An introduction to the R language

Véronique Martin & Sophie Schbath

INRAE - MaIAGE

March 8-9, 2021

Tour de table

- Qui êtes-vous ? (nom, unité, activité)
- Vos besoins par rapport à votre activité ?
- Vos attentes par rapport à la formation ?

Déroulé de la formation

- Deux jours : 8 et 9 mars 2021
- Horaires : 9h30 - 17h00
- Déjeuners : 12h30 - 14h00
- Pauses : 11h et 15h

Objectif pédagogique

À l'issue de la formation :

- les stagiaires connaîtront les principales fonctionnalités du langage R et ses principes.
- Ils seront capables de les appliquer pour effectuer des calculs ou des représentations graphiques simples.
- Ils seront autonomes pour manipuler leurs tableaux de données.

Attention : ce module n'est ni un module de statistique, ni un module d'analyse statistique des données.

Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

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

R is a **free and open** environment for computational statitics and graphics (Open source, Open development, under GNU General Public Licence).

ıbout R
Vhat is R?
Contributors
Screenshots
Vhat's new?

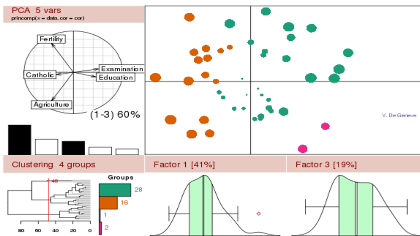
Download, Packages
CRAN

R Project
Foundation
Members & Donors
Mailing Lists
Bug Tracking
Developer Page
Conferences
Search

Documentation
Manuals
FAQs
The R Journal
Wiki
Books
Certification
Other

Misc
Bioconductor
Related Projects
User Groups
Links

The R Project for Statistical Computing



Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News:

- **R 2.14.1 prerelease versions** will appear starting December 12. Final release is scheduled for December 22, 2011.
- **useR! 2012**, will take place at Vanderbilt University, Nashville Tennessee, USA, June 12-15, 2012.
- **R version 2.14.0** (Great Pumpkin) has been released on 2011-10-31.
- **R version 2.13.2** has been released on 2011-09-30.
- **The R Journal Vol.3/1** is available.

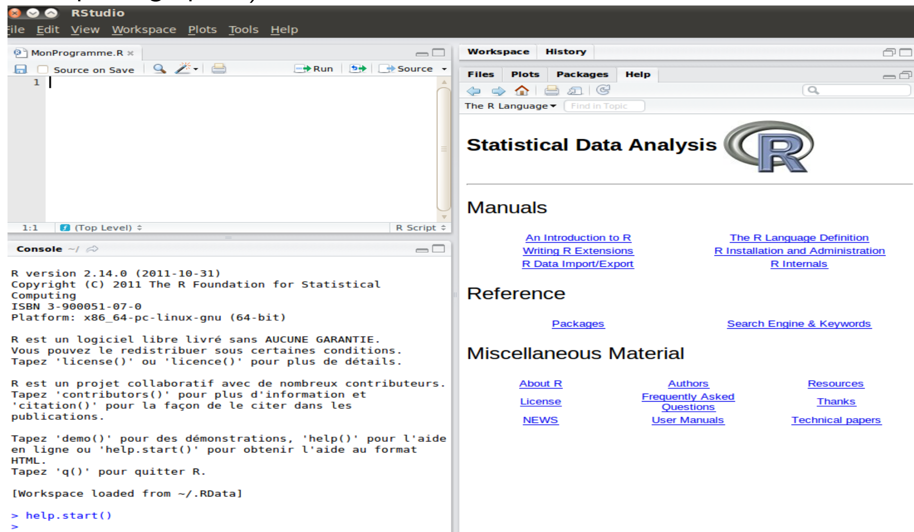
This server is hosted by the Institute for Statistics and Mathematics of the WU Wien.

Working environment

- R is an interpreted language
- There is no compilation
- One can work in the console (like during the practice moments) or in a script file (like for the exercices you will do)
- Good for interactive use of the language
- Bad for speed (when performing heavy computations)

RStudio https://www.rstudio.com/

Rstudio provides a nice front-end to R with 4 panel (script, console, workspace, graphics)



Rstudio server of Migale facility

For this training session, you will use the Rstudio server of the Migale facility :

1. access to `https://rstudio.migale.inrae.fr/`
2. and enter your training login/password

Console

The console is a glorified calculator,

- you submit some R code and press Enter
- R evaluates the expression and returns the answers

```
2+2
```

```
## [1] 4
```

Installing and loading packages

The main strength of `R` comes from the thousands of packages that provide nice functions and utilities to the language. Most are available from the CRAN (Comprehensive R Archive Network) and easy to install:

```
install.packages("ggplot2")
```

Loading packages is equally easy:

```
library(ggplot2)
```

Most packages must be loaded at each new session (see the “Packages” tab in R-studio)

Assigning a value to a variable

You can save the value of some R code using one of two ways:

- the "arrow operator": `<-` (or more rarely `->`)
- the "equal" sign: `=` (we recommend `<-` over `=`)

The syntax is simple: `variable_name <- value`.

```
a <- 2*4 # (the # sign signals a comment)
```

And you can access and manipulate the value of that variable

```
a # prints a

## [1] 8

a/2

## [1] 4
```

Changing the value of a variable

The arrow is also used to change the value of an object:

```
a <- 4
a

## [1] 4
```

Modifications made to a copy do no impact the original object:

```
b <- a; b <- 8 # use ";" to separate instructions on same line
a; b

## [1] 4
## [1] 8
```

Data in R

In R every *basic* object has four characteristics:

- a name: starting by a letter and containing only letters, digits, _ and dots
- a mode: mainly `numeric`, `logical` or `character`. It can be obtained via the `mode()` function
- a length: can be obtained via the `length()` function
- a content

Content and special values

Numeric	Character	Logical

`1/0`
`## [1] Inf`
`2/0 - 1/0`
`## [1] NaN`
	Use double (") quote
	`x <- "hello"`
	`mode(x)`
	`## [1] "character"`
		Can take only value TRUE or FALSE
		`x <- TRUE`
		`mode(x)`
		`## [1] "logical"`

Two other important special values in R : NA and NULL.

- NA stands for Not Available and is a code for missing data.
- NULL is the R code for a null object. It has length 0.

Data structures

R offers many data structures to organize data. We will see the 4 main ones:

- vector (1D array)
- matrix (2D array)
- list
- data.frame

Practice 1

Open the Rstudio interface and work in the console:

1. create a variable named "`height`" containing your height in meter
2. print the value of variable "`height`"
3. create a variable named "`weight`" containing your weight in kg
4. print the value of variable "`weight`"
5. compute and print the value of $\frac{\texttt{weight}}{\texttt{height}^2}$

Practice 2

Guess the result of the following code and check your guess in the console:

```
height <- 2
weight <- 80
bmi <- weight/(height^2)
bmi
weight <- 84
bmi
bmi <- weight/(height^2)
bmi
```

Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

Vectors: creation (1)

Multiple elements of the same mode (numeric, character, logical) can be collected in a `vector` (1D array) using the `c()` function:

```
x <- c(2, 4, 8, 9, 0) ; x

## [1] 2 4 8 9 0
```

```
length(x)

## [1] 5
```

Vectors containing regular sequences of numbers can be created thanks to the operator `:`

```
5:12 # consecutive integers

## [1]  5  6  7  8  9 10 11 12
```

or the `rep()` function

```
rep(0, times=5) # to repeat values

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

Vectors: creation (2)

The `seq()` function can also be used:

```
seq(from=2, to=20, by=2) # set the gap size

##  [1]  2  4  6  8 10 12 14 16 18 20

seq(from=2, to=20, length=5) # set the vector length

## [1]  2.0  6.5 11.0 15.5 20.0
```

Do not hesitate to check the help page of an R function if you forget its argument names.

Vectors: accessing/modifying component values

Elements of `x` can be accessed with the indexing operations:

```
x ; x[1] # first element

## [1] 2 4 8 9 0
## [1] 2
```

```
x[1:3] # 3 first elements

## [1] 2 4 8
```

```
x[c(3, 5)] # 3rd and 5th elements

## [1] 8 0
```

Elements of `x` can be modifyed with the indexing operations:

```
x[1] <- 10  # new value for the first element
x

## [1] 10  4  8  9  0
```

Vectors: removing or adding components

Elements of x can be filtered by using negative indexes:

```
x ; x[c(-3, -5)] ## all except the third and fifth elements

## [1] 10  4  8  9  0
## [1] 10  4  9
```

If one really wants to remove these elements from x:

```
x <- x[c(-3, -5)] ## all except the third and fifth elements
x

## [1] 10  4  9
```

To add new elements at the beginning or end of a vector, use c():

```
c(1,1,1,x,4:7)

##  [1]  1  1  1 10  4  9  4  5  6  7
```

Practice 3

Compare in the console the following instructions:

```r
n <- 8
1:(n-2)
1:n-2
```

Compare in the console the following instructions:

```r
rep(1:5, times=2)
rep(1:5, each=2)
```

Practice 4

Guess the result of the following code, check your guess in the console:

```
# Indexing
x <- c("O", "F", "I", "L", "R")
x[c(4, 3, 1, 2, 5)]
```

Matrices: creation

Matrices are essentially 2-D vectors: all elements must have the same mode.

```
x <- matrix(data=1:18, nrow=3, ncol=6)
x

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    4    7   10   13   16
## [2,]    2    5    8   11   14   17
## [3,]    3    6    9   12   15   18
```

Dimensions can be obtained via `dim()`, `ncol()`, `nrow()`; note the result of the `length()`

```
dim(x); length(x)

## [1] 3 6
## [1] 18
```

```
ncol(x); nrow(x)

## [1] 6
## [1] 3
```

Matrices: accessing values

Indexing works the same way than for vectors but with **two** indices: the first for rows, the second for columns.

```
x

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    4    7   10   13   16
## [2,]    2    5    8   11   14   17
## [3,]    3    6    9   12   15   18

x[2, 4] ## element in 2nd row, 4th column

## [1] 11
```

```
x[, 2] ## 2nd column

## [1] 4 5 6
```

```
x[2, ] ## 2nd row

## [1]  2  5  8 11 14 17
```

Matrices: adding/removing rows/columns

To remove rows or columns, use negative indexes:

```
x[,c(-1,-2)] ## remove 1st and 2nd columns

##      [,1] [,2] [,3] [,4]
## [1,]    7   10   13   16
## [2,]    8   11   14   17
## [3,]    9   12   15   18
```

To add a new column (resp. row), use the `cbind()` function (resp. `rbind()`); this new column (resp. row) should have the good length!

```
new.x <- cbind(x, c(0,1,0))
new.x

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

Practice 5

```
x
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    4    7   10   13   16
## [2,]    2    5    8   11   14   17
## [3,]    3    6    9   12   15   18
```

Try to guess what the following commands do, check in the console

```
x[ , c(1, 4, 6)]
x[c(1, 3), ]
x[c(1, 3), c(1, 4, 6)]
x[ , -1]
```

Practice 6

1. create the following matrix x

```
x

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

and modify its value as follows:

2. set the element on first row and 2nd column to value 5,
3. add a third column with values 1 and 2,
4. set the second row to (3,4,6),
5. and print the new value of x

List: creation

A `list` is an indexed collection of objects (also called **components**). These components **can have different type and different mode**. It is the preferred way to store possibly unrelated objects in one place.

```
myseq <- list(species = "E. coli", length = 3600000,
       composition = c(0.23, 0.31, 0.19, 0.27))
```

```
myseq

## $species
## [1] "E. coli"
##
## $length
## [1] 3600000
##
## $composition
## [1] 0.23 0.31 0.19 0.27
```

To know the number of components stored in a list:

```
length(myseq)

## [1] 3
```

`ls(myseq)` will return the component names.

List: accessing to components

Each component of a list can be individually accessed with `[[ ]]`:

```
myseq[[1]]

## [1] "E. coli"

mode(myseq[[1]])

## [1] "character"
```

```
myseq[[2]]

## [1] 3600000

mode(myseq[[2]])

## [1] "numeric"
```

or by using the operator `$` with the component names:

```
myseq$composition # equivalent to myseq[[3]]

## [1] 0.23 0.31 0.19 0.27
```

List: adding/removing components

To remove or select some components, use `[ ]`

```
myseq[1:2] # components 1 and 2

## $species
## [1] "E. coli"
##
## $length
## [1] 3600000
```

```
myseq[-2] #all except the 2nd

## $species
## [1] "E. coli"
##
## $composition
## [1] 0.23 0.31 0.19 0.27
```

To add a new component to a given list, use `c()`:

```
myseq.new <- c(myseq, strain="K12")
length(myseq.new) ; ls(myseq.new)

## [1] 4
## [1] "composition" "length"      "species"     "strain"
```

Practice 7

1. What's the difference between `myseq[3]` and `myseq[[3]]`? Check in the console.
2. Create a list named `trainee` and containing the 3 following components:
   - your first name
   - the year you entered your institution
   - the countries you went to from 2015

   Imagine you forgot to mention *Italy* in the set of countries you went to. Modify the present list `trainee` to add this country.

Data.frame: creation

A `data.frame` is a table-like structure used to store contextual data of different modes. Technically a `data.frame` is a **list** of equal-length **vectors** and/or **factors**[1].

```
x <- data.frame(age = c(23,42,18,31),
                city = c("Paris", "Rio", "Bath", "LA"),
                surname = c("Tom", "Ana", "Jim", "Joe"))
```

```
x
```

```
##   age  city surname
## 1  23 Paris     Tom
## 2  42   Rio     Ana
## 3  18  Bath     Jim
## 4  31    LA     Joe
```

A `data.frame` has two dimensions: rows and columns (just like a matrix)

```
dim(x);nrow(x);ncol(x)
```

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

[1]factors will not be treated in this tutorial; options(stringsAsFactors=FALSE)

Data.frame: accessing to elements

```
x[2, 3]
```

```
## [1] "Ana"
```

```
x[ , 2] # second column
```

```
## [1] "Paris" "Rio"   "Bath"  "LA"
```

When columns are named, they can also be accessed with the special operator $.

```
x$surname # or x[ , 3]
```

```
## [1] "Tom" "Ana" "Jim" "Joe"
```

```
x
```

```
##   age  city surname
## 1  23 Paris     Tom
## 2  42   Rio     Ana
## 3  18  Bath     Jim
## 4  31    LA     Joe
```

Warning: rows cannot be extracted as vectors

```
x[2 , ]
```

```
##   age city surname
## 2  42  Rio     Ana
```

```
class(x[2 , ])
```

```
## [1] "data.frame"
```

Data.frame: removing columns/rows

To remove or select some columns, use `[ ]`

```
x[1:2] # columns 1 and 2

##   age  city
## 1  23 Paris
## 2  42   Rio
## 3  18  Bath
## 4  31    LA
```

```
x[-2] #all except the 2nd

##   age surname
## 1  23     Tom
## 2  42     Ana
## 3  18     Jim
## 4  31     Joe
```

For rows, use the matrix syntax:

```
x[1:2,] # rows 1 and 2

##   age  city surname
## 1  23 Paris     Tom
## 2  42   Rio     Ana
```

```
x[-2,] #all except the 2nd

##   age  city surname
## 1  23 Paris     Tom
## 3  18  Bath     Jim
## 4  31    LA     Joe
```

Data.frame: adding columns

You can add columns (variables) to an existing data frame by using the `cbind()` function or directly as follows

```
x$year <- 2011:2014 # it changes x
x

##   age  city surname year
## 1  23 Paris     Tom 2011
## 2  42   Rio     Ana 2012
## 3  18  Bath     Jim 2013
## 4  31    LA     Joe 2014
```

or equivalently :

```
cbind(x,year=2011:2014) # x not changed
```

Note: `colnames(x)` gives the names of the columns.

The simplest way to import a tabulated text file[2] is `read.table()`. `read.table()` outputs a `data.frame` and is very flexible. Its main arguments are:

Argument	Description
`file`	File name, or complete path to file (can be an URL)
`header`	Are variable names there? (`FALSE` by default)
`sep`	Field separator character (white character by default)
`dec`	Character used for decimal points ("." by default)
`row.names`	Row names (or labels) where the rownames are stored.
`skip`	Number of top lines (usually comments) to skip.

Exemple

```
mydata <- read.table(file="data/virus-count-tri.tab", header=TRUE,
skip=1)
```

[2] think excel worksheet, but in text format

Data.frame: export data

Matrix-like objects (matrices, data.frame) can be exported as tabulated text files (human-readable) with `write.table()`. The typical use is:

```
## for tsv
write.table(matrix_object, file = "my_file.tsv", sep = "\t",
            quote = FALSE)
```

To save general objects as R -readable objects (more compact), use `save()` (and `load()` to load them back).

```
save(object1, object2, file = "data.Rdata")
load("data.Rdata")
```

Practice 8

Objectif : à partir du fichier `data/short-virus.tab` qui contient les longueurs et comptages des 9 premiers trinucléotides pour 3 virus, on souhaite créer un tableau de données contenant plutôt les fréquences des trinuclotides.

Avant

```
##    virus1 virus2 virus3
## 1   23814  20869  31787
## 2     728    760    833
## 3     453    422    531
## 4     588    553    463
## 5     732    503   1087
## 6     431    544    695
## 7     234    223    294
## 8     141    145    310
## 9     485    396    836
## 10    571    726    390
```

Après

```
##          virus1      virus2      virus3
## aaa 0.030572820 0.036421143 0.026207331
## aac 0.019024022 0.020223319 0.016705993
## aag 0.024693432 0.026501174 0.014566619
## aat 0.030740803 0.024105046 0.034198521
## caa 0.018100118 0.026069871 0.021865660
## cac 0.009826978 0.010686730 0.009249646
## cag 0.005921384 0.006948771 0.009753028
## cat 0.020367882 0.018977333 0.026301715
## gaa 0.023979506 0.034791776 0.012269939
```

Practice 8 : étapes

1. Importer le tableau de données original
2. Copier ce tableau dans un nouveau tableau de données (ne pas écraser le tableau d'origine)
3. Remplacer les comptages par les fréquences
4. Supprimer la première ligne
5. Modifier les labels de ligne (`aaa`, `aac` etc.)

Fundamental data types: summary

- `vector` (and `matrix`): 1-D (and 2-D) array of basic data, all of the same mode (integer, numeric, logical, character)
- `list`: indexed/labelled collection of objects that do not need to be of the same type or the same mode, and can be arbitrarily complex
- `data.frame`: used for experimental results, a table-like structure (technically, a list of equal-length vectors). All elements in a column have the same mode but different columns may have different modes.

Indexing: summary

- **position**: index elements by position in a `vector` (`x[i]`) or positions (row, column) in a `matrix/data.frame` (`x[i, j]`)
- **label**:
  - index elements by label if any in a `vector` (`x["first"]`) or in a `matrix/data.frame` (`x["row", "column"]`)
  - extract list components by using their labels (`myseq$composition`). Works for `data.frame` which are a special kind of `list` (`x$surname`)

More than one element (or row, column) can be indexed at the same time: `x[c(i1, i2, ..., in)]`

Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

Introduction

Graphics are displayed into the active graphical device which can be

- a graphical window (plot panel in Rstudio), by default
- or a pdf file, for instance.

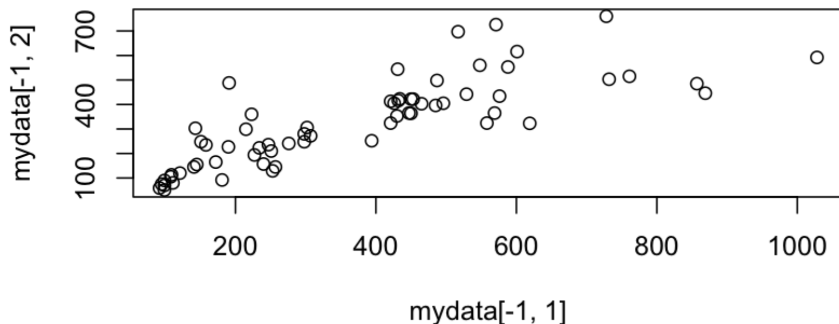
There are two kinds of graphical functions in `R` ("graphics" package):

- master functions that reset the graphical device: `plot()`, `hist()`, `boxplot()`, etc.
- secondary functions that allow to superimpose graphical elements to an existing graphics: `abline()`, `points()`, `legend()`, etc.

Scatter plot: basic

To plot a cloud of points in 2D (x-axis and y-axis), use `plot()`:

```
# the two vectors must have the same length
plot(x = mydata[-1,1], y = mydata[-1,2])
```



Scater plot: arguments

The `plot()` function has many arguments; its main arguments are:

Argument	Description
`main`	title of the graphics into " "
`pch`	point symbol; can be any integer in 1:25 or any character into " "
`xlab`, `ylab`	axis labels into " "
`xlim`, `ylim`	vectors (length 2) setting the range of coordinates
`col`	color[3] of points (eg. "blue")
`cex`	multiplicative coefficient for point size (by default `cex=1`)
`log="x"`	to use a logaritmic scale on x-axis (use `y` or `xy` for y-axis or both)
`type`	"p" (for points), "l" (for lines), "b" (both); by default `type="p"`
`lty`	for line type (by default `lty=1`; 2 for "-"; 3 for "...", up to 6.)

To change the size of the title, the axes or their labels: `cex.main`, `cex.axis`, `cex.lab`

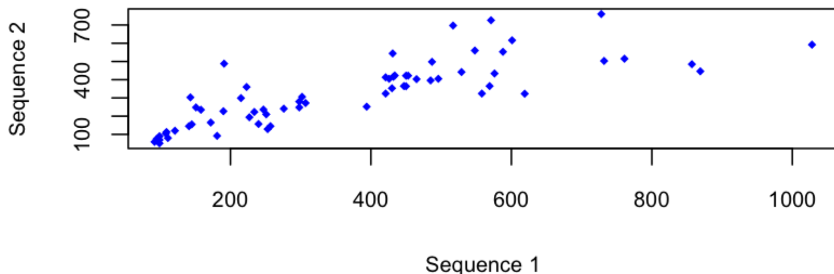
To change the color of the title, the axes or their labels: `col.main`, `col.axis`, `col.lab`

[3]Possible colors can be obtain via `colors()`.

Scatter plot

With the same data as previously, one can then obtain:

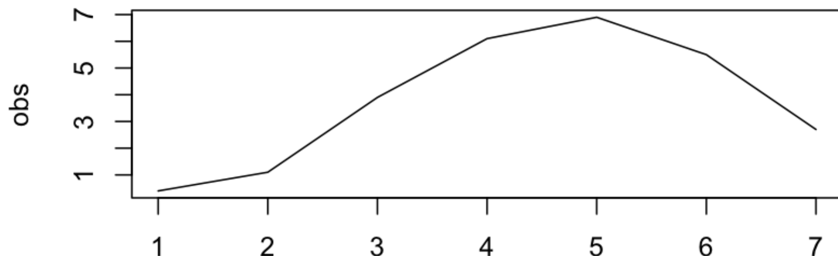
```
plot(x = mydata[-1,1], y = mydata[-1,2],
     main="3-mer counts in the first two sequences", cex.main=0.7,
     xlab = "Sequence 1", cex.lab=0.8, cex=0.7, cex.axis=0.8,
     ylab = "Sequence 2", pch = 18, col = "blue")
```



Curves (I)

To draw curves or time series, one needs to join consecutive points thanks to the `type="l"` argument of the `plot()` function:

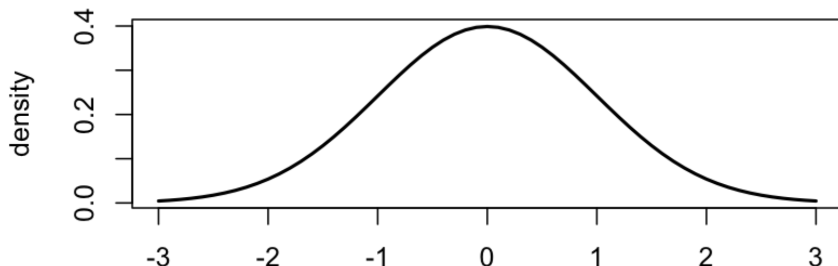
```
t <- 1:7 # times of observation (in hours for instance)
obs <- c(0.4, 1.1, 3.9, 6.1, 6.9, 5.5, 2.7)
plot (x=t, y=obs, main="Evolution of the variable", type="l")
```



Curves (II)

Another example (use argument `lwd` to change the line width):

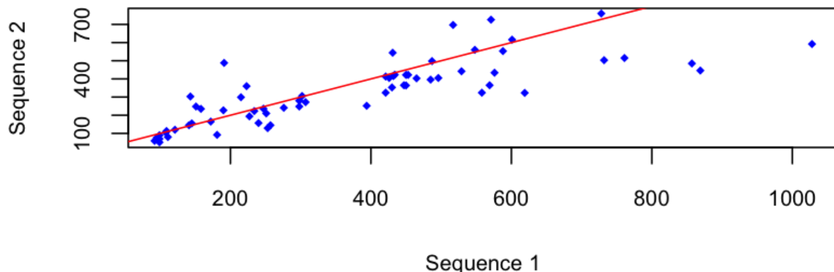
```
xdisc <- seq(from=-3, to=3, by=0.1) # discretization of [-3 , 3]
plot (x=xdisc, y=dnorm(xdisc), xlab="", ylab="density",
      main="Gaussian distribution N(0,1)", type="l", lwd=2)
```



Adding a line

To superimpose other graphical elements, one has to use secondary graphical functions like `abline()`:

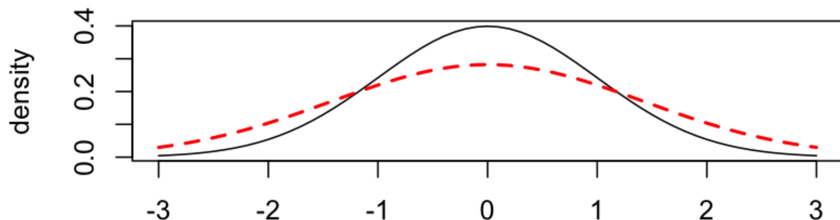
```
abline (a = 0, b = 1, col="red") # draw the y=a+bx line
# to draw vertical lines, use argument v= instead of a and b
```



Adding a curve or a set of points

Another example with `points()`

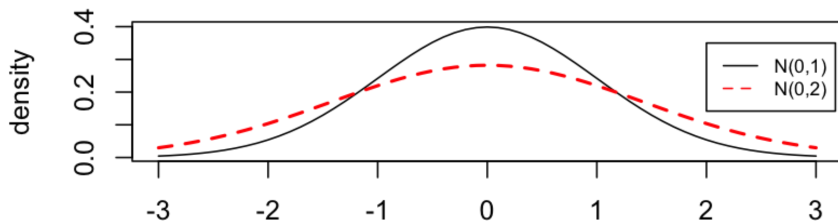
```
xdisc <- seq(from=-3, to=3, by=0.1) # discretization of [-3 , 3]
plot (x=xdisc, y=dnorm(xdisc), xlab="", ylab="density",
      main="Gaussian distributions N(0,1) and N(0,2)", type="l")
points(x=xdisc, y=dnorm(xdisc, sd=sqrt(2)), col="red", type="l",
       lty=2, lwd=2)
```



Adding a legend

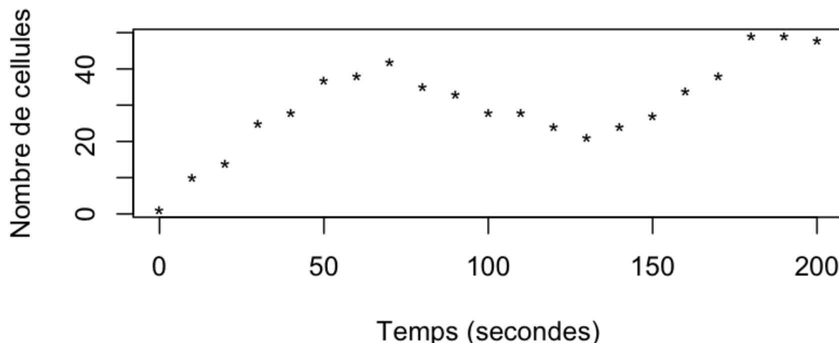
Use the `legend()` function:

```
legend(x=2, y=0.35, cex=0.7, legend=c("N(0,1)","N(0,2)"),
       lty=c(1,2), col=c("black","red"))
# argument order doesn't matter if argument names are used
```



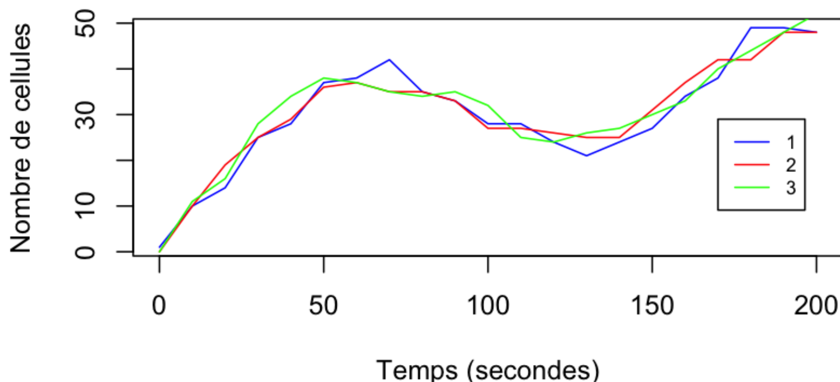
Practice 9

Dans le répertoire `data` se trouvent 20 fichiers nommés `cinetique.1` à `cinetique.20`. Ils contiennent, pour 20 expériences différentes, le nombre de cellules observées à différents temps.
Faire un script R qui reproduit la courbe de cinétique associée à l'expérience 1.



Practice 10

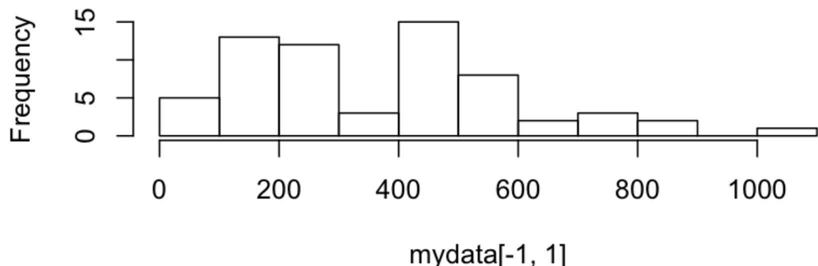
Faire un script R avec la superposition des cinétiques des 3 premières expériences. Pour distinguer chacune des courbes, on utilisera différentes couleurs et on ajoutera la légende correspondante.



Histogram: basic

The `hist()` function is a master one like `plot()`

```
hist(x=mydata[-1,1], main = "3-mer counts in sequence 1")
```

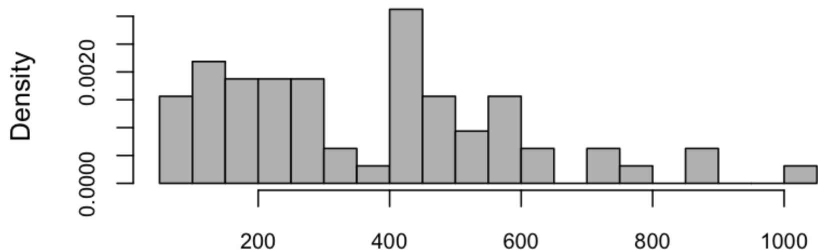


Have a look to the various arguments of the `hist()` function (Help); `proba=TRUE` allows to represent densities instead of counts.

Histogram

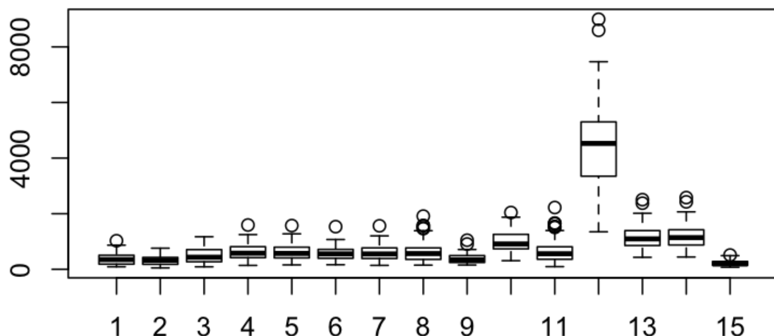
With the same data as previously, one can then obtain:

```
hist(x=mydata[-1,1], main="3-mer counts in sequence 1",
     proba=TRUE, xlab="", breaks=20, col="grey", cex.axis=0.8)
```



Boxplot: basic

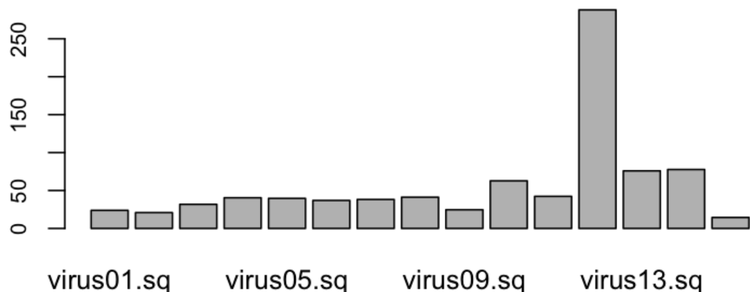
```
counts<-mydata[-1,1:15] # extraction of counts for seq 1 to 15
colnames(counts)<-1:15 # to rename the columns
boxplot (x=counts, main="3-mer counts for each sequence")
```



Barplot: basic

To graphically represent a **vector**:

```
long<-mydata["long",1:15] # extraction of length of seq 1 to 15
vect.long <- colSums(long) # trick to convert a data frame row into vector
barplot(height=vect.long/1000, cex.axis=0.8, main="Length of seq 1 to 15 (kb)")
```



To go further with graphics in R

Use the ggplot2 package:

- Migale training #12bis: "*Graphics in R with ggplot2*"
- Many tutorials on the web.

Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

for loop: syntax

A for loop is recommended as soon as some instructions have to be repeated a given number of times in the code. For instance, the code from "Solution 8"

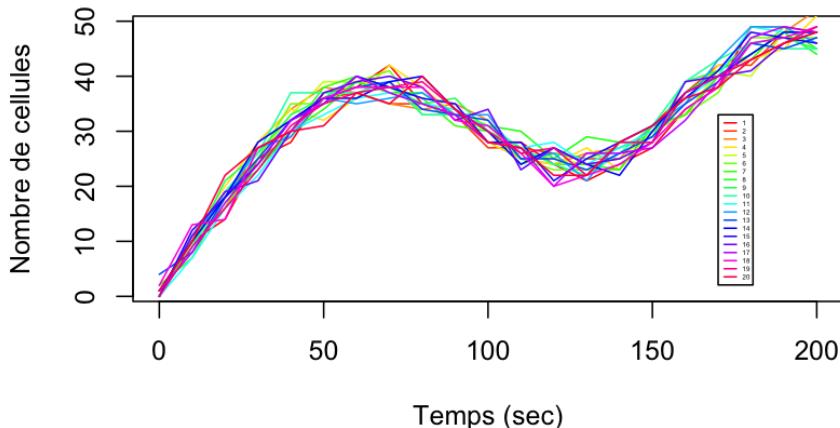
```
newdf[1]<-df[1]/(df[1,1]-2)
newdf[2]<-df[2]/(df[1,2]-2)
newdf[3]<-df[3]/(df[1,3]-2)
```

is equivalent to

```
for (i in 1:3){
  newdf[i]<-df[i]/(df[1,i]-2)
}
```

Practice 11

Dans la continuité des Practices 9 et 10, faire un script R qui produit la superposition des 20 courbes de cinétique.



Practice 11 (suite)

Indication: the `paste()` function allows to create a new character string by concatenation as follows:

```
directory<-"data"
paste(directory, "/", "cinetique.1", sep="")

## [1] "data/cinetique.1"

i<-3
paste(directory, "/", "cinetique.", i, sep="")

## [1] "data/cinetique.3"
```

By default, the separator `sep=" "`

Test `if`: syntax

Some instructions may be executed only `if` a given condition is true. For instance

```
if (length(v1)==length(v2)){
  plot(x=v1,y=v2)
}else{
cat("Error: v1 and v2 do not have the same length")
}
```

The "else" instructions are not mandatory; however, if there are "else" instructions, the keyword `else` must be placed on the same line than the } of the "if" instructions.

The comparison operators are: `==`, `>`, `>=`, `<`, `<=`.
Logical conditions can be associated with `&` (and), `|` (or), `!` (no).

Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

Vector operations

One can compute with vectors as one computes with single elements:

```
x <- 1:5 ; x

## [1] 1 2 3 4 5

y <- c(1, -4, 1, 0, -10) ; y

## [1]   1  -4   1   0 -10
```

```
x+y

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

x*y

## [1]   1  -8   3   0 -50

x^2

## [1]  1  4  9 16 25
```

Some mathematical functions (1)

Functions `sum()`, `min()`, `which.min()`

```
x<-mydata$virus01
sum(x) # computes the sum of all the components of x

## [1] 47620

min(x) # returns the minimum value of x

## [1] 92

which.min(x) # returns the index where x is minimum

## [1] 40
```

Guess what the following functions do: `prod(x)`, `max(x)`, `which.max(x)`

Check in the console what does the `colSums()` function:

```
colSums(mydata[,c(8,9,10)])
```

Some mathematical functions (2)

Here are some functions to round numbers:

```
round(1.234567, digits=3)

## [1] 1.235

floor(1.234567) # integer part (inferior)

## [1] 1

ceiling(1.234567) # integer part (superior)

## [1] 2
```

Sorting a vector can be also very useful:

```
sort(c(5,7,2,4,4,8,1)) # decreasing=FALSE by default

## [1] 1 2 4 4 5 7 8
```

Statistical quantities

```r
mean(x) # computes the empirical mean of the components of x

## [1] 732.6154

median(x) # ... the mediane

## [1] 394

var(x) ; sd(x) # ... the empirical variance and standard deviation

## [1] 8501833
## [1] 2915.79

y <- mydata$virus04.sq
cor(x,y, method="spearman") # ... the spearman correlation

## [1] 0.931429
```

Practice 12

On lance 155 fois un dé à 6 faces et on note le nb de fois où chaque chiffre a été tiré :

1	2	3	4	5	6
28	21	30	23	20	33

Pour tester si le tirage aléatoire est équilibré (H_0), cad si chaque chiffre a une proba $1/6$, on effectue classiquement un test du Chi-deux. Cela consiste à comparer les effectifs observés o_i (dans le tableau ci-dessus) avec les effectifs théoriques attendus t_i ($155/6$ ici) en calculant la statistique de test :

$$s = \sum_{i=1}^{n} \frac{(o_i - t_i)^2}{t_i} \text{ où } n \text{ est le nombre de valeurs possibles, } n = 6 \text{ ici.}$$

Sous H_0, s suit une loi du Chi-deux à $(n-1)$ degrés de liberté : $\chi^2(n-1)$.

Faire un script R qui calcule la statistique de test s et la p-value $\mathbb{P}(\chi^2(n-1) \geq s)$.

Note: `pchisq(q=s, df=d)` returns the cumulative probability $\mathbb{P}(\chi^2(d) < s)$.

Probability distributions

Many probability distributions are implemented. One can easily:

- generate n random variables X from a given distribution (ex: `rnorm(n, ...)` for the Normal distribution)
- compute the cumulative probability $P(X < q)$ for a given quantile q (ex: `pnorm(q, ...)`)
- compute the quantile q associated to a given cumulative probability p (ex: `qnorm(p, ...)`)
- compute the density function at x (ex: `dnorm(x, ...)`)

`(...)` stands for the parameter of the given distribution.

Probability distributions (II)

Distribution	function	parameters by default
Normal	`rnorm()`	`mean=0, sd=1`
Uniform	`runif()`	`min=0, max=1`
Poisson	`rpois()`	`lambda`
Exponential	`rexp()`	`rate=1`
χ^2	`rchisq()`	`df`
Binomial	`rbinom()`	`size, prob`

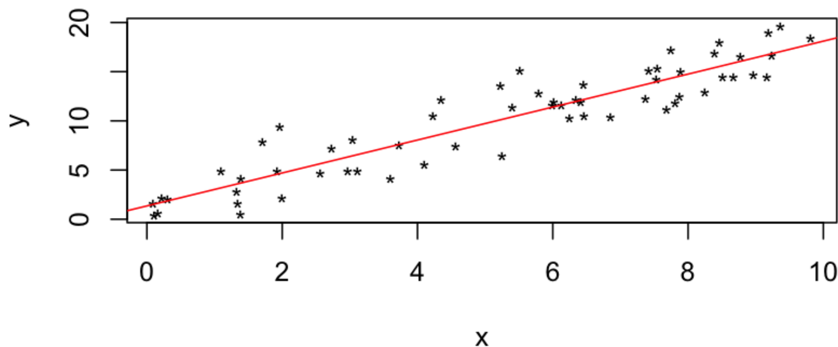
Practice 13

Le fichier `data/donnees.txt` contient un ensemble de mesures (abscisses et ordonnées).

Tracer le nuage de points avec sa droite de régression.

Note : la fonction `lm(y~x)` fait la régression linéaire de `y` en fonction de `x` et retourne une liste de plusieurs objets, dont les coefficients de la droite de régression (`$coef`).

Practice 13 (suite)

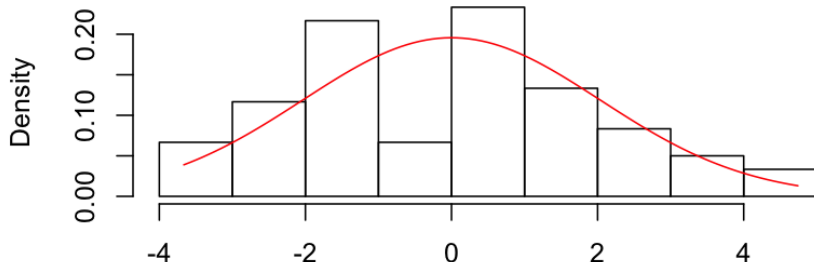


Optionnel : En guise de sous-titre (option `sub` de la fonction `plot`), on affichera l'équation de la droite de régression (les coefficients n'auront que deux chiffres après la virgule).

Practice 13bis

Tracer l'histogramme de densité des résidus $\varepsilon = y - a - bx$ (où $a + bx$ est l'équation de la droite de régression).

Y superposer la densité gaussienne de même moyenne et de même écart-type que les résidus ε.



A few pointers for beginners

- http://www.r-project.org/
- http://www.bioconductor.org/help/publications/

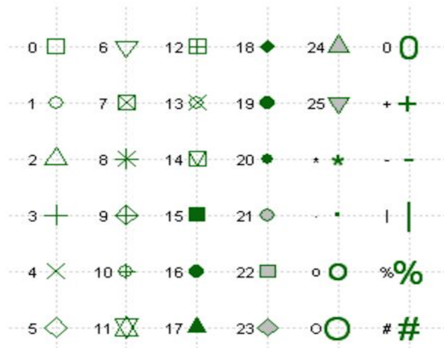
Outline

1. Introduction
2. Simple data structures
   - Vectors
   - Matrices
   - Lists
   - Data frames
   - Summary
3. Basic graphics
   - Scatter plots and curves
   - Histograms, barplots, boxplots
4. Towards programming
   - For loop
   - Test if
5. Doing mathematics and probabilities
   - Mathematical calculus
   - Probability distributions and simulations
6. Appendix

Graphics: pch option

Plotting Symbols

Use the **pch=** option to specify symbols to use when plotting points. For symbols 21 through 25, specify border color (col=) and fill color (bg=).

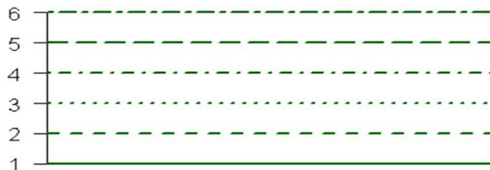


Graphics: lty option

Lines

You can change lines using the following options. This is particularly useful for reference lines, axes, and fit lines.

option	description
lty	line type. see the chart below.
lwd	line width relative to the default (default=1). 2 is twice as wide.



Logical Indexing

A vector `x` can be indexed by a logical vector `index` specifying which elements should be kept. In that case, `index` and `x` should have the **same length**...

```
x <- 10:15 # = c(10, 11, 12, 13, 14, 15)
index <- c(TRUE, FALSE, TRUE, TRUE, FALSE, FALSE)
x[index] # = x[c(1, 3, 4)]

## [1] 10 12 13
```

...otherwise strange things can happen.

```
index <- c(TRUE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE)
x[index] ## = x[c(1, 3, 4, 7)] but x[7] does not exist

## [1]  1  3  4 NA
```

Building logical filters (I)

R provides a built-in way to build logical indexes using logical operations (*e.g.* to filter data)

```
x <- 1:5
z <- (x < 3); z ## the first command returns a logical vector

## [1]  TRUE  TRUE FALSE FALSE FALSE

z <- (x < 4) & (x > 1); z ## logical AND

## [1] FALSE  TRUE  TRUE FALSE FALSE

z <- (x < 2) | (x > 4); z ## logical OR

## [1]  TRUE FALSE FALSE FALSE  TRUE

!z ## logical NOT

## [1] FALSE  TRUE  TRUE  TRUE FALSE
```

Building logical filters (II)

The logical indexes can be transformed to integer indexes using `which`

```
which(z)

## [1] 1 5
```

and used to extract part of the data

```
z <- (x < 4)
x[z]

## [1] 1 2 3

## or equivalently
x[x < 4]

## [1] 1 2 3
```

Practice 4bis

Guess the result of the following code, check your guess in the console:

```
# Filtering
y <- (-3:2)^2
which(y>2)
y[y>2]
```

apply() function

To apply any defined function (eg `mean()`) to each column of a matrix/data frame, use the function `apply()`:

```
counts<-mydata[,8:10]
apply(counts, MARGIN = 2, FUN = mean)

## virus08.sq virus09.sq virus10.sq
##  1266.7538   758.5846  1930.1231
```

(it generalizes the `colSums()` function).

For rows: just set `MARGIN=1`

Create our own function: syntax

It is possible to create your own function and to use it like other R functions. For that, you have to choose the name function (not an already existing one), the name of the arguments (eventually their default value).

```
bmi <- function(weight, height){
  res <- weight/(height^2)
  return(res)
}
bmi(weight=80, height=1.7)

## [1] 27.68166
```

Note that a function can only return a unique object (use a list if more than one object to return).

Practice 14

Only for pedagogical purpose, create a function which takes as argument `x` and returns its absolute value. Since this function already exists in R (`abs()`), choose another name.

Do not forget to test your new function!

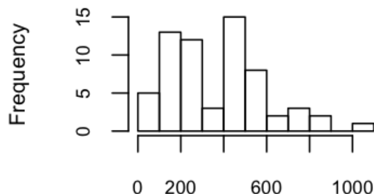
Practice 12bis

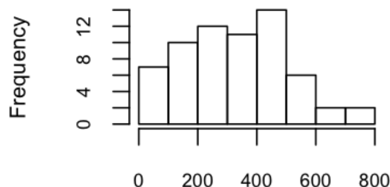
Improve Practice 12 by creating a function that takes the vector of observations as input and returns both the χ^2 statistic and the associated p-value.

Multiple display

To draw several graphics on the same page, split the page into rows/columns thanks to the option `mfrow` of the `par()` function which sets general parameters of the graphical device.
Warning: do not forget to put them back to their original values:

```
par(mfrow=c(1,2)) # 1 row but 2 columns
hist(x=mydata[-1,1], main = "3-mer counts in seq 1", cex.axis=0.7, xlab="")
hist(x=mydata[-1,2], main = "3-mer counts in seq 2", cex.axis=0.7, xlab="")
par(mfrow=c(1,1))
```





Exporting a figure

To save the content of the graphical device **from the console**:

```
dev.copy(device = pdf, file = "myfigure.pdf")
dev.off() # do not forget this line to properly close the pdf devic
```

To draw a graphics directly into a pdf file, without using the console:

```
pdf(file = "myfigure-2.pdf")
plot(x = mydata[-1,1], y = mydata[-1,2], col="red")
dev.off() # do not forget this line to properly close the pdf devic
```

You can also use the Rstudio interface.