Introduction au logiciel R

11-13 janvier 2017

Intervenant : Cyril Dalmasso
Support : Julien Chiquet - Cyril Dalmasso

Laboratoire de Mathématiques et Modélisation d'Évry

Programme

PART 1: Introduction
PART 2: Structures de données
PART 3: Entrées, sorties et statistique descriptive
PART 4: Programmation

Première partie I

Introduction

Plan

Introduction

Qu'est-ce que R ?

En deux mots,

R est un logiciel de développement scientifique spécialisé dans le calcul et l'analyse statistique.

R est aussi

- un langage,
- un environnement,
- un projet open source (projet GNU),
- un logiciel multi-plateforme (Linux, Mac, Windows),

Principales fonctionnalités

1. Gestionnaire de données
   - Lecture, manipulation, stockage.
2. Algèbre linéaire
   - Opérations classiques sur vecteurs, tableaux et matrices
3. Statistiques et analyse de données
   - Dispose d'un *grand* nombre de méthodes d'analyse de données (des plus anciennes et aux plus récentes)
4. Moteur de sorties graphiques
   - Sorties écran ou fichier
5. Système de modules
   - Alimenté par la communauté
6. Interface « facile » avec `C/C++`, `Fortran`,...

Historique

Approche chronologique

- 1970s développement de S au Bell labs.
- 1980s développement de S-PLUS au AT&T. Lab
- 1993 développement de R sur le modèle de S par Robert Gentleman et Ross Ihaka au département de statistique de l'université d'Auckland.
- 1995 dépôts des codes sources sous licence GNU/GPL
- 1997 élargissement du groupe
- 2002 la fondation R dépose ses statuts sous la présidence de Gentleman et Ihaka
- 2007 création de revolution analytics
- 2011 première version public de R-studio
- 2015 Microsoft Opensource R

Mode de diffusion

Développement

- « R development core team » (20aine de personnes)
- Participation de *nombreux* chercheurs (>7000 packages en Août 2015)
- Avec l'essor du « Big Data », apparition d'outils payants « autour de R » (revolution, rstudio entreprise, etc.) version payante

Qualités et défauts de R

Plus

1. Libre et gratuit,
2. Richesse des modules,
3. Souplesse,
4. Prise en main rapide (Syntaxe intuitive et compact),
5. Développement rapide (langage de scripts),
6. Nombreuses possibilités graphiques.

Qualités et défauts de R

Moins

1. Aide intégrée succincte,
2. Code parfois illisible (compacité),
3. Facile de « mal » coder,
4. Lent par rapport à C/C++,
5. Personnalisation des graphiques un peu lourde.

Les concurrents plus ou moins directs

Les logiciels de développement scientifique sont spécialisés en

1. algèbre linéaire
   - `Matlab` (Mathworks), la référence,
   - `Scilab` (INRIA), l'alternative libre,
   - `Octave` (GNU), l'alternative open source,
2. statistiques
   - `SAS` (SAS Inc.), la référence,
   - `S-PLUS` (TIBCO), le concurrent,
   - `R` (GNU), l'alternative open source,
3. calcul symbolique
   - `Mathematica` (Wolfram), la référence,
   - `Maple` (Maplesoft), la référence aussi,
   - `Maxima` (GNU), l'alternative open source,

+ `Python` + `SciPy` et `Julia`

Se tenir informé

1. La page web de la fondation R
   - les statuts, des liens, des références.
   - http://www.r-project.org/
2. La page web du CRAN (Comprehensive R Arxiv Network)
   - binaires d'installation, packages, documentations, ...
   - http://cran.r-project.org/
3. La conférence des utilisateurs de R
   - annuelle, prochaine édition à Stanford
   - http://user2016.org/
4. *The R journal* propose des articles sur
   - de nouvelles extensions, des applications, des actualités.
   - http://journal.r-project.org/

Se tenir informé

1. RSTUDIO,
   - Interface de développement multiplateforme pour R
   - binaires d'installation, packages, documentations, blog, etc.
   - https://www.rstudio.com/

2. REVOLUTION ANALYTICS, version « entreprise » de R
   - Support technique, développement spécifiques
   - passage à l'échelle/bigData orienté
   - http://www.revolutionanalytics.com

3. Datacamp, une plate-forme de cours en ligne
   - MOOC ne nécessitant pas d'installation préalable de R
   - https://www.datacamp.com/
   - voir aussi http://tryr.codeschool.com/

4. Blogs et plateformes alimentés par la communauté
   - http://www.inside-r.org/
   - http://www.r-statistics.com/

Plan

Introduction

Installation

Rendez-vous sur la page du CRAN `http://cran.r-project.org/`

Mac

Télécharger `R-3.3.0.pkg`, cliquer.

Windows

Télécharger `R-3.3.0-win32.exe`.

Linux

Systèmes supportants `apt` (Debian, Ubuntu, ...)

```
$ sudo apt-get up
$ sudo apt-get in
```

Lancer R

Dans un terminal, taper 'R'

Premiers pas (mode console)

```
$ R
R version 3.3.0 (2016-05-03) -- "Suppo
Copyright (C) 2016 The R Foundation fo
Platform: x86_64-pc-linux-gnu (64-bit)
[...]
Tapez 'demo()' pour des démonstrations
en ligne ou 'help.start()' pour obteni
Tapez 'q()' pour quitter R.
> 1+1
[1] 2
```

Quitter R

```
> q()
Save workspace image? [y/n/c]:y
```

$\rightsquigarrow$ Sauve l'environnement et le réouvre au démarrage de la session suivante

Organiser un projet R

Solution sans R-studio)

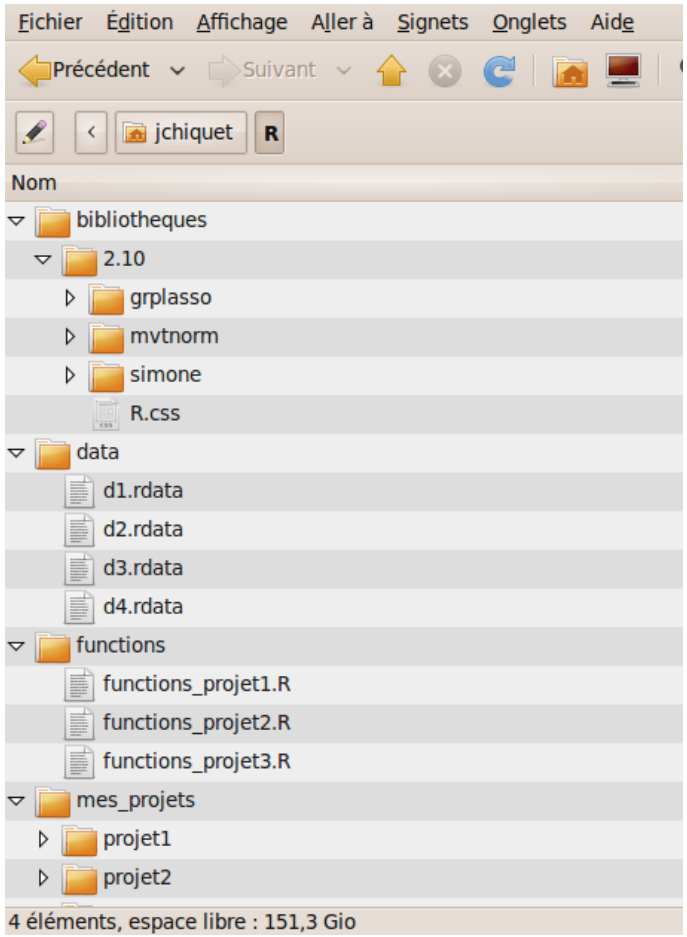


FIGURE : Arborescence type

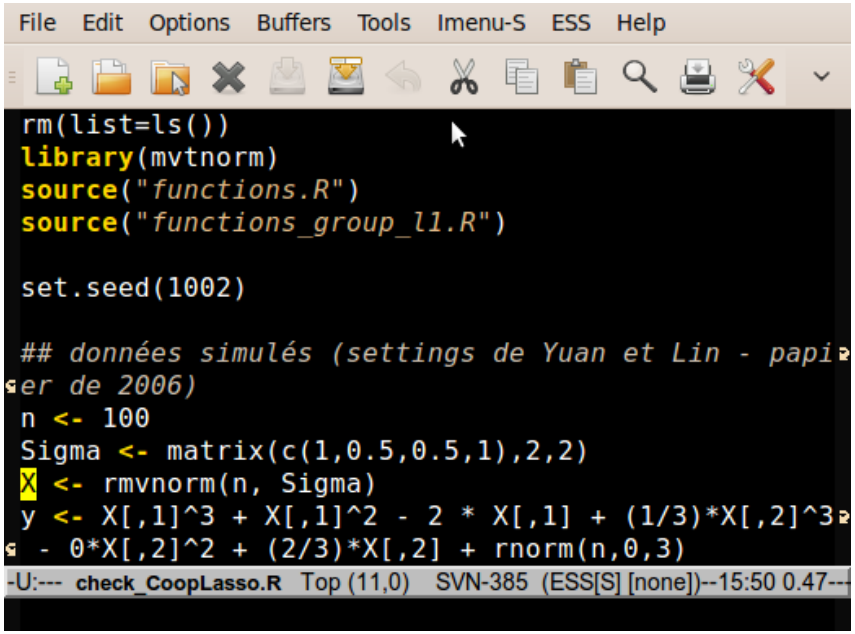
- Dans un répertoire R, placer
  - un répertoire `data`
  - un répertoire `mes_projets`
  - un répertoire `functions`
- Créer un répertoire par projet
  - sauvegarde des données

    `save.image(file = "f.RData")`
  - sauvegarde des instructions

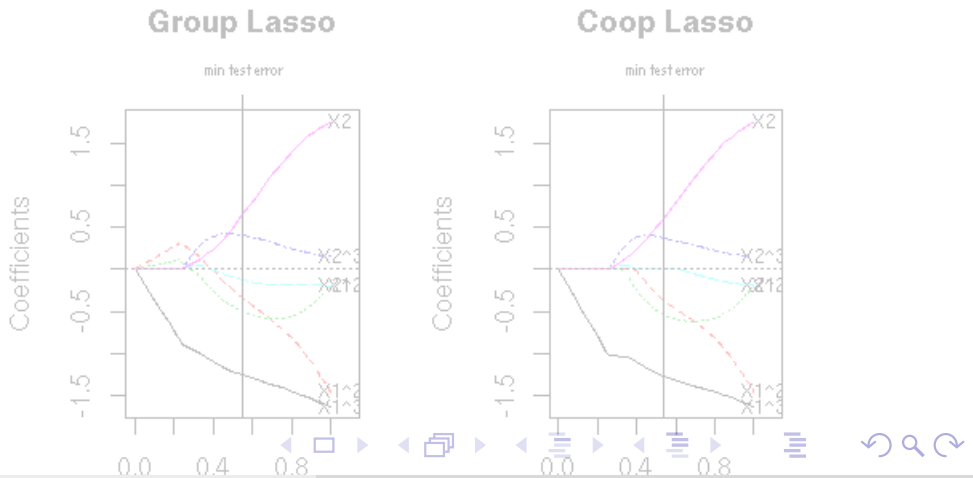
    `savehistory(file = "f.Rhistory")`
- `bibliotheques` contient les extensions installées.

Environnement de travail sous Linux

Un bureau de développement avec R(solution sans R-studio)

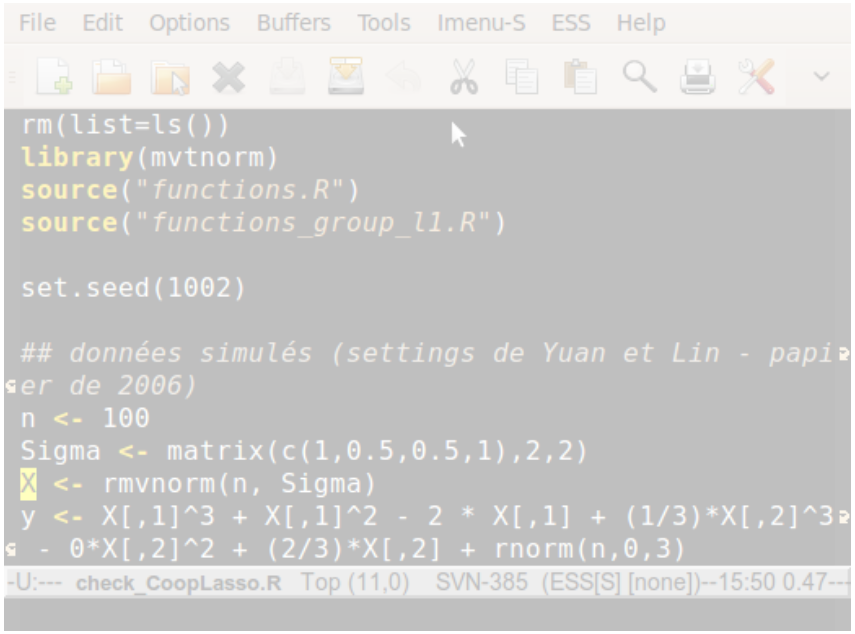


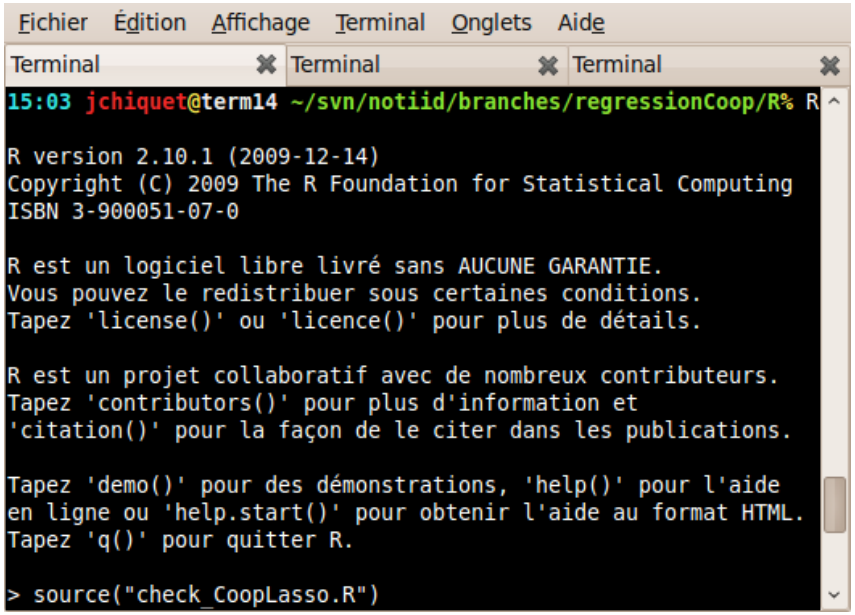
1. un éditeur de texte
2. un terminal avec R
3. des sorties graphiques



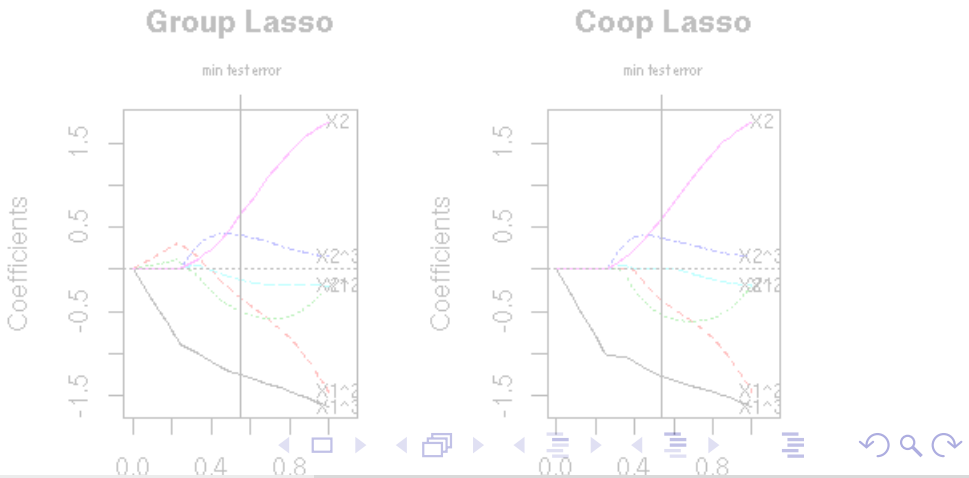
Environnement de travail sous Linux

Un bureau de développement avec R(solution sans R-studio)



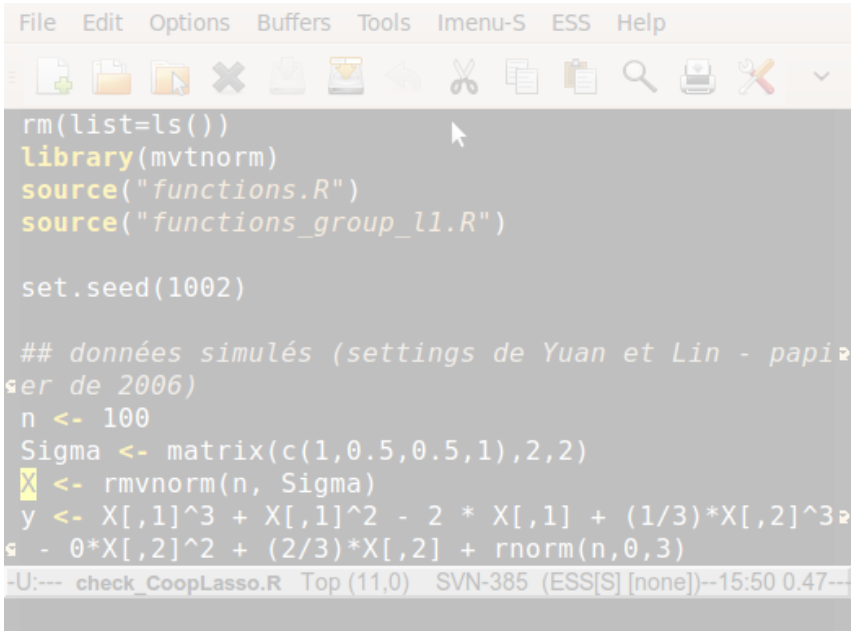


1. un éditeur de texte
2. un terminal avec R
3. des sorties graphiques

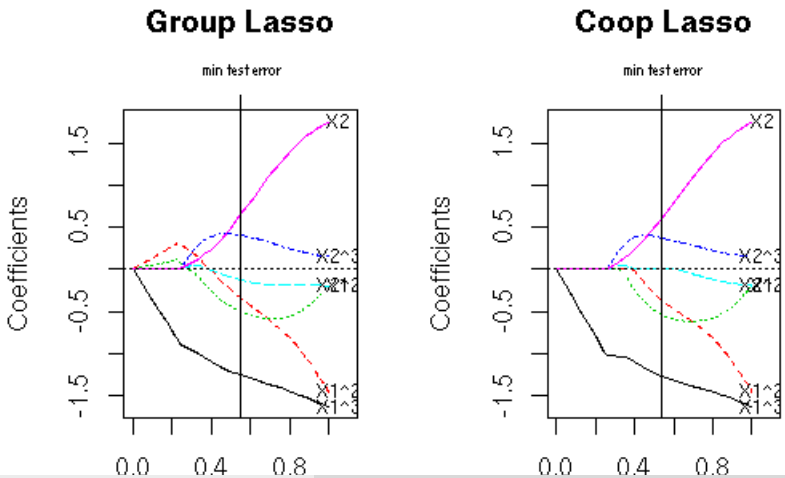


Environnement de travail sous Linux

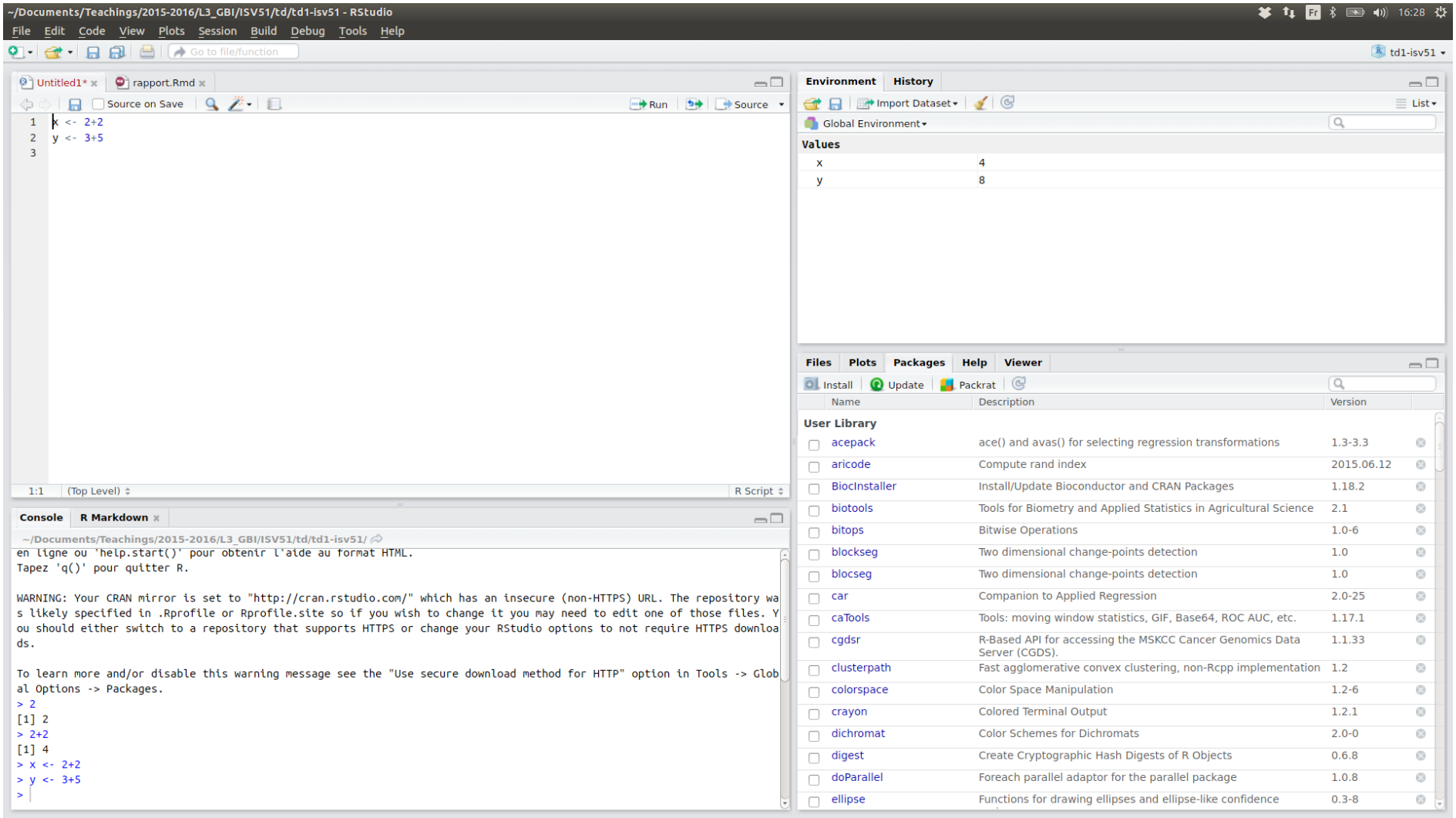
Un bureau de développement avec R(solution sans R-studio)



1. un éditeur de texte
2. un terminal avec R
3. des sorties graphiques



R-studio, environnement de travail intégré



Trouver de l'aide

Depuis R

- `help(str)` : lance l'aide associée à la commande `str`,
- `help.search("factorial")` : cherche les commandes contenant le mot-clé `factorial`,
- `help.start()` : lance l'aide HTML.

Sur le Web

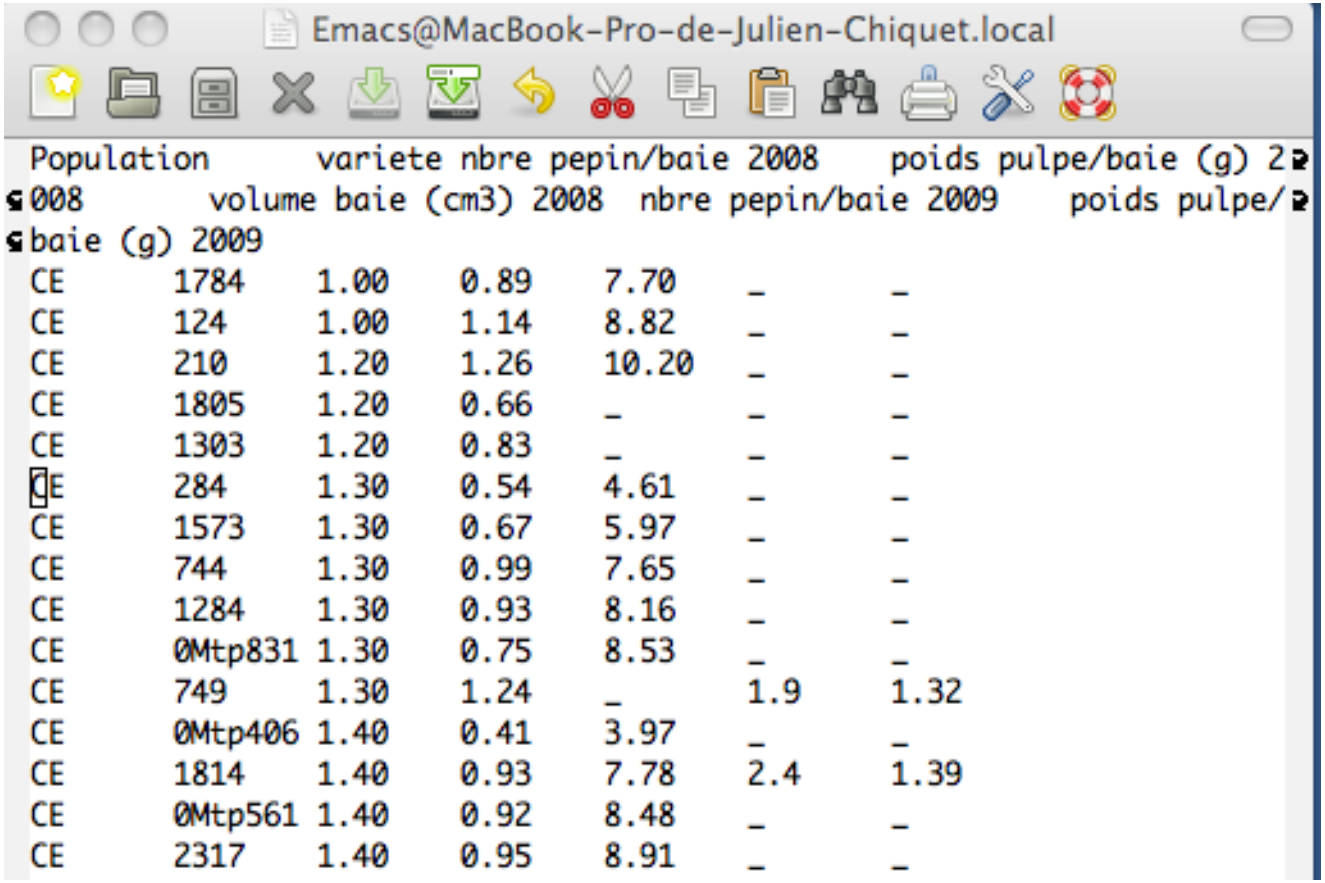
En utilisant les media mentionné précédemment.

Plan

Introduction

Analyse élémentaire d'un jeu de données

Quelle tête ont les données ? On ouvre avec Emacs :



Population	variete	nbre pepin/baie 2008	poids pulpe/baie (g) 2008	volume baie (cm3) 2008	nbre pepin/baie 2009	poids pulpe/baie (g) 2009
CE	1784	1.00	0.89	7.70	_	_
CE	124	1.00	1.14	8.82	_	_
CE	210	1.20	1.26	10.20	_	_
CE	1805	1.20	0.66	_	_	_
CE	1303	1.20	0.83	_	_	_
CE	284	1.30	0.54	4.61	_	_
CE	1573	1.30	0.67	5.97	_	_
CE	744	1.30	0.99	7.65	_	_
CE	1284	1.30	0.93	8.16	_	_
CE	0Mtp831	1.30	0.75	8.53	_	_
CE	749	1.30	1.24	_	1.9	1.32
CE	0Mtp406	1.40	0.41	3.97	_	_
CE	1814	1.40	0.93	7.78	2.4	1.39
CE	0Mtp561	1.40	0.92	8.48	_	_
CE	2317	1.40	0.95	8.91	_	_

FIGURE : données baies de vignes 2008/2009

Importation des données I

`getwd()` et `setwd()` gèrent le répertoire de travail :

```
setwd("/home/Enseignement/2015_2016/FE
getwd()
```

```
## [1] "/home/Enseignement/2015_2016/F
```

Importation des données II

Contenu du répertoire de travail :

```
dir()
```

```
##  [1] "figures0"
##  [2] "main.aux"
##  [3] "main.log"
##  [4] "main.nav"
##  [5] "main.out"
##  [6] "main.pdf"
##  [7] "main.Rnw"
##  [8] "main.snm"
##  [9] "main.tex"
## [10] "main.toc"
## [11] "mesures_baie_raisin_2008-2009
```

Chargeons les données (délimitation par des tabulations)

Importation des données III

```
donnees <- read.delim("mesures_baie_ra
```

Importation des données IV

Contenu de l'itinéraire de recherche :

```
ls()
```

```
##  [1] "C"          "cov.plot"   "don
##  [6] "matgeno"    "matpheno2"  "mat
## [11] "pheno"      "variable"
```

```
objects()
```

```
##  [1] "C"          "cov.plot"   "don
##  [6] "matgeno"    "matpheno2"  "mat
## [11] "pheno"      "variable"
```

Importation des données V

Quelle « tête » (au sens propre !) ont les données ?

```
head(donnees)
```

```
##   Population variete nbre.pepin.bai
## 1         CE    1784
## 2         CE     124
## 3         CE     210
## 4         CE    1805
## 5         CE    1303
## 6         CE     284
##   volume.baie..cm3..2008 nbre.pepin
## 1                   7.70
## 2                   8.82
## 3                  10.20
## 4                     NA
## 5                     NA
## 6                   4.61
```

Importation des données VI

Quelles sont ses attributs ?

```
str(donnees)
```

```
## 'data.frame': 245 obs. of  7 variab
##  $ Population               : Facto
##  $ variete                  : Facto
##  $ nbre.pepin.baie.2008     : num
##  $ poids.pulpe.baie..g..2008: num
##  $ volume.baie..cm3..2008   : num
##  $ nbre.pepin.baie.2009     : num
##  $ poids.pulpe.baie..g..2009: num
```

Analyse statistique I

Le « nécessaire » résumé statistique :

Analyse statistique II

```
summary(donnees)
```

```
##  Population     variete    nbre.pep
##  CE:84      OMtp1004:  1   Min.   :
##  CO:89      OMtp1005:  1   1st Qu.:
##  TE:72      OMtp1033:  1   Median :
##             OMtp1068:  1   Mean   :
##             OMtp1072:  1   3rd Qu.:
##             OMtp1073:  1   Max.   :
##             (Other) :239   NA's   :
##  volume.baie..cm3..2008 nbre.pepin.
##  Min.   : 3.44          Min.   :1.0
##  1st Qu.: 7.14          1st Qu.:1.5
##  Median : 8.91          Median :2.0
##  Mean   :10.47          Mean   :2.0
##  3rd Qu.:11.90          3rd Qu.:2.6
##  Max.   :33.80          Max.   :3.6
##  NA's   :44             NA's   :196
```

Analyse statistique III

Et si je veux le nombre de baies moyen en 2008 pour chaque population ?

```
with(donnees,tapply(nbre.pepin.baie.20
```

```
##       CE       CO       TE
## 1.850649 2.034667 1.666667
```

Et le volume moyen des baies ?

```
with(donnees,tapply(volume.baie..cm3..
```

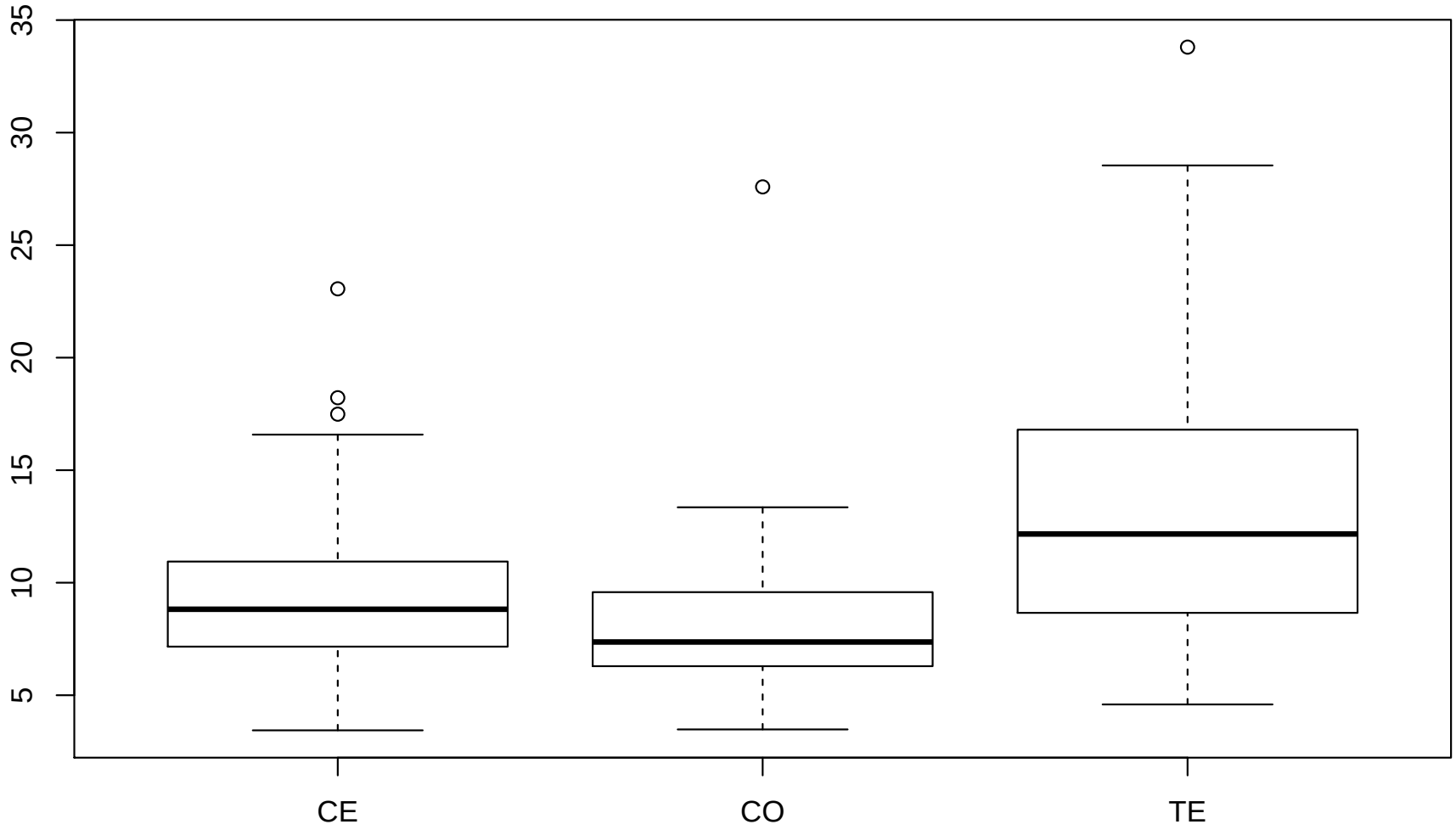
```
##        CE        CO        TE
##  9.667971  8.003000 14.132097
```

Analyse statistique IV

Ça a l'air disparate. Qu'est-ce que ça donne graphiquement ?

```
boxplot(volume.baie..cm3..2008 ~ Popul
```

Analyse statistique V

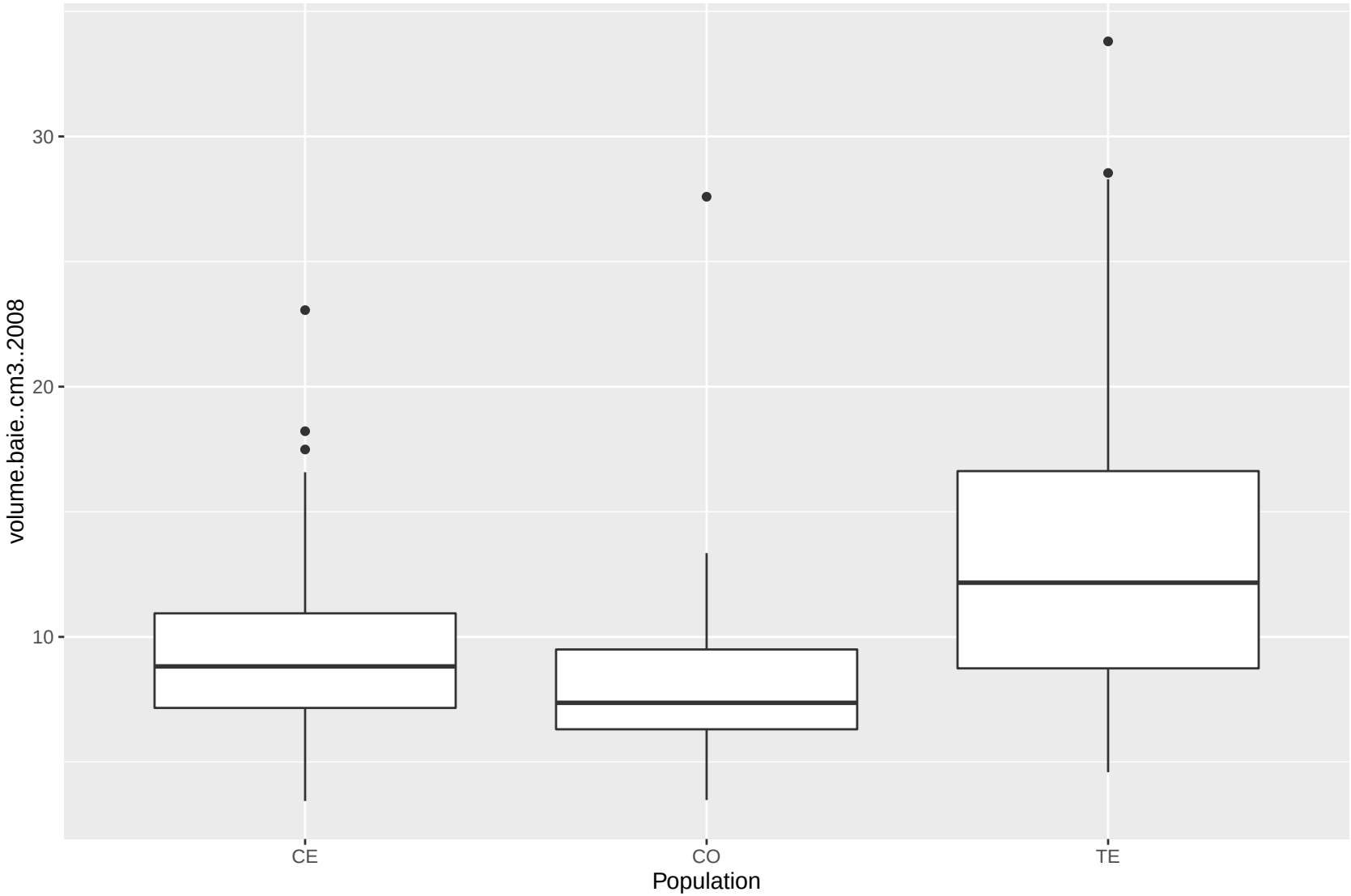


Analyse statistique IV I

Voyons cela en plus joli/moderne...

```
                                                    library(ggplot2)
                                                    qplot(Population,volume.baie..cm3..200

## Warning: Removed 44 rows containing non-finite values (stat_boxplot).
```

Analyse statistique IV II



Analyse de la variance

Finalement, y-a-t-il un effet « population » pour le volume des baies ?

```
anova(lm(volume.baie..cm3..2008 ~ Popul
```

```
## Analysis of Variance Table
##
## Response: volume.baie..cm3..2008
##              Df Sum Sq Mean Sq F val
## Population    2 1301.9  650.94   29.
## Residuals   198 4376.5   22.10
## ---
## Signif. codes:  0 '***' 0.001 '**'
```

etc. *beaucoup* d'autres choses sont possibles...

Un autre exemple : SNP et HIV (Dalmasso et al, 2008) I

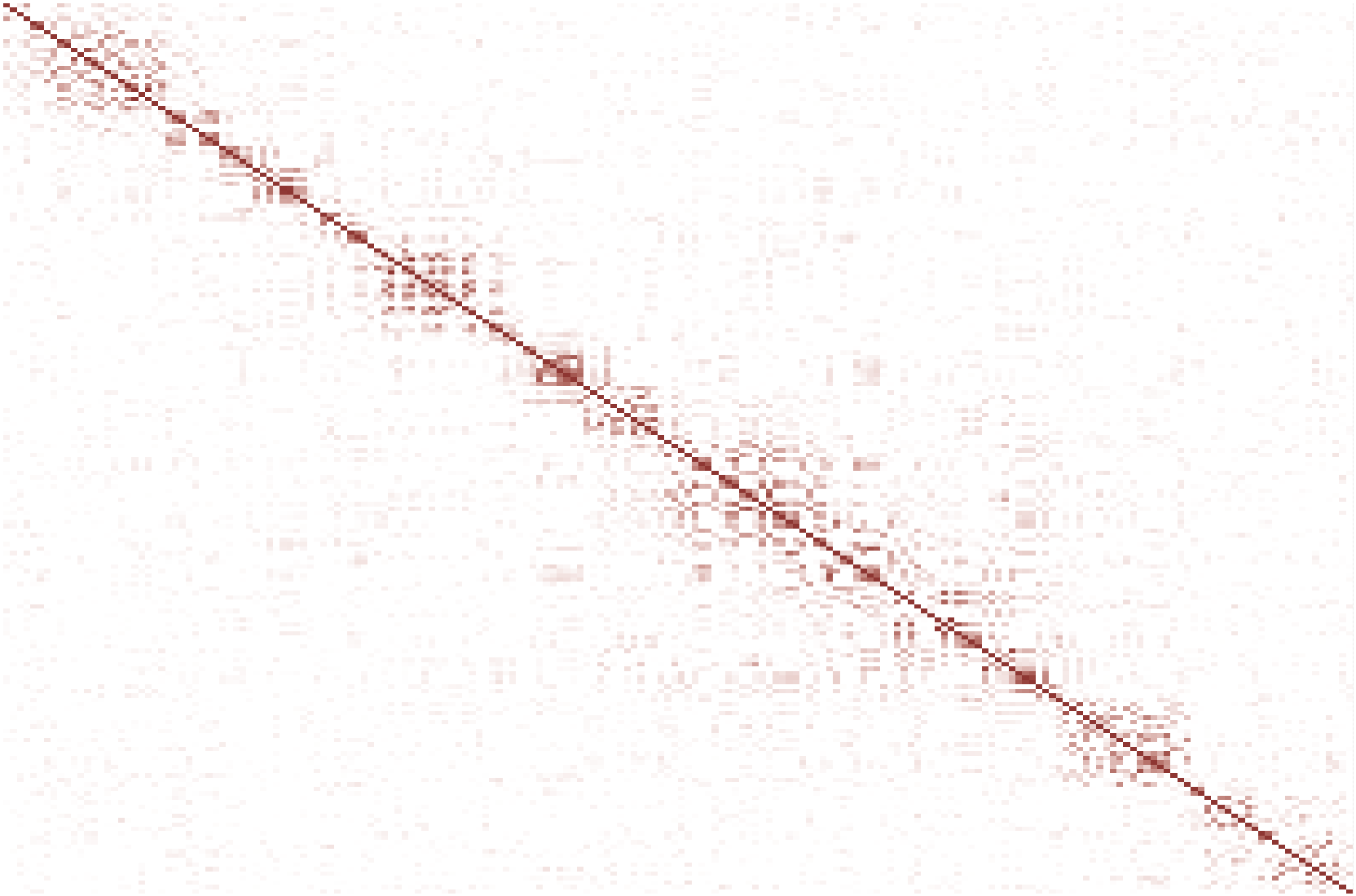
```
setwd("~/svn/hdr/code/examples/example
source("../plot_func.R") #a couple of
library(Matrix) # better matrix manipu
```

```
## LOAD PREPROCESSED SNP DATA
load("data/geno.Rdata")      # SNP data
load("data/matannot.RData") # annotati
load("data/pheno.Rdata")     # phenotyp
nsnp <- 200 # only consider the first
```

```
## Linkage desuiquilibrium / correlatio
C <- cov2cor(var(t(matgeno[1:nsnp, ]),
mat.plot(C)
```

```
## Using row as id variables
```

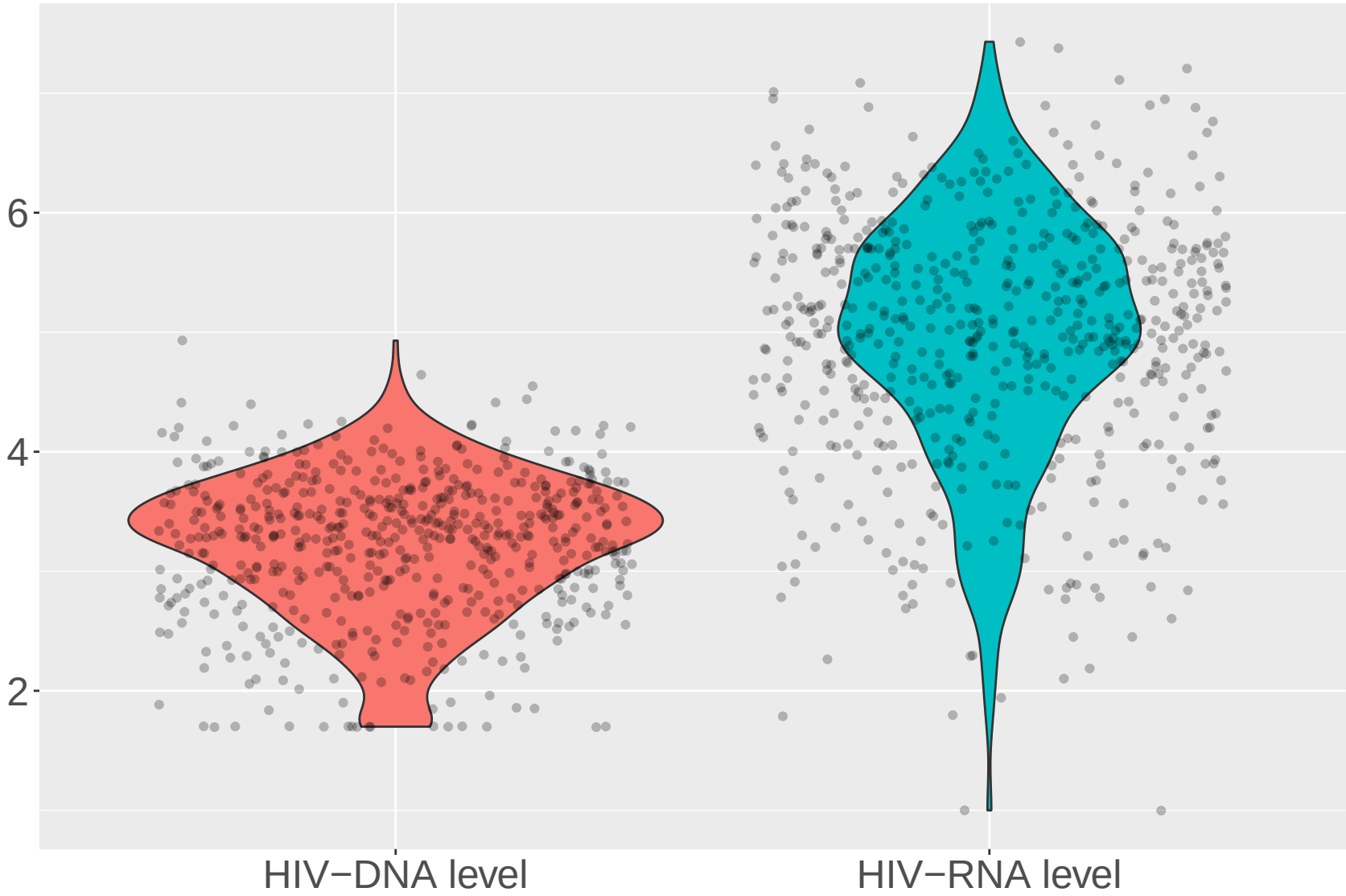
Un autre exemple : SNP et HIV (Dalmasso et al, 2008) II



Un autre exemple : SNP et HIV (Dalmasso et al, 2008) I

```
                                                    ## Phenotype distributions
                                                    pheno <- melt(matpheno2[, 4:5])


## No id variables; using all as measure variables

                                                    variable <- rep("HIV-RNA level", nrow(
                                                    variable[pheno$variable == "DNAinc"] <
                                                    pheno$variable <- factor(variable)
                                                    print(ggplot(pheno, aes(x=variable,y=va
                                                          theme(legend.position="none",
                                                                text=element_text(size=24)
                                                                axis.title=element_blank())


## Warning: Removed 34 rows containing non-finite values (stat_ydensity).
## Warning: Removed 34 rows containing missing values (geom_point).
```

Un autre exemple : SNP et HIV (Dalmasso et al, 2008) II



Deuxième partie II

Structures de données

Plan

Structures de données

Plan

Structures de données

Plan

Structures de données

Déclaration d'une variable

En R, affecter une variable revient à la déclarer.

Definition (affectation)

Opération consistant à attribuer une valeur à une variable.

Déclaration d'une variable

Plusieurs choix possibles

- l'opérateur usuel est '<-' (signe inférieur suivi de moins)

```
jo <- "l'indien"
jo
```

```
## [1] "l'indien"
```

- l'opérateur '=' peut être utilisé

```
mavariable = 3
mavariable
```

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

- la commande `assign`

```
assign("x", -pi)
x
```

Les principaux type et modes

Types et modes

Pour une variable élémentaire (atomique), on a principalement

type	`typeof`	**mode**	`mode`
flottant	`double`	numérique	`numeric`
entier	`integer`	numérique	`numeric`
complexe	`complex`	complex	`complex`
booléen	`logical`	logique	`logical`
caractère	`caracter`	caractère	`caracter`

Les principaux type et modes

Types et modes

Pour une variable élémentaire (atomique), on a principalement

type	`typeof`	**mode**	`mode`
flottant	`double`	numérique	`numeric`
entier	`integer`	numérique	`numeric`
complexe	`complex`	complex	`complex`
booléen	`logical`	logique	`logical`
caractère	`caracter`	caractère	`caracter`

Types et modes II

D'autres types sont possible pour des objets plus complexes

type	`typeof`	**mode**	`mode`
liste	`list`	liste	`list`
NULL	`NULL`	NULL	`NULL`
closure	`closure`	fonction	`function`
raw	raw	raw	raw

Valeurs spéciales, réservées par R

- TRUE/FALSE, indicateurs logiques
- NA, valeurs manquantes, de type logical
- NaN, résultat numérique aberrant, de type double
- Inf et -Inf, plus et moins ∞, de type double
- NULL, l'objet nul, de type NULL

```
c(4,2,NA,5)
```

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

```
0/0
```

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

```
1/0
```

Plan

Structures de données

Variable atomique numérique

Opérateurs arithmétiques élémentaires '+','-','/','*'

```
x <- pi
2*x
```

```
## [1] 6.283185
```

```
mode(x)
```

```
## [1] "numeric"
```

```
typeof(x)
```

```
## [1] "double"
```

ainsi que ^,%%,%/%,abs,log,exp,log10,sqrt,cos,tan,sin

Variable atomique logique

Opérateurs logiques élémentaires '&','|','!','xor'

```
x <- TRUE; y <- FALSE
x | y
```

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

```
mode(x)
```

```
## [1] "logical"
```

```
typeof(x)
```

```
## [1] "logical"
```

Chaîne de caractères

Opérations élémentaires : `'paste','substr','sub'` et bien d'autres

```
x <- "allo"; y <- "non mais"
paste(y,x)
```

```
## [1] "non mais allo"
```

```
substr(y,5,8)
```

```
## [1] "mais"
```

Tester le type d'une variable

Les fonctions de format `is.xx` permet de tester le type d'une variable

```
is.numeric(pi)
```

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

```
is.logical("chaîne")
```

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

```
is.character("")
```

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

```
is.integer(1) ## étonnant ?
```

Convertir une variable

La fonction générique as et ses dérivées le permet

```
is.integer(as.integer(1))
```

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

```
as(TRUE, "numeric")
```

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

```
as.character(pi)
```

```
## [1] "3.14159265358979"
```

On peut aussi forcer le type !

Plan

Structures de données

Plan

Structures de données

Vecteurs : définition

Propriétés

- objet le plus élémentaire sous R,
- collection d'entités de même nature,
- mode défini par la nature des entités qui le composent.

Création par initialisation I

Avec les fonctions issus des types :

```
integer(5)
```

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

```
double(5)
```

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

```
character(5)
```

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

```
logical(5)
```

Création par initialisation II

Avec la commande `vector`

```
v <- vector(mode="numeric", 3); v
```

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

```
v <- vector(mode="logical", 4); v
```

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

```
v <- vector(mode="character", 5); v
```

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

Création par concaténation I

En « combinant » des éléments de même type à l'aide de la fonction c()

1. Numérique

```
x1 <- c(-1,23,98.7)
mode(x1)
```

```
## [1] "numeric"
```

2. Caractère

```
y1 <- c("Pomme","Flore","Alexandr
mode(y1)
```

```
## [1] "character"
```

Création par concaténation II

3. Logique

```
z1 <- c(FALSE,TRUE,FALSE,TRUE,TRU
z2 <- c(T,F,F)
mode(z2)


## [1] "logical"
```

Création par concaténation III

Attention : un seul mode possible. . .

```
c(TRUE,"bonjour", 2)
```

```
## [1] "TRUE"    "bonjour" "2"
```

mais

```
"bonjour"
```

```
## [1] "bonjour"
```

Plan

Structures de données

Opérations arithmétiques I

s'effectuent terme-à-terme

Soient `x,y` de mode `numeric` tels que

```
x<-c(1,2,-3,-4)
y<-c(-5,-6,9,0)
```

'+' addition des éléments de deux vecteurs

```
x+y
```

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

'−' soustraction des éléments de deux vecteurs

```
x-y
```

```
## [1]   6   8 -12  -4
```

Opérations arithmétiques II

s'effectuent terme-à-terme

'*' multiplication des éléments de deux vecteurs

```
x*y
```

```
## [1]  -5 -12 -27   0
```

'/' division des éléments de deux vecteurs

```
x/y
```

```
## [1] -0.2000000 -0.333
```

'%%' division entière

```
abs(x) %% abs(y)
```

```
## [1]   1   2   3 NaN
```

Opérations arithmétiques III

s'effectuent terme-à-terme

'%/%' reste de la division entière

```
abs(y) %/% abs(x)
```

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

Le « recyclage » des éléments du vecteur

Lors d'une opération entre vecteurs, les vecteurs trop courts sont ajustés pour atteindre la taille du plus grand vecteur en recyclant les données.

```
x <- c(10,100,1000)
y <- c(1,2)
x+10
```

```
## [1]   20  110 1010
```

```
2*x + y - 1
```

```
## Warning in 2 * x + y: la taille d'un objet plus long n'est pas multiple de la
taille d'un objet plus court
```

```
## [1]   20  201 2000
```

$\rightsquigarrow$ souvent pratique mais attention aux effets de bords !

Opérateurs mathématiques I

Fonctions numériques élémentaires

`floor`,`ceiling`,`round`.

Opérateurs mathématiques II

```
x<-c(1,2,-3,-4); y<-c(-5,-6,9,0)
x/y
```

```
## [1] -0.2000000 -0.3333333 -0.333333
```

```
floor(x/y)
```

```
## [1]   -1   -1   -1 -Inf
```

```
ceiling(x/y)
```

```
## [1]    0    0    0 -Inf
```

```
round(x/y,3)
```

Opérateurs ensemblistes I

Fonctionnent pour tous les modes

`unique,intersect,union,setdiff,setequal,is.element`

Opérateurs ensemblistes II

```
unique(c("banane","citron","banane"))
```

```
## [1] "banane" "citron"
```

```
intersect(c("banane","citron"),c("orang
```

```
## [1] "banane"
```

```
union(c("banane","citron"),c("orange",
```

```
## [1] "banane" "citron" "orange"
```

```
setequal(c("banane","citron"),c("orang
```

Faire une recherche dans un vecteur

Fonctionnent pour tous les modes

`match`,`%in%` (voir aussi `pmatch`...)

```
x <- rep(1:5,2); y <- c(3,5,1)
y %in% x
```

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

```
x %in% y
```

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

```
match(x,y)
```

```
##  [1]  3 NA  1 NA  2  3 NA  1 NA  2
```

Tester le mode / les éléments d'un vecteur

Fonctionnent pour tous les modes

`is.xx,is.NA,is.infinite,all.equal,anyNA ...`

```
all.equal(c(1,12,3),c(1,12,3))
```

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

```
anyNA(c(1,NA,3.5))
```

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

```
is.numeric("allo?")
```

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

Nommer les éléments d'un vecteur

La fonction `names` permet d'accéder ou d'attribuer des noms aux éléments d'un vecteur.

```
x <- c(1,2,3)
names(x) <- c("one","two","three")
names(x)
```

```
## [1] "one"   "two"   "three"
```

La fonction `setNames` permet de créer directement un vecteur en nommant les éléments.

```
x <-setNames(c(1,2,3),c("one","two","t
x
```

```
##   one   two three
##     1     2     3
```

`names` confère un attribut à un objet vecteur

Plan

Structures de données

L'opérateur ':'

Génère une séquence de mode `numeric` par pas de un depuis le nombre `from` jusqu'à `to` (si possible).

```
-5:5
```

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

```
5:-5
```

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

```
pi:6
```

```
## [1] 3.141593 4.141593 5.141593
```

La commande seq

Génère une séquence de mode `numeric`

Plusieurs schémas possibles

- `seq(from,to)`
- `seq(from,to,by=)`
- `seq(from,to,length.out=)`

```
seq(1,10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
seq(2,10,by=2)
```

```
## [1]  2  4  6  8 10
```

La commande rep

Fonctionne pour tous les modes

- rep(x,times), où times peut être un vecteur,
- rep(x,each).
- rep(x,length).

```
rep(1,3)
```

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

```
rep(c("A","B","C"),c(3,2,4))
```

```
## [1] "A" "A" "A" "B" "B" "C" "C" "C"
```

```
rep(c(TRUE,FALSE), each=2)
```

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

Génération de vecteurs logiques

Obtenus par conditions avec

- les opérateurs logiques '<', '<=', '>', '>=', '==' '!='
- le ET, le OU, NON, OU exclusif : '&' (intersection), '|' (union), '!' (négation), `xor`.

```
note1 <- c(8,9,14,3,17.5,11)
note2 <- c("C","B","A","B","E","B")
admis <- (note1 >= 10) & (note2 == "A"
mention <- (note1 >= 15) & (note2 == "
admis
```

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

```
sum(admis)
```

```
## [1] 2
```

Par concaténation

Avec 'c()'

L'opérateur 'c()' peut s'appliquer à n'importe quoi pourvu que l'on concatène des vecteurs de même type.

```
c( c(1,2), c(3,4))
```

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

```
round(c(seq(-pi,pi,len=4),rep(c(1:3),ea
```

```
##  [1] -3.14 -1.05  1.05  3.14  1.00
```

Remarque

Dans le second exemple, les entiers composants c(1:3) ont été forcés au typage flottant.

Par concaténation II

Avec paste

Concaténation de chaînes de caractères. Convertit en caractères les éléments passés en argument avant toute opération.

```
paste("R","c'est","bien")
```

```
## [1] "R c'est bien"
```

```
paste(2:4,"ieme")
```

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

```
paste("A",1:5, sep="")
```

```
## [1] "A1" "A2" "A3" "A4" "A5"
```

Plan

Structures de données

Indexation des vecteurs

Extrêmement puissant !

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Permet également d'affecter de nouvelles valeurs.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

Indexation des vecteurs

Extrêmement puissant !

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Permet également d'affecter de nouvelles valeurs.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;

Indexation des vecteurs

Extrêmement puissant !

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Permet également d'affecter de nouvelles valeurs.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;

Indexation des vecteurs

Extrêmement puissant !

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Permet également d'affecter de nouvelles valeurs.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;

Indexation des vecteurs

Extrêmement puissant !

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Permet également d'affecter de nouvelles valeurs.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;
4. un vecteur de chaînes de caractères, qui spécifie les noms des éléments de `x` à conserver.

Indexation des vecteurs : exemples I

Vecteurs logiques

Indexation des vecteurs : exemples II

```
x <- c(3,6,-2,9,NA,sin(-pi/6))
x[x > 0]
```

```
## [1]  3  6  9 NA
```

```
x[!is.na(x)]
```

```
## [1]  3.0  6.0 -2.0  9.0 -0.5
```

```
x[!is.na(x) & x>0]
```

```
## [1] 3 6 9
```

```
x[x <= mean(x,na.rm=TRUE)]
```

Autres commandes d'indexation et de sélection

1. Classer
   - `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
   - `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,

Autres commandes d'indexation et de sélection

1. Classer
   - `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
   - `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,

2. Extraire
   - `which` renvoie les indices de `x` vérifiant une condition ;

Autres commandes d'indexation et de sélection

1. Classer
   - `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
   - `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,
2. Extraire
   - `which` renvoie les indices de `x` vérifiant une condition ;
3. Échantillonner
   - `sample` échantillonne aléatoirement dans un vecteur `x`, avec ou sans remise.

Exemples

```
x <- -5:5
y <- sample(x)
sort(y)
```

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

```
order(y)
```

```
##  [1]  9 11  3  6  8  4 10  1  7  2
```

```
y[order(y)]
```

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

```
y[order(y,decreasing=TRUE)]
```

Fonctions statistiques élémentaire

moyenne, médiance variance, écart-type :

```
x <- -2:5
mean(x)
```

```
## [1] 1.5
```

```
median(x)
```

```
## [1] 1.5
```

```
var(x)
```

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

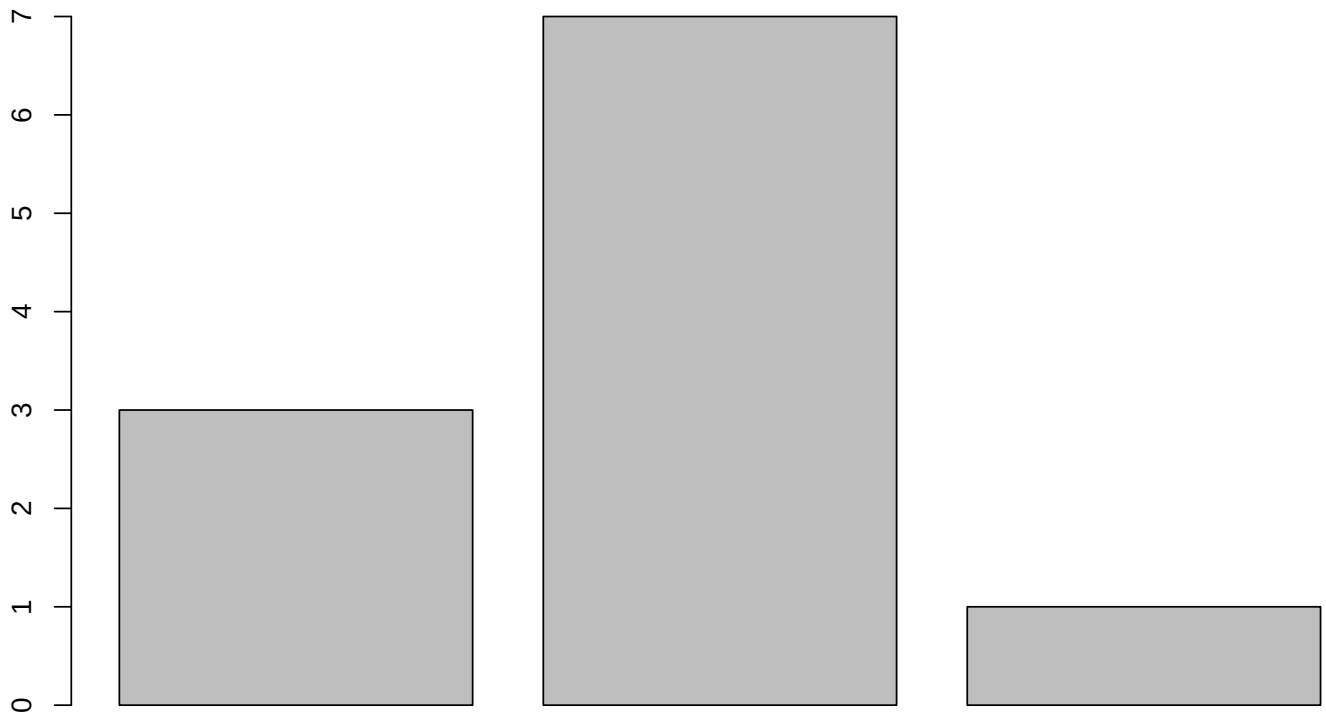
```
sd(x)
```

Plan

Structures de données

la fonction barplot

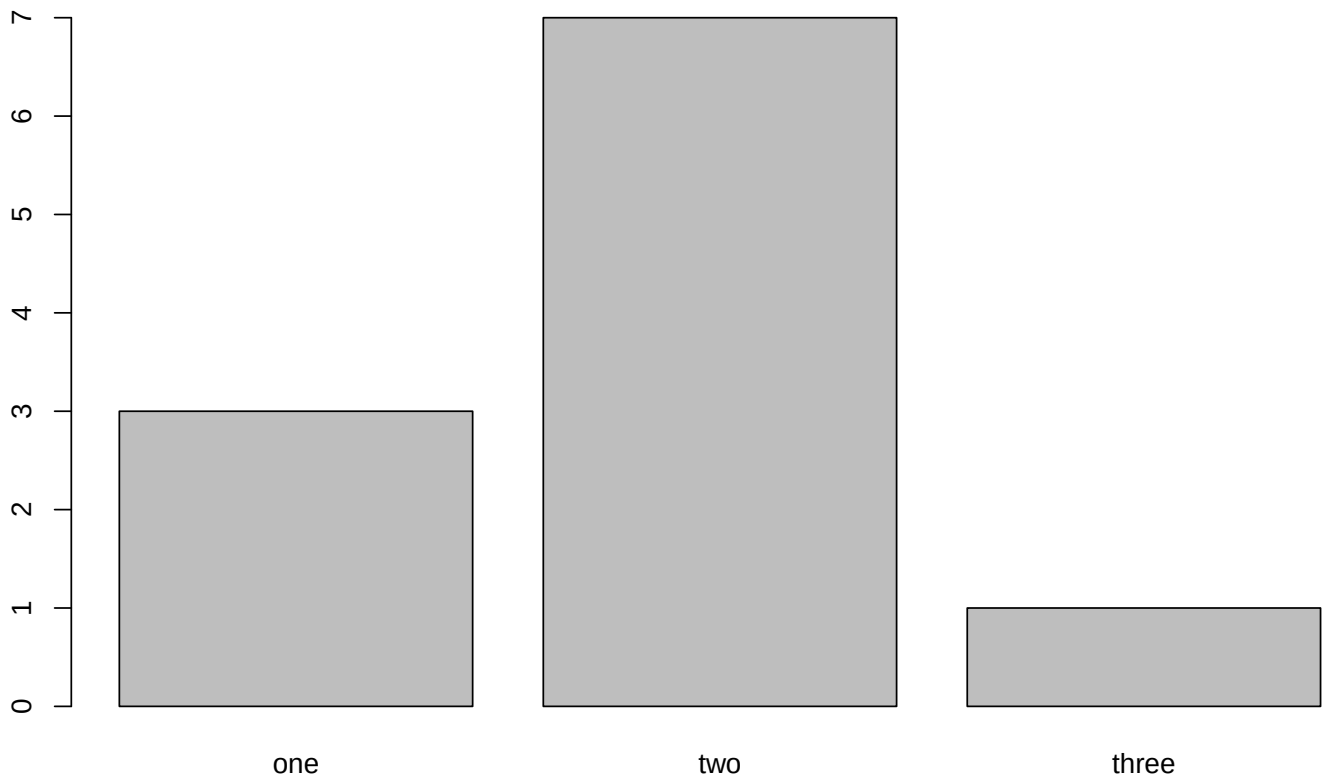
```
x <-c(3,7,1)
barplot(x)
```



la fonction barplot II

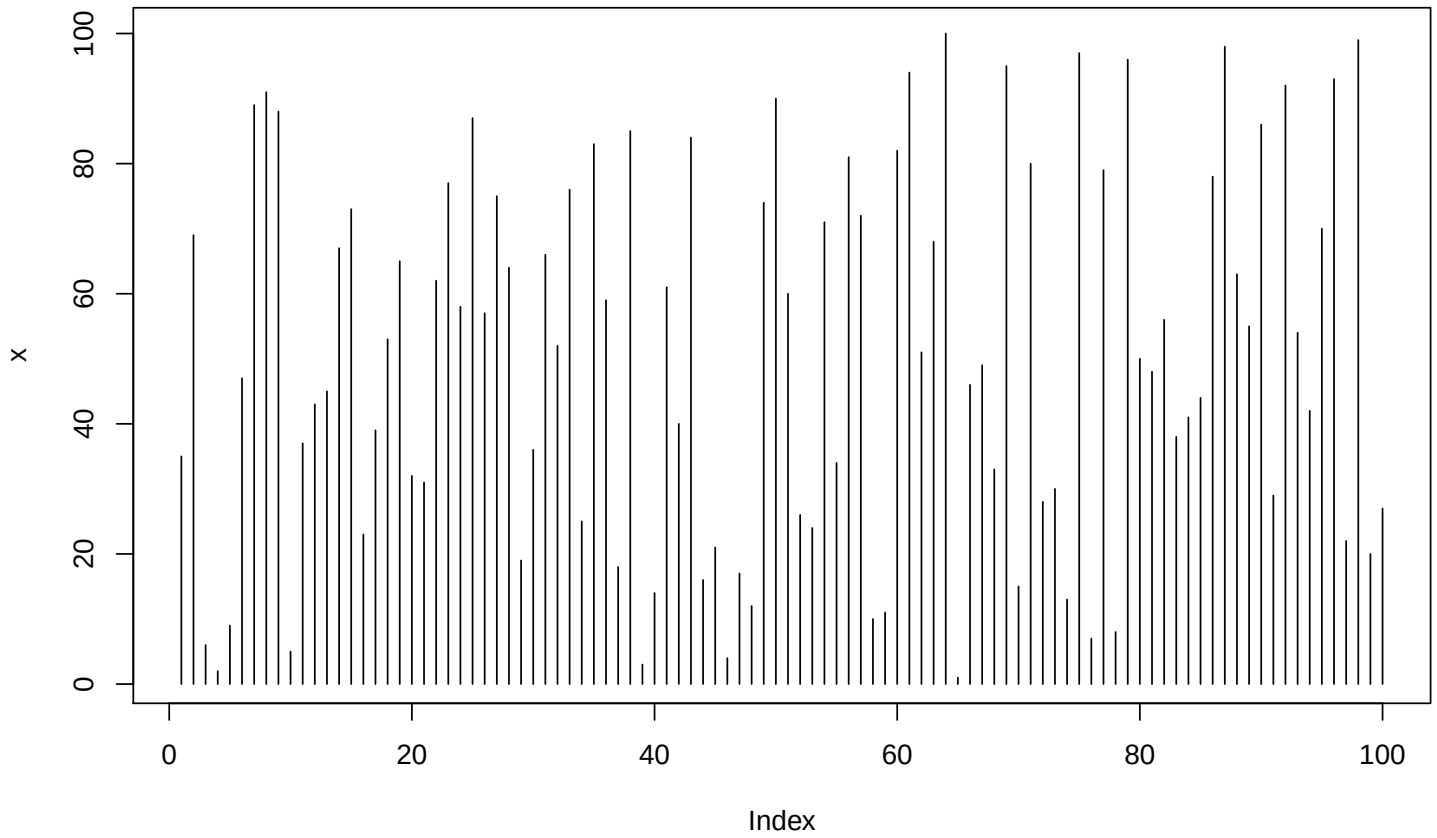
De l'utilité des attributs et du typage

```
x <-setNames(c(3,7,1),c("one","two","three"))
barplot(x)
```



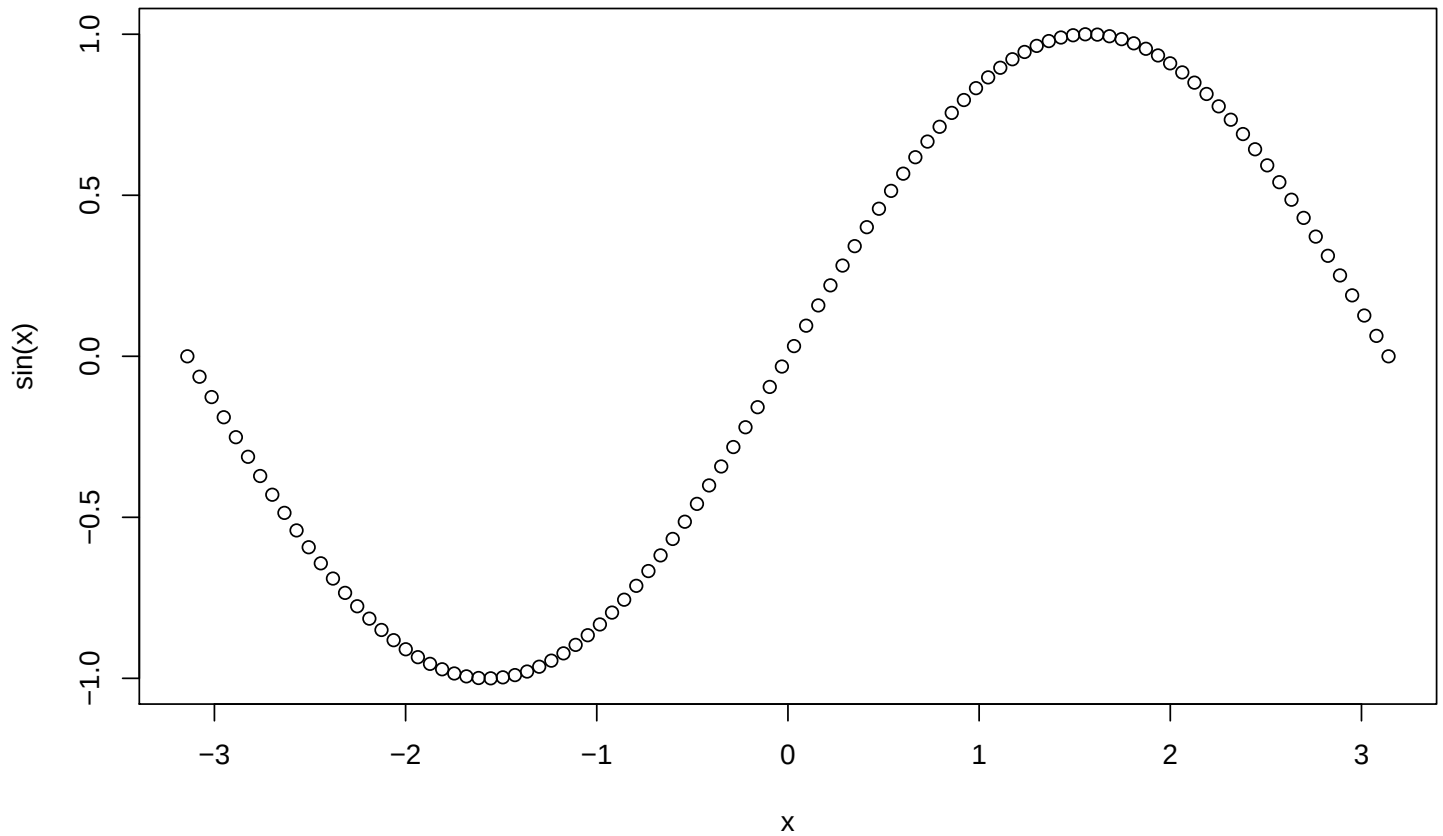
la fonction plot

```
x <- sample(1:100)
plot(x, type="h")
```



la fonction plot II

```
x <- seq(-pi,pi,len=100)
plot(x, sin(x))
```



Plan

Structures de données

Plan

Structures de données

Définition

Definition

Un *facteur* est un vecteur de variables catégorielles. Les *niveaux* du facteur peuvent être ordonnés ou pas.

Définition

Definition

Un *facteur* est un vecteur de variables catégorielles. Les *niveaux* du facteur peuvent être ordonnés ou pas.

Utilisation

les facteurs s'utilisent pour catégoriser les données d'un vecteur (ce qui s'avère très utile pour la gestion des variables qualitatives).

Définition

Definition

Un *facteur* est un vecteur de variables catégorielles. Les *niveaux* du facteur peuvent être ordonnés ou pas.

Utilisation

les facteurs s'utilisent pour catégoriser les données d'un vecteur (ce qui s'avère très utile pour la gestion des variables qualitatives).

$\rightsquigarrow$ un facteur est souvent associé à d'autres vecteurs pour en définir une partition.

Création

Création : la fonction factor

```
x <- sample(1:3,10,replace=TRUE)
factor(x)
```

```
##  [1] 3 3 2 1 1 1 3 2 1 3
## Levels: 1 2 3
```

```
as.factor(x)
```

```
##  [1] 3 3 2 1 1 1 3 2 1 3
## Levels: 1 2 3
```

```
factor(x,levels=1:5)
```

```
##  [1] 3 3 2 1 1 1 3 2 1 3
## Levels: 1 2 3 4 5
```

```
factor(x,levels=1:5, ordered=TRUE)
```

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

Le type d'objet facteur

Un facteur est en fait un vecteur d'entier muni d'un attribut `levels` :

```
x

##  [1] 3 3 2 1 1 1 3 2 1 3

fx <- factor(x)
attributes(fx)

## $levels
## [1] "1" "2" "3"
##
## $class
## [1] "factor"

as.numeric(fx)

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

Plan

Structures de données

Accès/affectation des attributs I

Gestion : nlevels,levels,table

```
levels <- c("MdC","thésard","CR")
data <- sample(levels,15,replace=TRUE)
fx  <- factor(data)
nlevels(fx)

## [1] 3

levels(fx)

## [1] "CR"       "MdC"      "thésard"

table(fx)

## fx
##       CR      MdC thésard
##        7        4        4
```

Accès/affectation des attributs II

```
levels(fx) <- c("Dr","CR","MdC","thésards")
nlevels(fx)

## [1] 4

levels(fx)

## [1] "Dr"       "CR"       "MdC"      "thésards"

table(fx)

## fx
##       Dr       CR      MdC thésards
##        7        4        4        0

is.ordered(fx)

## [1] FALSE
```

Accès/affectation des attributs III

```
fx <- factor(data, levels=c("thésard","MdC","CR","Dr"), ordered=TRUE)
nlevels(fx)

## [1] 4

levels(fx)

## [1] "thésard" "MdC"     "CR"      "Dr"

table(fx)

## fx
## thésard     MdC      CR      Dr
##       4       4       7       0

is.ordered(fx)

## [1] TRUE
```

Plan

Structures de données

Facteur associé à un vecteur

Permet de manipuler un vecteur conditionnellement à un facteur

Données

Dans le laboratoire, chacun me donne son âge et son grade [a]

```
age <- c(25,35,32,27,32,40,26,25,26,28,30,NA,36,30,30)
grd <- c("thd","CR","MdC","thd","thd","MdC","MdC","thd","thd","MdC","CR","MdC","CR"
```

a. sauf un qui refuse :'(

Question : nombre d'individus par catégorie ?

```
table(grd)

## grd
##  CR MdC thd
##   3   5   7
```

La fonction split

Découpe un vecteur selon les valeurs d'un facteur

```
split(age,grd)

## $CR
## [1] 35 30 36
##
## $MdC
## [1] 32 40 26 28 NA
##
## $thd
## [1] 25 27 32 25 26 30 30
```

La fonction tapply

Utilisation

Applique une fonction sur un vecteur partitionné en groupes. Généralise l'exemple de la fonction `split`, en appliquant n'importe quelle fonction au résultat obtenu

Question : âge moyen / écart-type par catégorie ?

```
tapply(age,grd,mean,na.rm=TRUE)

##       CR      MdC      thd
## 33.66667 31.50000 27.85714

tapply(age,grd,sd,na.rm=TRUE)

##       CR      MdC      thd
## 3.214550 6.191392 2.794553
```

Plan

Structures de données

la fonction barplot

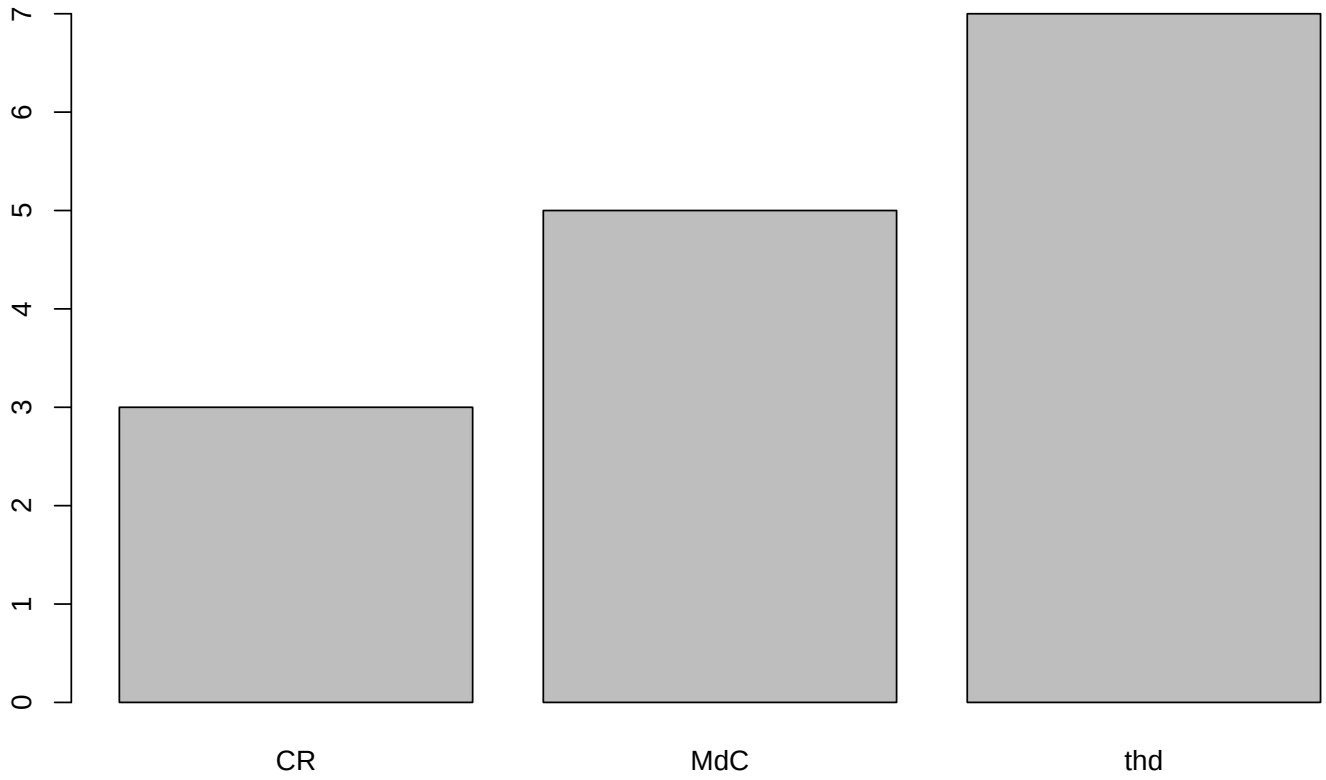
Aïe !

```
plot(grd)

## Warning in xy.coords(x, y, xlabel, ylabel, log): NAs introduits lors de la
conversion automatique
## Warning in min(x): aucun argument trouvé pour min ; Inf est renvoyé
## Warning in max(x): aucun argument pour max ; -Inf est renvoyé
## Error in plot.window(...): valeurs finies requises pour 'ylim'
```

la fonction barplot II

De l'utilité des attributs et du typage

```
plot(factor(grd))
```



Plan

Structures de données

Plan

Structures de données

Tableau : définition

Definition (objet array)

Un tableau est un vecteur muni d'un attribut dimension (`dim`), lui même défini par un vecteur. Il est défini par la commande
`array(data,dim,dimnames=)`

```
array(1:8,c(2,2,2))
```

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

Matrice : définition

Definition (objet `matrix`)

Une matrice est un tableau à deux dimensions. Elle est définie par la commande

`matrix(data,nrow=,ncol=,byrow)`

En conséquence

- Un objet `array` à deux dimensions est automatiquement converti en `matrix`
- Un vecteur auquel on ajoute un attribut dimension est automatiquement converti en `matrix`

```
class(array(1:4,c(2,2)))

## [1] "matrix"

x <- c(1,2,3,4)
dim(x) <- c(2,2)
class(x)
```

Matrice et tableau : attributs dimension

On accède aux attributs de dimension d'un tableau à l'aide des commandes `dim` et `nrow`,`ncol` dans le cas d'une matrice.

```
A <- array(1:12,c(2,2,3))
M <- matrix(1:6,c(2,3))
dim(A)

## [1] 2 2 3

dim(M)

## [1] 2 3

nrow(M)

## [1] 2

ncol(M)

## [1] 3
```

Remarques importantes

1. R range les éléments d'une matrice par défaut par colonne.

```
matrix(1:6,nrow=2)
```

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

```
matrix(1:6,nrow=2,byrow=TRUE)
```

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

2. Lors de la création d'une matrice, R recycle les éléments jusqu'à ce que les contraintes de dimension soient vérifiées.

```
matrix(1:3,nrow=2,ncol=2)
```

```
## Warning in matrix(1:3, nrow = 2, ncol = 2): la longueur des données [3]
n'est pas un diviseur ni un multiple du nombre de lignes [2]
```

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

Plan

Structures de données

Matrices : opérateurs élémentaires

Étant donné qu'une matrice est un vecteur pourvu d'une dimension, on a la proposition suivante :

Proposition

La plupart des opérateurs vectorielles s'appliquent (arithmétiques/mathématiques, ensemblistes, d'indexation).

```
a <- matrix(sample(-4:4,9),3,3)
cat(max(a),sum(a),prod(a))

## 4 0 0

which(a > 0)

## [1] 1 2 4 9

cumsum(a[a > 0])

## [1]  1  5  7 10
```

Manipulation de matrices

Opérateurs matriciels usuels

- `+,/,*,^` sont les opérateurs usuels terme-à-terme,
- `%*%` est le produit matriciel,
- `crossprod()` est le produit scalaire,
- `t()` transpose une matrice,
- `diag()` extrait / spécifie la diagonale.

```
a <- matrix(sample(-4:4,9),3,3)
b <- matrix(sample(a),3,3)
diag(a)

## [1] -4 -3  1

diag(a) <- diag(b) <- 1
diag(a)

## [1] 1 1 1
```

Indexation propres aux matrices

En ligne et/ou en colonne

Selon les mêmes techniques que pour un vecteur

```
a[1:2,]
```

```
##      [,1] [,2] [,3]
## [1,]    1   -2    3
## [2,]    0    1   -1
```

```
a[,-3]
```

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

```
a[-2,c(3,1)]
```

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

Indexation propres aux matrices II

À l'aide d'une matrice de booléen

Nécessite la création d'une matrice appropriée

```
upper.tri(a)

##       [,1]  [,2]  [,3]
## [1,] FALSE  TRUE  TRUE
## [2,] FALSE FALSE  TRUE
## [3,] FALSE FALSE FALSE

a[upper.tri(a)]

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

Indexation propres aux matrices III

À l'aide des noms de ligne et de colonnes

Nécessite d'avoir des attributs `colnames`/`rownames`

```
colnames(a) <- paste0("c",as.character(1:3))
rownames(a) <- paste0("r",as.character(1:3))
a
```

```
##    c1 c2 c3
## r1  1 -2  3
## r2  0  1 -1
## r3  2  4  1
```

```
a[rownames(a) == "r2", colnames(a) %in% c("c3","c2")]
```

```
## c2 c3
##  1 -1
```

Concaténation de matrices I

Trois fonctions selon l'effet voulu :

1. `c()` concatène les éléments en un vecteur,
2. `cbind()` empile horizontalement plusieurs matrices,
3. `rbind()` empile verticalement plusieurs matrices.

```
a <- matrix(1,2,3)
b <- matrix(2,2,3)
c(a,b)

##  [1] 1 1 1 1 1 1 2 2 2 2 2 2
```

Concaténation de matrices II

```
cbind(a,b)
```

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

```
rbind(a,b)
```

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

Plan

Structures de données

Quelques fonctions spécifiques

Opération en ligne/en colonne

Les commandes `colSums`,`rowSums`,`colMeans`,`rowMeans`

```
colSums(a)
```

```
## [1] 2 2 2
```

```
rowSums(a)
```

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

```
rowMeans(a)
```

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

```
colMeans(a)
```

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

Plus généralement et pour les tableau : apply

Opération en ligne/en colonne

Les commandes `apply(array, dim, function)`

```
apply(a, 2, max)

## [1] 1 1 1

a <- array(1:12, c(2,3,2))
apply(a, 3, colMeans)

##      [,1] [,2]
## [1,]  1.5  7.5
## [2,]  3.5  9.5
## [3,]  5.5 11.5
```

⇝ Très puissant !

Algèbre linéaire élémentaire

Résolution de systèmes linéaires, inversion matricielle

La commande `solve` résout

$$\mathbf{A}x = \mathbf{b},$$

```
A <- matrix(c(4,2,8,-3),2,2)
b <- c(2,3)
solve(A,b)

## [1]  1.0714286 -0.2857143
```

ou inverse une matrice :

```
round(solve(A) %*% A,8)

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

Commandes avancées d'algèbre linéaire

Utile pour l'analyse numérique

`R` dispose des outils classiques d'algèbre linéaire

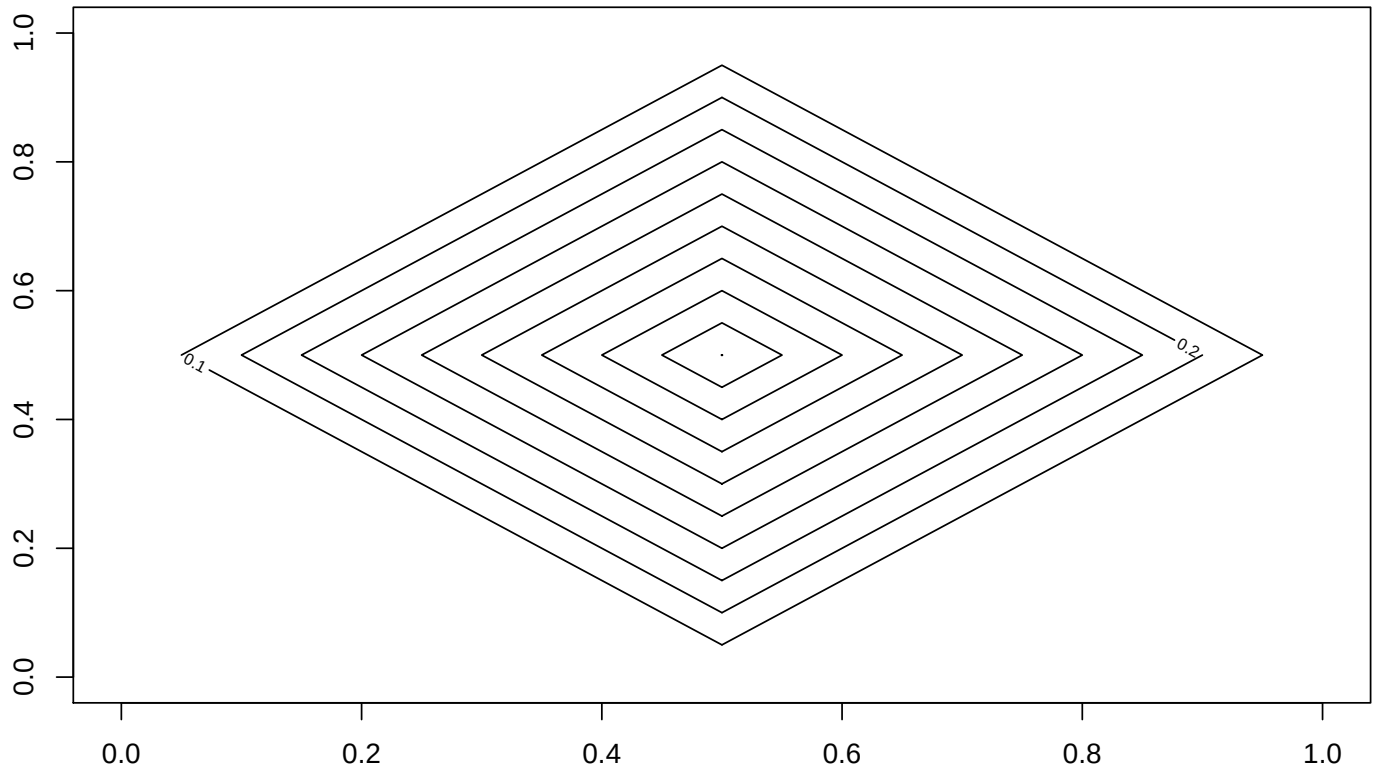
- `det` : calcule le déterminant d'une matrice ;
- `chol` : factorisation de Cholesky ($A = C^\intercal C$, avec A symétrique, C triangulaire supérieure) ;
- `qr` : factorisation QR ($A = QR$ avec Q orthogonale, R triangulaire supérieure) ;
- `eigen` : calcule valeurs propres et vecteurs propres d'une matrice ;
- `svd` : calcule la décomposition en valeurs singulières.
- ...

Plan

Structures de données

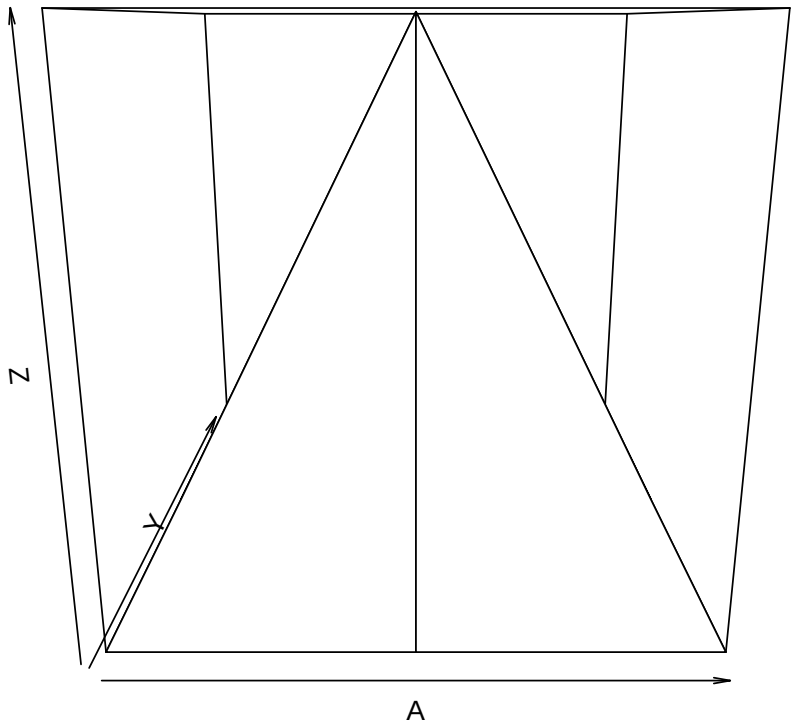
Représentation tridimensionnelle

```
A <- matrix(0,3,3); A[2,2] <- 1
contour(A)
```



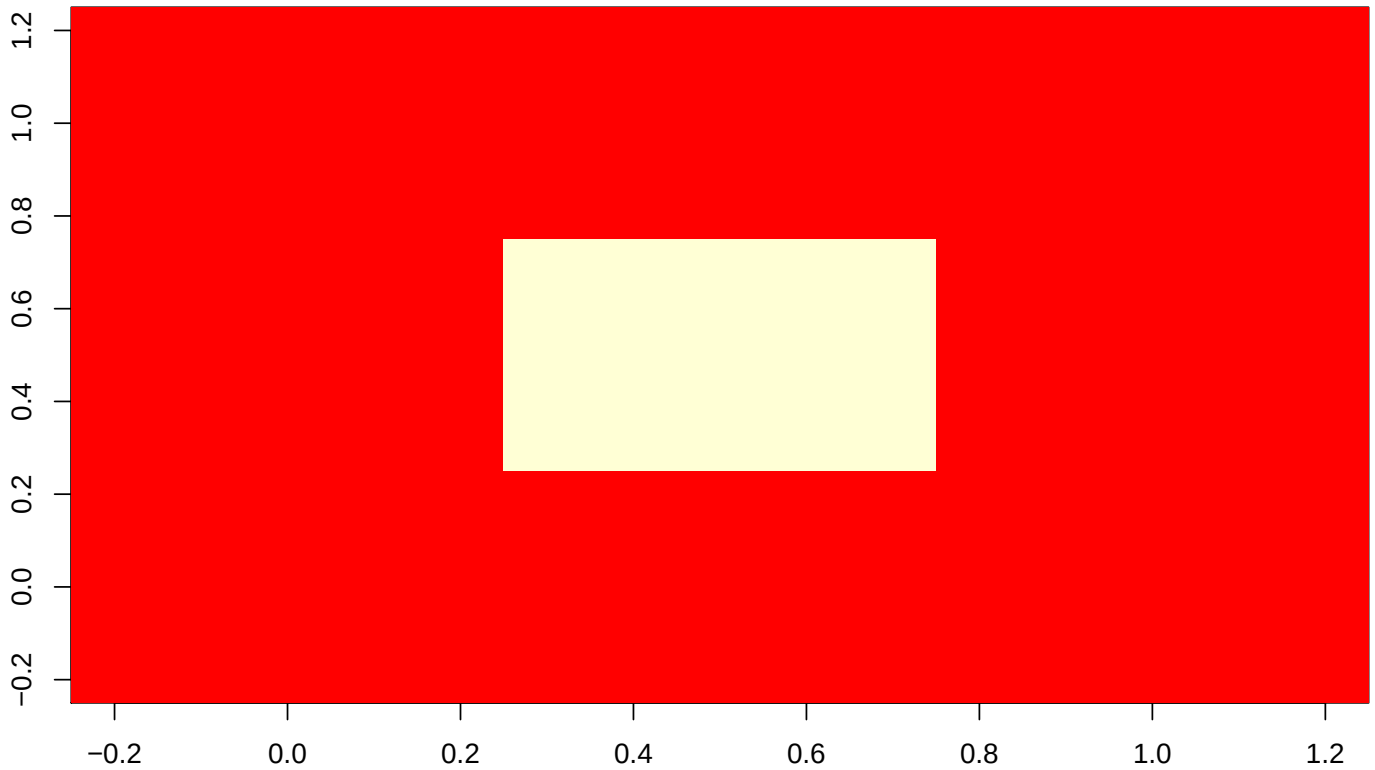
Représentation tridimensionnelle II

```
persp(A)
```



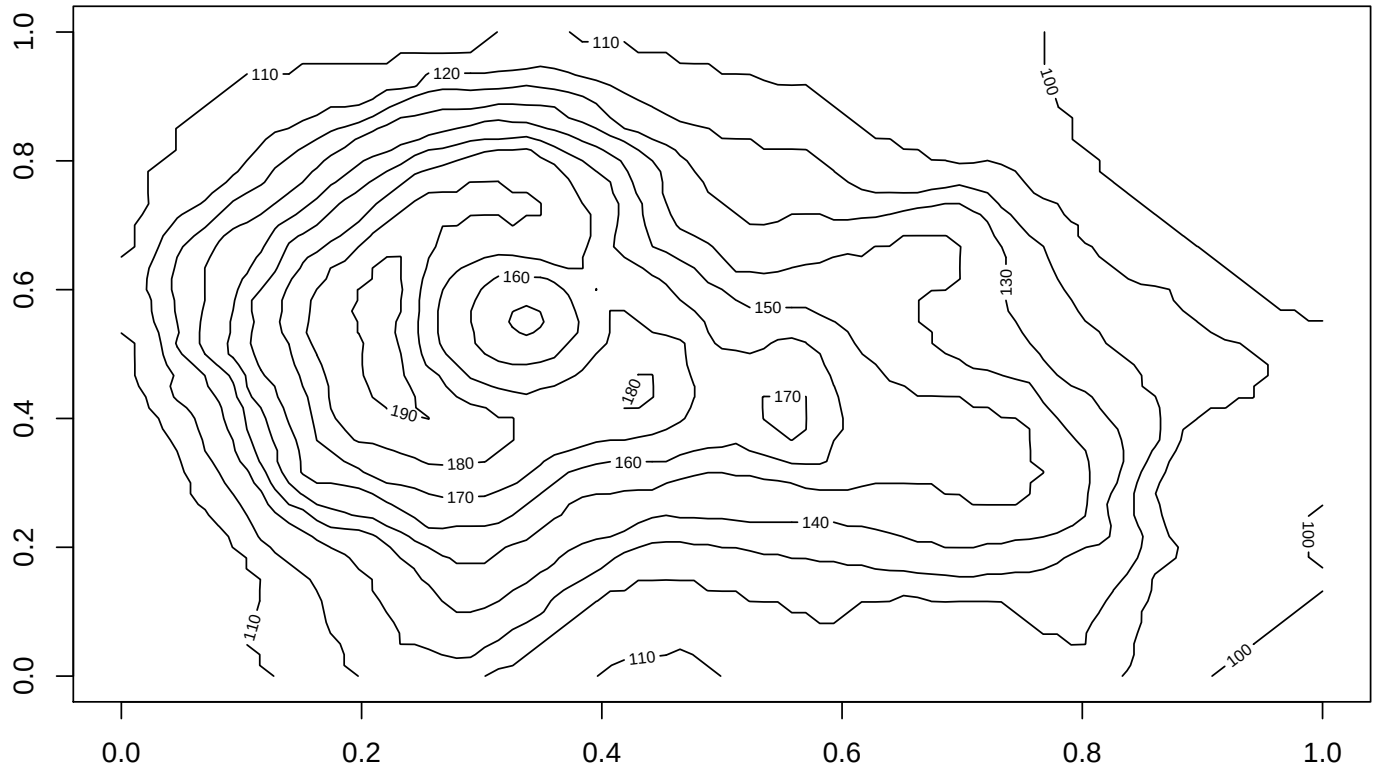
Représentation tridimensionnelle III

```
image(A)
```



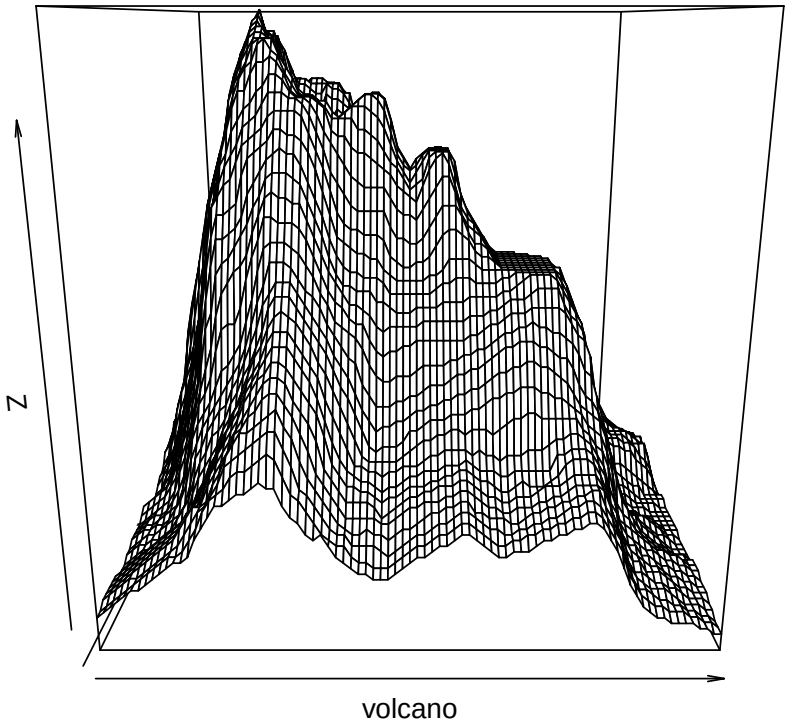
Représentation tridimensionnelle IV

```
contour(volcano)
```



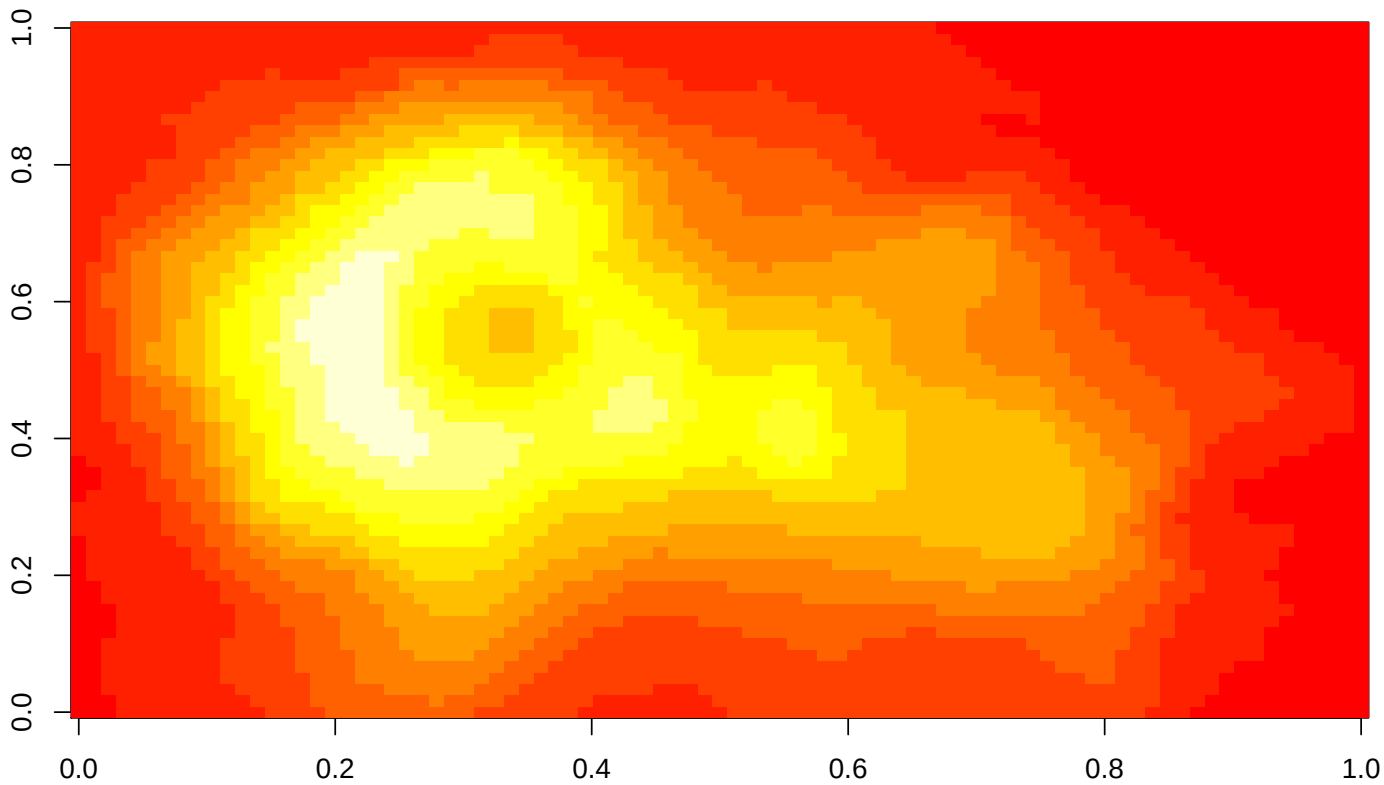
Représentation tridimensionnelle V

```
persp(volcano)
```



Représentation tridimensionnelle VI

```
image(volcano)
```



Plan

Structures de données

Plan

Structures de données

Liste I

Definition (objet `list`)

Une liste est une collection d'objets hétérogènes. Elle est définie par la commande `list(el1=, el2=, ...)`.

```
maliste <- list(c(1,2,3),c("robert","johnson"),matrix(rnorm(4),2,2))
maliste
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "robert"  "johnson"
##
## [[3]]
##             [,1]       [,2]
## [1,] 0.14333657  1.590061
## [2,] 0.01983328 -1.254152
```

Liste II

Peut aussi être initialisée par

```
vector("list", 3)

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

Liste III

Les éléments d'une liste peuvent posséder un nom à la définition ou être attribués *a posteriori*.

```
list(numero = c(1,2,3), noms = c("robert","johnson"), mat = matrix(rnorm(4),2,2))
```

```
## $numero
## [1] 1 2 3
##
## $noms
## [1] "robert"  "johnson"
##
## $mat
##           [,1]       [,2]
## [1,] 0.3877382  0.8330844
## [2,] 0.5940931 -0.2741880
```

```
l <- list(c(1,2,3), c("robert","johnson"), matrix(rnorm(4),2,2))
names(l) <- c("numero","noms","mat")
```

Plan

Structures de données

Accéder aux éléments d'une liste I

Les éléments de la liste ne sont pas nommés

On accède au i^{e} élément par indexation `nom_liste[[i]]` uniquement.

```
maliste[[2]]

## [1] "robert"  "johnson"

maliste[[2]][2]

## [1] "johnson"
```

Accéder aux éléments d'une liste II

Les éléments de la liste sont nommés

On peut accéder comme ci-dessus ou en utilisant le nom de l'élément `nom_liste$nom_elt`.

```
maliste <- list(numero = c(1,2,3), noms = c("robert","johnson"), mat = matrix(rnorm
maliste$nom

## [1] "robert"  "johnson"

maliste$nom[2]

## [1] "johnson"
```

Sélectionner des éléments I

Fonctionne (presque) comme pour les vecteurs

Attention à la différence `[[]]` et `[]`

```
l1 <- list(1:2,c("a","c","g","t"),matrix(NA,2,1))
l1[-c(1,3)]

## [[1]]
## [1] "a" "c" "g" "t"

mode(l1[3])

## [1] "list"

mode(l1[[3]])

## [1] "logical"
```

Dépiler une liste I

Commande `unlist`

mets à plat une liste et simplifie en vecteur si possible.

```
unlist(list(1,2:5))

## [1] 1 2 3 4 5

unlist(list("a",2:5))

## [1] "a" "2" "3" "4" "5"
```

Par defaut, récursivement

```
l <- list(list(1,"a",matrix(0,2,1)),c("b","c"),1:2)
unlist(l)

## [1] "1" "a" "0" "0" "b" "c" "1" "2"
```

Dépiler une liste II

```
unlist(l, recursive=FALSE)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
##      [,1]
## [1,]    0
## [2,]    0
##
## [[4]]
## [1] "b"
##
## [[5]]
## [1] "c"
##
## [[6]]
## [1] 1
##
## [[7]]
```

Commande lapply

Très puissant !

Applique une fonction à chaque élément d'une liste. La version sapply simplifie automatiquement.

```
lapply(maliste,length)

## $numero
## [1] 3
##
## $noms
## [1] 2
##
## $mat
## [1] 4

sapply(list(1,2:4,5:9),function(x) sum(x)/3)

## [1]  0.3333333  3.0000000 11.6666667
```

Plan

Structures de données

Plan

Structures de données

Tableau de données : définition

Un autre point fort de R

Definition (objet `data.frame`)

C'est une liste à laquelle on impose certaines contraintes afin de rassembler vecteurs et facteurs sous la forme d'un tableau de données.

Tableau de données : définition

Un autre point fort de R

Definition (objet `data.frame`)

C'est une liste à laquelle on impose certaines contraintes afin de rassembler vecteurs et facteurs sous la forme d'un tableau de données.

- Pratiquement, *un tableau de données est une matrice dont les colonnes sont de mode différent*,
- C'est l'objet idéal pour la manipulation de données (forcez-vous à l'utiliser).

Création de tableau de données

Syntaxe

On peut spécifier le nom des colonnes par le vecteur `row.names` ou directement comme pour une liste :

`data.frame(e1=,e2=,...,row.names=)`

```
age <- c(25,35,32,27,32,40,26,25,26,28,30,NA,36,30,30)
grd <- c("thd","CR","MdC","thd","thd","MdC","MdC","thd","thd","MdC","CR","MdC","CR"
sex <- factor(sample(c(rep("M",3),rep("F",12))))
donnees <- data.frame(age=age,grade=grd,sexe=sex)
head(donnees)
```

```
##   age grade sexe
## 1  25   thd    F
## 2  35    CR    F
## 3  32   MdC    F
## 4  27   thd    F
## 5  32   thd    M
## 6  40   MdC    F
```

Plan

Structures de données

Manipulation des éléments du tableau de données I

- Comme une liste !

```
donnees$age
```

```
##  [1] 25 35 32 27 32 40 26 25 26 28 30 NA 36 30 30
```

```
donnees[2]
```

```
##    grade
## 1    thd
## 2     CR
## 3    MdC
## 4    thd
## 5    thd
## 6    MdC
## 7    MdC
## 8    thd
## 9    thd
## 10   MdC
## 11    CR
## 12   MdC
## 13    CR
```

Manipulation des éléments du tableau de données II

- Mais aussi comme une matrice !

```
donnees[2, ]

##   age grade sexe
## 2  35    CR    F

donnees[1:2, c(1,3)]

##   age sexe
## 1  25    F
## 2  35    F
```

Attention tout de même

Un `data.frame` reste plus proche de la liste...

```
length(donnees)

## [1] 3

dim(donnees)

## [1] 15  3

class(donnees)

## [1] "data.frame"

mode(donnees)

## [1] "list"
```

Le couple attach/detach I

les commandes attach() / detach placent / ôtent les éléments du tableaux de données dans l'itinéraire de recherche

```
attach(donnees,warn.conflicts=FALSE)
grade

##  [1] thd CR  MdC thd thd MdC MdC thd thd MdC CR  MdC CR  thd thd
## Levels: CR MdC thd

age

##  [1] 25 35 32 27 32 40 26 25 26 28 30 NA 36 30 30

detach(donnees)
```

Les fonctions stack/unstack

Empile/dépile les colonnes d'une data.frame

```
data(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
head(stack(cars))
```

```
##   values   ind
## 1      4 speed
## 2      4 speed
## 3      7 speed
## 4      7 speed
## 5      8 speed
## 6      9 speed
```

La fonction summary

Réalise un résumé statistique adapté au type de chaque colonne.

```
summary(donnees)
```

```
##       age          grade   sexe
##  Min.   :25.00   CR :3   F:12
##  1st Qu.:26.25   MdC:5   M: 3
##  Median :30.00   thd:7
##  Mean   :30.14
##  3rd Qu.:32.00
##  Max.   :40.00
##  NA's   :1
```

La fonction by

Équivalent de `tapply` pour les tableaux de données ; à utiliser couplé à `with`.

```
attach(donnees,warn.conflicts=FALSE)
by(age,sexe,mean,na.rm=TRUE)

## sexe: F
## [1] 30.81818
## ------------------------------------------------------------
## sexe: M
## [1] 27.66667

detach(donnees)
with(donnees, by(age,grade,mean,na.rm=TRUE))

## grade: CR
## [1] 33.66667
## ------------------------------------------------------------
## grade: MdC
## [1] 31.5
## ------------------------------------------------------------
## grade: thd
```

La fonction aggregate

Permet un `tapply` multivarié ! Attention aux nœuds dans la tête...

```
aggregate(donnees, list(donnees$grade,donnees$sexe), length)
```

```
##   Group.1 Group.2 age grade sexe
## 1      CR       F   3     3    3
## 2     MdC       F   4     4    4
## 3     thd       F   5     5    5
## 4     MdC       M   1     1    1
## 5     thd       M   2     2    2
```

Note pour plus tard

Le `data.frame` est l'object idéal pour l'analyse statistique et la manipulation de données

```
anova(lm(age ~ sexe*grade,data=donnees))

## Analysis of Variance Table
##
## Response: age
##             Df  Sum Sq Mean Sq F value Pr(>F)
## sexe         1  23.411  23.411  1.4940 0.2526
## grade        2  65.931  32.966  2.1037 0.1780
## sexe:grade   1  33.338  33.338  2.1275 0.1787
## Residuals    9 141.033  15.670
```

Troisième partie III

Entrées, sorties et statistique descriptive

Plan

Entrées, sorties et statistique descriptive

Plan

Entrées, sorties et statistique descriptive

Plan

Entrées, sorties et statistique descriptive

Saisir des données

Alternative à la concaténation

commande `scan`

Une utilisation élémentaire de `scan` permet une saisie plus agréable que la saisie directe des éléments d'un vecteur.

```
> x<-scan()
1: 1
2: 2
3: 3
4: 4
5: 5
6:
Read 5 items
>
> x
[1] 1 2 3 4 5
```

$\rightsquigarrow$ valable pour les jeux de données d'au plus quelques dizaines d'éléments...

Éditer des données

commande `edit`

Permet d'éditer des données existantes à l'aide d'un mini-tableur. Utile pour faire de petites modifications.

```
> new.data <- edit(old.data)
```

FIGURE : Éditeur Mac OS 10.6 / R 2.10 (obsolète !

Fichiers binaires

commandes save et load

save sauvegarde un sous ensemble des variables de l'espace de travail dans un fichier binaire ; load permet de les recharger.

```
x <- rnorm(125)
y <- 1 + x + x^2
save(file="mes_simus",x,y)
rm(list=ls())
objects()

## character(0)

load(file="mes_simus")
objects()

## [1] "x" "y"
```

Jeux de données prédéfinies

commande data

R dispose d'une collection de données prédéfinies directement utilisables.
La commande `data()` permet de les lister puis de les charger.

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

- La description d'un jeu de données est accessible dans l'aide.
- L'installation d'un nouveau package rend souvent disponibles de nouveaux jeux de données accessibles par `data`.

Lecture de fichiers

Méthodologie

Un bon éditeur permet de constater le formatage d'un fichier texte et comment en « attaquer » l'importation.

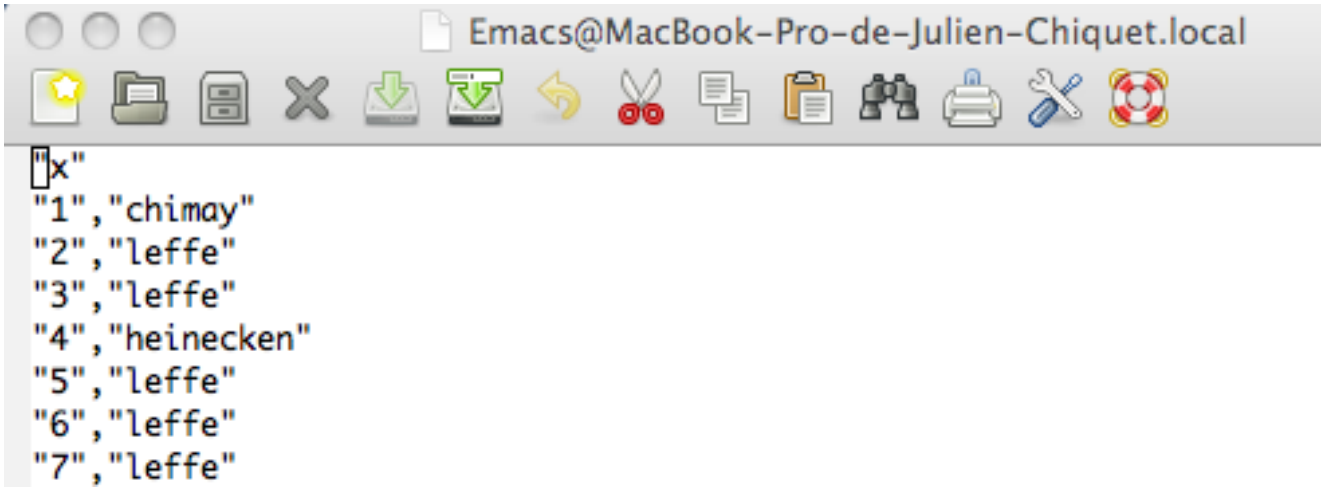


FIGURE : Fichier au formatage "csv"

Lecture de fichiers I

La fonction générique

commande `read.table`

Permet de lire un fichier formaté sous forme de table et de le convertir sous la forme du objet `data.frame`. Parmi les nombreuses options, les plus importantes sont

- `header` : présence ou pas d'une ligne nommant les colonnes du tableau
- `sep` : la chaîne de caractère définissant le séparateur (par défaut, un ou plusieurs espaces).

Lecture de fichiers II

La fonction générique

```
vignes <- read.table("data/baies_raisin.txt")

## Error in scan(file = file, what = what, sep = sep, quote = quote, dec = dec, :
la ligne 2 n'avait pas 20 éléments
```

Lecture de fichiers III

La fonction générique

```
vignes <- read.table("data/baies_raisin.txt", sep='\t')
head(vignes)
```

```
##           V1      V2                    V3                          V4
## 1 Population variete nbre pepin/baie 2008 poids pulpe/baie (g) 2008
## 2         CE    1784                 1.00                       0.89
## 3         CE     124                 1.00                       1.14
## 4         CE     210                 1.20                       1.26
## 5         CE    1805                 1.20                       0.66
## 6         CE    1303                 1.20                       0.83
##                       V5                    V6                          V7
## 1 volume baie (cm3) 2008 nbre pepin/baie 2009 poids pulpe/baie (g) 2009
## 2                 7.70
## 3                 8.82
## 4                10.20
## 5
## 6
```

Lecture de fichiers IV

La fonction générique

```
vignes <- read.table("data/baies_raisin.txt", sep='\t', header=TRUE)
head(vignes)
```

```
##   Population variete nbre.pepin.baie.2008 poids.pulpe.baie..g..2008
## 1         CE    1784                   1.0                      0.89
## 2         CE     124                   1.0                      1.14
## 3         CE     210                   1.2                      1.26
## 4         CE    1805                   1.2                      0.66
## 5         CE    1303                   1.2                      0.83
## 6         CE     284                   1.3                      0.54
##   volume.baie..cm3..2008 nbre.pepin.baie.2009 poids.pulpe.baie..g..2009
## 1                   7.70                    NA                       NA
## 2                   8.82                    NA                       NA
## 3                  10.20                    NA                       NA
## 4                     NA                    NA                       NA
## 5                     NA                    NA                       NA
## 6                   4.61                    NA                       NA
```

Lecture de fichiers

`read.csv,read.delim`

Commandes `read.csv` et `read.delim`

Raccourcis pour la fonction `read.table`, spécialisés dans l'importation des données « .csv » (*comma-separated value*) ou tabulées (le séparateur est la tabulation).

```
vignes <- read.delim(file="data/baies_raisin.txt", header=TRUE)
head(vignes)
```

```
##   Population variete nbre.pepin.baie.2008 poids.pulpe.baie..g..2008
## 1         CE    1784                   1.0                      0.89
## 2         CE     124                   1.0                      1.14
## 3         CE     210                   1.2                      1.26
## 4         CE    1805                   1.2                      0.66
## 5         CE    1303                   1.2                      0.83
## 6         CE     284                   1.3                      0.54
##   volume.baie..cm3..2008 nbre.pepin.baie.2009 poids.pulpe.baie..g..2009
## 1                   7.70                    NA                       NA
## 2                   8.82                    NA                       NA
## 3                  10.20                    NA                       NA
```

Écriture dans un fichiers I

write.table

commandes write.table, write.csv et write.delim

La fonction write.table permet d'imprimer les données issues d'un data.frame dans un fichier texte externe. write.csv et write.delim sont des raccourcis pour les données csv ou tabulée.

```
vignes2008 <- vignes[, 1:5]
write.table(vignes2008, file="data/baies_raisin2008.txt")
rm(vignes2008)
```

Écriture dans un fichiers II

write.table

```
head(read.table(file="data/baies_raisin2008.txt", header=TRUE))
```

```
##   Population variete nbre.pepin.baie.2008 poids.pulpe.baie..g..2008
## 1         CE    1784                   1.0                      0.89
## 2         CE     124                   1.0                      1.14
## 3         CE     210                   1.2                      1.26
## 4         CE    1805                   1.2                      0.66
## 5         CE    1303                   1.2                      0.83
## 6         CE     284                   1.3                      0.54
##   volume.baie..cm3..2008
## 1                   7.70
## 2                   8.82
## 3                  10.20
## 4                     NA
## 5                     NA
## 6                   4.61
```

Pour aller plus loin...

Beaucoup de choses sur l'importation des données dans

R Data Import /Export.
`http://cran.r-project.org/doc/manuals/R-data.pdf`

- Exemples avancés avec `read.table`,
- communication avec les bases de données (SQL),
- importation de données Excel,
- ...

Plan

Entrées, sorties et statistique descriptive

Paramètres récurrents

Forme générique

La plupart des fonctions graphique s'utilisent par un appel du type

1. `nom.function(object, options)`,
2. `nom.function(x, y , options)`.

Paramètres récurrents

Forme générique

La plupart des fonctions graphique s'utilisent par un appel du type

1. `nom.function(object, options)`,
2. `nom.function(x, y , options)`.

Parmi les options les plus courantes, on trouve :

- type="p" ; spécifie le type de tracé : "p" pour points, "l" pour lignes, "b" pour points liés par des lignes, "o" pour lignes superposées aux points...
- xlim= ; ylim=, spécifie les limites de axes x et y
- xlab= ; ylab=, annotation des axes x et y
- main= ; titre du graphe en cours
- sub= ; sous-titre du graphe en cours
- add=FALSE ; si TRUE superpose le graphe au précédent
- axes=TRUE ; si FALSE ne trace pas d'axes

Représenter un objet graphiquement I

commande `plot`

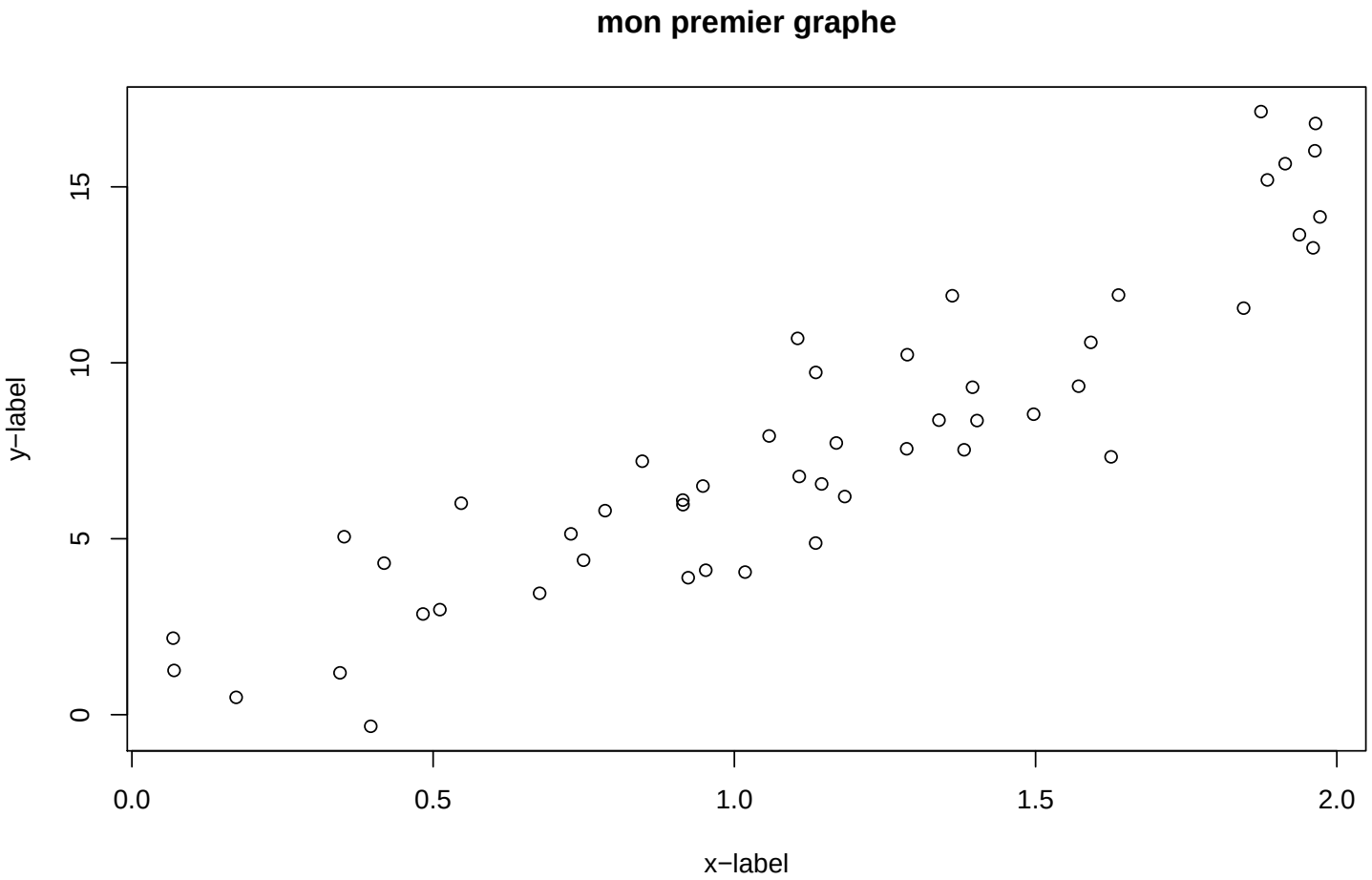
Fonction élémentaire de réprésentation graphique.

- `plot(vect)` représente le graphe des valeurs de `vect` sur l'axe des y.
- `plot(vect1,vect1)` représente le graphe des valeurs de `vect2` en fonction de `vect1`. `plot(object,...)` appelle la méthode `plot.class` si elle est définie pour l'objet de class `class`.

Par exemple, avec deux vecteurs :

```
x <- runif(50,0,2)
y <- 3 * x + 2 * x^2 + 1 + rnorm(50,sd=1.5)
plot(x, y, xlab="x-label",ylab="y-label",main="mon premier graphe")
```

Représenter un objet graphiquement II

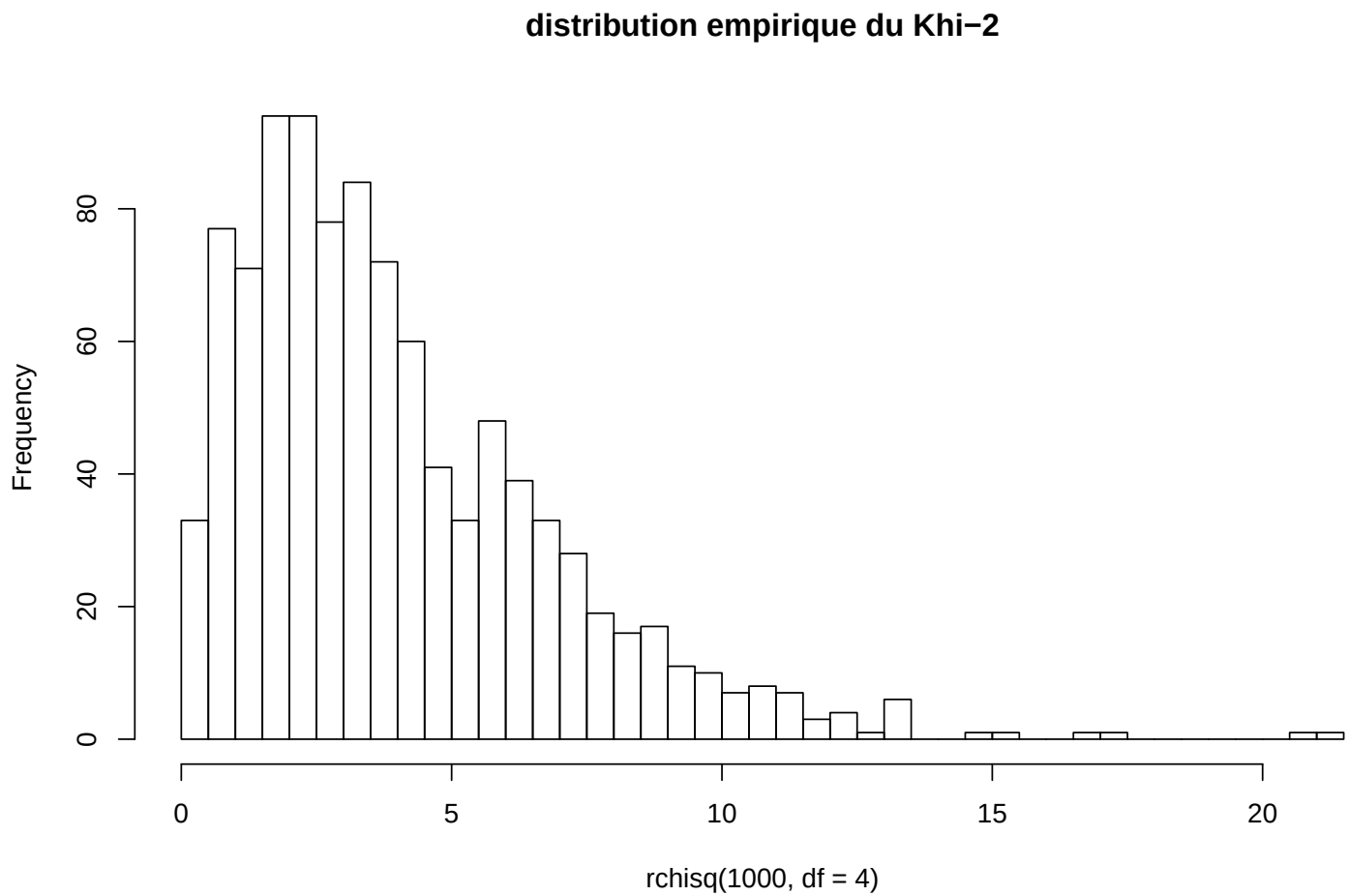


Autres exemples d'utilisation de plot (I) I

Beaucoup d'objet R accepte la commande plot ! En particulier, les histogrammes :

```
mon_histo <- hist(rchisq(1000,df=4),nclass=75, plot=FALSE)
plot(mon_histo,main="distribution empirique du Khi-2")
```

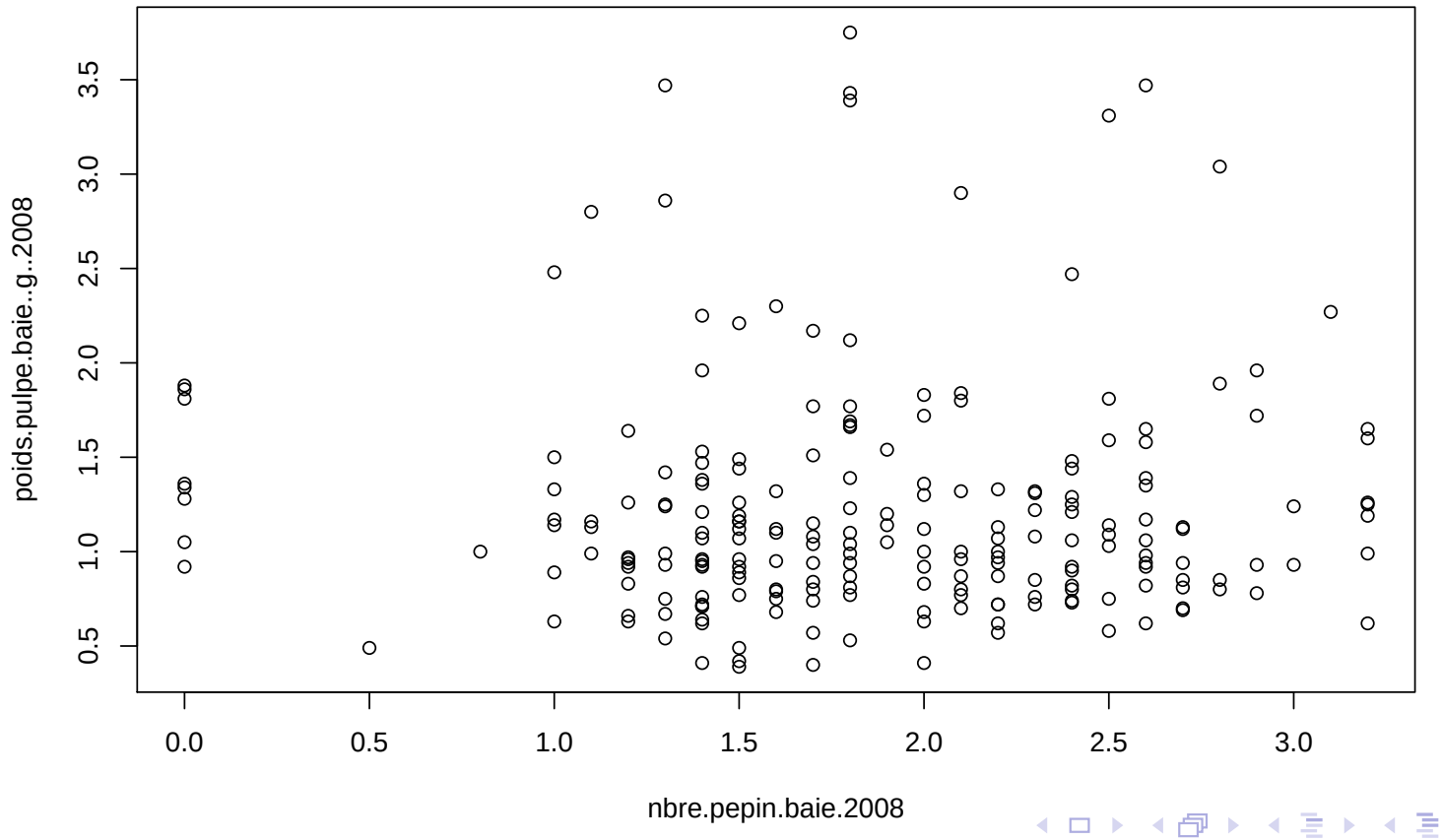
Autres exemples d'utilisation de plot (I) II



Autres exemples d'utilisation de plot (II)

Objet "formule" entre variables numériques : graphe de dispersion

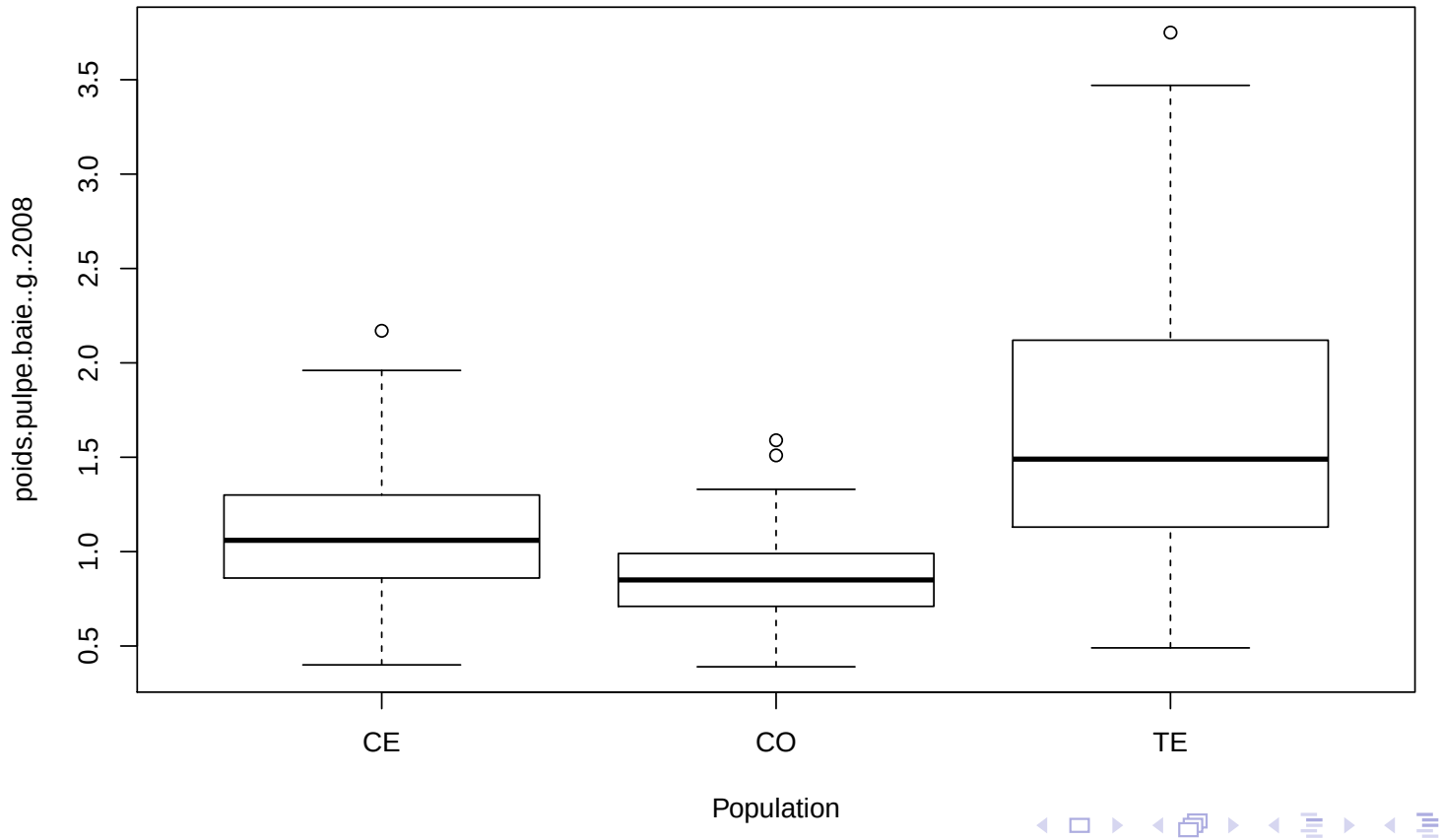
```
plot(poids.pulpe.baie..g..2008~nbre.pepin.baie.2008, vignes)
```



Autres exemples d'utilisation de plot (III)

Objet "formule" entre variables numérique et catégorielle : boxplot

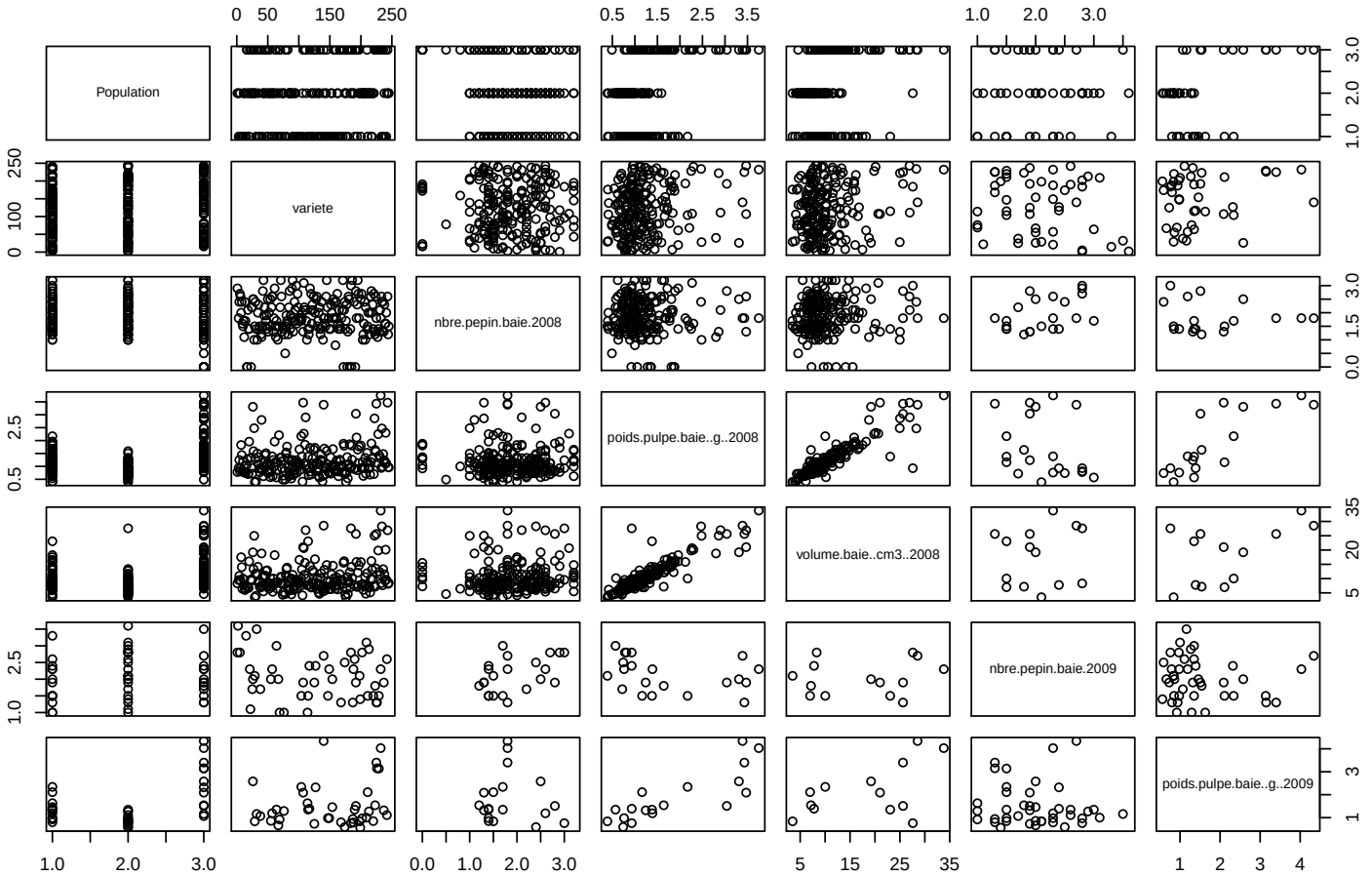
```
plot(poids.pulpe.baie..g..2008~Population, vignes)
```



Autres exemples d'utilisation de plot (IV)

Objet "data.frame" : graphes pair à pair

```
plot(vignes)
```



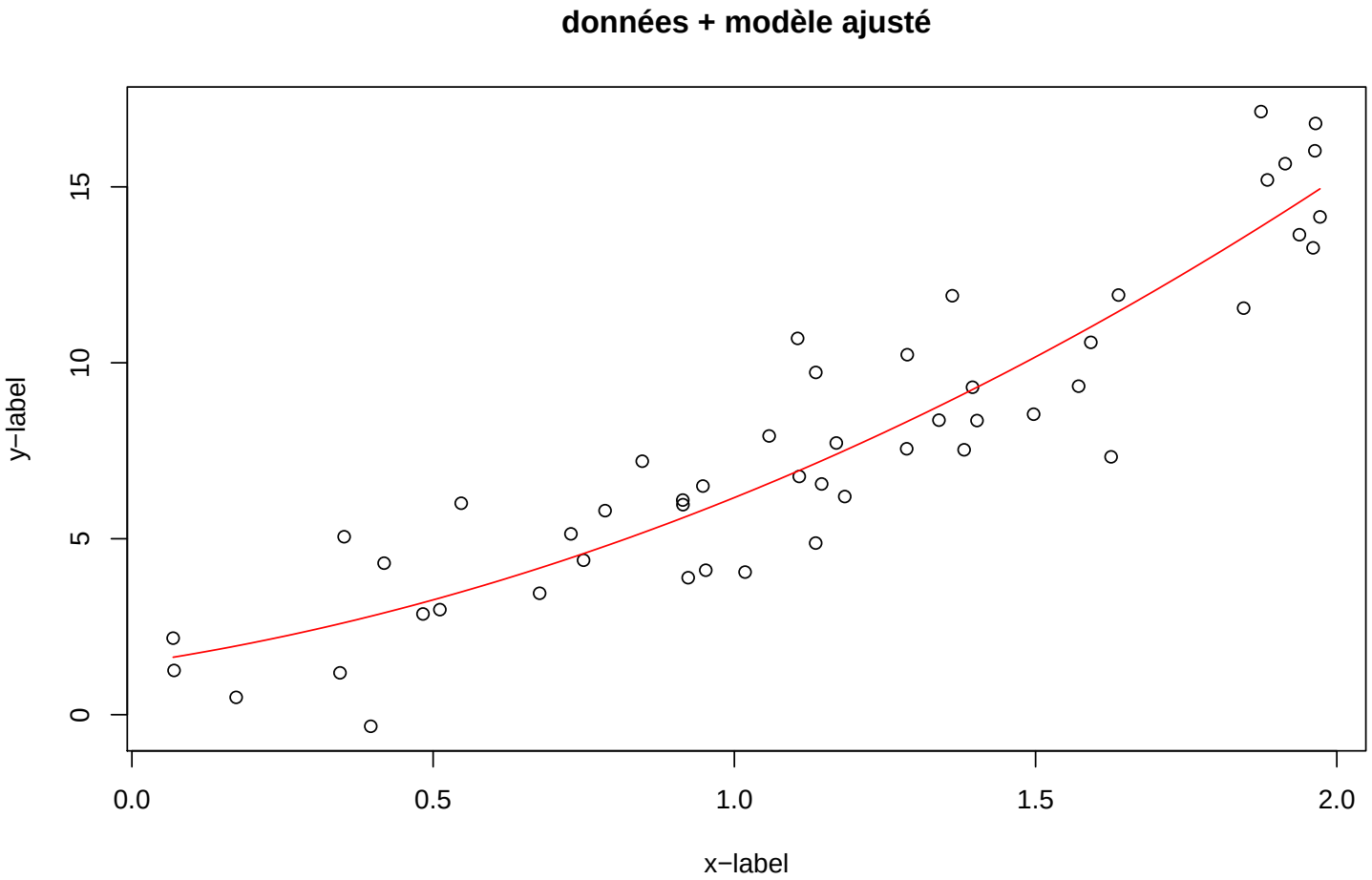
Tracer une fonction symbolique I

commande curve

Elle permet de tracer une fonction définie par une expression de x.

```
plot(x, y, main="données + modèle ajusté",
     xlab="x-label", ylab="y-label")
a <- coefficients(lm(y~1+x+I(x^2)))
curve(a[1] + a[2]*x + a[3]*x^2,add=TRUE,col="red")
```

Tracer une fonction symbolique II



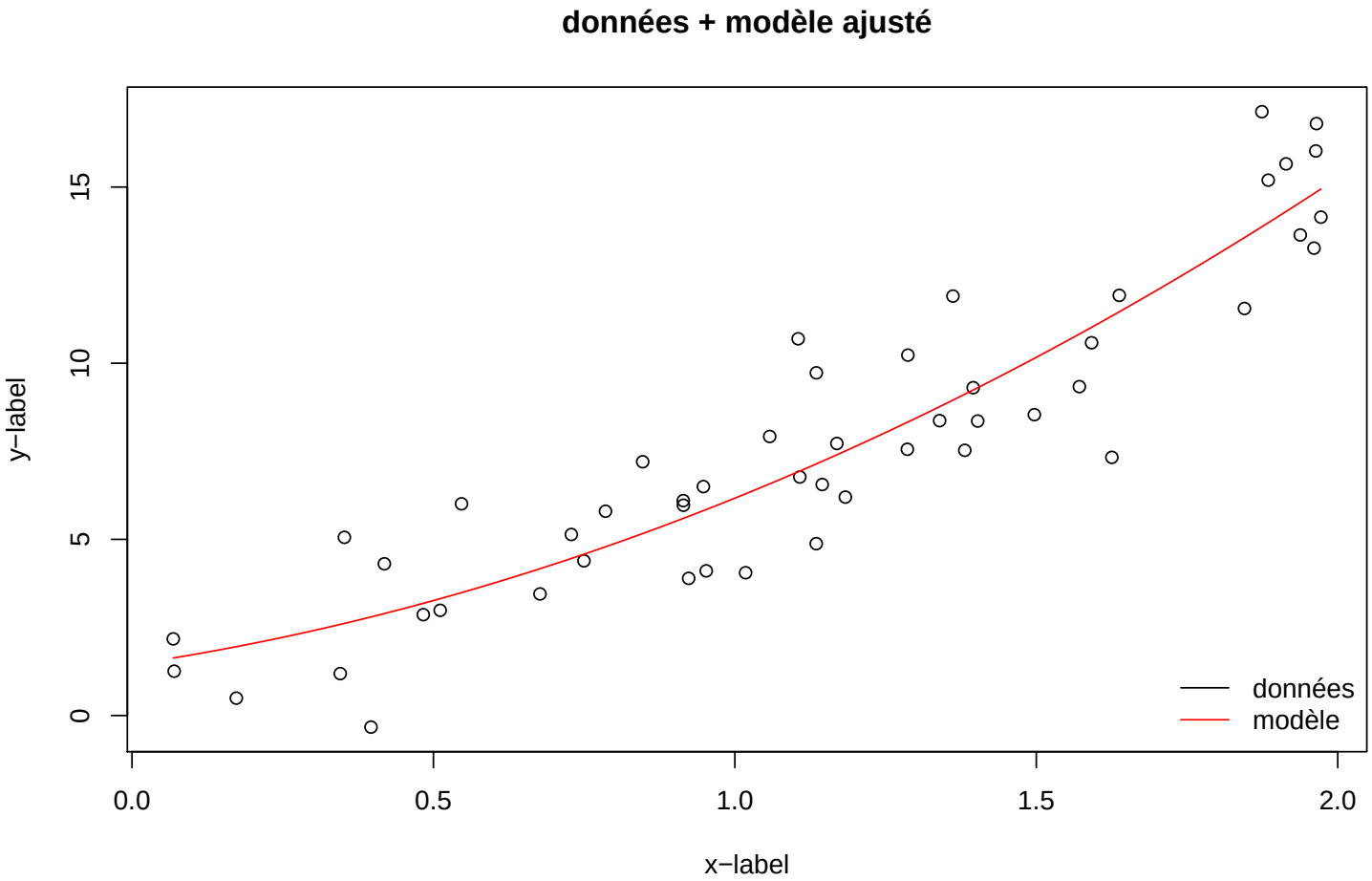
Ajouter une légende I

commande `legend`

Pour ajouter une légende. Attention aux options, assez nombreuses !

```
plot(x, y, main="données + modèle ajusté",
     xlab="x-label", ylab="y-label")
a <- coefficients(lm(y~1+x+I(x^2)))
curve(a[1] + a[2]*x + a[3]*x^2,add=TRUE,col="red")
 legend("bottomright",c("données","modèle"),lty=c(1,1),col=c("black","red"),bty="n")
```

Ajouter une légende II



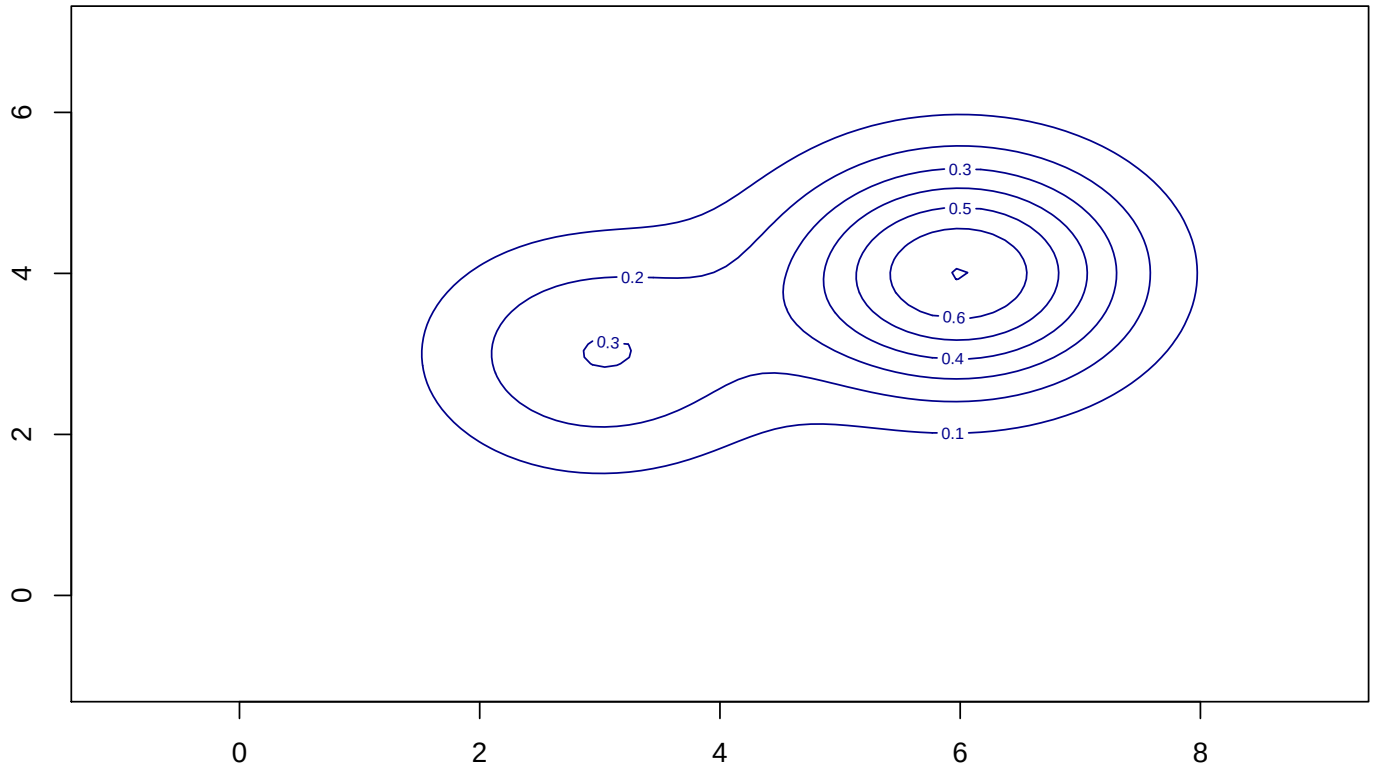
Représentation 3D (courbe de niveaux) I

commande contour

`contour(x,y,z)` permet de tracer des courbes de niveaux : `x` et `y` sont des vecteurs et `z` une matrice telle que les dimensions de `z` soient `length(x),length(y)`.

```
x<-seq(-1,9,length=100)
y<-seq(-1,7,length=100)
z<-outer(x,y,function(x,y) 0.3*exp(-0.5*((x-3)^2 +(y -3)^2)) +
         0.7*exp(-0.5*((x-6)^2 +(y -4)^2)))
contour(x,y,z,col="blue4")
```

Représentation 3D (courbe de niveaux) II



Ajout de droites I

commande abline

`abline` permet d'ajouter à un graphe courant

- des droites de décalage `a` et de coefficient directeur `b` avec `abline(a,b)`,
- des droites verticales avec `abline(v=)`,
- des droites horizontales avec `abline(h=)`.

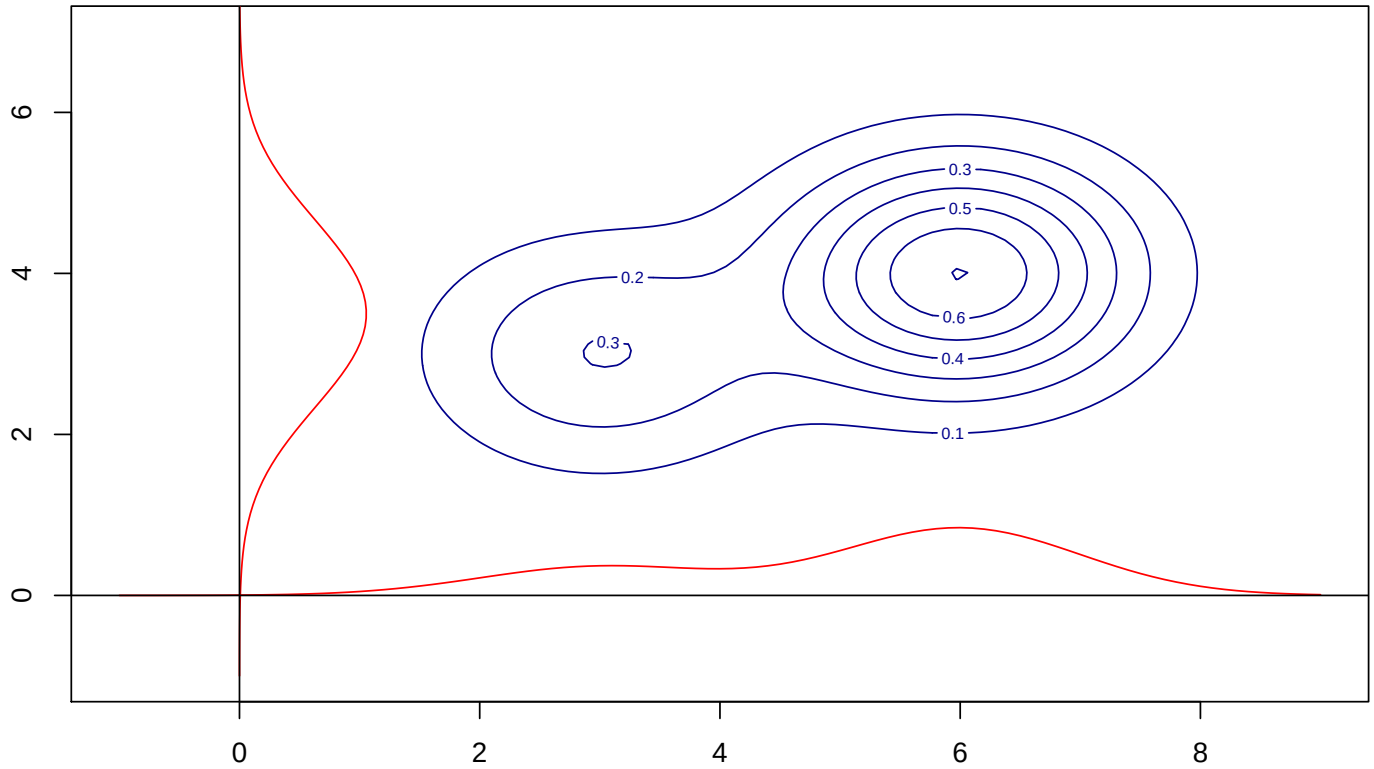
commandes lines et points

Pour ajouter une courbe ou des points : s'utilisent de manière similaire à `plot`.

Ajout de droites II

```
contour(x,y,z,col="blue4")
curve((0.3*dnorm(x,mean=3) + 0.7*dnorm(x,mean=6))*3,-1,9,col="red",ylim=c(-1,7),add=
x<-seq(-1,9,length=100)
lines((0.5*dnorm(x,mean=3) + 0.5*dnorm(x,mean=4))*3,x,col="red")
abline(h=0)
abline(v=0)
```

Ajout de droites III



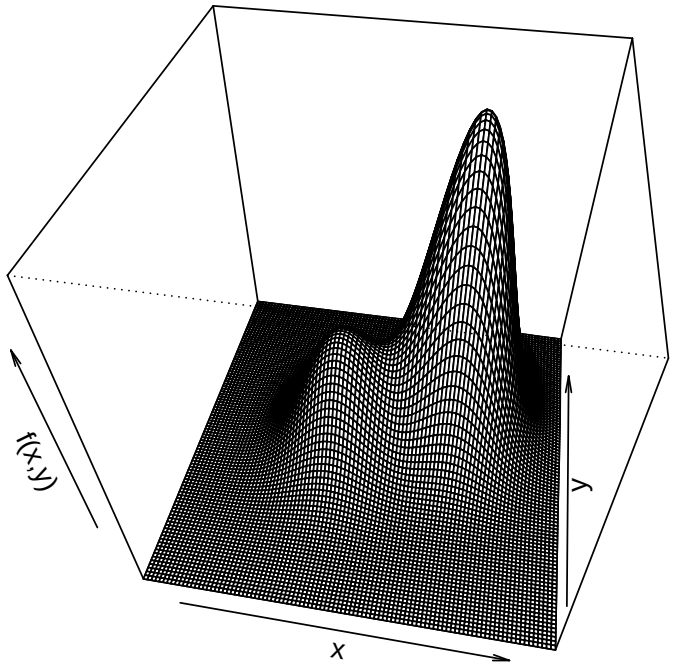
Graphe en 3D I

commande persp

Fonctionne comme la fonction contour en proposant une représentation en perpective.

```
persp(x,y,z, box=TRUE,theta = 10, phi = 45,xlab = "x", ylab = "y", zlab = "f(x,y)")
```

Graphe en 3D II



Rediriger la sortie graphique

Par défaut, R envoie les graphiques sur la sortie *écran*. De nombreuses

Exportation de graphes

Se réalise en encadrant les fonctions graphiques par les commandes `format_export(file="nom_fichier")` et `dev.off()`, où `format_fichier` peut prendre les valeurs `pdf`,`postscrip`,`png`, ...

```
pdf(file="ma_sortie.pdf")
plot(runif(20),runif(20))
dev.off()
```

Graphes multiples

Ouverture d'une nouvelle fenêtre graphique

Se fait, selon les plateformes, avec les commandes

- `x11())` pour Linux,
- `quartz()` ou `x11())` pour Mac OS,
- `windows()`.

Découpage d'une fenêtre

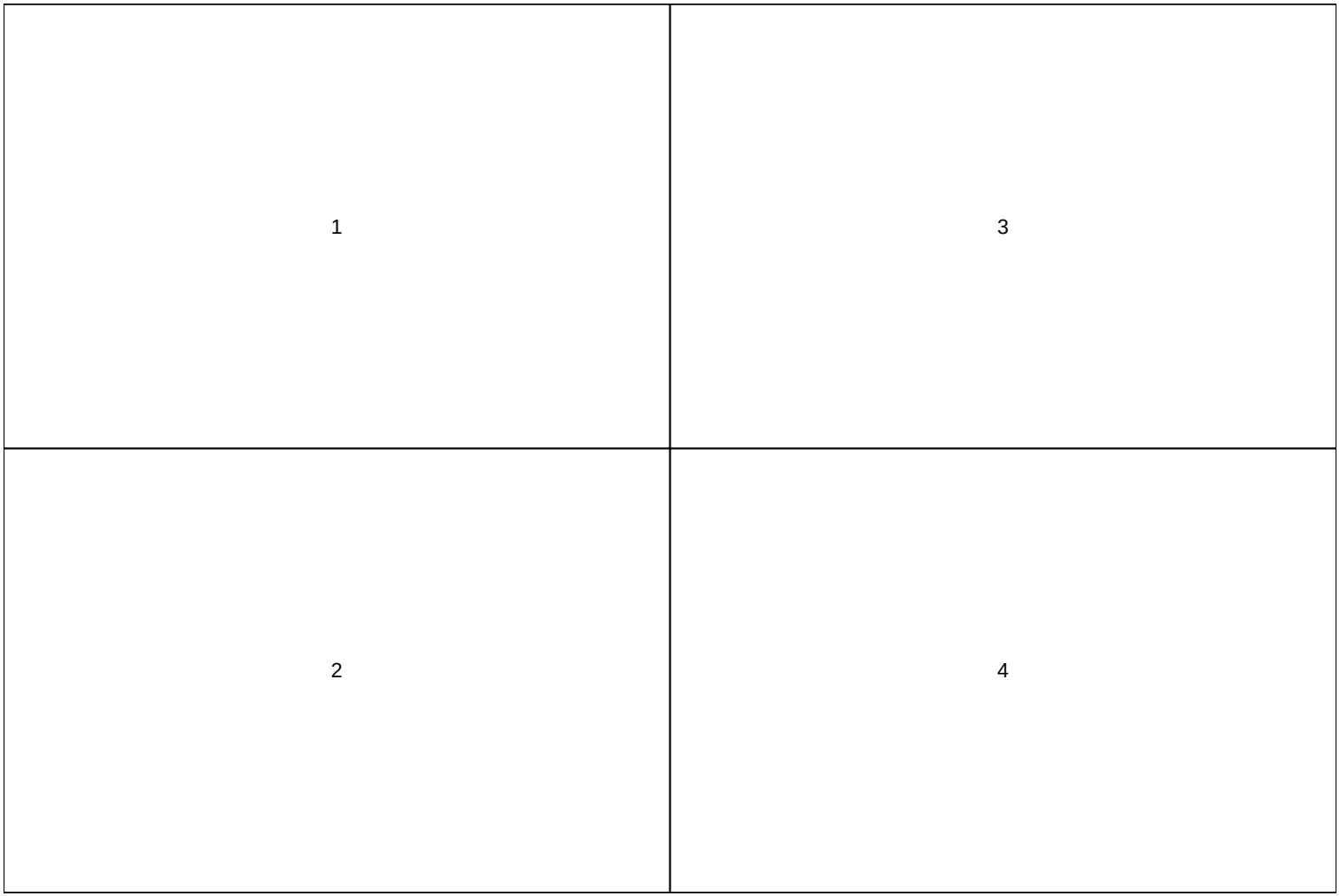
Plusieurs possibilités :

- `layout(mat,width=,height=)`, qui s'utilise en découpant l'écran via la matrice `mat`.
- `par(mfrow=vect)` ou `par(mfcol=vect)` qui découpent en n lignes et m colonne spécifiées par le vecteur `vect`. Le remplissage se fait par ligne ou par colonne selon la fonction choisie.

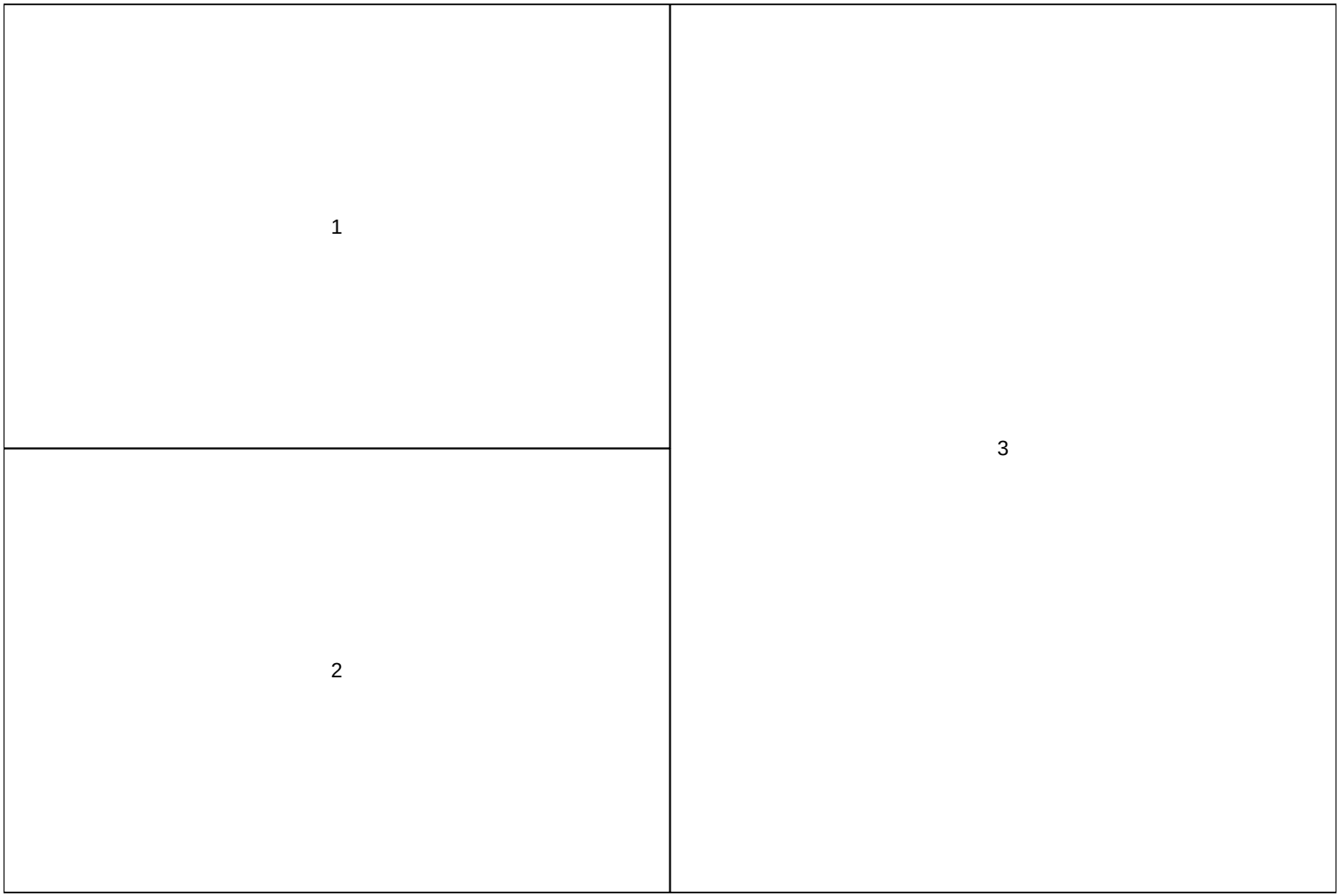
Découpage du support graphique I

```
m1 <- matrix(1:4,2,2)
layout(m1)

m2 <- matrix(c(1:3,3),2,2)
layout(m2)

m3 <- matrix(0:3,2,2)
layout(m3,c(1,3),c(1,3))
```

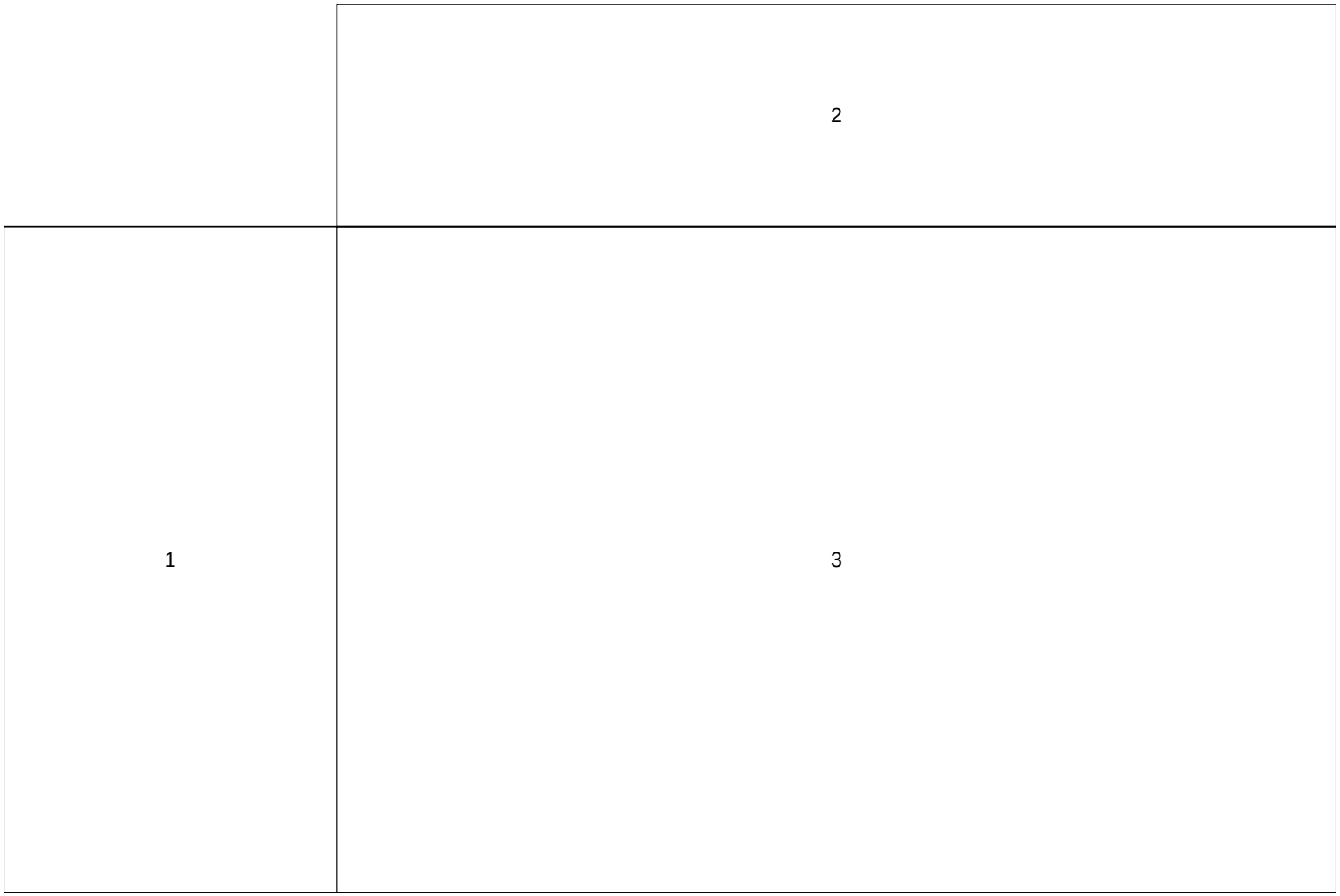
Découpage du support graphique I



Découpage du support graphique II



Découpage du support graphique III



Pour aller plus loin

- La commande `par` gère les options graphiques,
- Le package `lattice`, pour des graphes multivariés,
- Le package `ggplot2`, dont nous verrons une introduction en fin de module

Lattice : Multivariate Data Visualization with R Deepayan Sarkar
`http://lmdvr.r-forge.r-project.org/`

ggplot2 : Grammar of graphics, Hadley Wickham
`http://ggplot2.org/`

$\rightsquigarrow$ Au delà des mécanismes de représentation graphiques élémentaires, les possibilités graphiques de R sont **liées à la nature des résumés statistiques** opérés sur les données (cf. section suviante).

Plan

Entrées, sorties et statistique descriptive

Statistique

Statistique : étude de la variabilité

Définitions

Le terme **statistique** est utilisé pour désigner trois notions distinctes :

1. Recueil de données
2. Méthodes utilisées pour analyser ces données
3. Toute grandeur calculée à partir d'observations

Statistique

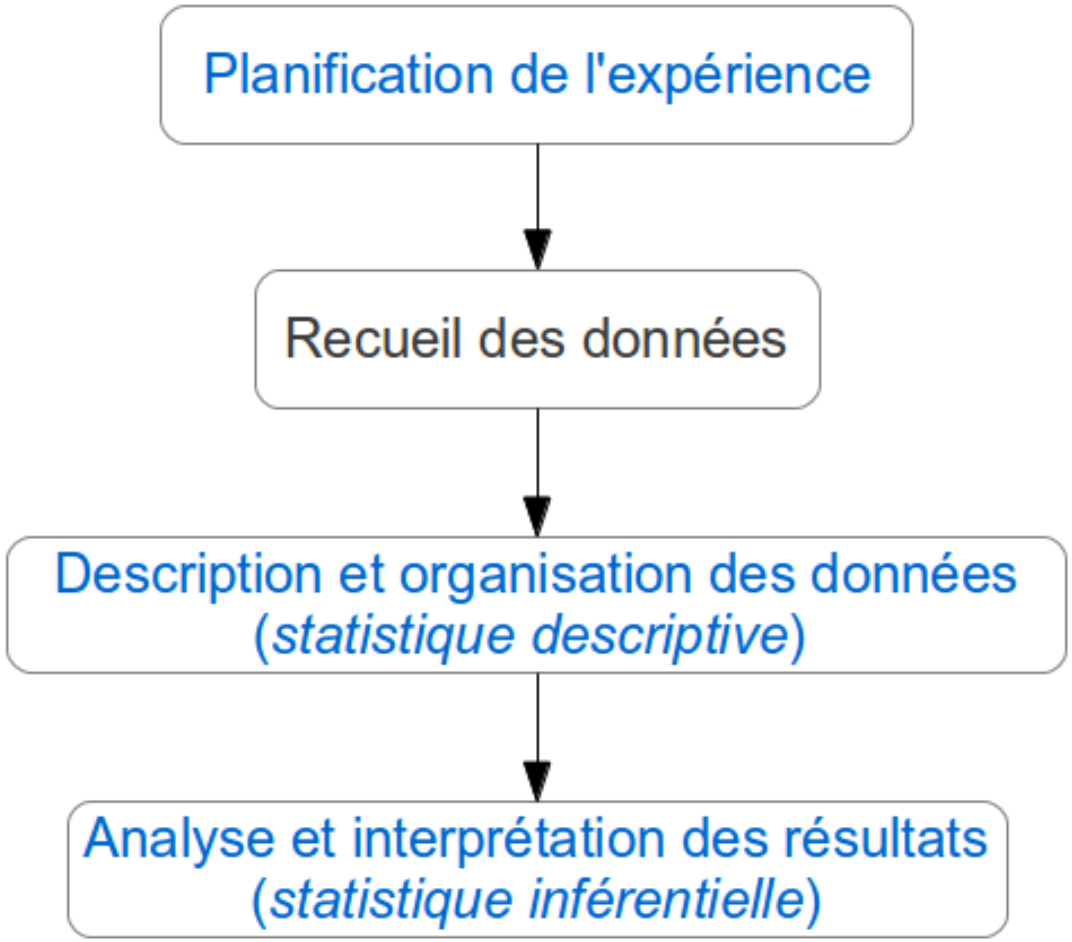
Définitions

- On appelle **statistique descriptive** l'ensemble des méthodes et techniques mathématiques permettant de représenter, de décrire et de résumer un ensemble de données.
- On appelle **statistique inférentielle** (ou inductive) l'ensemble des méthodes visant à modéliser un ensemble de données afin de tirer des conclusions sur un ensemble plus vaste.

Remarque

La statistique repose sur des modèles et des hypothèses issus des probabilités.

Démarche statistique



Statistique descriptive

Objectif

Organiser et résumer les données afin d'en dégager les caractéristiques principales sous une forme simple et intelligible

Remarque

La statistique descriptive permet notamment d'identifier des valeurs extrêmes ou aberrantes et de vérifier certaines hypothèses de modélisation

Types de représentation

- Tableaux
- Graphiques
- Indicateurs numériques

Vocabulaire

- **Population** : ensemble (grand, voire infini) d'individus ou d'objets de même nature
- **Echantillon** : sous ensemble de la population
- **Caractère / Variable** : une caractéristique de la population pouvant prendre différentes valeurs
- **Modalité** : toute valeur que peut prendre une variable
- **Série statistique** : ensemble des données recueillie pour un caractère donné à partir d'un échantillon

Types de variables

- Variable **quantitative** : variable/caractère à laquelle on peut associer un nombre
  - **discrète** : ne peut prendre qu'un nombre fini ou dénombrable de valeurs
  - **continue** : peut prendre toutes les valeurs d'un intervalle de l'ensemble des nombres réels
- Variable **qualitative** : variable/caractère dont les modalités ne sont pas quantifiables
  - **ordinale** : variable dont les modalités peuvent être ordonnées
  - **nominale** : variable dont les modalités ne peuvent pas être ordonnées

Plan

Entrées, sorties et statistique descriptive

Tableaux

Tableau individu-caractère

Dans le cadre d'une étude portant sur la contamination du lait par des spores de clostridia, on analyse 10 tubes de 1ml de lait et, pour chaque tube, on compte le nombre de spores présents :

Indice tube	Nombre de spores
1	0
2	1
3	0
4	3
5	2
6	0
7	0
8	1
9	2
10	1

Tableaux

Définitions

Soit X un caractère qualitatif ou quantitatif discret pouvant prendre k modalités $(x^1, ..., x^k)$ observé sur un échantillon de taille n.

- L'**effectif** n_i d'une modalité x^i est le nombre d'individus pour lesquels la modalité x^i a été observée
- La **fréquence** f_i d'une modalité x^i est le nombre f_i tel que :

$$f_i = \frac{n_i}{n}$$

Remarque

Pour un caractère quantitatif continu, la notion de fréquences suppose une répartition des observations en k **classes** (chaque classe étant définie par un intervalle)

Tableaux

Définitions

Soit X un caractère qualitatif **ordinal** ou quantitatif discret pouvant prendre k modalités $(x^1 < ... < x^k)$ observé sur un échantillon de taille n.

- L'**effectif cumulé** N_i d'une modalité x^i est le nombre d'individus pour lesquels une modalité inférieure ou égale à x^i a été observée

$$N_i = \sum_{j=1}^{i} n_j = n_1 + n_2 + ... + n_i$$

- La **fréquence cumulée** F_i d'une modalité x^i est le nombre F_i tel que :

$$F_i = \frac{N_i}{n} = \sum_{j=1}^{i} f_j = f_1 + f_2 + ... + f_i$$

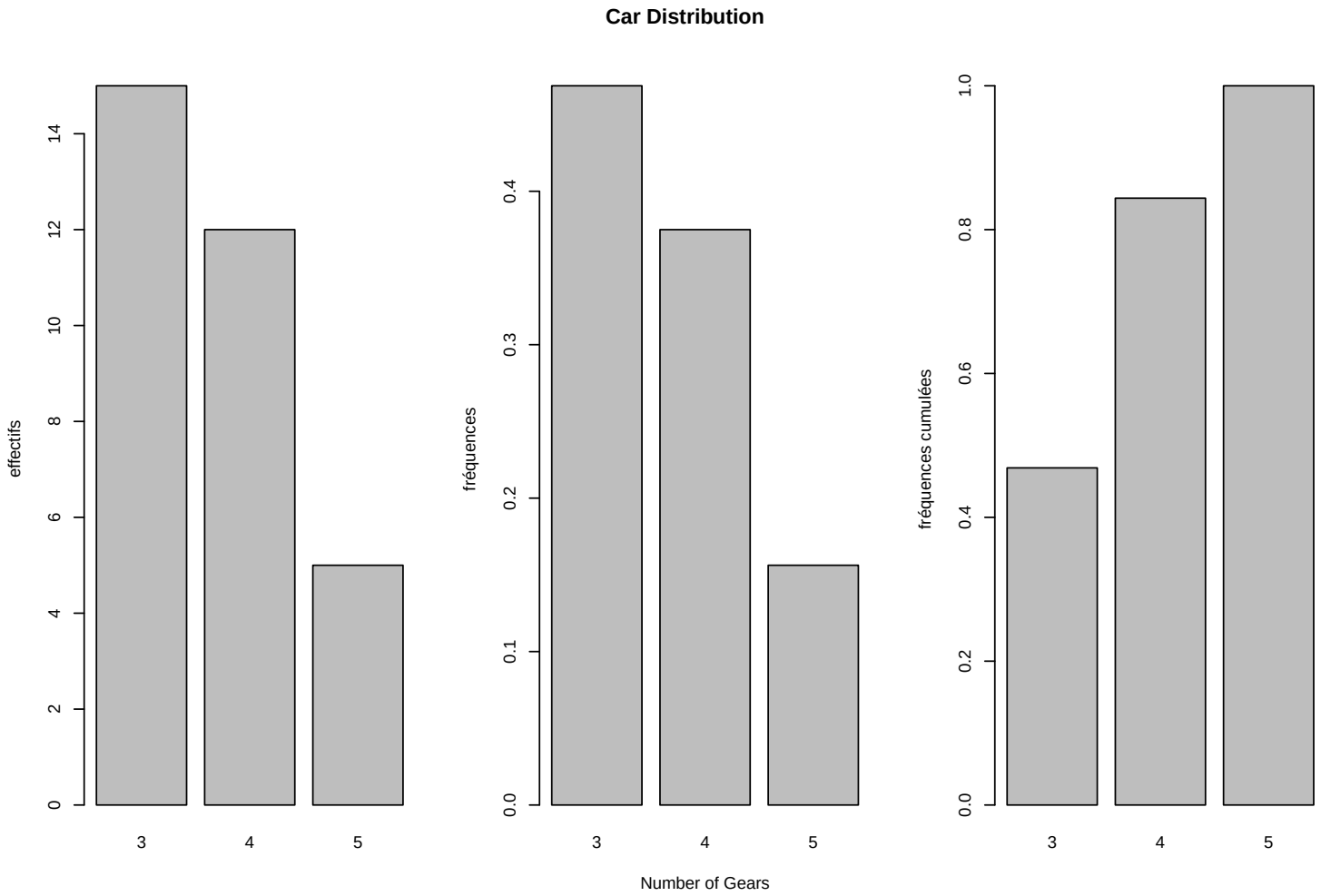
Fréquences I

```
data(mtcars)
print(counts <- table(mtcars$gear))

##
##  3  4  5
## 15 12  5

print(frequences <- counts/length(mtcars$gear))

##
##       3       4       5
## 0.46875 0.37500 0.15625

print(cumFreq     <- cumsum(frequences))

##       3       4       5
## 0.46875 0.84375 1.00000
```

Fréquences II

```
par(mfrow=c(1,3))
barplot(counts, ylab="effectifs", xlab="")
barplot(frequences, ylab="fréquences", xlab="Number of Gears")
barplot(cumFreq, ylab="fréquences cumulées", xlab="")
title(outer=TRUE,main="\nCar Distribution")
```

Fréquences III



Tableaux

Tableau des fréquences

Nombre de spores	0	1	2	3
Nombre de tubes	4	2	2	1
Fréquence	0.4	0.2	0.2	0.1

Tableaux

Table de contingence

On compare les réactions produites par deux vaccins BCG désignés par A et B.

Vaccin	Réaction légère	Réaction moyenne	Ulcération	Abcès	Total
A	12	156	8	1	177
B	29	135	6	1	171
Total	41	291	14	2	348

Plan

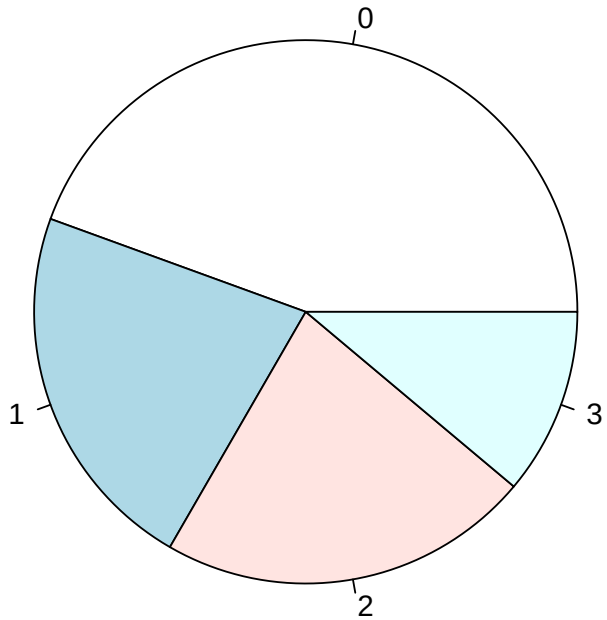
Entrées, sorties et statistique descriptive

Graphiques

Les graphiques donnent de manière immédiate une information sur la distribution des observations.

Diagramme circulaire

Spores de clostridia (variable quantitative discrète)

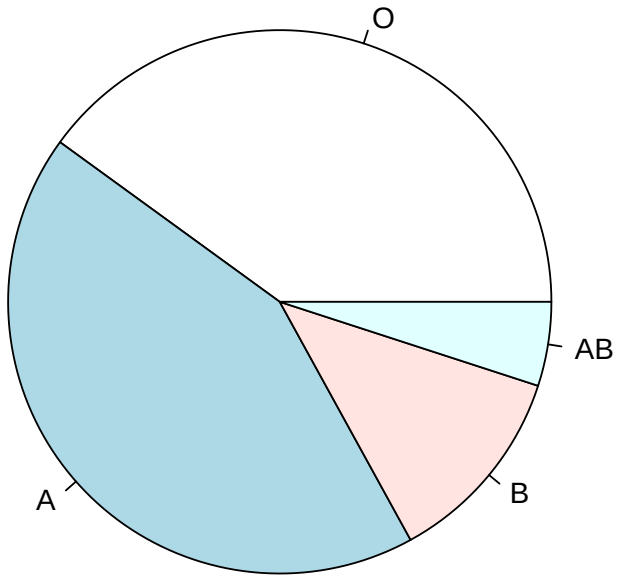


Graphiques

Les graphiques donnent de manière immédiate une information sur la distribution des observations.

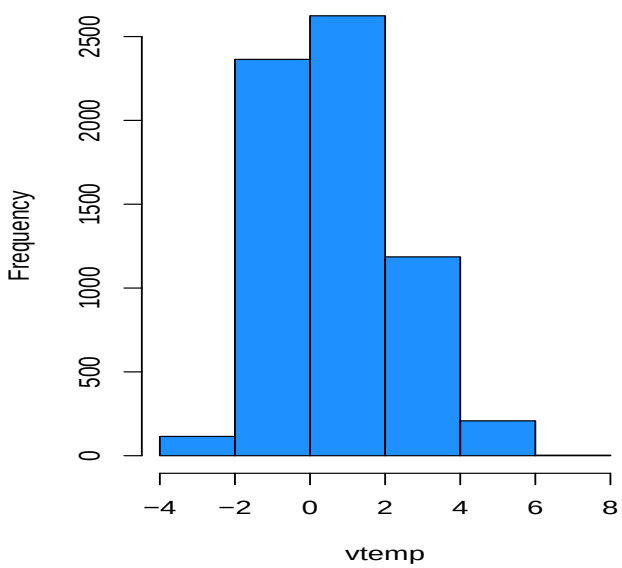
Diagramme circulaire

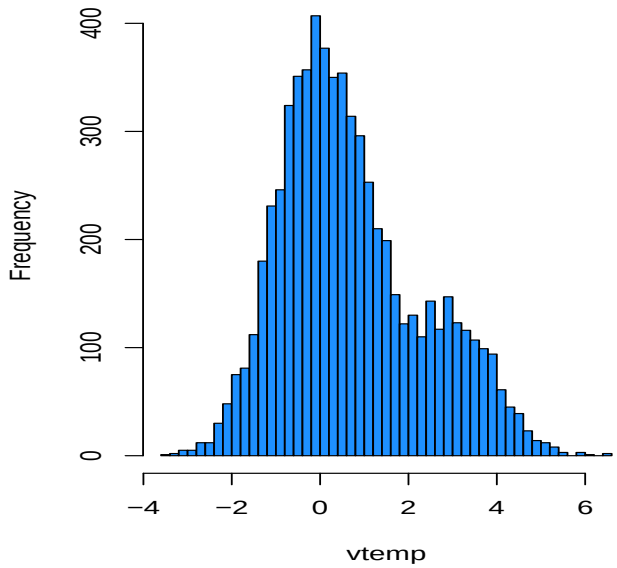
Groupes sanguins (variable qualitative nominale)



Graphiques

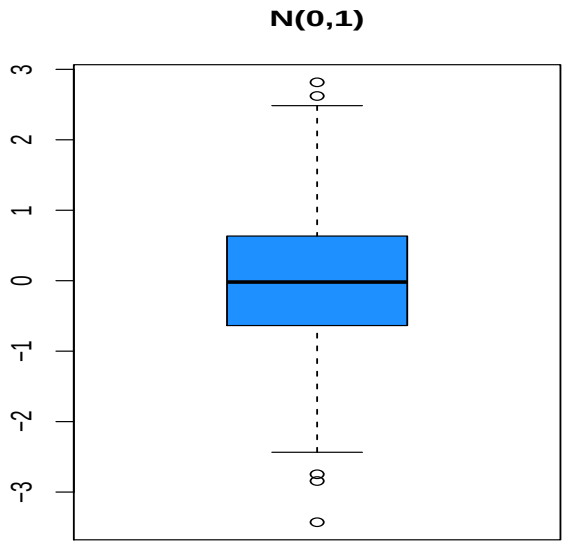
Histogramme

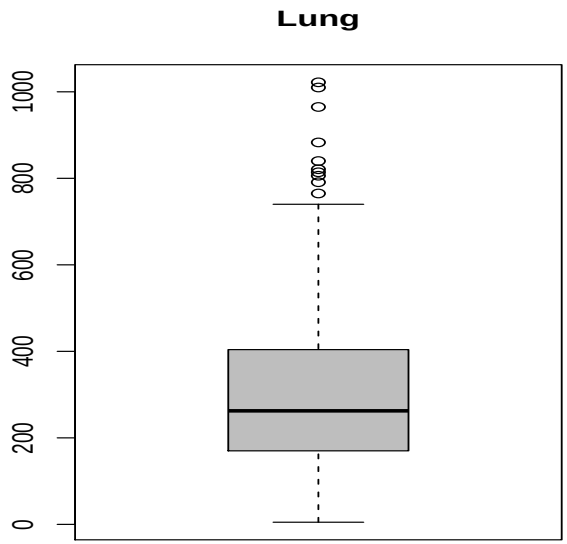




Graphiques

Boîtes à moustaches (boxplot)





Plan

Entrées, sorties et statistique descriptive

Résumés numériques

Indicateurs de tendance centrale

- moyenne empirique : `mean`
- moyenne pondérée : `weighted.mean`
- médiane : `median`

Indicateurs de dispersion

- variance empirique (corrigée) : `var`
- écart-type : `sd`
- étendu : `range`
- fractiles empirique : `quantile`

`summary`/`fivenum` reprend ces indicateurs numériques élémentaires...

Indicateurs de position

Moyenne empirique

La moyenne empirique d'un échantillon est la moyenne arithmétique des observations :

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Indicateurs de position

Quantiles empiriques

Le quantile empirique d'ordre $1/p$ (où p est un entier naturel) est la valeur $\tilde{q}_{1/p}$ qui partage l'échantillon en p parties de même effectif.

Quantiles particuliers

- **Médiane empirique** : quantile d'ordre $1/2$
- **Quartiles** : quantile d'ordre $i/4$
- **Déciles** : quantile d'ordre $i/10$
- **Centiles** : quantile d'ordre $i/100$

Indicateurs de position

Soit $x_1, ..., x_n$ les observations d'un échantillon et soit $x_{(1)} \leq ... \leq x_{(n)}$ les observations ordonnées.

Médiane empirique

La médiane empirique est le quantile d'ordre $1/2$:

- Si n est impair :

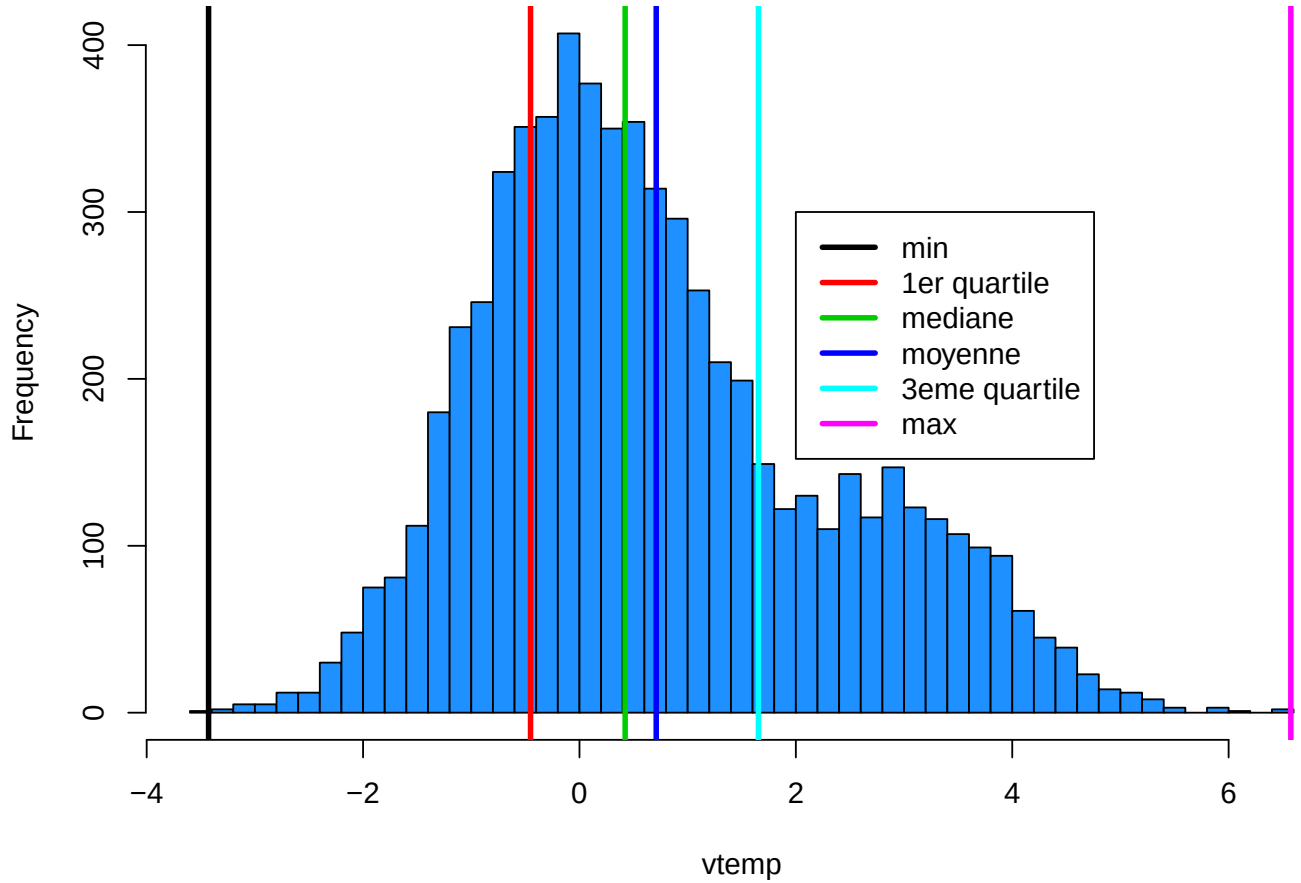
$$\tilde{x} = x_{(\frac{n+1}{2})}$$

- Si n est pair :

Toute valeur comprise entre $x_{(\frac{n}{2})}$ et $x_{(\frac{n}{2}+1)}$

Indicateurs de position

Exemple



Indicateurs de dispersion

Etendue

L'étendue mesure l'écart entre la plus grande et la plus petite des valeurs observées. Elle est définie par :

$$e_n = max(x_i) - min(x_i)$$

Indicateurs de dispersion

Distance interquartile

- L'**intervalle interquartiles** est l'intervalle :

$$[\tilde{q}_{1/4}; \tilde{q}_{3/4}]$$

. Il contient la moitié la plus centrale des observations.

- La longueur de cet intervalle

$$\Delta_q = q_3 - q_1$$

est appelée **distance interquartile**. Cette quantité est un indicateur de dispersion.

Indicateurs de dispersion

Variance empirique

La variance empirique d'un échantillon est définie par :

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \overline{x}_n^2$$

Elle mesure l'écart quadratique moyen de l'échantillon à sa moyenne.

Ecart-type

L'écart-type est défini par :

$$s = \sqrt{s^2}$$

Contrairement à la variance empirique, il est exprimé dans la même unité de mesure que le caractère X.

Indicateurs de dispersion

Variance empirique corrigée

La variance empirique corrigée est définie par :

$$s^*2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2 = \frac{1}{n-1}(\sum_{i=1}^{n} x_i^2 - n\overline{x}_n^2)$$

Elle possède de meilleures propriétés que la variance empirique (voir chapitre *Estimation*)

Ecart-type corrigé

L'écart-type corrigé est défini par :

$$s^* = \sqrt{s^*2}$$

Contrairement à la variance empirique, il est exprimé dans la même unité de mesure que le caractère X.

Résumés numériques I

Résumés numériques II

```
vol.cm3 <- vignes$volume.baie..cm3..2008; vol.cm3 <- vol.cm3[!is.na(vol.cm3)]
mean(vol.cm3)

## [1] 10.46512

median(vol.cm3)

## [1] 8.91

var(vol.cm3) ## version corrigée !

## [1] 28.39179

sum((vol.cm3 - mean(vol.cm3))^2)/length(vol.cm3)

## [1] 28.25053

sum((vol.cm3 - mean(vol.cm3))^2)/(length(vol.cm3)-1)

## [1] 28.39179
```

Fonction de répartition empirique I

Définition

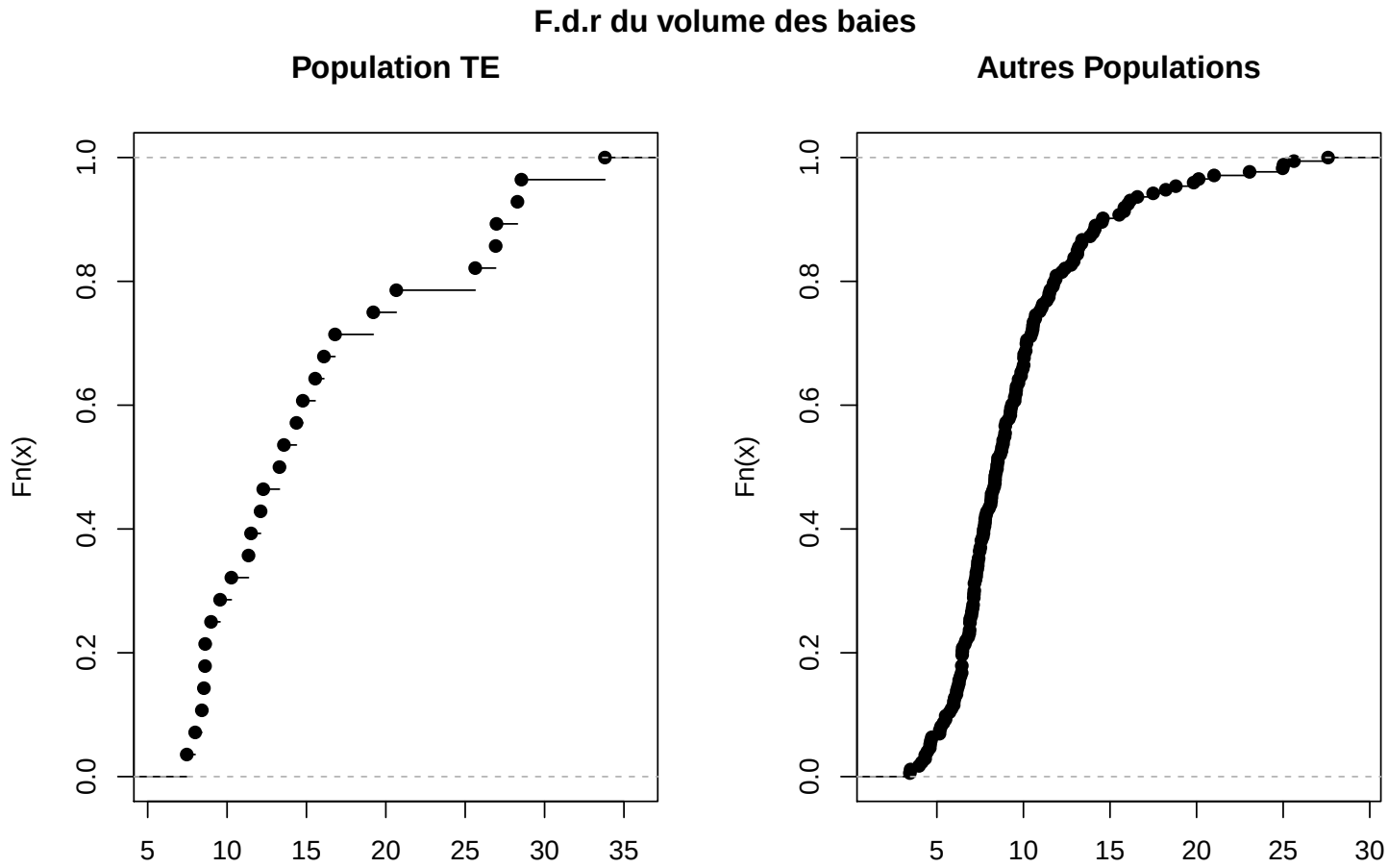
La version empirique de la fonction de répartition $F(x) = \mathbb{P}(X \leq x)$ s'écrit

$$\hat{F} : \mathbb{R} \mapsto [0,1], \ x \mapsto \frac{1}{n}\text{card}\{i : x_i \leq x\}$$

$\rightsquigarrow$ le graphe de la fonction de répartition est une fonction en escalier appelé **diagramme cumulatif**

```
par(mfrow=c(1,2))
plot(ecdf(vol.cm3[vignes$Population =="TE"]), main="Population TE", xlab="")
plot(ecdf(vol.cm3[vignes$Population !="TE"]), main="Autres Populations", xlab="")
title(outer=TRUE, main="\nF.d.r du volume des baies")
```

Fonction de répartition empirique II



Histogramme et estimateur à noyau I

Définition

Ce sont des estimateurs de la fonction de densité de x. On pose

$$\sum_i h_i 1_{[a_i, a_{i+1}[}(x) \text{ pour } \quad a_1 < ... < a_{k+1}.$$

On a $\sum_i h_i(a_{i+1} - a_i) = 1$ et $h_i(a_{i+1} - a_i) = \hat{\mathbb{P}}(X \in [a_i, a_{i+1}[)$.

Réalisation

1. Découpage en intervalles $[a_i, a_{i+1})$
2. Calcul de la fréquence f_i et de la hauteur h_i
3. Aire du rectangle proportionnel à la fréquence

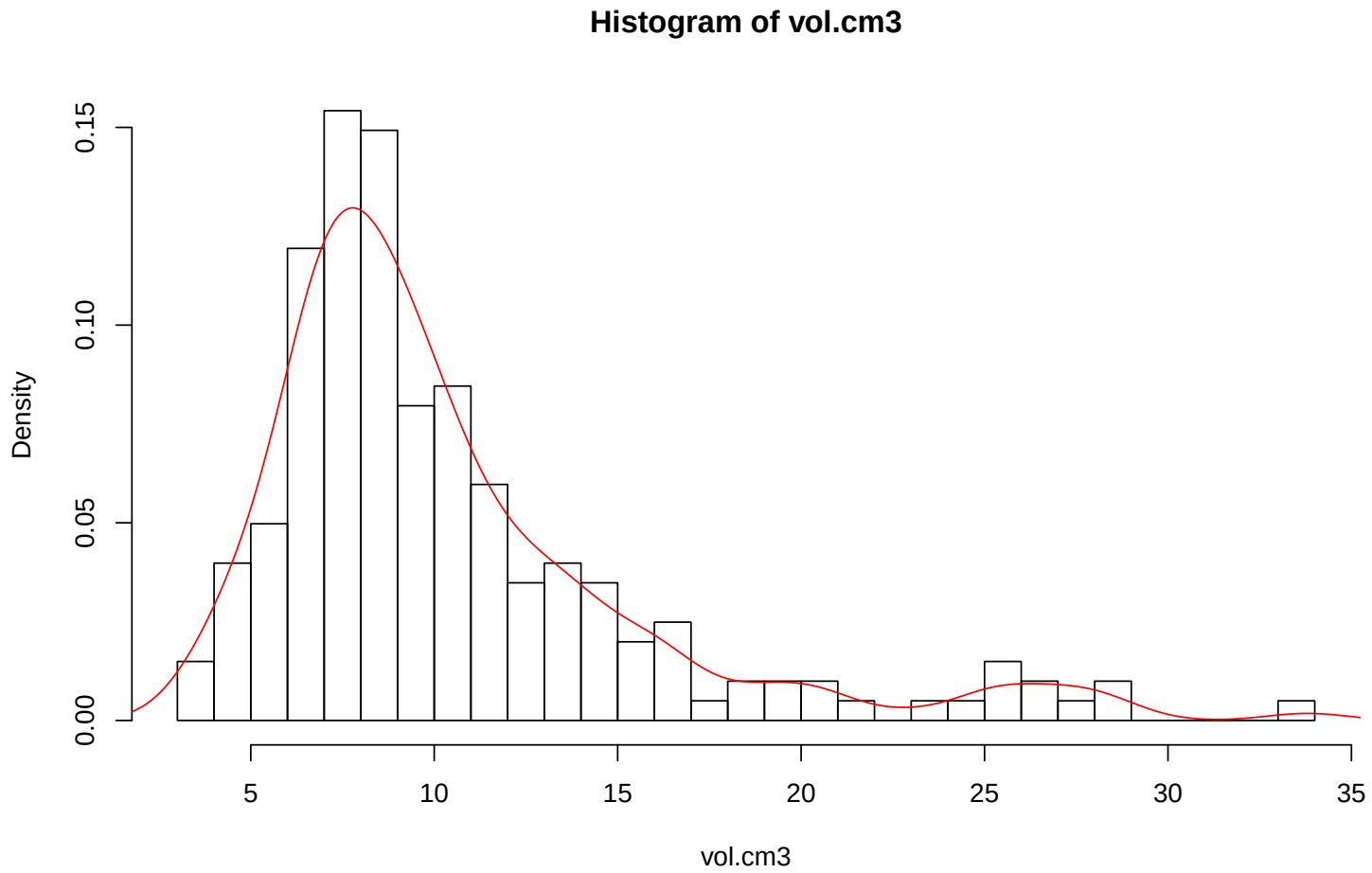
Histogramme et estimateur à noyau II

Remarques

- Attention : hauteur proportionnelle à la fréquence si et seulement si les intervalles ont tous la même largeur
- Nombre d'intervalles : Important, mais réglage difficile...

```
hist(vol.cm3,nclass=25,prob=TRUE)
lines(density(vol.cm3), col="red")
```

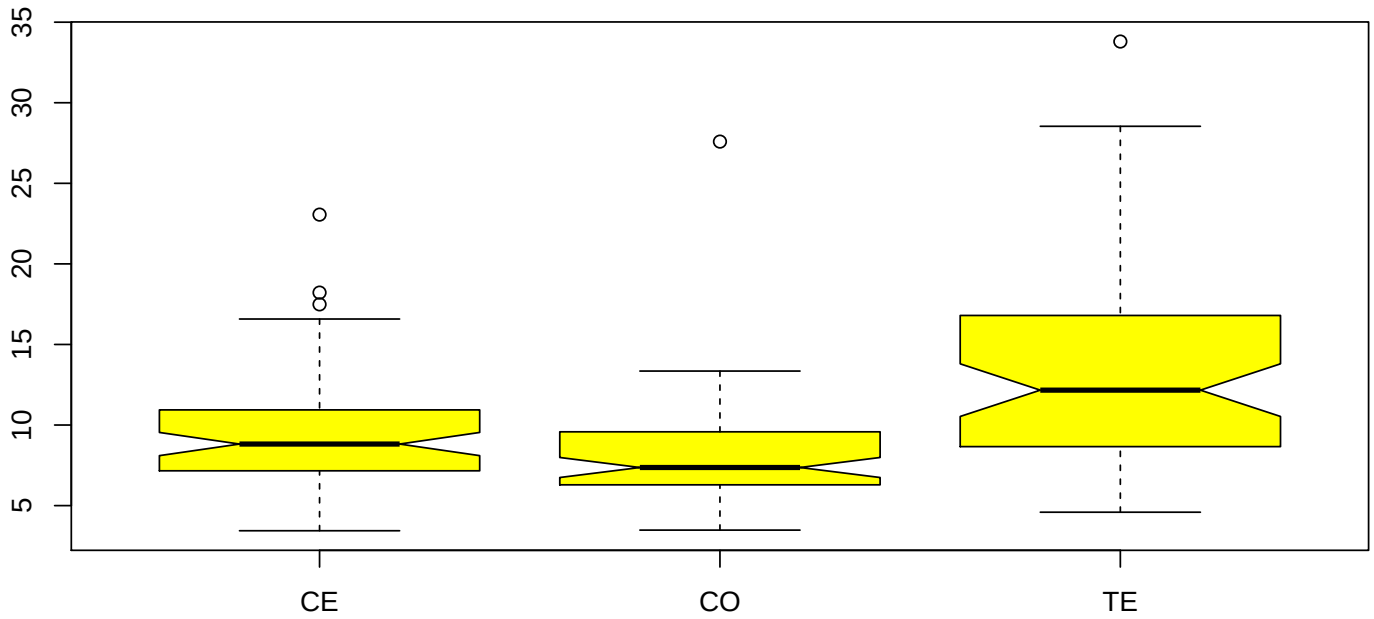
Histogramme et estimateur à noyau III



Représentation conditionnellement à un facteur

Les boîtes à moustaches se prettent bien à cet exercice

```
pop <- vignes$Population[!is.na(vignes$volume.baie..cm3..2008)]
boxplot(vol.cm3~pop,col="yellow",notch=T)
```



Graphe conditionné par une variable

```
pepin <- vignes$nbre.pepin.baie.2008[!is.na(vignes$volume.baie..cm3..2008)]
coplot(vol.cm3 ~ pepin | pop, show.given=FALSE)
```

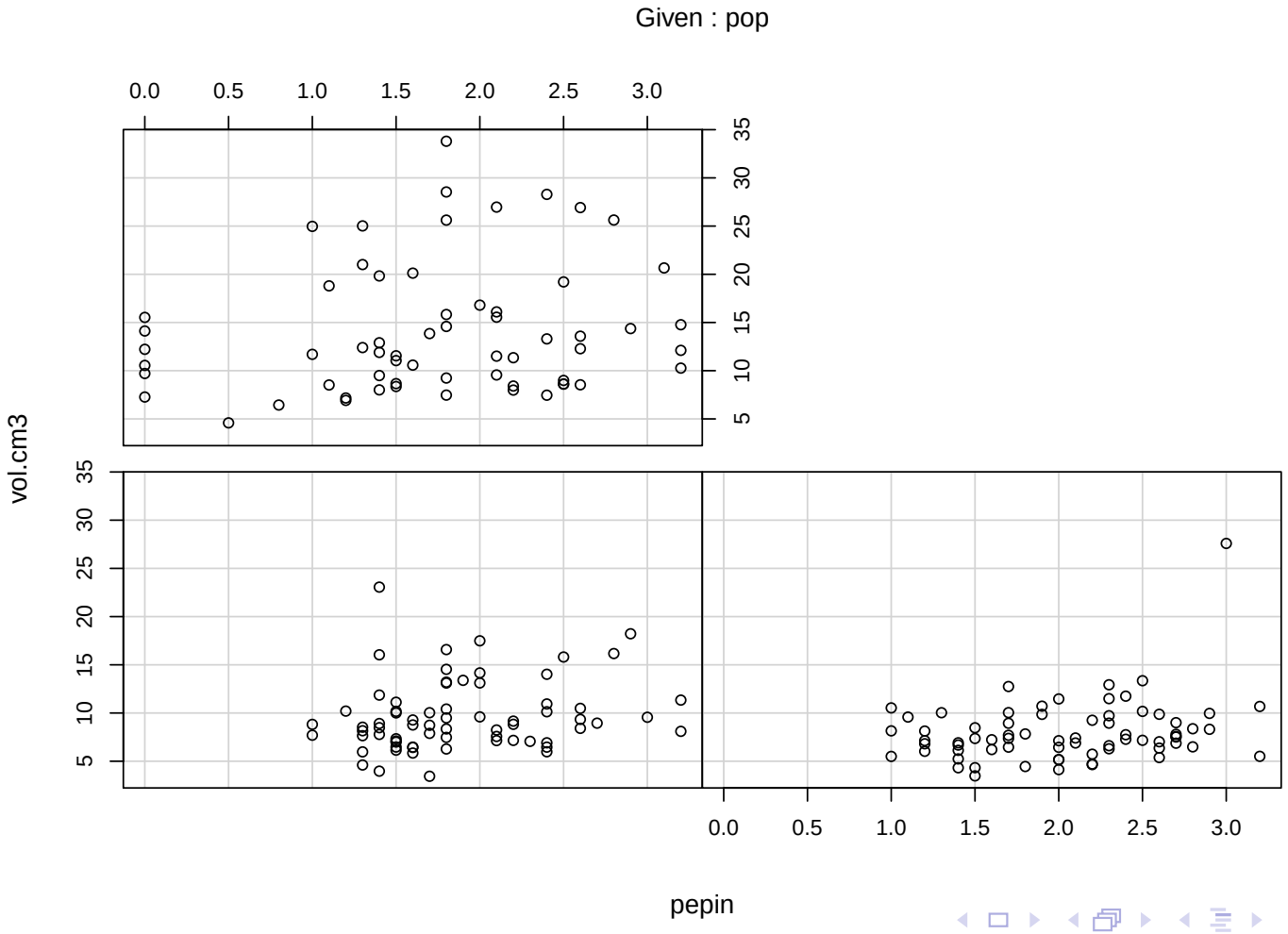


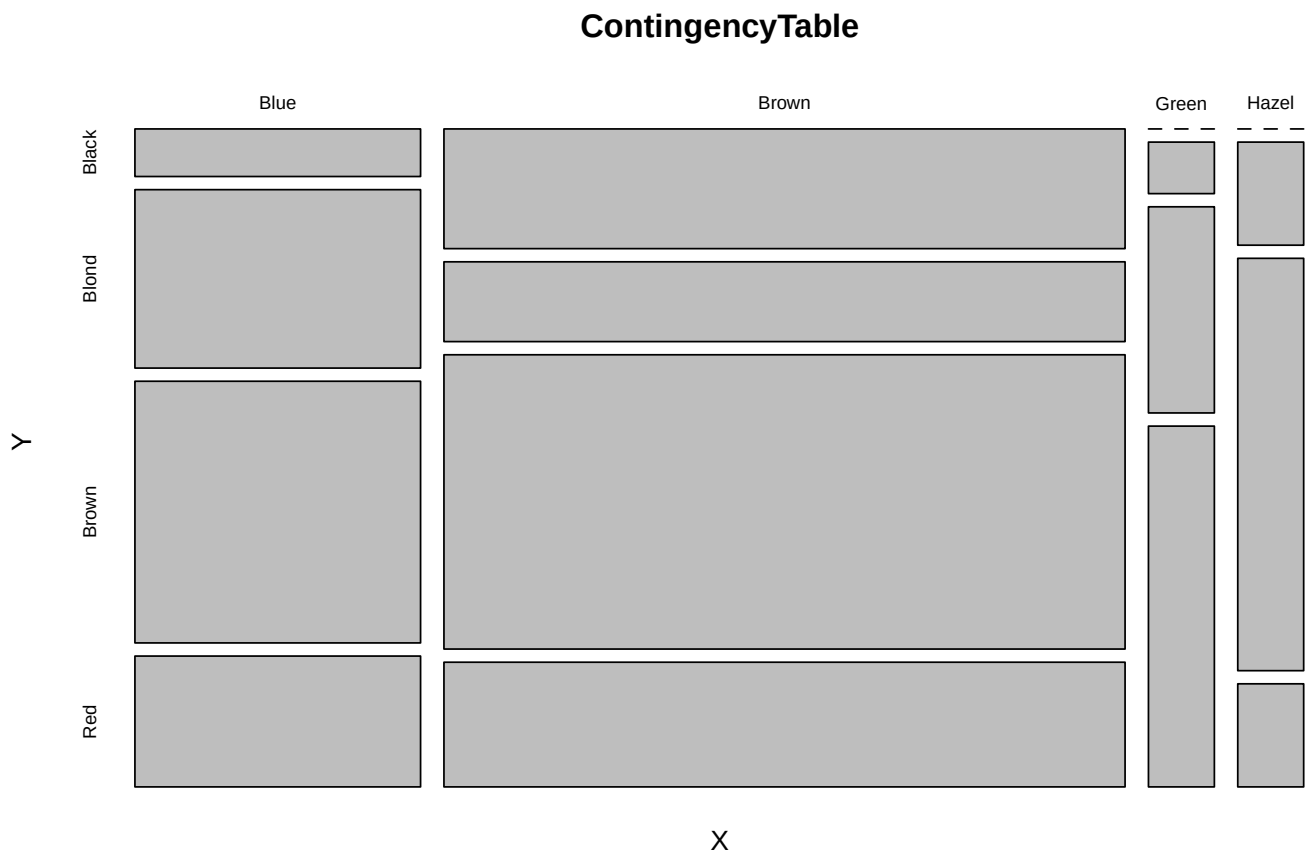
Diagramme mosaïque I

Représenter un tableau de contingence avec des informations sur ses marges

- chaque colonne j possède
  une largeur proportionnelle à sa marge $n_{\bullet j}$
- chaque case ij dans une colonne j possède
  une hauteur proportionnelle à $\frac{n_{ij}}{n_{\bullet j}}$
- la surface de chaque case ij est donc proportionnelle à son effectif n_{ij}

```
plot(ContingencyTable)
```

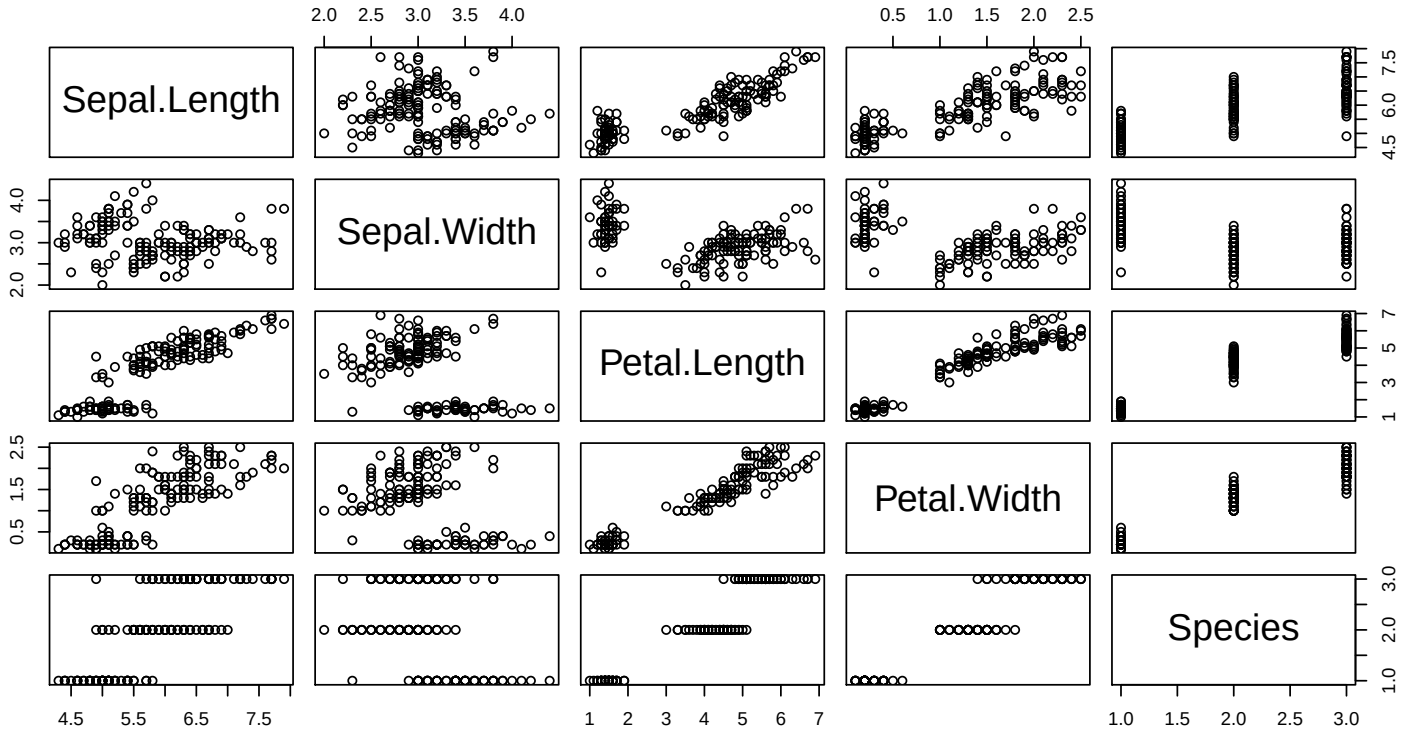
Diagramme mosaïque II



Graphes pair à pair

Représente toutes les paires de graphes naturels d'un tableau

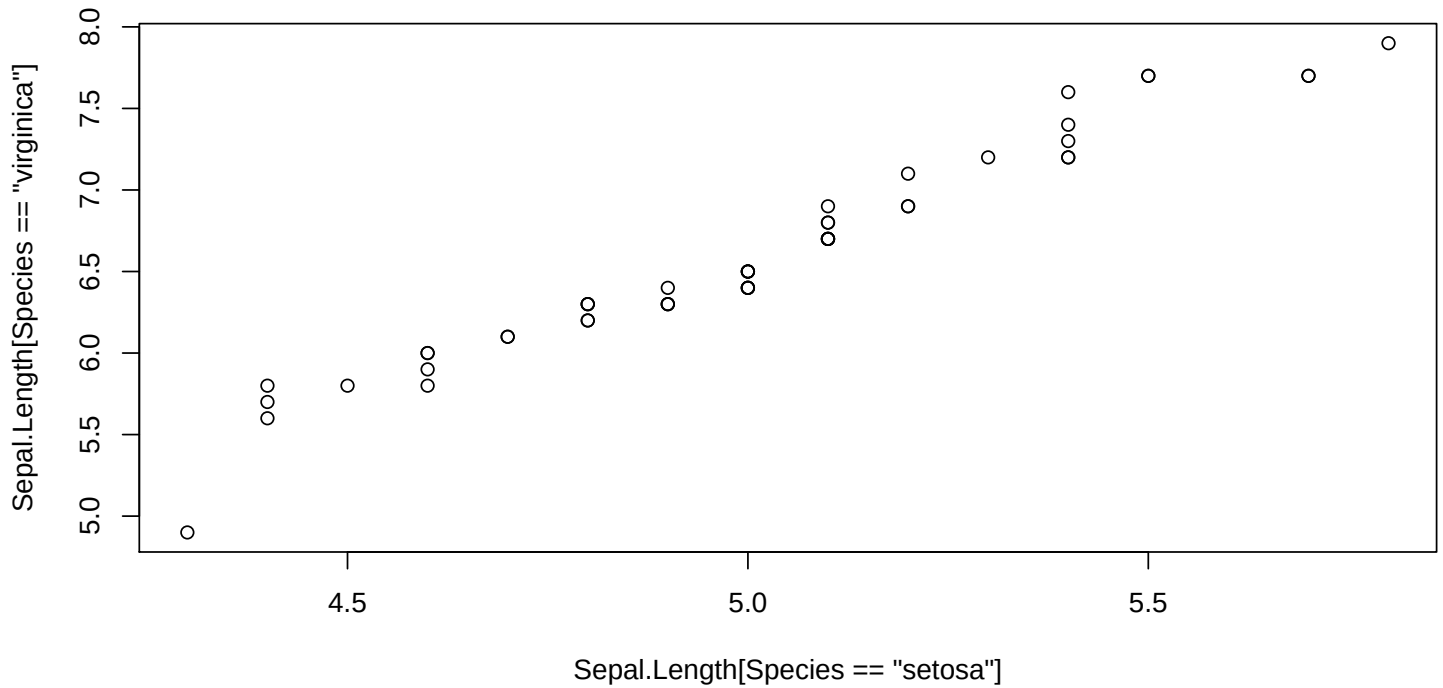
```
data(iris); pairs(iris)
```



Graphe quantile/quantile

Pour comparer visuellement les distributions de variables continues.

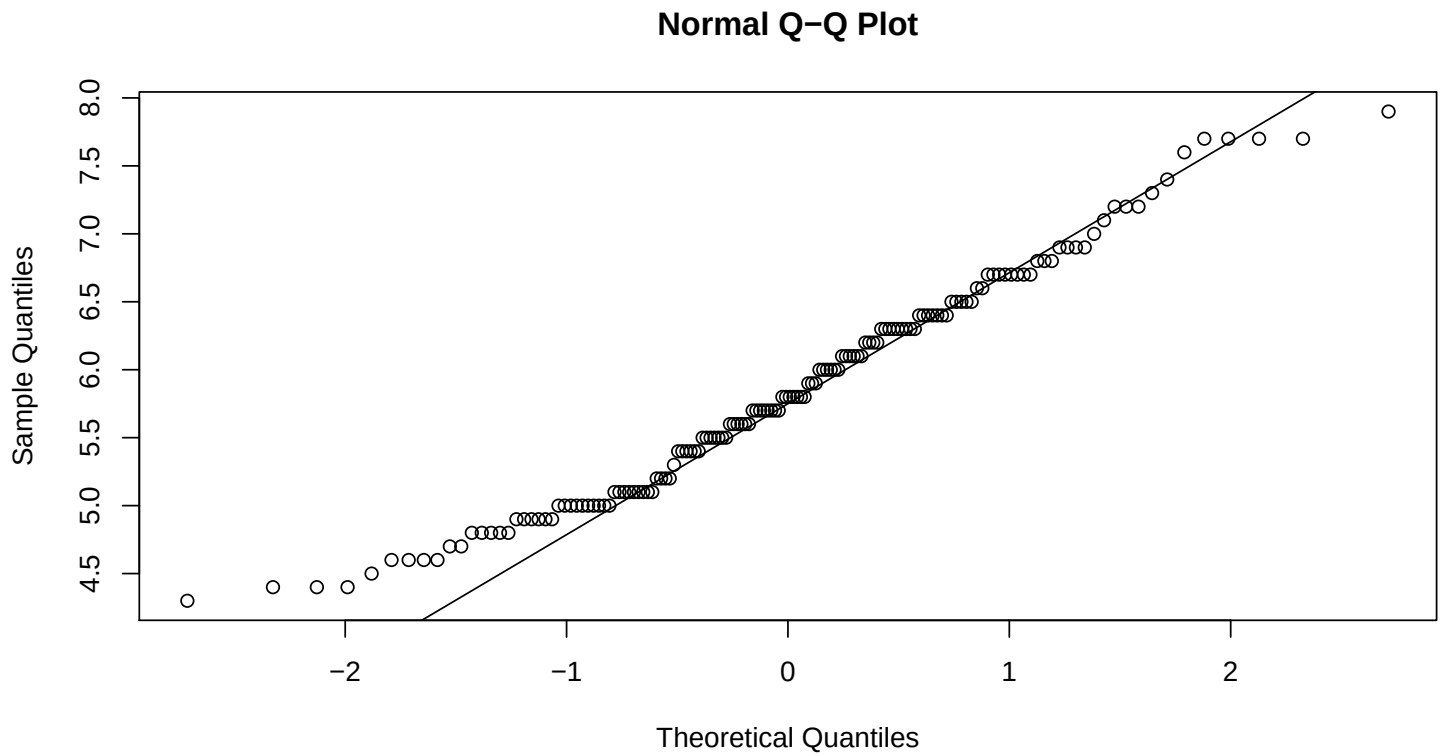
```
with(iris, qqplot(Sepal.Length[Species=="setosa"],Sepal.Length[Species=="virginica"]
```



Écart à la distribution normale

Une distribution est-elle normale ? `qqnorm/qqline` donne une indication.

```
with(iris, qqnorm(Sepal.Length))
with(iris, qqline(Sepal.Length))
```



Statistique du couple : covariance

Définition

Décrit l'écart conjoint de 2 variables à leurs espérances respectives

$$\text{cov}(X, Y) = \mathbb{E}\left[(X - \mathbb{E}X)(Y - \mathbb{E}Y)\right] = \mathbb{E}(XY) - EX\mathbb{E}Y$$

```
cov.mat <- cov(iris[,-5])
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	0.6856935	-0.0424340	1.2743154	0.5162707
Sepal.Width	-0.0424340	0.1899794	-0.3296564	-0.1216394
Petal.Length	1.2743154	-0.3296564	3.1162779	1.2956094
Petal.Width	0.5162707	-0.1216394	1.2956094	0.5810063

Statistique du couple : corrélation

Définition

Il s'agit de la version normalisé de la covariance

$$\mathrm{cor}(X, Y) = \frac{\mathrm{cov}(X, Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}}$$

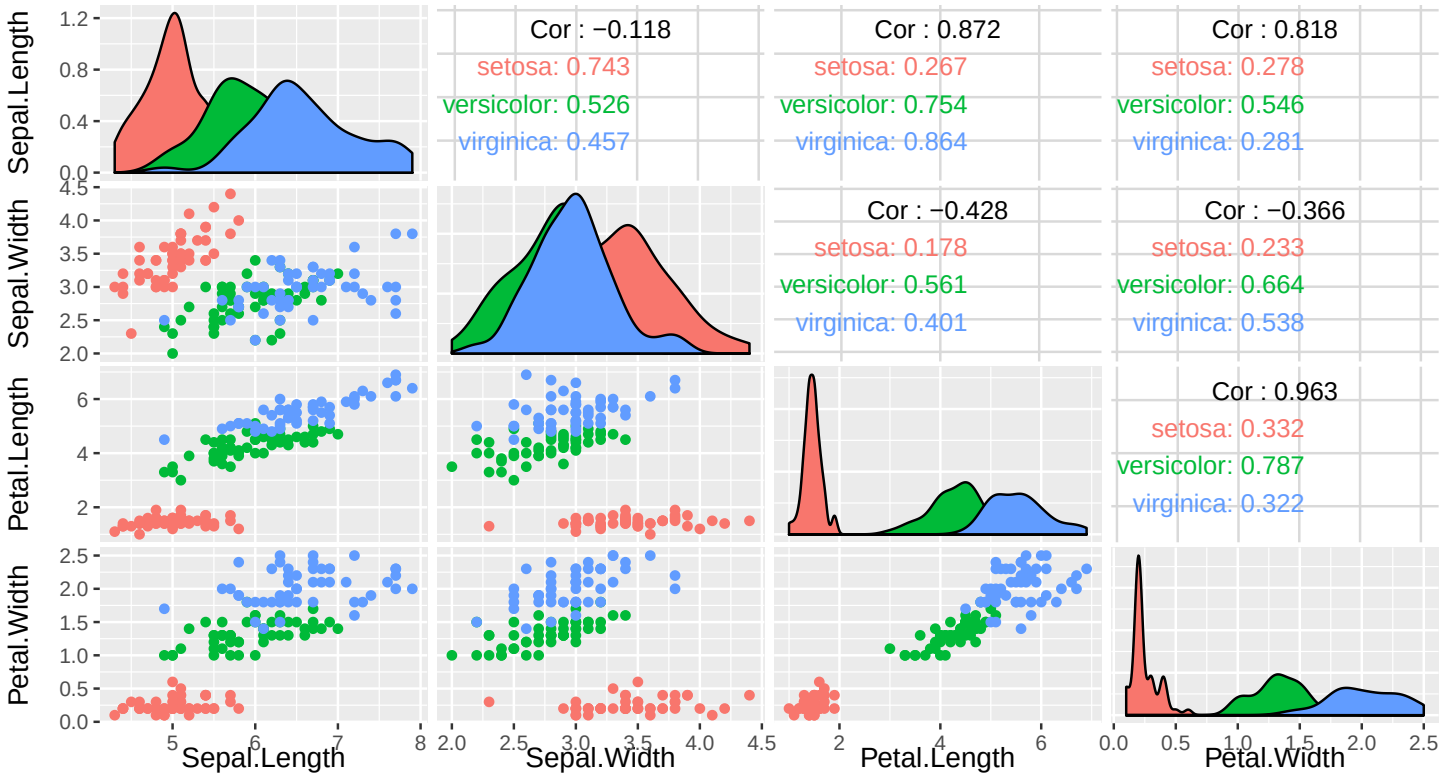
```
data(iris)
cor.mat <- cor(iris[,-5])
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.0000000	-0.1175698	0.8717538	0.8179411
Sepal.Width	-0.1175698	1.0000000	-0.4284401	-0.3661259
Petal.Length	0.8717538	-0.4284401	1.0000000	0.9628654
Petal.Width	0.8179411	-0.3661259	0.9628654	1.0000000

Graphe pair à pair et corrélation

dit à quel point deux variables s'expliquent linéairement l'une l'autre.

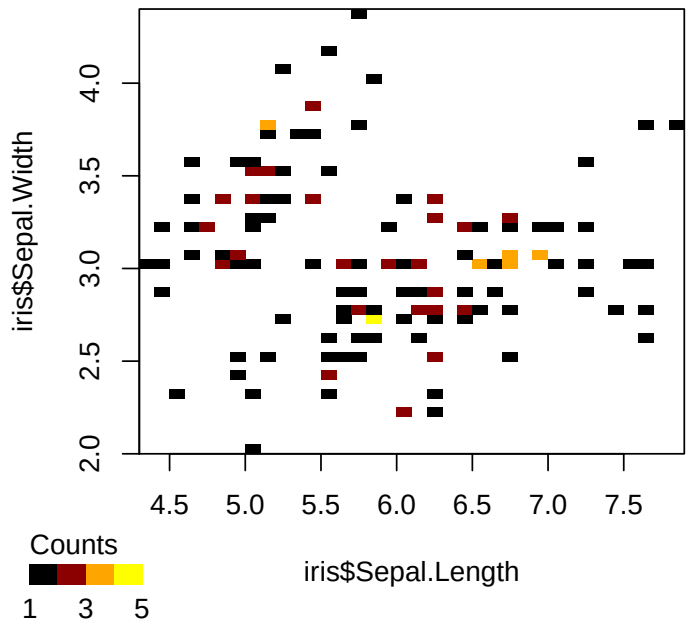
```
library(GGally)
ggpairs(iris, columns = 1:4, mapping = ggplot2::aes(colour = Species))
```

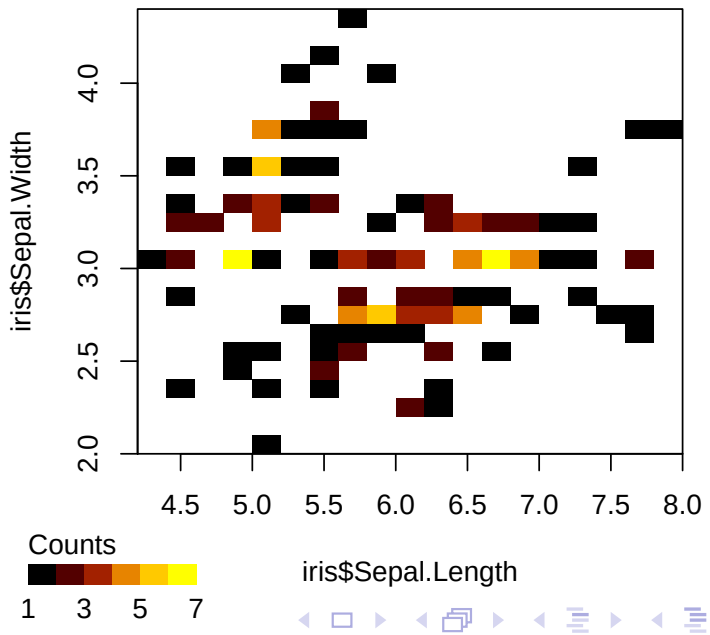


Histogramme bidimensionnelle

Regroupe les points d'un graphe de dispersion par pavé bidimensionnels.

```
library(squash);
par(mfrow=c(1,2))
hist2(iris$Sepal.Length,iris$Sepal.Width)
hist2(iris$Sepal.Length,iris$Sepal.Width, nx=20)
```

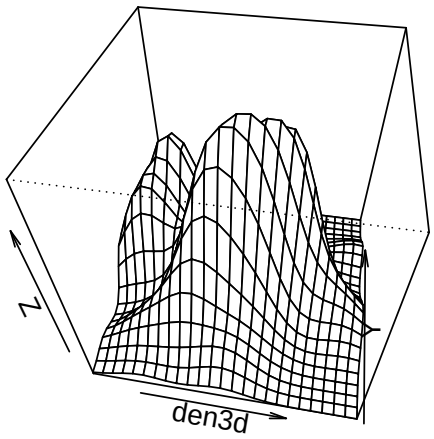


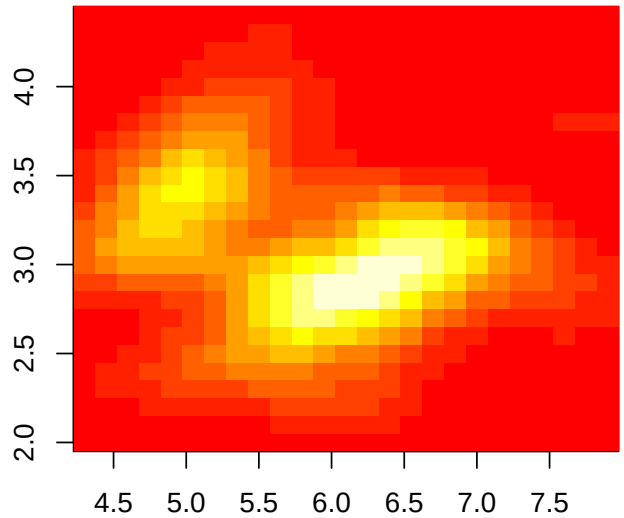


Histogramme bidimensionnelle lisse

Estime la densité associé à deux variables continues. Représentation 3D ou image possible

```
library(MASS)
den3d <- kde2d(iris$Sepal.Length, iris$Sepal.Width)
par(mfrow=c(1,2))
persp(den3d, box=TRUE,theta = 10, phi = 45)
image(den3d)
```





Plan

Entrées, sorties et statistique descriptive

Analyse en composantes principales

Idée

L'idée centrale de l'analyse en composante principales est de réduire la dimensionalité d'un jeu de données consitué d'un grand nombre de variables liées, tout en gardant autant d'information que possible.

Intuition

Définir de nouvelles variables

- les plus informatives (variance élevées) possibles, non correlées
- combinaison linéaires des variables originales

Objectifs

1. une analyse de ressemblances entre les individus : par exemple peut-on mettre en évidence une typologie des individus ?
2. une analyse des liaisons entre les variables : par exemple existe-t-il des groupes de variables corrélées entre elles ?

L'ACP en un transparent

Notation

- $\mathbf{x}$, un vecteur à p variables décrivant un individu
- $\mathbf{u}_k$, un vecteur de p coefficient
- $\mathbf{u}_k^t\mathbf{x} = \sum_j u_{kj}x_j = c_k$

Procédure

- Trouver une fonction linéaire de $\mathbf{x}$, $\mathbf{u}_1^t\mathbf{x}$ de variance maximum
- Trouver une autre combinaison linéaire de $\mathbf{x}$, $\mathbf{u}_2^t\mathbf{x}$ non corrélée avec $\mathbf{u}_1^t\mathbf{x}$ de variance maximum
- Itérer ....

Solution du problème de l'ACP

Notations et hypothèses

- Σ est la matrice de variance covariance du vecteur aléatoire $\mathbf{x}$
- Lorsque Σ est inconnue, elle est remplacée par son estimateur $\mathbf{S}$

Solution en bref

- $\forall k \ \mathbf{u_k}^t \ \mathbf{u_k} = 1$ (Tous les axes principaux sont de norme unité)
- Les $\mathbf{u_k}$ sont les vecteur propres de Σ associé aux valeurs propres λ_k
- L'ordre des $\mathbf{u_k}$ correspond à l'ordre inverse des valeurs propres λ_k
- $var(c_k) = \lambda_k$ avec $\mathbf{u}_k^t \mathbf{x} = \sum_j u_{kj} x_j = c_k$

En pratique

Représentations

- individus
- variables
- individus supplémentaires
- variables supplémentaires

Indicateurs de qualité

- Représentation globale
- Contribution relative d'un axe à un individu
- Contribution relative d'un individu à un axe

En R

Fonctions

- `prcomp()`
- `princomp()`

Modules

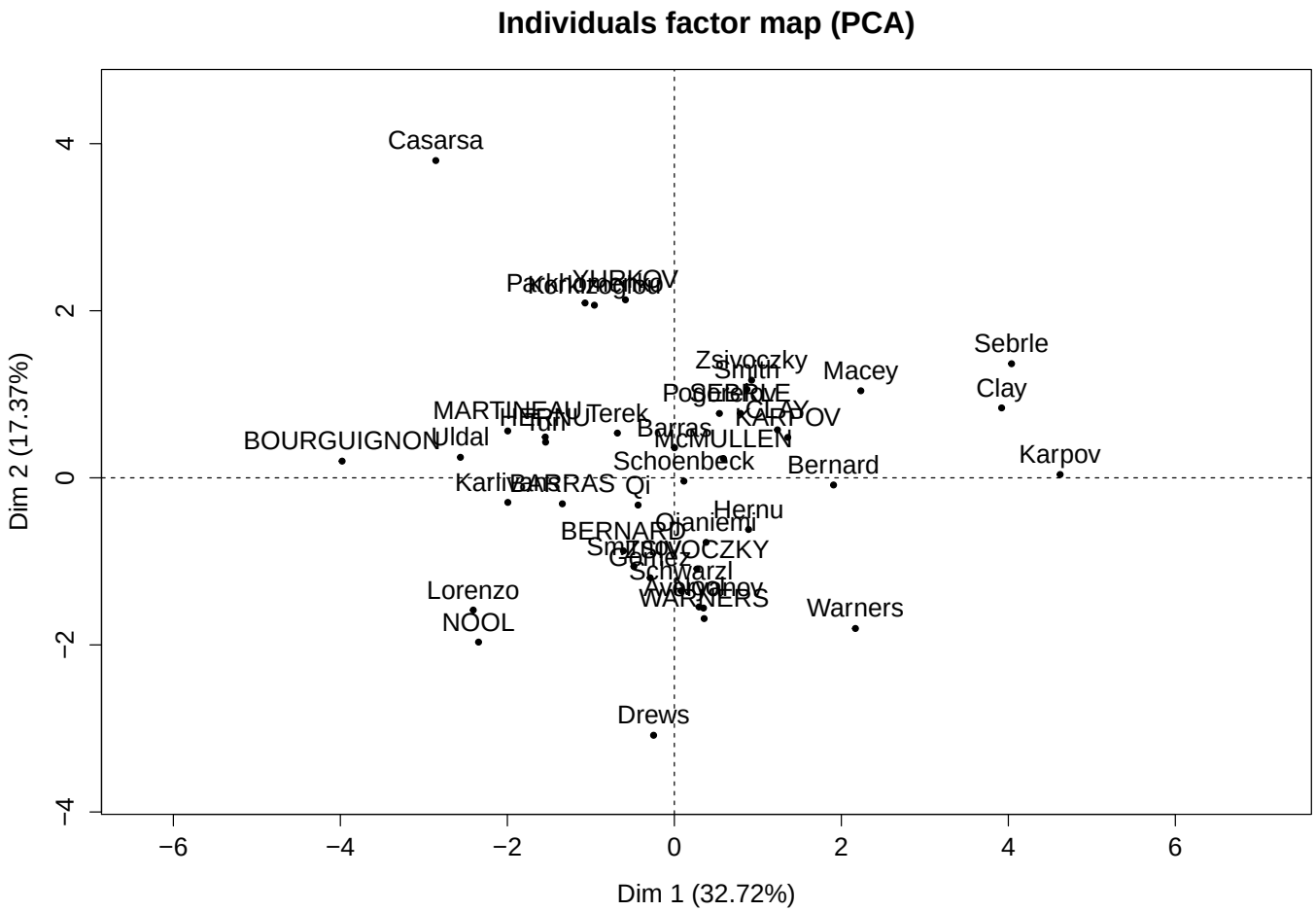
- `FactoMiner`
- `ade4`

Décathlon

	100m	Long.jump	Shot.put	High.jump	400m	110m.hurdle	Discus	Pole.vault	Javeline	1500m
SEBRLE	11.04	7.58	14.83	2.07	49.81	14.69	43.75	5.02	63.19	291.70
CLAY	10.76	7.40	14.26	1.86	49.37	14.05	50.72	4.92	60.15	301.50
KARPOV	11.02	7.30	14.77	2.04	48.37	14.09	48.95	4.92	50.31	300.20
BERNARD	11.02	7.23	14.25	1.92	48.93	14.99	40.87	5.32	62.77	280.10
YURKOV	11.34	7.09	15.19	2.10	50.42	15.31	46.26	4.72	63.44	276.40
WARNERS	11.11	7.60	14.31	1.98	48.68	14.23	41.10	4.92	51.77	278.10
ZSIVOCZKY	11.13	7.30	13.48	2.01	48.62	14.17	45.67	4.42	55.37	268.00
McMULLEN	10.83	7.31	13.76	2.13	49.91	14.38	44.41	4.42	56.37	285.10
MARTINEAU	11.64	6.81	14.57	1.95	50.14	14.93	47.60	4.92	52.33	262.10
HERNU	11.37	7.56	14.41	1.86	51.10	15.06	44.99	4.82	57.19	285.10
BARRAS	11.33	6.97	14.09	1.95	49.48	14.48	42.10	4.72	55.40	282.00
NOOL	11.33	7.27	12.68	1.98	49.20	15.29	37.92	4.62	57.44	266.60
BOURGUIGNON	11.36	6.80	13.46	1.86	51.16	15.67	40.49	5.02	54.68	291.70
Sebrle	10.85	7.84	16.36	2.12	48.36	14.05	48.72	5.00	70.52	280.01
Clay	10.44	7.96	15.23	2.06	49.19	14.13	50.11	4.90	69.71	282.00
Karpov	10.50	7.81	15.93	2.09	46.81	13.97	51.65	4.60	55.54	278.11
Macey	10.89	7.47	15.73	2.15	48.97	14.56	48.34	4.40	58.46	265.42
Warners	10.62	7.74	14.48	1.97	47.97	14.01	43.73	4.90	55.39	278.05
Zsivoczky	10.91	7.14	15.31	2.12	49.40	14.95	45.62	4.70	63.45	269.54
Hernu	10.97	7.19	14.65	2.03	48.73	14.25	44.72	4.80	57.76	264.35
Nool	10.80	7.53	14.26	1.88	48.81	14.80	42.05	5.40	61.33	276.33
Bernard	10.69	7.48	14.80	2.12	49.13	14.17	44.75	4.40	55.27	276.31
Schwarzl	10.98	7.49	14.01	1.94	49.76	14.25	42.43	5.10	56.32	273.56
Pogorelov	10.95	7.31	15.10	2.06	50.79	14.21	44.60	5.00	53.45	287.63
Schoenbeck	10.90	7.30	14.77	1.88	50.30	14.34	44.41	5.00	60.89	278.82
Barras	11.14	6.99	14.91	1.94	49.41	14.37	44.83	4.60	64.55	267.09
Smith	10.85	6.81	15.24	1.91	49.27	14.01	49.02	4.20	61.52	272.74
Averyanov	10.55	7.34	14.44	1.94	49.72	14.39	39.88	4.80	54.51	271.02
Ojaniemi	10.68	7.50	14.97	1.94	49.12	15.01	40.35	4.60	59.26	275.71
Smirnov	10.89	7.07	13.88	1.94	49.11	14.77	42.47	4.70	60.88	263.31

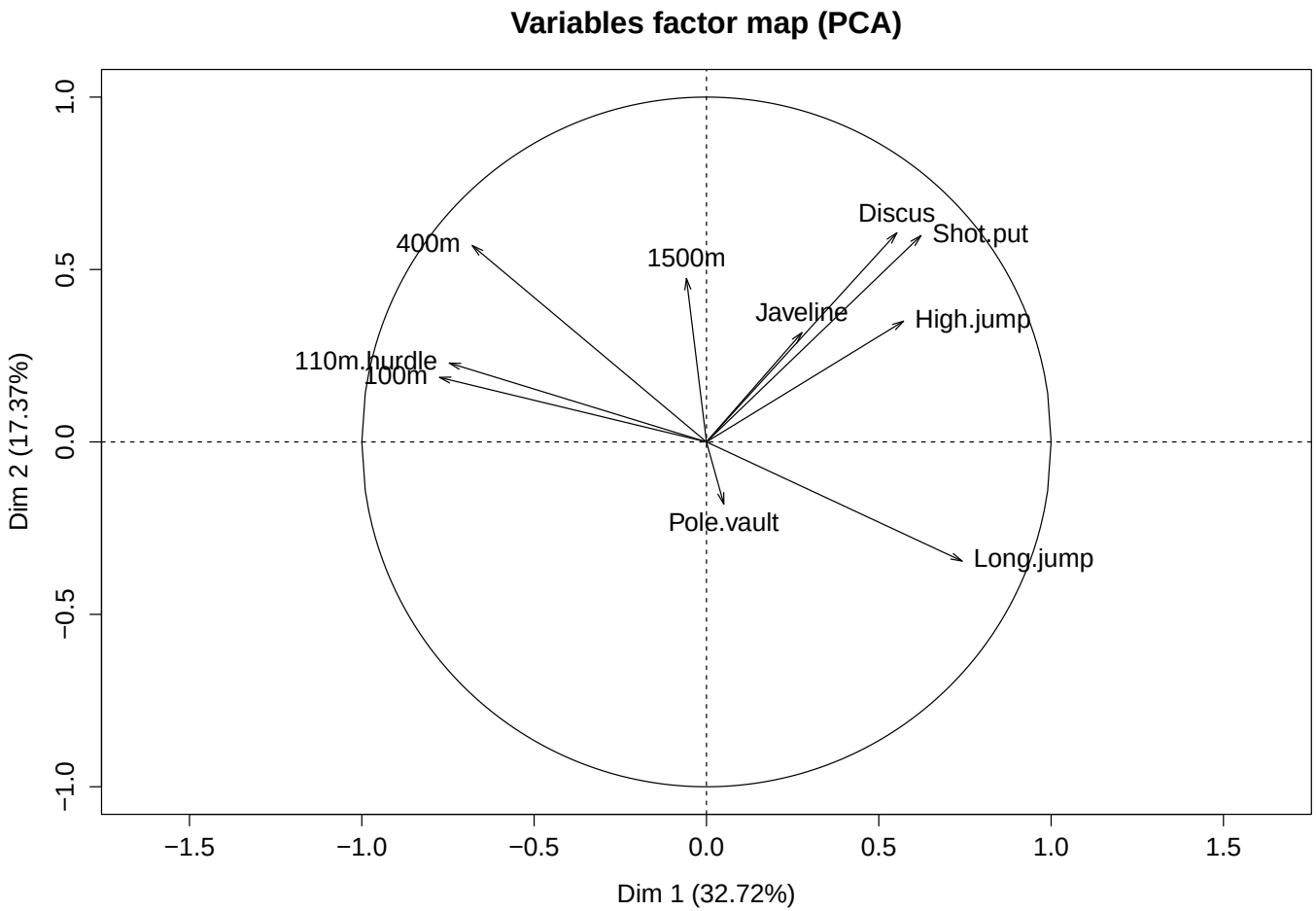
Représentation des individus

Nage des individus



Réprésentation des variables

Cercle des corrélations



Quatrième partie IV

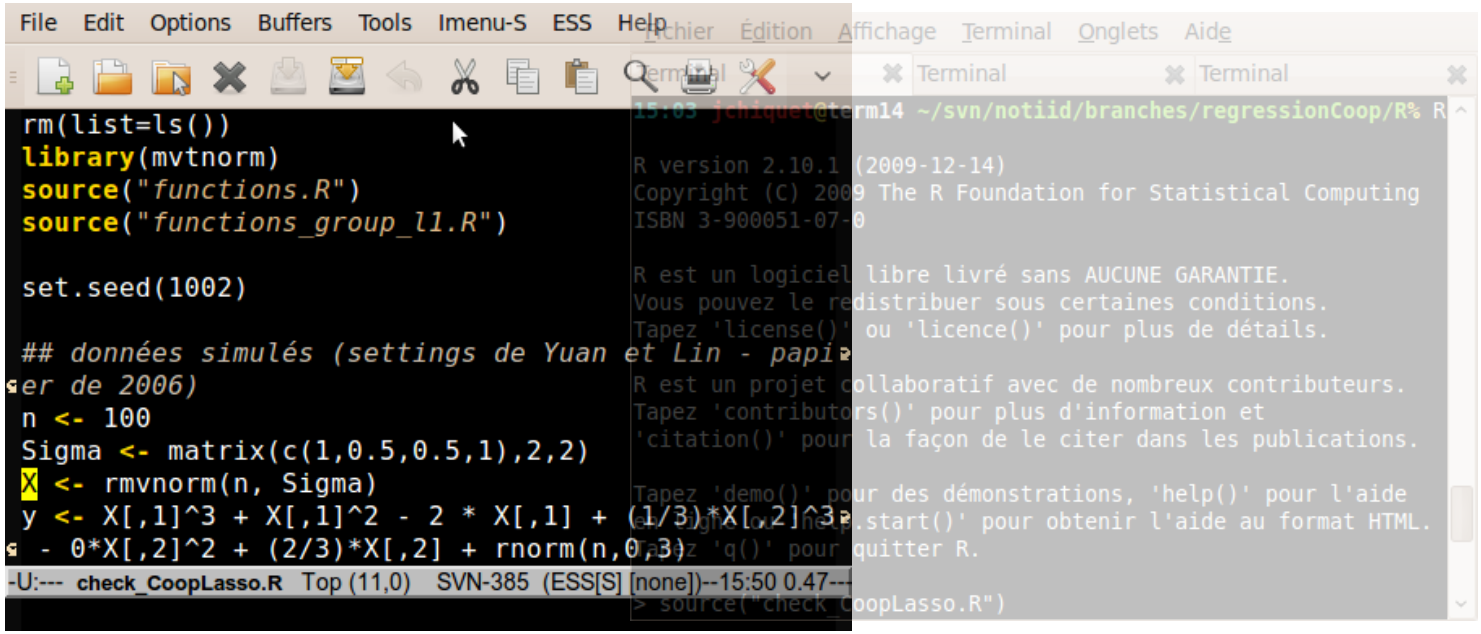
Programmation

Plan

Programmation

Environnement de développement

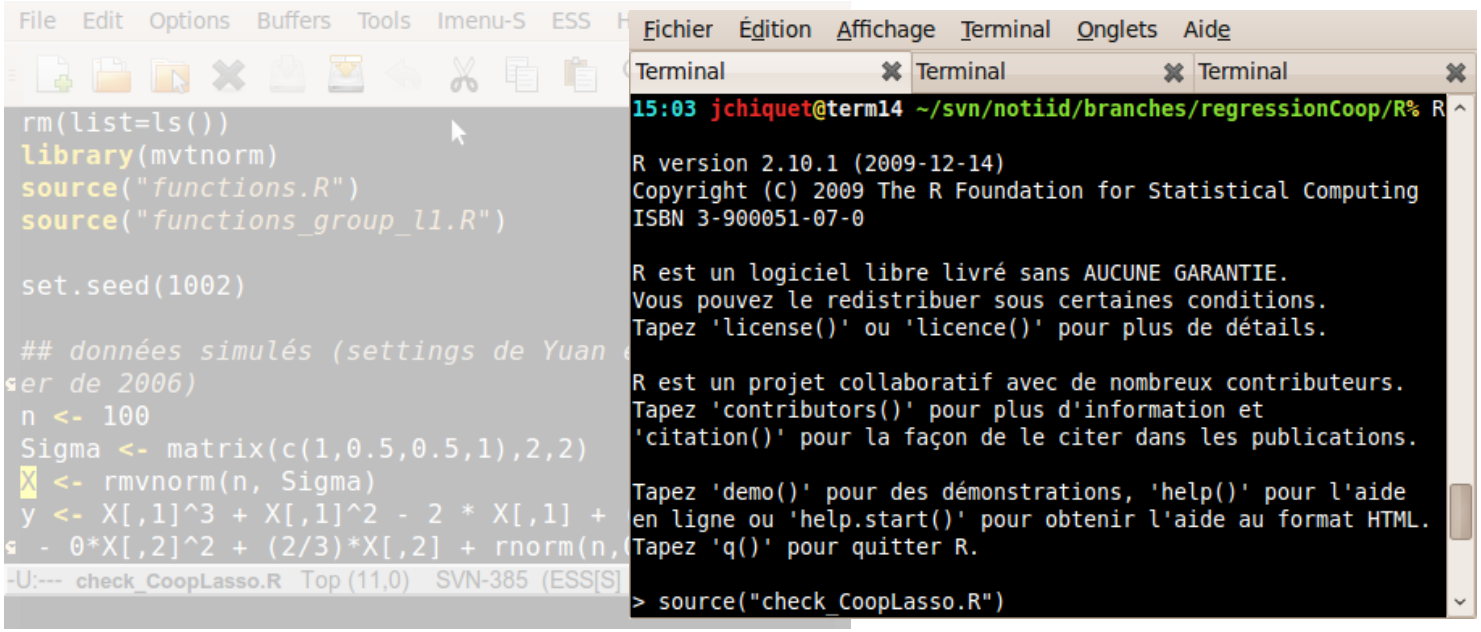
Première solution



1. un éditeur de texte (vos fonctions / scripts)
2. un terminal avec R (tester, « sourcer »)

Environnement de développement

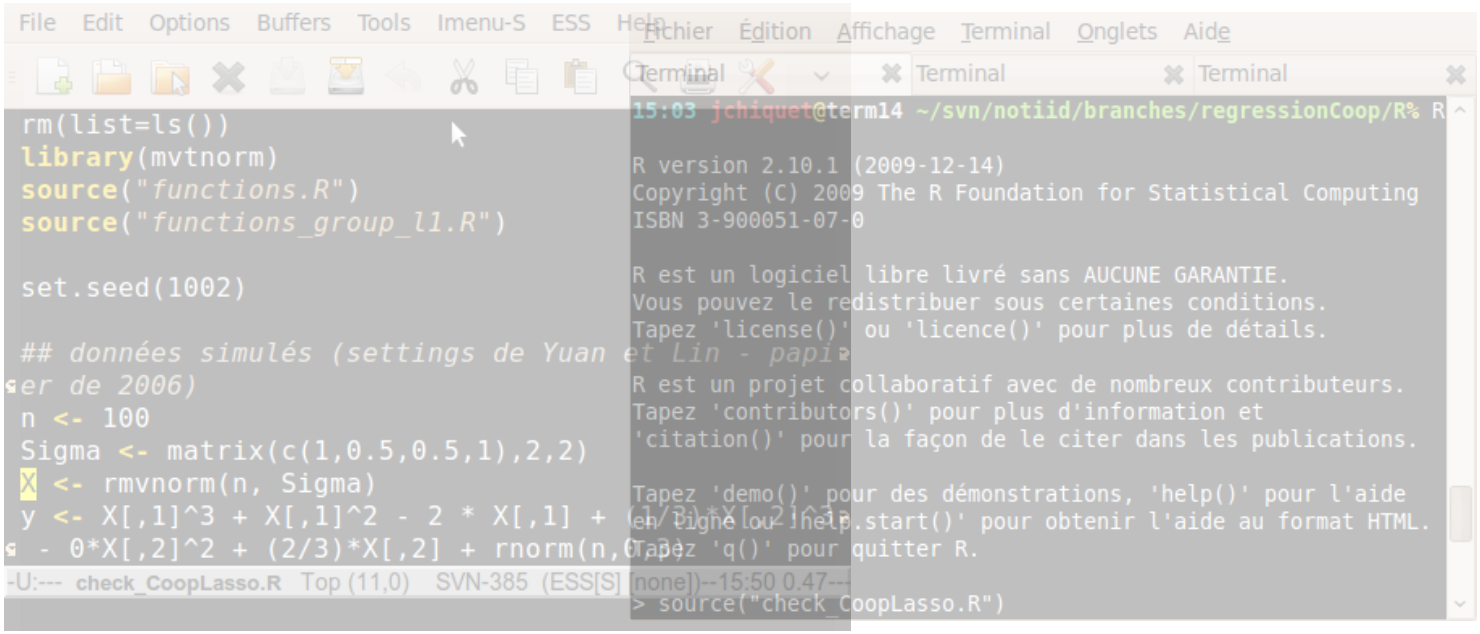
Première solution



1. un éditeur de texte (vos fonctions / scripts)
2. un terminal avec R (tester, « sourcer »)

Environnement de développement

Première solution



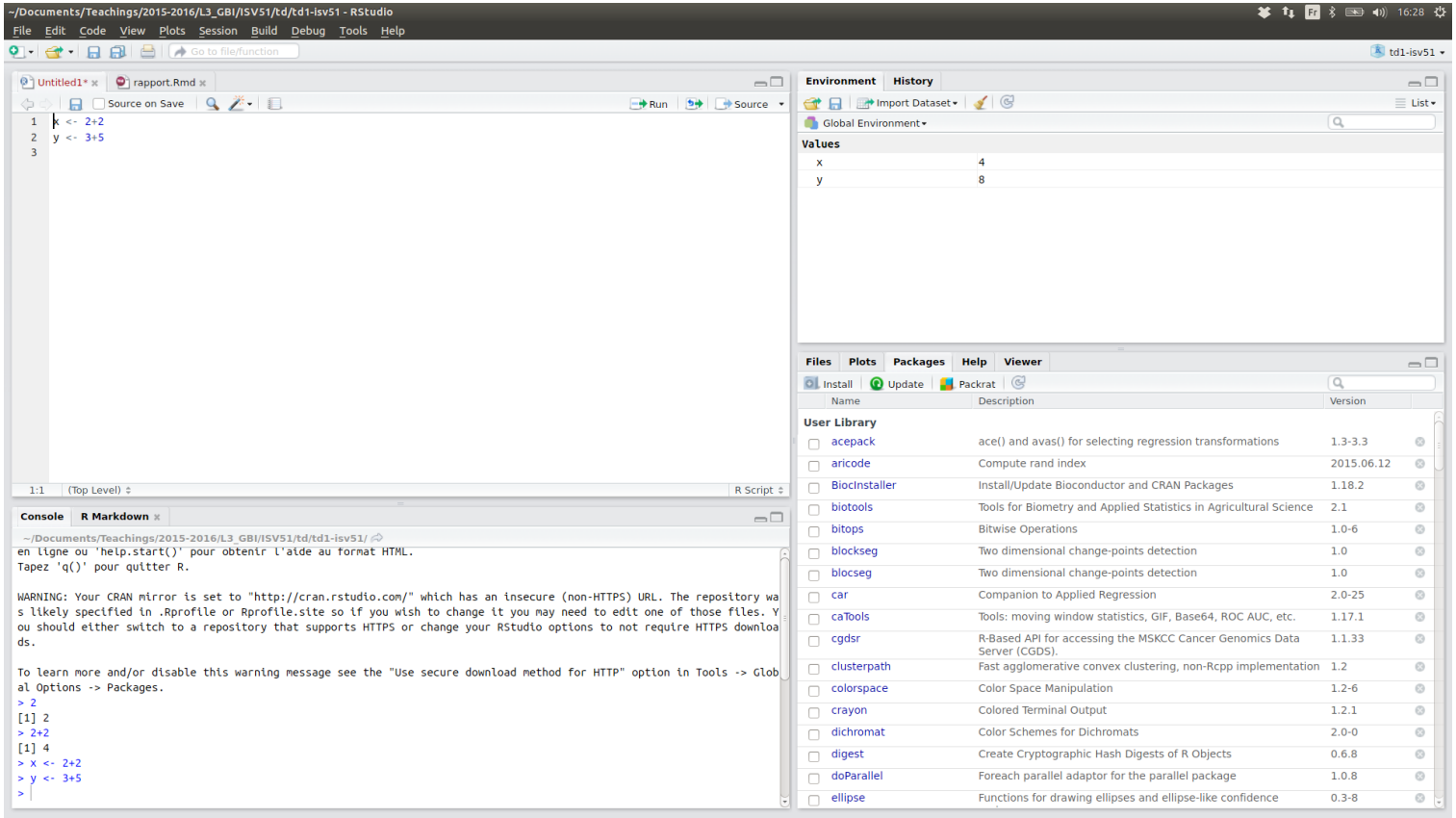
1. un éditeur de texte (vos fonctions / scripts)
2. un terminal avec R (tester, « sourcer »)

« Sourcer »

`source("un_script.R")` : exécute la série de commandes de `mon_script.R`

Environnement de développement

R-studio, environnement de travail intégré



Plan

Programmation

Regrouper les expressions

Syntaxe

```
{expr_1; expr_2; ...; expr_n }
```

ou

```
{
  expr_1
   ...
  expr_n
}
```

Remarques sur les groupes

- La dernière valeur du groupe est retournée ;
- un groupe d'expressions peut être passé à une fonction, réutilisé dans une expression plus grande, etc.

Regrouper les expressions

Examples

```
expr1 <- {a<-3; b<-5; a*b}
expr1

## [1] 15
```

```
tmp <- 12
expr2 <- {a<-3; b<-5; tmp<-a*b+tmp}
expr2

## [1] 27

tmp

## [1] 27
```

Exécution conditionnelle : `if`, `if/else`, `ifelse`

Syntaxe

```
if (condition) {
  expr_1
} else {
  expr_2
}
```

ou (forme vectorielle)

```
ifelse(condition, a, b)
```

Remarques

- `condition` est une valeur logique : penser à `&`, `|`, `!`, ...
- le `else` est optionnel,
- `elseif` permet d'imbriquer les conditionnements.

Exécution conditionnelle : if,if/else,ifelse

Exemples

```
partiel <- 11
DS <- 14
if (partiel > 6 & mean(DS,partiel) >10) {
    cat("\nreçu(e).")
} else {
    cat("\nrecalé(e).")
}

##
## reçu(e).
```

Fonctionnemt en vectoriel de ifelse

```
partiel <- c(11,5,6,12,9,8,14)
DS <- c(14,16,12,12,19,12,7)
ifelse(partiel > 6 & rowMeans(cbind(DS,partiel)) >10, "reçu", "recalé")

## [1] "reçu"   "recalé" "recalé" "reçu"   "reçu"   "recalé" "reçu"
```

Exécution conditionnelle : `switch`

Syntaxe

```
switch (expr,
 expr.1 = faire.1,
 expr.2 = faire.2,
  ...,
  faire.defaut
)
```

Remarques

- `expr` est une variable contenant une chaîne de caractère ou un entier.
- Si `expr` est un entier i, la i^{e} expression `faire.i` est évalué et renvoyée.
- Si `expr` contient une chaîne, l'expression `faire.i` telle que `expr == expr.1` est évaluée.

Exécution conditionnelle : switch

Exemples

Avec un entier

```
expr <- 2
switch(expr, cat("je vaux 1"), cat("je vaux 2"))

## je vaux 2

expr <- 3
switch(expr, cat("je vaux 1"), cat("je vaux 2"))
```

Avec une chaîne

```
stat <- "variance"
ma.fonction <- switch(stat,
                      "moyenne"  = mean,
                      "variance" = var, NULL)
ma.fonction(1:10)

## [1] 9.166667
```

Exécution répétée : boucle `for`

Syntaxe

```
for (var in set) {
  expr(var)
}
```

ou

```
for (var in set)
  expr(var)
```

à fuir pour éviter les effets de bords sournois !

Remarques sur la boucle `for`

- `var` est la variable incrémentée,
- `set` est un vecteur définissant les valeurs successives,
- *lente* comparée aux opérateurs matriciels.

Exécution répétée : boucle `for` I

Exemples

```
for (i in sample(1:5)) {
    cat(i)
}

## 35142
```

```
v <- numeric(7)
for (i in seq_along(v)) {
    v[i] <- i*3
}
v

## [1]  3  6  9 12 15 18 21
```

Exécution répétée : boucle `for` II

Exemples

```
data(iris)
cat("\nNoms des colonnes:")

##
## Noms des colonnes:

for (nom in colnames(iris)) {
    cat("",nom)
}

##  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
```

Exécution répétée : boucles `while` et `repeat`

Syntaxe

```
while (condition) {
  expr
}
```

ou

```
repeat {
  expr
}
```

Remarque

- Comme pour `for`, éviter les imbrications sources de lenteur.

Contrôle des boucles : `break`, `next`

Exemples d'utilisation

```
repeat {
  expr
  if (condition) {break}
}
```

ou

```
while (condition1){
  expr_1
  if (condition2) {next}
  expr_2
}
```

Remarque

- `break` est la seule manière d'interrompre une boucle `repeat`.

Exécution répétée : `while`

Exemples

Parcours des lignes du tableau `iris` tant qu'on a pas rencontré un individu ayant certaines caractéristiques.

```
data(iris)
i <- 0 ## numéro individu courant
cond <- FALSE
while (!cond) {
    i <- i + 1
    if (iris[i, ]$Sepal.Length > 6)
        cond <- TRUE
}
cat("\nL individu", i, "est le premier à avoir une longueur de sépale > 6.")

##
## L individu 51 est le premier à avoir une longueur de sépale > 6.
```

Plan

Programmation

Définir une fonction

Syntaxe

```
nom_de_la_fonction <- function(arg1,arg2, ...) {
  expression

  return(var)
}
```

Remarques

- `return` peut être omis (à éviter) : dans ce cas la dernière valeur calculée est renvoyée.
- peut être tapée directement dans l'interpréteur ou dans un fichier externe `functions.R`, chargé par `source`.

Un exemple simple

Moyenne empirique d'un vecteur

Avec suppression des valeurs manquantes.

```
moyenne <- function(x) {
  x.not.na <- x[!is.na(x)]
  ## moyenne empirique
  resultat <- sum(x.not.na) / length(x.not.na)

  return(resultat)
}
```

Tests

```
moyenne(rnorm(100))

## [1] 0.002743981

moyenne(c(1,-5,3,NA,8.7))

## [1] 1.925
```

Les arguments, leurs valeurs par défaut

Propriétés

- les arguments peuvent être passés dans le désordre s'ils sont nommés : `var=object`,

Les arguments, leurs valeurs par défaut

Propriétés

- les arguments peuvent être passés dans le désordre s'ils sont nommés : `var=object`,
- on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction : `var=10`.

Les arguments, leurs valeurs par défaut

Propriétés

- les arguments peuvent être passés dans le désordre s'ils sont nommés : `var=object`,
- on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction : `var=10`.
- en cas de sorties multiples, les sorties doivent être renvoyées sous forme de liste.

Les arguments, leurs valeurs par défaut

Propriétés

- les arguments peuvent être passés dans le désordre s'ils sont nommés : `var=object`,
- on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction : `var=10`.
- en cas de sorties multiples, les sorties doivent être renvoyées sous forme de liste.

Remarques

- Les valeurs par défaut rendent la lecture des fonctions beaucoup plus aisée pour l'utilisateur : imposer peu d'arguments obligatoires.
- Les noms des éléments de la liste définie dans la fonction sont conservés à l'extérieur de la fonction.

Un exemple (un tout petit peu) plus avancé

Résumé numérique d'un vecteur

```
resume <- function(x,na.rm=TRUE,affiche=FALSE) {
  mu <- mean(x,na.rm=na.rm)
  s2 <- var(x,na.rm=na.rm)
  if (affiche) {
    cat("\nMoyenne:",mu,"et variance:",s2)
  }
  return(list(moyenne = mu, variance = s2))
}
```

```
out <- resume(rnorm(100))
out$variance

## [1] 1.009893

out <- resume(affiche=TRUE,x=rexp(100,0.5))

##
## Moyenne: 2.229719 et variance: 4.20312
```

Fonction anonyme

Définition

Il s'agit d'une fonction qui ne porte pas de nom.
$\rightsquigarrow$ Elle est définie « à la volée » au moment de son utilisation.

Utilisation

- couplée à une fonction type `xapply`,
- généralement courte,
- possède peu d'arguments,
- fait sens "localement", dans un contexte particulier du programme.

Fonction anonyme : exemples

Utilisation avec tapply

Dans les données `iris`, on cherche la somme des carrés de la longeur de sépale pour chaque espèce.

```
with(iris, tapply(Sepal.Length, Species, function(x) {return(sum(x^2))}))
```

```
##     setosa versicolor  virginica
##    1259.09    1774.86    2189.90
```

Utilisation avec apply

On veut calculer la moyenne de chaque ligne de la matrice A en ne conservant que les valeurs entre 0 et 1.

```
A <- matrix(rnorm(100*200), 100,200)
head(apply(A, 1, function(x) {sum(x[x>0 & x<1])}))
```

```
## [1] 25.44734 26.36134 31.44209 32.89236 30.05999 35.07290
```

Fonction anonyme : exemples II

Utilisation avec `sapply/lapply`

On veut estimer la distribution discrétiser dans chaque colonne de `iris`.

```
lapply(iris, function(x) { ## x est la colonne courante
    if(is.factor(x)) {
        return(table(x))
    } else {
        return(table(cut(x, seq(min(x), max(x), len=5))))
    }
})
```

```
## $Sepal.Length
##
## (4.3,5.2] (5.2,6.1]   (6.1,7]   (7,7.9]
##        44        50        43        12
##
## $Sepal.Width
##
##   (2,2.6] (2.6,3.2] (3.2,3.8] (3.8,4.4]
##        23        83        37         6
##
## $Petal.Length
##
##    (1,2.48] (2.48,3.95] (3.95,5.43]  (5.43,6.9]
##          49          11          61          28
##
## $Petal.Width
```