

Introduction au logiciel R

Formation doctorale

Cyril Dalmasso
cyril.dalmasso@univ-evry.fr

Laboratoire de Mathématiques et Modélisation d'Evry (LaMME)
Université Paris-Saclay, CNRS, Université d'Evry

13 - 15 juin 2022

Présentation générale

Structures de données

Entrés, sorties et graphiques

Introduction à la programmation en R

Compléments sur ggplot2

Présentation générale

Qu'est-ce que R ?

- ▶ R est un logiciel de développement scientifique spécialisé dans le calcul et l'analyse statistique de données.
- ▶ R est aussi :
 - ▶ un langage (dérivé du langage S développé par John Chambers au sein des laboratoires Bell en 1975)
 - ▶ un environnement
 - ▶ un projet open source (projet GNU)
 - ▶ un logiciel multi-plateforme (Linux, Mac, Windows)

Principales fonctionnalités

- ▶ Gestion des données (lecture, manipulation, stockage)
- ▶ Algèbre linéaire (opérations classiques sur les vecteurs, tableaux et matrices)
- ▶ Statistiques et analyse de données (très grand nombre de méthodes d'analyse de données disponibles, des plus anciennes aux plus récentes)
- ▶ Moteur de sorties graphiques (sorties écran ou fichier)
- ▶ Système de packages/modules (alimenté par la communauté)
- ▶ Interface avec C/C++, Fortran,

Avantages et inconvénients de R

Avantages :

- ▶ Libre et gratuit
- ▶ Richesse des modules
- ▶ Souplesse
- ▶ Prise en main rapide (Syntaxe intuitive et compact)
- ▶ Développement rapide (langage de scripts)
- ▶ Nombreuses possibilités graphiques

Avantages et inconvénients de R

Inconvénients :

- ▶ Aide intégrée succincte
- ▶ Debugger un peu sec
- ▶ Code parfois illisible (compacité)
- ▶ Facile de mal coder
- ▶ Lent par rapport à C/C++
- ▶ Personnalisation des graphiques un peu lourde

Installation

Page du CRAN (Comprehensive R Archive Network)

{<http://cran.r-project.org/>}



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#) ([Debian](#), [Fedora/Redhat](#), [Ubuntu](#))
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-05-18, Camp Pontanezen) [R-4.1.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Premiers pas (mode console)

► Lancement de R :

```
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"  
Copyright (C) 2017 The R Foundation for Statistical Computing  
Platform: x86_64-apple-darwin15.6.0 (64-bit)
```

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

```
> |
```

Premiers pas (mode console)

- ▶ R est un interpréteur de commandes (pas de compilations)

```
2+2
```

- ▶ Plusieurs commandes/instructions peuvent être écrites sur une même ligne (séparées par ';').

```
2+1;3*7
```

- ▶ Tout ce qui suit le symbole `#` est ignoré (il s'agit d'un commentaire)

```
2+1 # ceci est une addition
```

Premiers pas (mode console)

- ▶ La plupart des instructions sont des fonctions appliquées à d'autres objets. La syntaxe est toujours `nomdelafunction(argument1,argument2,...)`

```
log(2,base=2)
```

- ▶ Quitter R :

```
q()
```

Tout environnement sauvé lors de la fermeture est rouvert au démarrage de la session suivante $\Rightarrow$ Attention à ne pas saturer la mémoire

Trouver de l'aide

Dans la console

- ▶ `help(str)` ou `?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.

```
?str
```

Par l'onglet "Help" de Rstudio

Sur le Web

Trouver de l'aide

Remarque

Pour obtenir un descriptif détaillé de chacune des fonctions citées dans ce document, consulter la page d'aide correspondante.

Les packages

Les packages sont des codes qui peuvent être inclus pour étendre les fonctionnalités de R

Installation

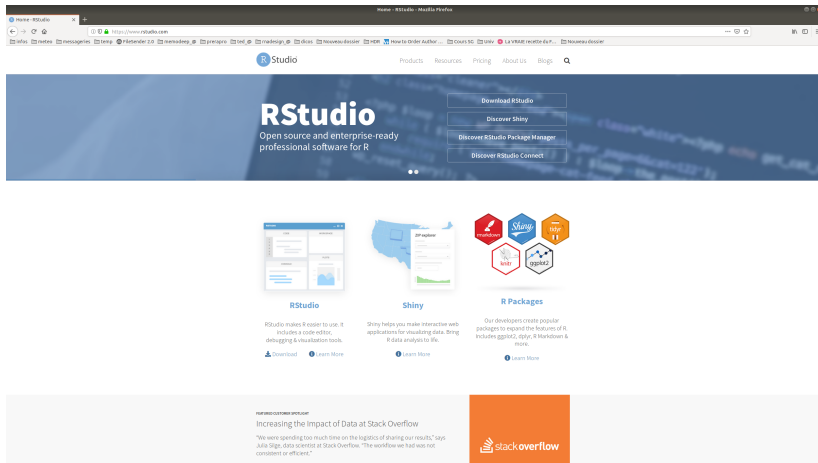
- ▶ Avec la fonction `install.packages`
- ▶ Avec des fonctions spécifiques (exemple : Bioconductor).

Chargement

- ▶ Avec la fonction `library`
- ▶ La fonction `require` permet de tester si un package est chargé et le charge si ce n'est pas le cas

R-studio

- R-studio est un environnement de travail intégré
{<https://www.rstudio.com/>}



The image shows a screenshot of the RStudio website homepage. At the top, there's a navigation bar with links for Products, Resources, Pricing, About us, and Blogs. The main header features the RStudio logo and the tagline "Open source and enterprise-ready professional software for R". Below this, there are four buttons: "Download RStudio", "Discover Shiny", "Discover RStudio Package Manager", and "Discover RStudio Connect". The main content area is divided into three sections: RStudio, Shiny, and R Packages. Each section has a brief description and a "Learn More" link. At the bottom, there's a section titled "INCREASE CUSTOMER IMPACT" with a quote from Julia Silge, a data scientist at Stack Overflow, and the Stack Overflow logo.

Home - RStudio - Mozilla Firefox

Home - RStudio

Products Resources Pricing About us Blogs

RStudio

Open source and enterprise-ready professional software for R

Download RStudio

Discover Shiny

Discover RStudio Package Manager

Discover RStudio Connect

RStudio

RStudio makes R easier to use. It includes a code editor, debugging & visualization tools.

Download Learn More

Shiny

Shiny helps you make interactive web applications for visualizing data. Bring R data analysis to life.

Learn More

R Packages

Our developers create popular packages to expand the features of R. Includes ggplot2, dplyr, R Markdown & more.

Learn More

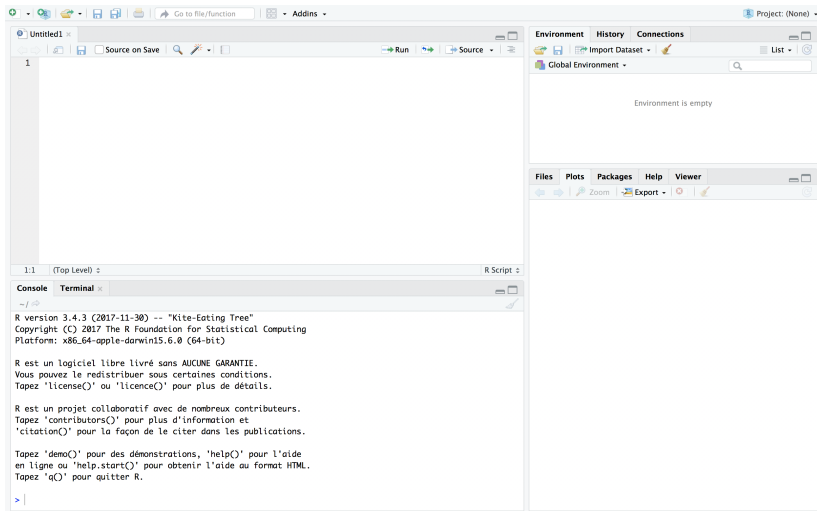
INCREASE CUSTOMER IMPACT

Increasing the Impact of Data at Stack Overflow

"We were spending too much time on the logistics of sharing our results," says Julia Silge, data scientist at Stack Overflow. "The workflow we had was not consistent or efficient."

stackoverflow

R-studio



R-studio- Editeur de script

- ▶ Il est préférable d'entrer les commandes dans l'éditeur de script
- ▶ et de sauvegarder le script
- ▶ envoyer la ligne actuelle ou le texte sélectionné à la console R à l'aide du raccourci Ctrl-Entrée

R Markdown- Le langage R Markdown

R Markdown offre une syntaxe simplifiée pour mettre en forme des documents contenant à la fois du texte, des instructions R. Cet outil permet de produire des rapports d'analyse détaillés et commentés.

Installer knitr

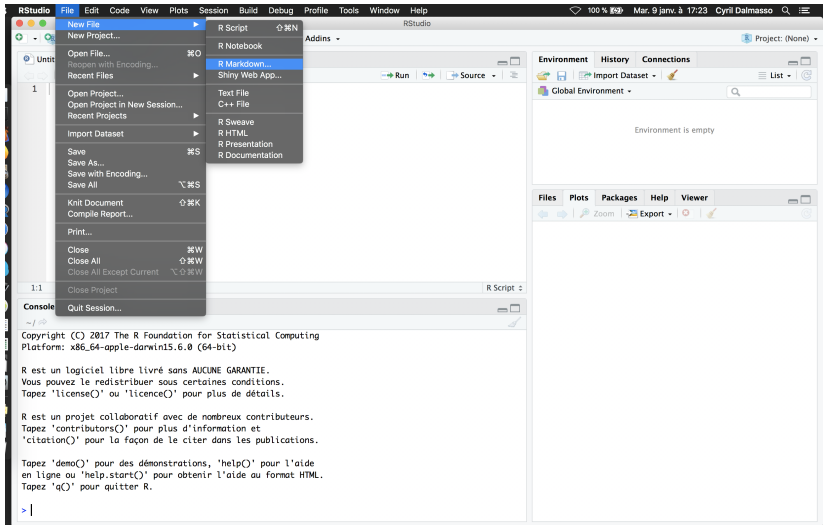
Dans la console

```
install.packages('knitr')
```

Par le menu

Tools -> Install Packages

R Markdown- Créer un fichier de type Rmarkdown (.Rmd)



R Markdown - YAML

Création du fichier avec une entête dite yaml qui précise comment peut être transformé le fichier.

```
---  
title: "An Example Using the Tufte Style"  
author: "John Smith"  
output:  
  tufte::tufte_handout: default  
  tufte::tufte_html: default  
---
```

R Markdown - Code chunks

Exemple :

```
171 ▾ ## Code chunks
172
173 Ceci est un test d'addition:
174 ```{r, eval=TRUE}`r ```
175 1 + 1
176 ▾ ```
```

Organiser un projet R - Sans R-studio

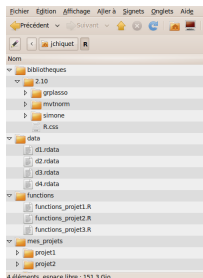


Figure 1: Arborescence type

Créer un répertoire par projet

- ▶ sauvegarde des données : `save.image(file = "f.RData")`
- ▶ sauvegarde des instructions : `savehistory(file = "f.Rhistory")`

Organiser un projet R - Avec R-studio

Utiliser le menu: File -> New Project...

Exercice

Créer un fichier Rmarkdown qui produira une sortie .pdf avec un code chunk de votre choix

Structures de données

Structures de données

- ▶ Variables de types élémentaires
- ▶ Vecteurs
- ▶ Facteurs
- ▶ Matrices
- ▶ Listes
- ▶ Tableaux de données

Variables de types élémentaires

Déclaration/affectation d'une variable

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

```
x <- "test"
```

- ▶ l'opérateur '=' (déconseillé)

```
y = 56
```

- ▶ autre possibilité : la commande assign

```
assign("z",-pi) # ne pas oublier ""
```

Variables de types élémentaires

Remarques

- ▶ L'instruction `y <- 56` consiste à assigner la valeur '56' à l'objet nommé 'y' : rien ne s'affiche.
- ▶ Pour afficher l'objet 'y', il faut utiliser la commande 'y' (ou bien `print(y)`)
- ▶ Attention à ne pas écraser un objet préexistant lors d'une assignation

Variables de types élémentaires

Type, mode et classe

- ▶ Mode : Le mode détermine la structure de base (unique) d'un objet.
 - ▶ objets atomiques : `character`, `numeric`, `logical`, `complex`
 - ▶ objets recursifs : `list`, `function`, ...

Variables de types élémentaires

Type, mode et classe

- Type : Le type détermine la structure interne (à R) d'un objet.
'Type' et 'mode' sont généralement identiques.

Exceptions :

type	mode
double	numeric
integer	
special	functions
builtin	
symbol	names
language	call

Variables de types élémentaires

Type, mode et classe

- ▶ Classe : La classe d'un objet permet de déterminer la façon dont les fonctions génériques (`plot`, `summary`, etc) vont se comporter (programmation orientée objet). Si aucune classe particulière n'est définie, la classe d'un l'objet correspond à son mode.
 - ▶ `list`, `matrix`, `data.frame`, `factor`, ...

Variables de types élémentaires

Valeurs spéciales, réservées par R

- ▶ TRUE/ FALSE, indicateurs logiques
- ▶ NA ('Not Available'), valeurs manquantes
- ▶ NaN ('Not a Number'), résultat numérique aberrant
- ▶ Inf et -Inf, plus et moins ∞ , de type double
- ▶ NULL, l'objet nul, de type NULL

Exemples :

```
c(4,NA,1)
```

```
0/0
```

```
1/0
```

```
names(1)
```

Variables de types élémentaires

Manipulation de variable atomique numérique

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

```
2+2
```

```
2*6
```

```
5/2
```

```
1-4
```

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

```
3^2
```

```
5%%2
```

```
5%/%2
```

```
log(1)
```

```
sqrt(4)
```

```
cos(pi)
```

Variables de types élémentaires

Manipulation de variable atomique logique

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

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

```
x|y
```

```
x&y
```

```
x==y
```

```
x!=y
```

```
xor(x,y)
```

Variables de types élémentaires

Manipulation de chaîne de caractères

- ▶ Les chaînes de caractères sont toujours notées entre ""
- ▶ Opérations élémentaires:
'paste', 'substr', 'strsplit', 'sub' et bien d'autres

```
x <- "Hello"; y <- "world"  
paste(x,y)  
paste(x,y,sep=" ")  
substr(x,2,4)  
strsplit(x,"l")
```

Variables de types élémentaires

Tester le type d'une variable

Les fonctions de format `is.xx` permettent de tester le mode/type d'une variable

```
is.numeric(pi)
is.integer(pi)
is.double(pi)
is.character(pi)
is.na(pi)
is.null(pi)
```

Variables de types élémentaires

Convertir une variable

La fonction générique `as.xx()` et ses dérivées le permet

```
x <- 2  
is.character(x)  
x <- as.character(x)  
is.character(x)  
as.numeric("toto")  
as.logical(sqrt(2))
```

Vecteurs

Définition

Un vecteur est une collection d'entités de même nature

Propriétés

- ▶ objet le plus élémentaire sous $\mathbb{R}$
- ▶ le mode d'un vecteur est défini par celui des entités qui le composent

Vecteurs

Création par initialisation

- ▶ avec les fonctions issues des types

```
x <- integer(3)
double(5)
character(2)
logical(6)
```

- ▶ avec la fonction vector()

```
vector("integer",3)
vector("double",5)
vector("character",2)
vector("logical",6)
```

Vecteurs

Création par concaténation

- ▶ avec la fonction `c()`

```
x <- c(5,3,9,1)
y <- c("pim","pam","poum")
```

- ▶ Remarque : Si tous les éléments n'ont pas le même mode, la fonction `c()` effectue des conversions automatiques

```
z <- c(5,TRUE,"toto",1)
z
mode(z)
```

Vecteurs

Création par génération de séquences

- ▶ avec l'opérateur ':'

```
1:5
```

```
-1:5
```

```
1:-5
```

- ▶ avec la fonction `seq()`

```
seq(1,5)
```

```
seq(1,5,by=0.5)
```

```
seq(1,5,length.out=12)
```

Vecteurs

Création par répétition d'éléments

- ▶ avec la fonction `rep()`

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

```
rep(NA,5)
```

Vecteurs

Opérations arithmétiques

- Les opérations arithmétiques s'effectuent terme à terme

```
x <- c(1,6,2); y <- c(3,4,-2)
```

```
x+y
```

```
x*y
```

```
x^y
```

- Lorsque les dimensions des deux vecteurs sont différentes, la longueur du vecteur le plus court est ajustée en recyclant les données

```
x <- c(1,6,2); y <- c(3,7)
```

```
x+y
```

Vecteurs

Opérateurs mathématiques

- ▶ Fonctions numériques élémentaires : `floor()`, `ceiling()`, `round()`

```
x <- c(1,2,-3,-4); y <- c(5,-6,9,0)
x/y
floor(x/y)
ceiling(x/y)
round(x/y)
round(x/y,digit=2)
```

Vecteurs

Opérateurs mathématiques

- Fonctions arithmétiques élémentaires : `abs()`, `log()`, `exp()`, `sqrt()`, `sin()`, `cos()`, `tan()`, ...

```
log(c(1,2,3))  
abs(c(-1,5,-8))  
exp(c(0,1))  
sqrt(c(4,25,9,16))  
sin(c(0,pi/2))
```

Vecteurs

Opérateurs mathématiques

- Fonctions déterminant le minimum et le maximum terme à terme : `pmin()`, `pmax()`

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


Vecteurs

Opérateurs mathématiques

- Fonctions caractérisant un vecteur : `length()`, `prod()`, `sum()`, `max()`, `min()`, `range()`, `cumsum()`, `diff()`, `cumprod()`, ...

```
x <- c(1,2,-3,-4)
length(x)
prod(x)
sum(x)
max(x)
cumsum(x)
```

Attention : $\min(x,y) = \min(c(x,y)) \neq \text{pmin}(x,y)$

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

Vecteurs

Opérateurs ensemblistes

- ▶ fonctions `unique()`, `intersect()`, `union()`, `setdiff()`, `setequal()`, `is.element()`

```
x <- c("banane", "citron", "citron")
y <- c("orange", "banane")
unique(x)
union(x,y)
intersect(x,y)
setdiff(x,y)
setequal(x,y)
is.element("orange",y)
```

Vecteurs

Opérateurs logiques

► '&', '|', '==', '!=', 'xor', '<', '>', '<=', '>='

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

```
x==y # attention à ne pas taper x=y
```

```
x<y
```

```
x<=y
```

Vecteurs

Nommer les éléments d'un vecteur

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

```
x <- 1:3  
names(x) <- c("pim", "pam", "poum")  
x  
names(x)
```

Vecteurs

Indexation des vecteurs

L'indexation des vecteurs

- ▶ permet la sélection d'un sous-ensemble du vecteur
- ▶ permet d'affecter de nouvelles valeurs
- ▶ le sous-ensemble est spécifié entre crochets `x[subset]`

Vecteurs

L'objet `subset` (dans `x[subset]`) est

- ▶ un vecteur logique (de même taille que `x`)
- ▶ un vecteur numérique d'entiers positifs correspondant aux indices des valeurs à inclure
- ▶ un vecteur numérique d'entiers négatifs correspondant aux indices des valeurs à exclure
- ▶ un vecteur numérique de chaînes de caractères donnant le nom des éléments à conserver

```
x <- c(-3,2,NA,5)
names(x) <- c("v1","v2","v3","v4")
# ou bien : names(x) <- paste("v",1:4,sep="")
x[c(1,4)]
x[-2]
x[!is.na(x)]
x[!is.na(x)&x>0]
x[c("v2","v3")]
```

Vecteurs

Autres commandes d'indexation et de sélection

- ▶ Classer avec les fonctions `sort()`, `order()`, `rank()`

```
x <- sample(10:15); x # voir l'aide de 'sample'
sort(x)
order(x)
rank(x)
```

- ▶ Extraire avec la fonction `which()`

```
which(x>12)
```

Vecteurs

Rechercher des éléments dans un vecteur

- Fonctions `match()`, `%in%`, ...

```
x <- c(1,3,2,5,7); y <- c(4,5,1)
x%in%y
y%in%x
match(x,y)
match(y,x)
```


Vecteurs

Fonctions statistiques élémentaires

- moyenne (`mean()`), médiane (`median()`), variance (`var()`) et écart-type (`sd()`)

```
x <- c(1,5,-2,9)
mean(x)
median(x)
var(x)
sd(x)
```

- La fonction `summary()` calcule plusieurs indicateurs numériques

```
summary(x)
```

Vecteurs

Représentation graphique

- ▶ avec les fonctions `barplot()`, `hist()`, `plot()`, ...

```
x <- sample(1:10,replace=TRUE)
x <- rnorm(1000)
names(x) <- paste("v",1:length(x))
plot(x)
barplot(x)
hist(x)
boxplot(x)
```

Facteurs

Définition

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

Facteurs

Création

- ▶ avec les fonctions `factor()`, `as.factor()`

```
factor(c(1,6,6,2))  
as.factor(c(1,6,6,2))  
factor(c(1,6,6,2),levels=1:6)  
factor(c(1,6,6,2),levels=1:3)  
factor(c(1,6,6,2),ordered=TRUE)
```

Facteurs

Un facteur est en fait un vecteur d'entiers muni d'un attribut
levels

```
x <- factor(c(1,6,6,2)); x  
attributes(x)  
as.numeric(x)
```

Facteurs

Gestion des attributs

- ▶ avec les fonctions `levels()`, `nlevels()`, `relevel()`, `table()`

```
x <- factor(c(1.3,1.3,1.7,"manquante"))
x <- factor(c(1.3,6,6,2)); x
nlevels(x)
levels(x)
levels(x) <- c("faible","fort","moyen"); x
is.ordered(x)
x <- factor(x,levels=c("fort","moyen","faible"),ordered=TRUE)
table(x)
```

Facteurs

Les fonctions `split()` et `tapply()`

- ▶ La fonction `split()` découpe un vecteur `x` selon les valeurs d'un facteur `y`
- ▶ La fonction `tapply()` applique une fonction au vecteur `x` pour chaque catégorie du facteur `y`

Exemple : étude de la longueur des pétales d'iris en fonction de l'espèce

```
x <- c(1.5, 4.4, 5.0, 1.6, 4.7)
y <- factor(c("setosa","versicolor","virginica",
              "setosa","versicolor"))

split(x,y)
tapply(x,y,mean)
tapply(x,y,var)

mean(x[y=="setosa"])
```

Facteurs

Représentation graphique

- ▶ avec la fonction `plot()`

```
plot(y)
```


Matrices

Définitions

- ▶ Un tableau (array) est un vecteur muni d'un attribut dimension (dim) lui-même défini par un vecteur.
- ▶ Une matrice (matrix) est un tableau à deux dimensions.

```
y <- array(1:24,c(3,4,2)); y  
x <- matrix(1:12,2,3); x  
class(y); class(x)
```

Matrices

Remarques

- ▶ Un objet array à deux dimensions est automatiquement converti en matrix
- ▶ Un vecteur auquel on ajoute un attribut dimension est automatiquement converti en array
- ▶ Par défaut, R range les éléments par colonne (argument byrow)
- ▶ Lors de la création d'un tableau, R recycle les éléments jusqu'à ce que les contraintes de dimension soient vérifiées

```
matrix(1:6,2,3)
matrix(1:6,2,3,byrow=TRUE)
matrix(1:5,2,3)
```

Matrices

Dimensions

- ▶ On accède aux attributs de dimension d'un tableau à l'aide des fonctions `dim()`, `ncol()`, `nrow()`

```
x <- matrix(1:12,2,3); x  
dim(x)  
nrow(x)  
ncol(x)
```

Matrices

Opérateurs

- ▶ Etant donné qu'une matrice est un vecteur muni d'une dimension, la plupart des opérateurs vectoriels s'appliquent.
- ▶ Opérateurs matriciels usuels :
 - ▶ produit matriciel : `%*%`
 - ▶ produit scalaire : `crossprod`
 - ▶ transposée d'une matrice : `t`
 - ▶ diagonale d'une matrice : `diag`

```
x <- matrix(1:12,2,3)
diag(x)
diag(x) <- 1; x
t(x)
y <- matrix(runif(9),3,3)
y%*%(y^2)
crossprod(y,y^2)
```

Matrices

Nommer les éléments d'une matrice

- Les fonctions `rownames()` et `colnames()` permettent d'accéder/attribuer des noms aux lignes et colonnes d'une matrice

```
x <- matrix(1:12,2,3); x
rownames(x) <- c("tic","tac")
colnames(x) <- c("pim","pam","poum")
x
```

Matrices

Indexation

- ▶ Même principe que pour les vecteurs, suivant chaque dimension (`x[subset1,subset2]`)

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

```
x[2,2:3]; x[2,-1]
```

```
x[2,]
```

```
x[x[,1]>1,]
```

```
x[,c("pim", "poum")]
```

Matrices

Concaténation

- ▶ avec les fonctions `c()`, `cbind()`, `rbind()`

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

Matrices

Exécuter une opération par ligne ou par colonne

- La fonction `apply()` est un outil très puissant

```
x <- matrix(1:12,2,3); x  
apply(x,1,sum)  
apply(x,2,mean)
```


Matrices

Représentations graphiques

- ▶ avec les fonctions `contour()`, `persp()`, `image()`, ...

```
?volcano  
dim(volcano)  
head(volcano)  
volcano[1:3,1:5]  
contour(volcano)  
persp(volcano)  
image(volcano)  
filled.contour(volcano)
```

Matrices

Résolution de systèmes linéaires, inversion matricielle

La commande `solve` résout :

$$Ax = b$$

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

Matrices

Autres fonctions pour l'algèbre linéaire

R dispose des outils classiques d'algèbre linéaire

- ▶ `det`: calcule le déterminant d'une matrice
- ▶ `chol`: factorisation de Cholesky ($A = C^T C$, 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.
- ▶ ...

Listes

Définition

- Une liste est une collection d'objets hétérogènes. Elle est définie par la commande `list`.

```
maliste <- list(matrix(1:4,2,2),"toto",c(1,10,3))  
names(maliste) <- c("pim","pam","poum"); maliste
```

```
maliste <- list(pim=matrix(1:4,2,2),pam="toto",  
               poum=c(1,10,3)); maliste
```

```
maliste <- list()  
maliste$pim <- matrix(1:4,2,2); maliste  
maliste$pam <- "toto"; maliste  
maliste$poum <- c(1,10,3); maliste
```

Listes

Accéder aux éléments d'une liste

- ▶ On accède au i^{eme} élément de la liste x par indexation $x[[i]]$ ou $x[i]$
- ▶ Si les éléments sont nommés, on peut accéder aux éléments en utilisant le nom : $x[[nom]]$ ou xnom$

```
maliste[[2]]  
maliste[["pam"]]  
maliste$pam
```

Listes

Dépiler une liste

- ▶ La fonction `unlist()` met à plat et simplifie en vecteur si possible

```
unlist(maliste)
```

Listes

Fonctions `lapply()` et `sapply()`

Les fonctions `lapply()` et `sapply()` permettent d'appliquer une fonction à chaque élément de la liste. Seule la présentation / organisation des résultats diffère (entre les deux fonctions)

```
lapply(maliste,length)
sapply(maliste,length)
```

Tableaux de données (data.frame)

Définition

Un tableau de données (data.frame) est une liste ayant certaines contraintes permettant de rassembler vecteurs et facteurs sous la forme d'un tableau.

Création

- ▶ avec la fonction data.frame()

```
data.frame(tic=1:3,tac=c("pim","pam","poum"))
```


Tableaux de données (`data.frame`)

Remarques

- ▶ Un tableau de données peut être vu comme une matrice dont les colonnes peuvent être de modes différents
- ▶ C'est l'objet idéal pour manipuler et analyser des données.

Tableaux de données (data.frame)

Manipulations d'un tableau de données

- Un data.frame se manipule comme une liste, mais aussi comme une matrice

```
x <- data.frame(tic=1:3,tac=c("pim","pam","poum"))  
x[1:2,2]  
x$tac[1:2]  
x[["tac"]][1:2]
```

Tableaux de données (data.frame)

Remarque

Les chaînes de caractères sont généralement converties en facteurs.
Attention lors de l'importation de données

```
x <- data.frame(1:3,c(4,"missing",5))  
x  
sum(x[1,])
```

Tableaux de données (data.frame)

Fonctions attach() et detach()

La fonction attach() (et la fonction detach()) place (et enlève) les éléments du tableau dans l'itinéraire de recherche

```
x <- data.frame(tic=1:3,tac=c("pim","pam","poum"))
tic
attach(x)
tic
detach(x)
tic
```

Tableaux de données (data.frame)

Fonctions `stack()` et `unstack()`

La fonction `stack()` (et la fonction `unstack()`) empile (et dépile) les colonnes d'un `data.frame`

```
x <- data.frame(speed=c(4,4,7,7),dist=c(2,10,4,22)); x
stack(x)
y <- data.frame( tic=1:6, tac=rep(c("pim","pam","poum"),2))
y
unstack(y)
```

Tableaux de données (data.frame)

Fonction summary()

La fonction `summary()` réalise un résultat statistique adapté au type de chaque colonne

```
x <- data.frame(tic=1:3,tac=c("pim","pam","poum"))  
summary(x)
```

```
data(iris)  
iris[1:6,]  
head(iris)  
summary(iris)
```

Entrés, sorties et graphiques

Jeux de données disponibles sous R

Fonction data()

R dispose d'un ensemble de jeux de données directement utilisables. La fonction data() permet de les lister puis de les charger

Remarques

- ▶ La description d'un jeu de données est accessible dans l'aide
- ▶ L'installation de nouveaux packages permet souvent d'accéder à de nouveaux jeux de données

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


Gestion des fichiers et répertoires

Gestion des chemins

- ▶ La fonction `getwd()` indique le répertoire de travail courant
- ▶ La fonction `setwd()` permet de définir un nouveau répertoire de travail

```
getwd()
```

```
setwd("/Users/cdalmasso/Cyril/0_Skmac/Enseignement/2017_2018")
```

- ▶ Les raccourcis suivants peuvent être utilisés :
 - ▶ '~' est remplacé par le chemin du répertoire de l'utilisateur
 - ▶ '.' est remplacé par le chemin du répertoire courant
 - ▶ '..' est remplacé par le répertoire parent du répertoire courant

Gestion des fichiers et répertoires

Gestion des répertoires

- ▶ Les fonctions `dir()` et `list.files()` permettent de connaître le contenu d'un répertoire.
- ▶ La fonction `dir.create()` permet de créer un nouveau répertoire

Gestion des fichiers et répertoires

Gestion des fichiers

- Certaines opérations de base peuvent être réalisées sur les fichiers à l'aide des fonctions `file.create()`, `file.remove()`, `file.rename()`, `file.copy()`, `file.append()`, ...

Fichiers binaires

Fonctions save() et load()

- ▶ La fonction save() sauvegarde un ensemble d'objets R dans un fichier binaire
- ▶ La fonction load() permet de les recharger

```
x <- rnorm(10); y <- "toto"
save(x,y,file="objetsxety.RData")
rm(x,y) # -> supprime les objets x et y
ls() # -> liste tous les objets de l'environnement

load(file="objetsxety.RData")
ls()
```

Lecture de fichiers

Fonction `read.table()`

- ▶ La fonction `read.table()` permet de lire un fichier formaté sous forme de table et de le convertir sous la forme d'un objet `data.frame`
- ▶ Examiner au préalable le formatage du fichier à l'aide d'un éditeur de texte afin de paramétrer les nombreuses options disponibles

```
vignes <- read.table("baies_raisin.txt")  
vignes <- read.table("baies_raisin.txt", sep="\t")  
head(vignes)  
str(vignes)  
vignes <- read.table("baies_raisin.txt",  
                      sep="\t", header=TRUE)  
head(vignes)  
str(vignes)
```

Lecture de fichiers

Fonctions `read.csv()` et `read.delim()`

- ▶ Les fonctions `read.csv()` et `read.delim()` sont des raccourcis pour la fonction `read.table()`

```
vignes <- read.delim("baies_raisin.txt")  
head(vignes)
```

- ▶ Sous RStudio, il est possible d'utiliser les menus

Ecriture de fichiers

Fonctions `write.table()`, `write.csv()`, `write.delim()`

- ▶ La fonction `write.table()` permet d'exporter un `data.frame` dans un fichier.
- ▶ Les fonctions `write.csv()`, `write.delim()` sont des raccourcis pour la fonction `write.table`

```
x <- data.frame(var1=1:3,var2=c("pim","pam","poum")); x
write.table(x, file="objetx.txt")
write.table(x, file="objetx.2.txt",quote=FALSE,
            sep="\t",row.names=FALSE)
```

Bases des graphiques

Fonction plot

La fonction `plot` est une fonction générique dont le comportement dépend du type d'objets auxquels elle s'applique

```
data(iris); attach(iris); iris[1:5,]  
plot(Sepal.Length)  
plot(Sepal.Length~Species)  
plot(Sepal.Length~Sepal.Width)  
plot(iris)  
plot(hist(Sepal.Length))  
plot(lm(Sepal.Length~Sepal.Width))
```


Bases des graphiques

Options graphiques

La liste de nombreuses options graphiques applicables à la plupart des fonctions graphiques peut être obtenue par la fonction `par()`

```
?par  
par(mfrow=c(1,2))  
plot(Sepal.Length)  
plot(Sepal.Length, pch=19, col="dodgerblue", cex=.5,  
      main="Longueurs", las=1)
```

Bases des graphiques

Ajouter...

- ▶ une légende : `legend`
- ▶ des droites : `abline`
- ▶ des courbes : `lines` (voir aussi la fonction `curve()` pour tracer le graphe d'une fonction)

```
curve(sqrt,from=0,to=2,col="dodgerblue",lwd=2,  
      main="Exemple")  
abline(a=.5,b=.5,col="red",lwd=2,lty=2)  
legend("bottomright",c("courbe","tangeante"),lty=c(1,2),  
      col=c("dodgerblue","red"),bty="n",lwd=2)
```

Bases des graphiques

Rediriger la sortie graphique

Il est possible d'exporter les graphiques à l'aide des fonctions `png()`, `pdf()`, `jpeg()`, `bmp()`, `tiff()`

```
png(file="legraphe.png")  
curve(sqrt,from=0,to=2,col="dodgerblue")  
dev.off()
```

Bases des graphiques

Ouverture/fermeture de fenêtre

- ▶ Ouverture d'une nouvelle fenêtre avec les fonctions `X11()` (Linux, mac OS), `quartz()` (mac OS) ou `windows()` (Windows)
- ▶ Fermeture d'une fenêtre graphique avec la fonction `dev.off()`

```
quartz()  
plot(1:5)  
dev.off()
```

Bases des graphiques

Découpage de fenêtres

- ▶ Découpage d'une fenêtre avec les fonctions
 - ▶ `par(mfrow=c(n,m))`
 - ▶ `layout()`

```
par(mfrow=c(2,2))  
plot(1:5); barplot(1:5); hist(1:5); boxplot(1:5)  
  
layout(matrix(c(1,2,3,4),2,2))  
plot(1:5); barplot(1:5); hist(1:5); boxplot(1:5)  
layout(matrix(c(1,2,3,3),2,2))  
plot(1:5); barplot(1:5); boxplot(1:5)  
layout(matrix(c(0,1,2,3),2,2),c(1,2),c(2,1.5))  
plot(1:5); barplot(1:5); boxplot(1:5)
```

Bases des graphiques

Le package ggplot2

```
data("iris")
plot(iris[,1:4])
library(GGally)
ggpairs(iris, columns=1:4, mapping=ggplot2::aes(color=Species))

vignes <- read.table("baies_raisin.txt", sep="\t", header=TRUE)
boxplot(volume.baie..cm3..2008~Population, data=vignes)
library(ggplot2)
qplot(Population, volume.baie..cm3..2008, data=vignes, geom="bar")
```

Bases des graphiques

Barplots avec intervalles de confiance

```
library(gplots)
barplot2(c(2,4,3), plot.ci = TRUE, ci.l = c(1.9,3.6,2.7),
         ci.u = c(2.1,4.4,3.3), xpd = FALSE)
```

Bases des graphiques

Diagrammes de Venn

```
donnees <- matrix(sample(c(TRUE,FALSE),3000,replace=TRUE),nrow=100,ncol=30)

library(VennDiagram)
venn.diagram(data.frame(donnees),filename="testvenn.png")

library(limma)
quartz()
a <- vennCounts(donnees)
vennDiagram(a,cex=1.5,lwd=2, circle.col=c("blue","green","red"))
```


Introduction à la programmation en R

Structures de contrôle

Regrouper les expressions

► Syntaxe :

```
{exprs1; exprs2; exprs3}  
# ou  
{  
    exprs1  
    exprs2  
    exprs3  
}
```

- Remarque : Toutes les instructions sont effectuées mais seule la dernière valeur est retournée

Structures de contrôle

Regrouper les expressions

► Exemples :

```
a <- 10  
exprs1 <- {a <- 2; b <- 3; c <- a+b }  
exprs1  
a; b
```

Structures de contrôle

Executions conditionnelles : if, if/else, ifelse

► Syntaxe :

```
if(coindition){  
    exprs1  
} else {  
    exprs2  
}
```

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

- Remarque : chacune des expressions peut elle-même être une expression conditionnelle

Structures de contrôle

Executions conditionnelles : if, if/else, ifelse

► Exemples :

```
x <- rnorm(1); x
if(x>=0){
  y <- sqrt(x)
} else{
  y <- 5
}
y
```

#ou

```
y <- ifelse(x>=0,sqrt(x),5); y
```

Structures de contrôle

Executions répétées : for, while, repeat

► Syntaxe :

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

```
while (condition){  
  exprs  
}
```

```
repeat{  
  exprs  
}
```

► Remarques :

- privilégier les opérateurs matriciels (apply, etc)
- break est la seule manière d'interrompre une boucle {repeat}

Structures de contrôle

Executions répétées : for, while, repeat

► Exemples :

```
v <- numeric(5)
for(i in list(matrix(1:4,2,2),"toto")){
  print(i)
  #v[i] <- 2*i
}
v
```

Structures de contrôle

Executions répétées : for, while, repeat

► Exemples :

```
i <- 0
while (i<10) {
  print(i)
  i <- i+1
}
```


Structures de contrôle

Executions répétées : for, while, repeat

► Exemples :

```
i <- 0
repeat{
  if(i>10){
    break
  } else{
    print(i)
    i <- i+1
  }
}
```

Fonctions

Définir une fonction

- Syntaxe :

```
nomdelafunction <- function(arg1,arg2,...){  
  expression  
  return(res)  
}
```

- Remarque : return peut être omis (à éviter) : dans ce cas, la dernière valeur calculée est renvoyée

Fonctions

Définir une fonction

► Exemple :

```
mafonction <- function(x){  
  z <- x-5  
  y <- 3*x+5  
  return(y)  
}  
yres <- mafonction(2)  
mafonction(c(3,1,7))
```

Fonctions

Définir une fonction

- ▶ Remarques :
 - ▶ Les arguments peuvent être passés dans le désordre s'ils sont nommés : `arg=valeur`
 - ▶ On peut définir des valeurs par défaut pour n'importe quel argument lors de la création de la fonction
 - ▶ En cas de résultats multiples, les résultats doivent être renvoyé sous forme de liste

```
mafonction <- function(x,a=3,b=5){  
  y <- a*x+b  
  z <- x^2  
  res <- list(res1=y,res2=z)  
  return(res)  
}  
mafonction(2)  
mafonction(b=4,x=1,a=6)  
mafonction(7,2,4)
```

Fonctions

Fonctions anonymes

Ce sont des fonctions qui ne portent pas de nom. Elles sont définies à la volée lors de l'utilisation (généralement avec des fonctions de type apply)

```
x <- matrix(1:4,2,2)
apply(x,2, function(vecteur){mean(vecteur)+100} )
```

Compléments sur ggplot2

Compléments sur ggplot2

Principe général

Construction de graphiques par assemblage de couches.

Éléments de la grammaire ggplot2 :

- ▶ des données (data)
- ▶ de types de représentations graphiques (geom)
- ▶ un système de coordonnées (coord)
- ▶ des propriétés esthétiques (aes)
- ▶ des transformations statistiques (stats)
- ▶ des échelles (scale)
- ▶ des regroupements de données (facet)

Compléments sur ggplot2

Etapes de la construction d'un graphique

1. Préparation des données
2. Initialisation du graphique (fonction ggplot)
3. Ajout de calques (fonctions geom_* et stat_*)
4. Eléments complémentaires

Syntaxe

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +  
<GEOM_FUNCTION>(stat = <STAT>, position = <POSITION>) +  
<COORDINATE_FUNCTION> +  
<FACET_FUNCTION>
```


Compléments sur ggplot2

Exemples

```
ggplot(diamonds, aes(x=price, y=volume)) +  
  geom_point()
```

```
ggplot(diamonds,  
       aes(x=price, y=volume, color=cut, shape=cut)) +  
  geom_point()
```

```
ggplot(diamonds, aes(x=price, y=volume, color=cut)) +  
  geom_point() +  
  geom_smooth(method=lm)
```

```
ggplot(diamonds, aes(x=price, y=volume, color=cut)) +  
  geom_point() +  
  geom_smooth(method=lm) +  
  coord_cartesian(ylim = c(0, 600))
```

Compléments sur ggplot2

Exemples (suite)

```
ggplot(diamonds, aes(x=cut, y=price,color=cut)) +  
  geom_boxplot()
```

```
ggplot(diamonds) +  
  geom_boxplot(aes(x=cut, y=price,color=cut))
```

```
ggplot(diamonds) +  
  geom_violin(aes(x=cut, y=price,color=cut))
```

Compléments sur ggplot2

Aide-mémoire

<https://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf>