

Introduction

Il existe déjà de multiples ouvrages traitant de S-Plus ou R. Dans la majorité des cas, toutefois, ces deux logiciels sont présentés dans le cadre d'applications statistiques spécifiques. Le présent ouvrage se concentre plutôt sur l'apprentissage du langage de programmation sous-jacent aux diverses fonctions statistiques, le S.

Cette seconde édition est principalement une réorganisation du contenu de la première édition. Les chapitres sur la régression linéaire et les séries chronologiques ont été éliminés et les annexes C et D ont été déplacées dans le corps du document. Nous avons également corrigé plusieurs coquilles suite à la révision effectuée par Mme Mireille Côté.

Le texte est la somme de notes et d'exercices de cours donnés par l'auteur à l'École d'actuariat de l'Université Laval. Les six premiers chapitres, qui constituent le cœur du document, proviennent d'une partie d'un cours où l'accent est mis sur l'apprentissage d'un (deuxième) langage de programmation par des étudiants de premier cycle en sciences actuarielles. Les applications numériques et statistiques de S-Plus et R présentées aux chapitres ??, ?? et ?? sont étudiées plus tard dans le cursus universitaire.

Les cours d'introduction au langage S sont donnés à raison de une heure par semaine de cours magistral suivie de deux heures en laboratoire d'informatique. C'est ce qui explique la structure des six premiers chapitres : les éléments de théorie, contenant peu voire aucun exemple, sont présentés en rafale en classe. Puis, lors des séances de laboratoire, les étudiantes et étudiants sont appelés à lire et exécuter les exemples se trouvant à la fin des chapitres. Chaque section d'exemples couvre l'essentiel des concepts présentés dans le chapitre et les complémente souvent. L'étude de ces sections fait donc partie intégrante de l'apprentissage du langage S.

Le texte des sections d'exemples est disponible en format électronique dans le site Internet

http://vgoulet.act.ulaval.ca/intro_S

Certains exemples et exercices trahissent le premier public de ce document :

on y fait à l'occasion référence à des concepts de base de la théorie des probabilités et des mathématiques financières. Les contextes actuariels demeurent néanmoins peu nombreux et ne devraient généralement pas dérouter le lecteur pour qui ces notions sont moins familières.

Les chapitres ?? (fonctions d'optimisation), ?? (générateurs de nombres aléatoires) et ?? (planification d'une simulation en S) sont structurés de manière plus classique, notamment parce que le texte y est en prose.

Le texte prend parti en faveur de l'utilisation de GNU Emacs et du mode ESS pour l'édition de code S. Les annexes contiennent de l'information sur l'utilisation de S-Plus et R avec cet éditeur.

Dans la mesure du possible, cet ouvrage tâche de présenter les environnements S-Plus et R en parallèle, en soulignant leurs différences s'il y a lieu. Les informations propres à S-Plus ou à R sont d'ailleurs signalées en marge par les marques «S+» et «R», respectivement. Étant donné la nette préférence de l'auteur pour R, les divers extraits de code ont généralement été exécutés avec ce moteur S.

À moins d'erreurs et d'omissions (que les lecteurs sont invités à nous faire connaître), les informations données à propos de S-Plus sont exactes pour les versions 6.1 (Linux et Windows), 6.2 Student Edition (Windows) et 7.0 (Linux et Windows). Pour R, la version 2.4.1 (Linux et Windows), soit la plus récente lors de la rédaction, a été utilisée comme référence.

On notera enfin que cet ouvrage n'a aucune prétention d'exhaustivité. C'est ce qui explique les nombreux renvois au livre de [Venables et Ripley \(2002\)](#), plus complet.

L'auteur tient à remercier M. Mathieu Boudreault pour sa collaboration dans la rédaction des exercices.

*Vincent Goulet <vincent.goulet@act.ulaval.ca>
Québec, janvier 2007*

Table des matières

Introduction	i
1 Présentation du langage R	1
1.1 Bref historique	1
1.2 Description sommaire de R	2
1.3 Interfaces	3
1.4 Stratégies de travail	4
1.5 Éditeurs de texte	5
1.6 Anatomie d'une session de travail	8
1.7 Répertoire de travail	9
1.8 Consulter l'aide en ligne	9
1.9 Où trouver de la documentation	9
1.10 Exemples	10
1.11 Exercices	11
2 Bases du langage R	13
2.1 Commandes R	13
2.2 Conventions pour les noms d'objets	15
2.3 Les objets R	16
2.4 Vecteurs	20
2.5 Matrices et tableaux	21
2.6 Listes	24
2.7 <i>Data frames</i>	26
2.8 Indixage	26
2.9 Exemples	28
2.10 Exercices	39
3 Opérateurs et fonctions	41
3.1 Opérations arithmétiques	41

3.2	Opérateurs	42
3.3	Appels de fonctions	43
3.4	Quelques fonctions utiles	44
3.5	Structures de contrôle	50
3.6	Fonctions additionnelles	51
3.7	Exemples	52
3.8	Exercices	60
A	GNU Emacs et ESS : la base	63
A.1	Mise en contexte	63
A.2	Installation	64
A.3	Description sommaire	64
A.4	<i>Emacs-ismes</i> et <i>Unix-ismes</i>	65
A.5	Commandes de base	66
A.6	Anatomie d’une session de travail (bis)	69
A.7	Configuration de l’éditeur	70
A.8	Aide et documentation	70
B	GNU Free Documentation License	71
B.1	APPLICABILITY AND DEFINITIONS	71
B.2	VERBATIM COPYING	73
B.3	COPYING IN QUANTITY	74
B.4	MODIFICATIONS	74
B.5	COMBINING DOCUMENTS	76
B.6	COLLECTIONS OF DOCUMENTS	77
B.7	AGGREGATION WITH INDEPENDENT WORKS	77
B.8	TRANSLATION	78
B.9	TERMINATION	78
B.10	FUTURE REVISIONS OF THIS LICENSE	78
	ADDENDUM: How to use this License for your documents	79
	Bibliographie	81
	Index	83

1 Présentation du langage R

Objectifs du chapitre

- ▶ Connaître la provenance du langage R et les principes ayant guidé son développement.
- ▶ Comprendre ce qu'est un langage de programmation interprété.
- ▶ Savoir démarrer une session R et exécuter des commandes simples.
- ▶ Comprendre l'utilité des fichiers de script R et savoir les utiliser de manière interactive.
- ▶ Savoir créer, modifier et sauvegarder ses propres fichiers de script R.

1.1 Bref historique

À l'origine fut le S, un langage pour «programmer avec des données» développé chez Bell Laboratories à partir du milieu des années 1970 par une équipe de chercheurs menée par John M. Chambers. Au fil du temps, le S a connu quatre principales versions communément identifiées par la couleur du livre dans lequel elles étaient présentées : version «originale» (*Brown Book*; Becker et Chambers, 1984), version 2 (*Blue Book*; Becker et collab., 1988), version 3 (*White Book*; Chambers et Hastie, 1992) et version 4 (*Green Book*; Chambers, 1998); voir aussi Chambers (2000) et Becker (1994) pour plus de détails.

Dès la fin des années 1980 et pendant près de vingt ans, le S a principalement été popularisé par une mise en œuvre commerciale nommée S-PLUS. En 2008, Lucent Technologies a vendu le langage S à Insightful Corporation, ce qui a effectivement stoppé le développement du langage par ses auteurs originaux. Aujourd'hui, le S est commercialisé de manière relativement confidentielle sous le nom Spotfire S+ par TIBCO Software.

Ce qui a fortement contribué à la perte d'influence de S-PLUS, c'est une nouvelle mise en œuvre du langage développée au milieu des années 1990. Inspirés à

la fois par le S et par Scheme (un dérivé du Lisp), Ross Ihaka et Robert Gentleman proposent un langage pour l'analyse de données et les graphiques qu'ils nomment R (Ihaka et Gentleman, 1996). À la suggestion de Martin Maechler de l'ETH de Zurich, les auteurs décident d'intégrer leur nouveau langage au projet GNU¹, faisant de R un logiciel libre.

Ainsi disponible gratuitement et ouvert aux contributions de tous, R gagne rapidement en popularité là même où S-PLUS avait acquis ses lettres de noblesse, soit dans les milieux académiques. De simple dérivé «*not unlike S*», R devient un concurrent sérieux à S-PLUS, puis le surpasse lorsque les efforts de développement se rangent massivement derrière le projet libre. D'ailleurs John Chambers place aujourd'hui ses efforts de réflexion et de développement dans le projet R (Chambers, 2008).

1.2 Description sommaire de R

R est un environnement intégré de manipulation de données, de calcul et de préparation de graphiques. Toutefois, ce n'est pas seulement un «autre» environnement statistique (comme SPSS ou SAS, par exemple), mais aussi un langage de programmation complet et autonome.

Tel que mentionné précédemment, le R est un langage principalement inspiré du S et de Scheme (Abelson et collab., 1996). Le S était à son tour inspiré de plusieurs langages, dont l'APL (autrefois un langage très prisé par les actuaire) et le Lisp. Comme tous ces langages, le R est *interprété*, c'est-à-dire qu'il requiert un autre programme — l'*interprète* — pour que ses commandes soient exécutées. Par opposition, les programmes de langages *compilés*, comme le C ou le C++, sont d'abord convertis en code machine par le compilateur puis directement exécutés par l'ordinateur.

Le programme que l'on lance lorsque l'on exécute R est en fait l'interprète. Celui-ci attend que l'on lui soumette des commandes dans le langage R, commandes qu'il exécutera immédiatement, une à une et en séquence.

Par analogie, Excel est certes un logiciel de manipulation de données, de mise en forme et de préparation de graphiques, mais c'est aussi au sens large un langage de programmation interprété. On utilise le langage de programmation lorsque l'on entre des commandes dans une cellule d'une feuille de calcul. L'interprète exécute les commandes et affiche les résultats dans la cellule.

Le R est un langage particulièrement puissant pour les applications mathématiques et statistiques (et donc actuarielles) puisque précisément développé dans ce but. Parmi ses caractéristiques particulièrement intéressantes, on note :

1. <http://www.gnu.org>

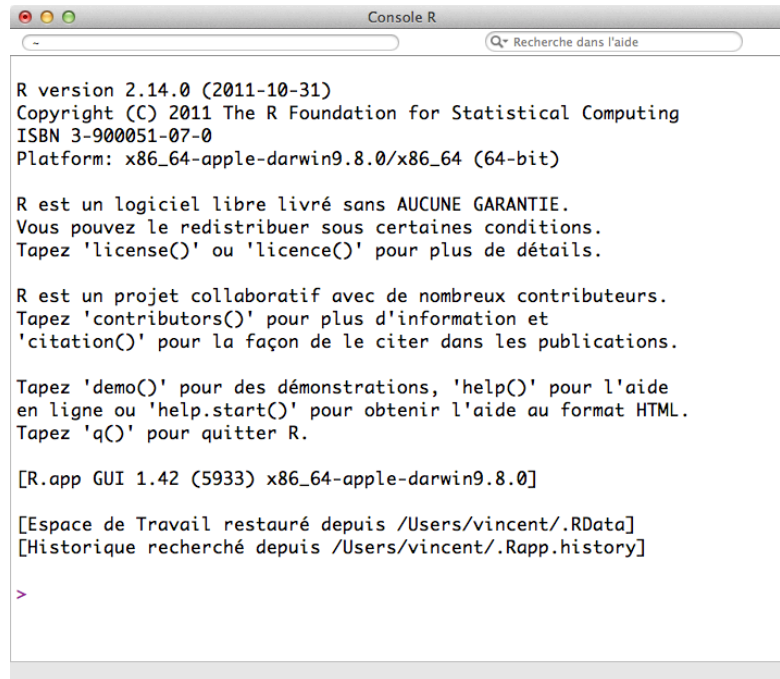


FIG. 1.1: Fenêtre de la console sous Mac OS X au démarrage de R

- ▶ langage basé sur la notion de vecteur, ce qui simplifie les calculs mathématiques et réduit considérablement le recours aux structures itératives (boucles) ;
- ▶ pas de typage ni de déclaration obligatoire des variables ;
- ▶ programmes généralement courts, en général quelques lignes de code seulement ;
- ▶ temps de développement très court.

1.3 Interfaces

R est d'abord et avant tout une application n'offrant qu'une invite de commande du type de celle présentée à la figure 1.1. En soi, cela n'est pas si différent d'un tableur tel que Excel : la zone d'entrée de texte dans une cellule n'est rien d'autre qu'une invite de commande², par ailleurs aux capacités d'édition plutôt réduites.

2. Merci à Markus Gesmann pour cette observation.

- ▶ Sous Unix et Linux, R n'est accessible que depuis la ligne de commande du système d'exploitation (terminal). Aucune interface graphique n'est offerte avec la distribution de base de R.
- ▶ Sous Windows, une interface graphique plutôt rudimentaire est disponible. Elle facilite certaines opérations tel que l'installation de packages externes, mais elle offre autrement peu de fonctionnalités additionnelles pour l'édition de code R.
- ▶ L'interface graphique de R sous Mac OS X est la plus élaborée. Outre la console présentée à la figure 1.1, l'application R. app comporte de nombreuses fonctionnalités, dont un éditeur de code assez complet.

1.4 Stratégies de travail

Dans la mesure où R se présente essentiellement sous forme d'une invite de commande, il existe deux grandes stratégies de travail avec cet environnement statistique.

1. On entre des expressions à la ligne de commande pour les évaluer immédiatement :

```
> 2 + 3  
[1] 5
```

On peut également créer des objets contenant le résultat d'un calcul. Ces objets sont stockés en mémoire dans l'espace de travail de R :

```
> x <- exp(2)  
> x  
[1] 7.389056
```

Lorsque la session de travail est terminée, on sauvegarde une image de l'espace de travail sur le disque dur de l'ordinateur afin de pouvoir conserver les objets pour une future séance de travail :

```
> save.image()
```

Par défaut, l'image est sauvegardée dans un fichier nommé `.RData` dans le dossier de travail actif (voir la section 1.7) et cette image est automatiquement chargée en mémoire au prochain lancement de R, tel qu'indiqué à la fin du message d'accueil :

```
[Sauvegarde de la session précédente restaurée]
```


Cette approche, dite de «code virtuel et objets réels» a un gros inconvénient : le code utilisé pour créer les objets n'est pas sauvegardé entre les sessions de travail. Or, celui-ci est souvent bien plus compliqué que l'exemple ci-dessus. De plus, sans accès au code qui a servi à créer l'objet *x*, comment savoir ce que la valeur 7,389056 représente au juste ?

2. L'approche dite de «code réel et objets virtuels» considère que ce qu'il importe de conserver d'une session de travail à l'autre n'est pas tant les objets que le code qui a servi à les créer. Ainsi, on sauvegardera dans ce que l'on nommera des *fichiers de script* nos expressions R et le code de nos fonctions personnelles. Par convention, on donne aux fichiers de script un nom se terminant avec l'extension `.R`.

Avec cette approche, les objets sont créés au besoin en exécutant le code des fichiers de script. Comment ? Simplement en copiant le code du fichier de script et en le collant dans l'invite de commande de R. La figure 1.2 illustre schématiquement ce que le programmeur R a constamment sous les yeux : d'un côté son fichier de script et, de l'autre, l'invite de commande R dans laquelle son code a été exécuté.

La méthode d'apprentissage préconisée dans cet ouvrage suppose que le lecteur utilisera cette seconde approche d'interaction avec R.

1.5 Éditeurs de texte

Dans la mesure où l'on a recours à des fichiers de script tel qu'expliqué à la section précédente, l'édition de code R bénéficie grandement d'un bon éditeur de texte pour programmeur. Dans certains cas, l'éditeur peut même réduire l'opération de copier-coller à un simple raccourci clavier.

- Un éditeur de texte est différent d'un traitement de texte en ce qu'il s'agit d'un logiciel destiné à la création, l'édition et la sauvegarde de fichiers textes purs, c'est-à-dire dépourvus d'information de présentation et de mise en forme. Les applications Bloc-notes sous Windows ou TextEdit sous OS X sont deux exemples d'éditeurs de texte simples.
- Un éditeur de texte pour programmeur saura en plus reconnaître la syntaxe d'un langage de programmation et assister à sa mise en forme : indentation automatique du code, coloration syntaxique, manipulation d'objets, etc.

Le lecteur peut utiliser l'éditeur de texte de son choix pour l'édition de code R. Certains éditeurs offrent simplement plus de fonctionnalités que d'autres.

```
## Fichier de script simple contenant des expressions R pour
## faire des calculs et créer des objets.
2 + 3

## Probabilité d'une loi de Poisson(10)
x <- 7
10^x * exp(-10) / factorial(x)

## Petite fonction qui fait un calcul trivial
f <- function(x) x^2

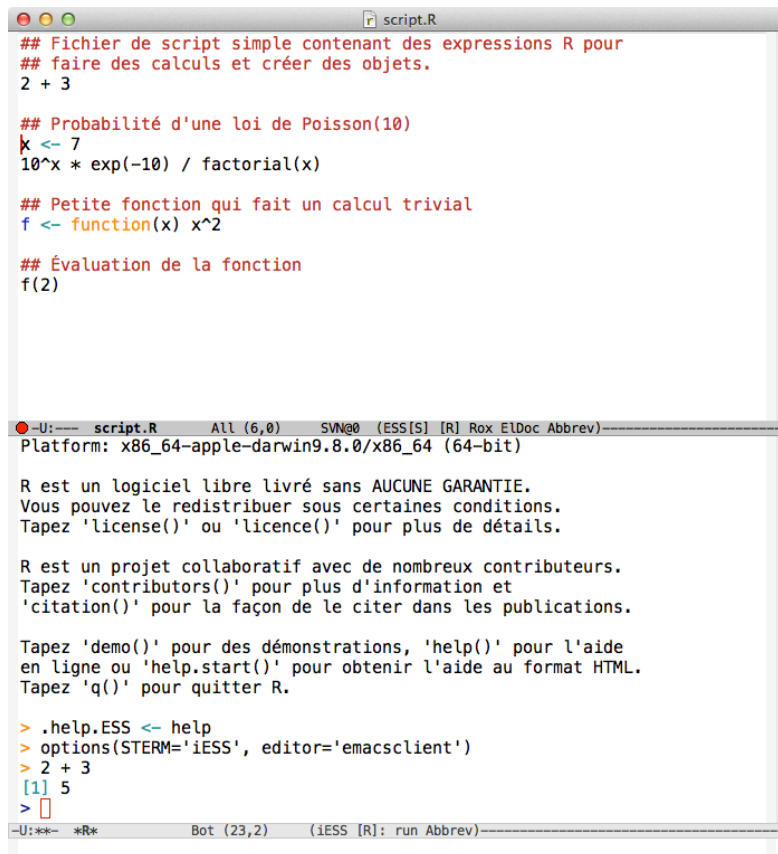
## Évaluation de la fonction
f(2)
```

```
R version 2.14.0 (2011-10-31)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

[...]

> ## Fichier de script simple contenant des expressions R pour
> ## faire des calculs et créer des objets.
> 2 + 3
[1] 5
>
> ## Probabilité d'une loi de Poisson(10)
> x <- 7
> 10^x * exp(-10) / factorial(x)
[1] 0.09007923
>
> ## Petite fonction qui fait un calcul trivial
> f <- function(x) x^2
>
> ## Évaluation de la fonction
> f(2)
[1] 4
```

FIG. 1.2: Fichier de script (en haut) et invite de commande R dans laquelle les expressions R ont été exécutées (en bas). Les lignes débutant par # dans le fichier de script sont des commentaires ignorés par l'interprète de commandes.



```

script.R
## Fichier de script simple contenant des expressions R pour
## faire des calculs et créer des objets.
2 + 3

## Probabilité d'une loi de Poisson(10)
k <- 7
10^x * exp(-10) / factorial(x)

## Petite fonction qui fait un calcul trivial
f <- function(x) x^2

## Évaluation de la fonction
f(2)

-U:--- script.R All (6,0) SVN@0 (ESS[S] [R] Rox EIDoc Abbrev)
Platform: x86_64-apple-darwin9.8.0/x86_64 (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.

> .help.ESS <- help
> options(STERM='iESS', editor='emacsclient')
> 2 + 3
[1] 5
> 
-U:*** *R* Bot (23,2) (iESS [R]: run Abbrev)

```

FIG. 1.3: Fenêtre de GNU Emacs sous OS X en mode d'édition de code R. Dans la partie du haut, on retrouve le fichier de script de la figure 1.2 et dans la partie du bas, l'invite de commandes R.

- GNU Emacs est un très ancien, mais aussi très puissant éditeur pour programmeur. À la question 6.2 de la foire aux questions de R (Hornik, 2011), «Devrais-je utiliser R à l'intérieur de Emacs?», la réponse est : «Oui, absolument.»

En effet, combiné avec le mode ESS (*Emacs Speaks Statistics*), Emacs offre un environnement de développement aussi riche qu'efficace. Entre autres fonctionnalités uniques à Emacs, le fichier de script et l'invite de commandes R sont regroupés dans la même fenêtre, comme on peut le voir à la figure 1.3.

Emblème du logiciel libre, Emacs est disponible gratuitement et à l'identique sur toutes les plateformes supportées par R, dont Windows, OS X et Linux.

- Consulter l'annexe A pour en savoir plus sur GNU Emacs et apprendre les com-

mandes essentielles pour y faire ses premiers pas.

- ▶ Malgré tous ses avantages (ou à cause de ceux-ci), Emacs est un logiciel difficile à apprivoiser, surtout pour les personnes moins à l'aise avec l'informatique.
- ▶ Il existe plusieurs autres options que Emacs pour éditer efficacement du code R — et le Bloc-notes de Windows n'en fait *pas* partie ! Nous recommandons plutôt :
 - sous Windows, l'éditeur Notepad++ muni de l'extension NppToR ([Redd, 2010](#)), tous deux des logiciels libres ;
 - toujours sous Windows, le logiciel WinEdt muni de l'extension libre R-WinEdt ([Ligges, 2003](#)) ;
 - sous OS X, tout simplement l'éditeur de texte très complet intégré à l'application R.app, ou alors l'éditeur de texte commercial TextMate (essai gratuit de 30 jours) ;
 - sous Linux, Vim et Kate semblent les choix les plus populaires après Emacs dans la communauté R.

1.6 Anatomie d'une session de travail

Dans ses grandes lignes, toute session de travail avec R se réduit aux étapes ci-dessous.

1. Ouvrir un fichier de script existant ou en créer un nouveau à l'aide de l'éditeur de texte de son choix.
2. Démarrer une session R en cliquant sur l'icône de l'application si l'on utilise une interface graphique, ou alors en suivant la procédure expliquée à l'annexe A si l'on utilise GNU Emacs.
3. Au cours de la phase de développement, on fera généralement de nombreux aller-retours la ligne de commande où l'on testera des commandes et le fichier de script où l'on consignera le code R que l'on souhaite sauvegarder et les commentaires qui nous permettront de s'y retrouver plus tard.
4. Sauvegarder son fichier de script et quitter l'éditeur.
5. Si nécessaire — et c'est rarement le cas — sauvegarder l'espace de travail de la session R avec `save.image()`. En fait, on ne voudra sauvegarder nos objets R que lorsque ceux-ci sont très longs à créer comme, par exemple, les résultats d'une simulation.
6. Quitter R en tapant `q()` à la ligne de commande ou en fermant l'interface graphique par la procédure usuelle. Encore ici, la manière de procéder est quelque peu différente dans GNU Emacs ; voir l'annexe A.

Évidemment, les étapes 1 et 2 sont interchangeables, tout comme les étapes 4, 5 et 6.

1.7 Répertoire de travail

Le répertoire de travail (*workspace*) de R est le dossier par défaut dans lequel le logiciel 1) va rechercher des fichiers de script ou de données ; et 2) va sauvegarder l'espace de travail dans le fichier `.RData`. Le dossier de travail est déterminé au lancement de R.

- ▶ Les interfaces graphiques démarrent avec un répertoire de travail par défaut. Pour le changer, utiliser l'entrée appropriée dans le menu Fichier (Windows) ou Divers (Mac OS X). Consulter aussi les foires aux questions spécifiques aux interfaces graphiques (Ripley et Murdoch, 2011; Iacus et collab., 2011) pour des détails additionnels sur la gestion des répertoires de travail.
- ▶ Avec GNU Emacs, la situation est un peu plus simple puisque l'on doit spécifier un répertoire de travail chaque fois que l'on démarre un processus R ; voir l'annexe A.

1.8 Consulter l'aide en ligne

Les rubriques d'aide des diverses fonctions disponibles dans R contiennent une foule d'informations ainsi que des exemples d'utilisation. Leur consultation est tout à fait essentielle.

- ▶ Pour consulter la rubrique d'aide de la fonction `foo`, on peut entrer à la ligne de commande

```
> ?foo
```

ou

```
> help(foo)
```

1.9 Où trouver de la documentation

La documentation officielle de R se compose de six guides accessibles depuis le menu Aide des interfaces graphiques ou encore en ligne dans le site du projet R³. Pour le débutant, seuls *An Introduction to R* et, possiblement, *R Data Import/Export* peuvent s'avérer des ressources utiles à court terme.

3. <http://www.r-project.org>

Plusieurs livres — en versions papier ou électronique, gratuits ou non — ont été publiés sur R. On en trouvera une liste exhaustive dans la section Documentation du site du projet R.

Depuis plusieurs années maintenant, les ouvrages de [Venables et Ripley \(2000, 2002\)](#) demeurent des références standards *de facto* sur les langages S et R. Plus récent, [Braun et Murdoch \(2007\)](#) participe du même effort que le présent ouvrage en se concentrant sur la programmation en R plutôt que sur ses applications statistiques.

1.10 Exemples

```
### Générer deux vecteurs de nombres pseudo-aléatoires issus
### d'une loi normale centrée réduite.
x <- rnorm(50)
y <- rnorm(x)

### Graphique des couples (x, y).
plot(x, y)

### Graphique d'une approximation de la densité du vecteur x.
plot(density(x))

### Générer la suite 1, 2, ..., 10.
1:10

### La fonction 'seq' sert à générer des suites plus générales.
seq(from = -5, to = 10, by = 3)
seq(from = -5, length = 10)

### La fonction 'rep' sert à répéter des valeurs.
rep(1, 5)          # répéter 1 cinq fois
rep(1:5, 5)        # répéter le vecteur 1,...,5 cinq fois
rep(1:5, each = 5) # répéter chaque élément du vecteur cinq fois

### Arithmétique vectorielle.
v <- 1:12          # initialisation d'un vecteur
v + 2              # additionner 2 à chaque élément de v
v * -12:-1         # produit élément par élément
v + 1:3            # le vecteur le plus court est recyclé

### Vecteur de nombres uniformes sur l'intervalle [1, 10].
v <- runif(12, min = 1, max = 10)
v
```

```
### Pour afficher le résultat d'une affectation, placer la
### commande entre parenthèses.
( v <- runif(12, min = 1, max = 10) )

### Arrondi des valeurs de v à l'entier près.
( v <- round(v) )

### Créer une matrice 3 x 4 à partir des valeurs de
### v. Remarquer que la matrice est remplie par colonne.
( m <- matrix(v, nrow = 3, ncol = 4) )

### Les opérateurs arithmétiques de base s'appliquent aux
### matrices comme aux vecteurs.
m + 2
m * 3
m ^ 2

### Éliminer la quatrième colonne afin d'obtenir une matrice
### carrée.
( m <- m[, -4] )

### Transposée et inverse de la matrice m.
t(m)
solve(m)

### Produit matriciel.
m %*% m          # produit de m avec elle-même
m %*% solve(m)   # produit de m avec son inverse
round(m %*% solve(m)) # l'arrondi donne la matrice identité

### Consulter la rubrique d'aide de la fonction 'solve'.
?solve

### Liste des objets dans l'espace de travail.
ls()

### Nettoyage.
rm(x, y, v, m)
```

1.11 Exercices

- 1.1 Démarrer une session R et entrer une à une les expressions ci-dessous à la ligne de commande. Observer les résultats.

```
> ls()
> pi
> (v <- c(1, 5, 8))
> v * 2
> x <- v + c(2, 1, 7)
> x
> ls()
> q()
```

- 1.2 Ouvrir dans un éditeur de texte le fichier de script contenant le code de la section précédente. Exécuter le code ligne par ligne et observer les résultats. Répéter l'exercice avec un ou deux autres éditeurs de texte afin de les comparer et de vous permettre d'en choisir un pour la suite.
- 1.3 Consulter les rubriques d'aide d'une ou plusieurs des fonctions rencontrées lors de l'exercice précédent. Observer d'abord comment les rubriques d'aide sont structurées — elles sont toutes identiques — puis exécuter quelques expressions tirées des sections d'exemples.
- 1.4 Exécuter le code de l'exemple de session de travail R que l'on trouve à l'annexe A de [Venables et collab. \(2011\)](#). En plus d'aider à se familiariser avec R, cet exercice permet de découvrir les fonctionnalités du logiciel en tant qu'outil statistique.

2 Bases du langage R

Objectifs du chapitre

- ▶ Connaître la syntaxe et la sémantique du langage R.
- ▶ Comprendre la notion d'objet et connaître les principaux types de données dans R.
- ▶ Comprendre et savoir tirer profit de l'arithmétique vectorielle de R.
- ▶ Comprendre la différence entre les divers modes d'objets R (en particulier `numeric`, `character` et `logical`) et la conversion automatique de l'un à l'autre.
- ▶ Comprendre la différence entre un vecteur, une matrice, un tableau, une liste et un *data frame* et savoir créer ces divers types d'objets.
- ▶ Savoir extraire des données d'un objet ou y affecter de nouvelles valeurs à l'aide des diverses méthodes d'indiciage.

Avant de pouvoir utiliser un langage de programmation, il faut en connaître la syntaxe et la sémantique, du moins dans leurs grandes lignes. C'est dans cet esprit que ce chapitre introduit des notions de base du langage R telles que l'expression, l'affectation et l'objet. Le concept de vecteur se trouvant au cœur du langage, le chapitre fait une large place à la création et à la manipulation des vecteurs et autres types d'objets de stockage couramment employés en programmation en R.

2.1 Commandes R

Tel que vu au chapitre précédent, l'utilisateur de R interagit avec l'interprète R en entrant des commandes à l'invite de commande. Toute commande R est soit une *expression*, soit une *affectation*.

- ▶ Normalement, une expression est immédiatement évaluée et le résultat est affiché à l'écran :

```
> 2 + 3
```

```
[1] 5
> pi
[1] 3.141593
> cos(pi/4)
[1] 0.7071068
```

- Lors d'une affectation, une expression est évaluée, mais le résultat est stocké dans un objet (variable) et rien n'est affiché à l'écran. Le symbole d'affectation est `<-`, c'est-à-dire les deux caractères `<` et `-` placés obligatoirement l'un à la suite de l'autre :

```
> a <- 5
> a
[1] 5
> b <- a
> b
[1] 5
```

- Pour affecter le résultat d'un calcul dans un objet et simultanément afficher ce résultat, il suffit de placer l'affectation entre parenthèses pour ainsi créer une nouvelle expression¹ :

```
> (a <- 2 + 3)
[1] 5
```

- Le symbole d'affectation inversé `->` existe aussi, mais il est rarement utilisé.
- Éviter d'utiliser l'opérateur `=` pour affecter une valeur à une variable puisque cette pratique est susceptible d'engendrer de la confusion avec les constructions `nom = valeur` dans les appels de fonction.

Astuce. Dans les anciennes versions de S et R, l'on pouvait affecter avec le caractère de soulignement `<_>`. C'est l'emploi n'est plus permis, mais la pratique subsiste dans le mode ESS de Emacs. Ainsi, taper le caractère `<_>` hors d'une chaîne de caractères dans Emacs génère automatiquement `<_<-_>`. Si l'on souhaite véritablement obtenir le caractère de soulignement, appuyer deux fois successives sur `<_>`.

Que ce soit dans les fichiers de script ou à la ligne de commande, on sépare les commandes R les unes des autres par un point-virgule ou par un retour à la ligne.

1. En fait, cela devient un appel à l'opérateur `" ("` qui ne fait que retourner son argument.

- ▶ On considère généralement comme une mauvaise pratique d'employer les deux, c'est-à-dire de placer des points-virgules à la fin de chaque ligne de code, surtout dans les fichiers de script.
- ▶ Le point-virgule peut être utile pour séparer deux courtes expressions ou plus sur une même ligne :

```
> a <- 5; a + 2  
[1] 7
```

C'est le seul emploi du point-virgule que l'on rencontrera dans cet ouvrage.

On peut regrouper plusieurs commandes en une seule expression en les entourant d'accolades { }.

- ▶ Le résultat du regroupement est la valeur de la dernière commande :

```
> {  
+   a <- 2 + 3  
+   b <- a  
+   b  
+ }  
[1] 5
```

- ▶ Par conséquent, si le regroupement se termine par une assignation, aucune valeur n'est retournée ni affichée à l'écran :

```
> {  
+   a <- 2 + 3  
+   b <- a  
+ }
```

- ▶ Les règles ci-dessus joueront un rôle important dans la composition de fonctions ; voir le chapitre ??.
- ▶ Comme on peut le voir ci-dessus, lorsqu'une commande n'est pas complète à la fin de la ligne, l'invite de commande de R change de >_ à +_ pour nous inciter à compléter notre commande.

2.2 Conventions pour les noms d'objets

Les caractères permis pour les noms d'objets sont les lettres minuscules a–z et majuscules A–Z, les chiffres 0–9, le point «.» et le caractère de soulignement «_». Selon l'environnement linguistique de l'ordinateur, il peut être permis d'utiliser des lettres accentuées, mais cette pratique est fortement découragée puisqu'elle risque de nuire à la portabilité du code.

- Les noms d'objets ne peuvent commencer par un chiffre. S'ils commencent par un point, le second caractère ne peut être un chiffre.
- Le R est sensible à la casse, ce qui signifie que `foo`, `Foo` et `F00` sont trois objets distincts. Un moyen simple d'éviter des erreurs liées à la casse consiste à n'employer que des lettres minuscules.
- Certains noms sont utilisés par le système, aussi vaut-il mieux éviter de les utiliser. En particulier, éviter d'utiliser

`c, q, t, C, D, I, diff, length, mean, pi, range, var.`

- Certains mots sont réservés pour le système et il est interdit de les utiliser comme nom d'objet. Les mots réservés sont :

`break, else, for, function, if, in, next, repeat, return, while,`
`TRUE, FALSE,`
`Inf, NA, NaN, NULL,`
`NA_integer_, NA_real_, NA_complex_, NA_character_,`
`..., ..1, ..2, etc.`

- Les variables `T` et `F` prennent par défaut les valeurs `TRUE` et `FALSE`, respectivement, mais peuvent être réaffectées :

```
> T
```

```
[1] TRUE
```

```
> TRUE <- 3
```

```
Error in TRUE <- 3 : membre gauche de l'assignation (do_set) incorrect
```

```
> (T <- 3)
```

```
[1] 3
```

- Nous recommandons de toujours écrire les valeurs booléennes `TRUE` et `FALSE` au long pour éviter des bogues difficiles à détecter.

2.3 Les objets R

Tout dans le langage R est un objet : les variables contenant des données, les fonctions, les opérateurs, même le symbole représentant le nom d'un objet est lui-même un objet. Les objets possèdent au minimum un *mode* et une *longueur* et certains peuvent être dotés d'un ou plusieurs *attributs*

- Le mode d'un objet est obtenu avec la fonction `mode` :

Mode	Contenu de l'objet
numeric	nombres réels
complex	nombres complexes
logical	valeurs booléennes (vrai/faux)
character	chaînes de caractères
function	fonction
list	données quelconques
expression	expressions non évaluées

TAB. 2.1: Modes disponibles et contenus correspondants

```
> v <- c(1, 2, 5, 9)
> mode(v)

[1] "numeric"
```

- La longueur d'un objet est obtenue avec la fonction `length` :

```
> length(v)

[1] 4
```

2.3.1 Modes et types de données

Le mode prescrit ce qu'un objet peut contenir. À ce titre, un objet ne peut avoir qu'un seul mode. Le tableau 2.1 contient la liste des principaux modes disponibles en R. À chacun de ces modes correspond une fonction du même nom servant à créer un objet de ce mode.

- Les objets de mode "numeric", "complex", "logical" et "character" sont des objets *simples* (*atomic* en anglais) qui ne peuvent contenir que des données d'un seul type.
- En revanche, les objets de mode "list" ou "expression" sont des objets *récur-sifs* qui peuvent contenir d'autres objets. Par exemple, une liste peut contenir une ou plusieurs autres listes ; voir la section 2.6 pour plus de détails.
- La fonction `typeof` permet d'obtenir une description plus précise de la représentation interne d'un objet (c'est-à-dire au niveau de la mise en œuvre en C). Le mode et le type d'un objet sont souvent identiques.

2.3.2 Longueur

La longueur d'un objet est égale au nombre d'éléments qu'il contient.

- ▶ La longueur, au sens R du terme, d'une chaîne de caractères est toujours 1. Un objet de mode `character` doit contenir plusieurs chaînes de caractères pour que sa longueur soit supérieure à 1 :

```
> v1 <- "actuariat"
> length(v1)
[1] 1
> v2 <- c("a", "c", "t", "u", "a", "r", "i", "a", "t")
> length(v2)
[1] 9
```

- ▶ On obtient le nombre de caractères dans un chaîne avec la fonction `nchar` :

```
> nchar(v1)
[1] 9
> nchar(v2)
[1] 1 1 1 1 1 1 1 1 1
```

- ▶ Un objet peut être de longueur 0 et doit alors être interprété comme un contenant qui existe, mais qui est vide :

```
> v <- numeric(0)
> length(v)
[1] 0
```

2.3.3 L'objet spécial NULL

L'objet spécial `NULL` représente «rien», ou le vide.

- ▶ Son mode est `NULL`.
- ▶ Sa longueur est 0.
- ▶ Toutefois différent d'un objet vide :
 - un objet de longueur 0 est un contenant vide ;
 - `NULL` est «pas de contenant».
- ▶ La fonction `is.null` teste si un objet est `NULL` ou non.

2.3.4 Valeurs manquantes, indéterminées et infinies

Dans les applications statistiques, il est souvent utile de pouvoir représenter des données manquantes. Dans R, l'objet spécial NA remplit ce rôle.

- ▶ Par défaut, le mode de NA est `logical`, mais NA ne peut être considéré ni comme `TRUE`, ni comme `FALSE`.
- ▶ Toute opération impliquant une donnée NA a comme résultat NA.
- ▶ Certaines fonctions (`sum`, `mean`, par exemple) ont par conséquent un argument `na.rm` qui, lorsque `TRUE`, élimine les données manquantes avant de faire un calcul.
- ▶ La valeur NA n'est égale à aucune autre, pas même elle-même (selon la règle ci-dessus, le résultat de la comparaison est NA) :

```
> NA == NA
[1] NA
```

- ▶ Par conséquent, pour tester si les éléments d'un objet sont NA ou non il faut utiliser la fonction `is.na` :

```
> is.na(NA)
[1] TRUE
```

La norme IEEE 754 régissant la représentation interne des nombres dans un ordinateur (IEEE, 2003) prévoit les valeurs mathématiques spéciales $+\infty$ et $-\infty$ ainsi que les formes indéterminées du type $\frac{0}{0}$ ou $\text{inf}-\text{inf}$. R dispose d'objets spéciaux pour représenter ces valeurs.

- ▶ Inf représente $+\infty$.
- ▶ -Inf représente $-\infty$.
- ▶ NaN (*Not a Number*) représente une forme indéterminée.
- ▶ Ces valeurs sont testées avec les fonctions `is.infinite`, `is.finite` et `is.nan`.

2.3.5 Attributs

Les attributs d'un objet sont des éléments d'information additionnels liés à cet objet. La liste des attributs les plus fréquemment rencontrés se trouve au tableau 2.2. Pour chaque attribut, il existe une fonction du même nom servant à extraire l'attribut correspondant d'un objet.

- ▶ Plus généralement, la fonction `attributes` permet d'extraire ou de modifier la liste des attributs d'un objet. On peut aussi travailler sur un seul attribut à la fois avec la fonction `attr`.

Attribut	Utilisation
<code>class</code>	affecte le comportement d'un objet
<code>dim</code>	dimensions des matrices et tableaux
<code>dimnames</code>	étiquettes des dimensions des matrices et tableaux
<code>names</code>	étiquettes des éléments d'un objet

TAB. 2.2: Attributs les plus usuels d'un objet

- On peut ajouter à peu près ce que l'on veut à la liste des attributs d'un objet. Par exemple, on pourrait vouloir attacher au résultat d'un calcul la méthode de calcul utilisée :

```
> x <- 3
> attr(x, "methode") <- "au pif"
> attributes(x)
$methode
[1] "au pif"
```

- Extraire un attribut qui n'existe pas retourne NULL :

```
> dim(x)
NULL
```

- À l'inverse, donner à un attribut la valeur NULL efface cet attribut :

```
> attr(x, "methode") <- NULL
> attributes(x)
NULL
```

2.4 Vecteurs

En R, à toutes fins pratiques, *tout* est un vecteur. Contrairement à certains autres langages de programmation, il n'y a pas de notion de scalaire en R ; un scalaire est simplement un vecteur de longueur 1. Comme nous le verrons au chapitre 3, le vecteur est l'unité de base dans les calculs.

- Dans un vecteur simple, tous les éléments doivent être du même mode. Nous nous restreignons à ce type de vecteurs pour le moment.
- Les fonctions de base pour créer des vecteurs sont :
 - `c` (concaténation) ;

- `numeric` (vecteur de mode `numeric`);
 - `logical` (vecteur de mode `logical`);
 - `character` (vecteur de mode `character`).
- Il est possible (et souvent souhaitable) de donner une étiquette à chacun des éléments d'un vecteur.

```
> (v <- c(a = 1, b = 2, c = 5))
a b c
1 2 5
> v <- c(1, 2, 5)
> names(v) <- c("a", "b", "c")
> v
a b c
1 2 5
```

Ces étiquettes font alors partie des attributs du vecteur.

- L'indixage dans un vecteur se fait avec `[]`. On peut extraire un élément d'un vecteur par sa position ou par son étiquette, si elle existe (auquel cas cette approche est beaucoup plus sûre).

```
> v[3]
c
5
> v["c"]
c
5
```

La section 2.8 traite plus en détail de l'indixage des vecteurs et des matrices.

2.5 Matrices et tableaux

Le R étant un langage spécialisé pour les calculs mathématiques, il supporte tout naturellement et de manière intuitive — à une exception près, comme nous le verrons — les matrices et, plus généralement, les tableaux à plusieurs dimensions.

Les matrices et tableaux ne sont rien d'autre que des vecteurs dotés d'un attribut `dim`. Ces objets sont donc stockés, et peuvent être manipulés, exactement comme des vecteurs simples.

- Une matrice est un vecteur avec un attribut `dim` de longueur 2. Cela change implicitement la classe de l'objet pour `"matrix"` et, de ce fait, le mode d'affichage de l'objet ainsi que son interaction avec plusieurs opérateurs et fonctions.

- La fonction de base pour créer des matrices est `matrix` :

```
> matrix(1:6, nrow = 2, ncol = 3)
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- La généralisation d'une matrice à plus de deux dimensions est un tableau (*array*). Le nombre de dimensions du tableau est toujours égal à la longueur de l'attribut `dim`. La classe implicite d'un tableau est "array".
- La fonction de base pour créer des tableaux est `array` :

```
> array(1:24, dim = c(3, 4, 2))
, , 1

      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

      [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```



On remarquera ci-dessus que les matrices et tableaux sont remplis en faisant d'abord varier la première dimension, puis la seconde, etc. Pour les matrices, cela revient à remplir par colonne. On conviendra que cette convention, héritée du Fortran, n'est pas des plus intuitives.

La fonction `matrix` a un argument `byrow` qui permet d'inverser l'ordre de remplissage, mais il vaut mieux s'habituer à la convention de R que d'essayer constamment de la contourner.

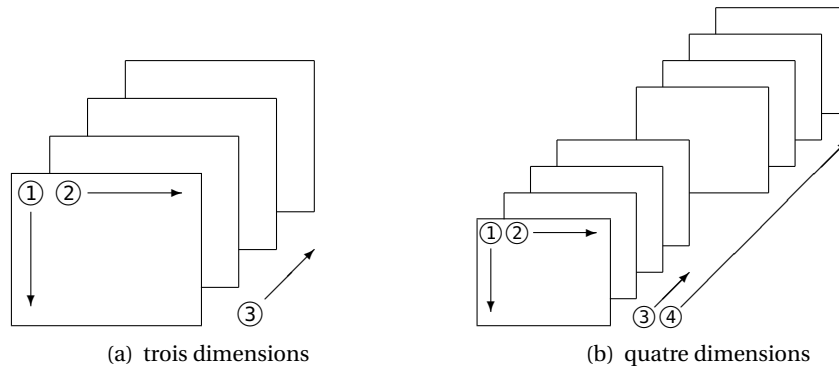


FIG. 2.1: Représentation schématique de tableaux. Les chiffres encadrés identifient l'ordre de remplissage.

L'ordre de remplissage inhabituel des tableaux rend leur manipulation difficile si on ne les visualise pas correctement. Imaginons un tableau de dimensions $3 \times 4 \times 5$.

- Il faut voir le tableau comme cinq matrices 3×4 (remplies par colonne !) les unes *derrière* les autres.
- Autrement dit, le tableau est un prisme rectangulaire haut de 3 unités, large de 4 et profond de 5.
- Si l'on ajoute une quatrième dimension, cela revient à aligner des prismes les uns derrière les autres, et ainsi de suite.

La figure 2.1 fournit une représentation schématique des tableaux à trois et quatre dimensions.

Comme pour les vecteurs, l'indéçage des matrices et tableaux se fait avec `[]`.

- On extrait un élément d'une matrice en précisant ses positions dans chaque dimension de celle-ci, séparées par des virgules :

```
> (m <- matrix(c(40, 80, 45, 21, 55, 32), nrow = 2, ncol = 3))
      [,1] [,2] [,3]
[1,]   40   45   55
[2,]   80   21   32
> m[1, 2]
[1] 45
```

- On peut aussi ne donner que la position de l'élément dans le vecteur sous-jacent :

```
> m[3]
```

```
[1] 45
```

- Lorsqu'une dimension est omise dans les crochets, tous les éléments de cette dimension sont extraits :

```
> m[2, ]
```

```
[1] 80 21 32
```

- Les idées sont les mêmes pour les tableaux.
- Pour le reste, les règles d'indilage de vecteurs exposées à la section 2.8 s'appliquent à chaque position de l'indice d'une matrice ou d'un tableau.

Des fonctions permettent de fusionner des matrices et des tableaux ayant au moins une dimension identique.

- La fonction `rbind` permet de fusionner verticalement deux matrices (ou plus) ayant le même nombre de colonnes.

```
> n <- matrix(1:9, nrow = 3)
```

```
> rbind(m, n)
```

```
      [,1] [,2] [,3]
[1,]   40   45   55
[2,]   80   21   32
[3,]    1    4    7
[4,]    2    5    8
[5,]    3    6    9
```

- La fonction `cbind` permet de fusionner horizontalement deux matrices (ou plus) ayant le même nombre de lignes.

```
> n <- matrix(1:4, nrow = 2)
```

```
> cbind(m, n)
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]   40   45   55    1    3
[2,]   80   21   32    2    4
```

2.6 Listes

La liste est le mode de stockage le plus général et polyvalent du langage R. Il s'agit d'un type de vecteur spécial dont les éléments peuvent être de n'importe quel mode, y compris le mode `list`. Cela permet donc d'emboîter des listes, d'où le qualificatif de *récuratif* pour ce type d'objet.

- La fonction de base pour créer des listes est `list` :

```
> (x <- list(size = c(1, 5, 2), user = "Joe", new = TRUE))  
$size  
[1] 1 5 2  
  
$user  
[1] "Joe"  
  
$new  
[1] TRUE
```

Dans l'exemple ci-dessus, le premier élément de la liste est de mode "numeric", le second de mode "character" et le troisième de mode "logical".

- Nous recommandons de nommer les éléments d'une liste. En effet, les listes contiennent souvent des données de divers type et il peut s'avérer difficile d'identifier les éléments s'ils ne sont pas nommés. De plus, comme nous le verrons ci-dessous, il est très simple d'extraire les éléments d'une liste par leur étiquette.
- La liste demeure un vecteur. On peut donc l'indicer avec l'opérateur `[]`. Cependant, cela retourne une liste contenant le ou les éléments indicés. C'est rarement ce que l'on souhaite.
- Pour indicer un élément d'une liste et n'obtenir que cet élément, et non une liste contenant l'élément, il faut utiliser l'opérateur d'indilage `[[]]`. Comparer

```
> x[1]  
$size  
[1] 1 5 2  
  
et  
  
> x[[1]]  
[1] 1 5 2
```

- Évidemment, on ne peut extraire qu'un seul élément à la fois avec les crochets doubles `[[]]`.
- Petite subtilité peu employée, mais élégante. Si l'indice utilisé dans `[[]]` est un vecteur, il est utilisé récursivement pour indicer la liste : cela sélectionnera la composante de la liste correspondant au premier élément du vecteur, puis l'élément de la composante correspondant au second élément du vecteur, et ainsi de suite.

- Une autre — et, en fait, la meilleure — façon d'indicer un seul élément d'une liste est par le biais de l'opérateur `$`, avec une construction `x$etiquette` :

```
> x$size  
[1] 1 5 2
```

- La fonction `unlist` convertit une liste en un vecteur simple. Elle est surtout utile pour concaténer les éléments d'une liste lorsque ceux-ci sont des scalaires. Attention, cette fonction peut être destructrice si la structure interne de la liste est importante.

2.7 Data frames

Les vecteurs, les matrices, les tableaux et les listes sont les types d'objets les plus fréquemment utilisés en programmation en R. Toutefois, un grand nombre de procédures statistiques — pensons à la régression linéaire, par exemple — repose davantage sur les *data frames* pour le stockage des données.

- Un *data frame* est une liste de classe "data.frame" dont tous les éléments sont de la même longueur (ou comptent le même nombre de lignes si les éléments sont des matrices).
- Il est généralement représenté sous la forme d'un tableau à deux dimensions. Chaque élément de la liste sous-jacente correspond à une colonne.
- Bien que visuellement similaire à une matrice un *data frame* est plus général puisque les colonnes peuvent être de modes différents ; pensons à un tableau avec des noms (mode `character`) dans une colonne et des notes (mode `numeric`) dans une autre.
- On crée un *data frame* avec la fonction `data.frame` ou, pour convertir un autre type d'objet en *data frame*, avec `as.data.frame`.
- Le *data frame* peut être indicé à la fois comme une liste et comme une matrice.
- Les fonctions `rbind` et `cbind` peuvent être utilisées pour ajouter des lignes ou des colonnes à un *data frame*.
- On peut rendre les colonnes d'un *data frame* (ou d'une liste) visibles dans l'espace de travail avec la fonction `attach`, puis les masquer avec `detach`.

2.8 Indichage

L'indichage des vecteurs et matrices a déjà été brièvement présenté aux sections 2.4 et 2.5. La présente section contient plus de détails sur cette procédure

des plus communes lors de l'utilisation du langage R. On se concentre toutefois sur le traitement des vecteurs.

L'indicage sert principalement à deux choses : extraire des éléments d'un objet avec la construction `x[i]` ou les remplacer avec la construction `x[i] <- y`.

- ▶ Il est utile de savoir que ces opérations sont en fait traduites par l'interprète R en des appels à des fonctions nommées `[]` et `[]<-`, dans l'ordre.
- ▶ De même, les opérations d'extraction et de remplacement d'un élément d'une liste de la forme `x$etiquette` et `x$etiquette <- y` correspondent à des appels aux fonctions `$` et `$<-`.

Il existe quatre façons d'indicer un vecteur dans le langage R. Dans tous les cas, l'indicage se fait à l'intérieur de crochets `[]`.

1. Avec un vecteur d'entiers positifs. Les éléments se trouvant aux positions correspondant aux entiers sont extraits du vecteur, dans l'ordre. C'est la technique la plus courante :

```
> x <- c(A = 2, B = 4, C = -1, D = -5, E = 8)
> x[c(1, 3)]
  A  C
2 -1
```

2. Avec un vecteur d'entiers négatifs. Les éléments se trouvant aux positions correspondant aux entiers négatifs sont alors *éliminés* du vecteur :

```
> x[c(-2, -3)]
  A  D  E
2 -5  8
```

3. Avec un vecteur booléen. Le vecteur d'indicage doit alors être de la même longueur que le vecteur indicé. Les éléments correspondant à une valeur TRUE sont extraits du vecteur, alors que ceux correspondant à FALSE sont éliminés :

```
> x > 0
      A      B      C      D      E
TRUE TRUE FALSE FALSE TRUE
> x[x > 0]
  A  B  E
2  4  8
```

4. Avec un vecteur de chaînes de caractères. Utile pour extraire les éléments d'un vecteur à condition que ceux-ci soient nommés :

```
> x[c("B", "D")]
  B  D
4 -5
```

5. L'indice est laissé vide. Tous les éléments du vecteur sont alors sélectionnés :

```
> x[]
  A  B  C  D  E
2  4 -1 -5  8
```

Remarquer que cela est différent d'indicer avec un vecteur vide ; cette opération retourne un vecteur vide.

2.9 Exemples

```
###
### COMMANDES R
###

## Les expressions entrées à la ligne de commande sont
## immédiatement évaluées et le résultat est affiché à
## l'écran, comme avec une grosse calculatrice.
1                # une constante
(2 + 3 * 5)/7    # priorité des opérations
3^5              # puissance
exp(3)           # fonction exponentielle
sin(pi/2) + cos(pi/2) # fonctions trigonométriques
gamma(5)         # fonction gamma

## Lorsqu'une expression est syntaxiquement incomplète,
## l'invite de commande change de '>' à '+'.
2 -              # expression incomplète
5 *              # toujours incomplète
3                # complétée

## Taper le nom d'un objet affiche son contenu. Pour une
## fonction, c'est son code source qui est affiché.
pi                # constante numérique intégrée
letters           # chaîne de caractères intégrée
LETTERS           # version en majuscules
matrix            # fonction

## Ne pas utiliser '=' pour l'affectation. Les opérateurs
```



```
## d'affectation standard en R sont '<-' et '->'.
x <- 5                # affecter 5 à l'objet 'x'
5 -> x                # idem, mais peu usité
x                     # voir le contenu
(x <- 5)              # affecter et afficher
y <- x                # affecter la valeur de 'x' à 'y'
x <- y <- 5           # idem, en une seule expression
y                     # 5
x <- 0                # changer la valeur de 'x'...
y                     # ... ne change pas celle de 'y'

## Pour regrouper plusieurs expressions en une seule commande,
## il faut soit les séparer par un point-virgule ';', soit les
## regrouper à l'intérieur d'accolades { } et les séparer par
## des retours à la ligne.
x <- 5; y <- 2; x + y  # compact; éviter dans les scripts
x <- 5;                # éviter les ';' superflus
{                      # début d'un groupe
  x <- 5                # première expression du groupe
  y <- 2                # seconde expression du groupe
  x + y                # résultat du groupe
}                      # fin du groupe et résultat
{x <- 5; y <- 2; x + y} # valide, mais redondant

###
### NOMS D'OBJETS
###

## Quelques exemples de noms valides et invalides.
foo <- 5               # valide
foo.123 <- 5           # valide
foo_123 <- 5           # valide
123foo <- 5            # invalide; commence par un chiffre
.foo <- 5              # valide
.123foo <- 5           # invalide; point suivi d'un chiffre

## Liste des objets dans l'espace de travail. Les objets dont
## le nom commence par un point sont considérés cachés.
ls()                   # l'objet '.foo' n'est pas affiché
ls(all.names = TRUE)   # objets cachés aussi affichés

## R est sensible à la casse
foo <- 1
Foo
FOO
```

```
###
### LES OBJETS R
###

## MODES ET TYPES DE DONNÉES

## Le mode d'un objet détermine ce qu'il peut contenir. Les
## vecteurs simples ("atomic") contiennent des données d'un
## seul type.
mode(c(1, 4.1, pi))      # nombres réels
mode(c(2, 1 + 5i))       # nombres complexes
mode(c(TRUE, FALSE, TRUE)) # valeurs booléennes
mode("foobar")           # chaînes de caractères

## Si l'on mélange dans un même vecteur des objets de mode
## différents, il y a conversion automatique vers le mode pour
## lequel il y a le moins de perte d'information, c'est-à-dire
## vers le mode qui permet le mieux de retrouver la valeur
## originale des éléments.
c(5, TRUE, FALSE)        # conversion en mode 'numeric'
c(5, "z")                 # conversion en mode 'character'
c(TRUE, "z")              # conversion en mode 'character'
c(5, TRUE, "z")           # conversion en mode 'character'

## La plupart des autres types d'objets sont récursifs. Voici
## quelques autres modes.
mode(seq)                 # une fonction
mode(list(5, "foo", TRUE)) # une liste
mode(expression(x <- 5))  # une expression non évaluée

## LONGUEUR

## La longueur d'un vecteur est égale au nombre d'éléments
## dans le vecteur.
(x <- 1:4)
length(x)

## Une chaîne de caractères ne compte que pour un seul
## élément.
(x <- "foobar")
length(x)

## Pour obtenir la longueur de la chaîne, il faut utiliser
## nchar().
```

```
nchar(x)

## Un objet peut néanmoins contenir plusieurs chaînes de
## caractères.
(x <- c("f", "o", "o", "b", "a", "r"))
length(x)

## La longueur peut être 0, auquel cas on a un objet vide,
## mais qui existe.
(x <- numeric(0))      # création du contenant
length(x)              # l'objet 'x' existe...
x[1] <- 1               # possible, 'x' existe
X[1] <- 1               # impossible, 'X' n'existe pas

## L'OBJET SPECIAL 'NULL'
mode(NULL)             # le mode de 'NULL' est NULL
length(NULL)           # longueur nulle
x <- c(NULL, NULL)      # s'utilise comme un objet normal
x; length(x); mode(x)   # mais donne toujours le vide

## L'OBJET SPÉCIAL 'NA'
x <- c(65, NA, 72, 88)  # traité comme une valeur
x + 2                  # tout calcul avec 'NA' donne NA
mean(x)                # voilà qui est pire
mean(x, na.rm = TRUE)  # éliminer les 'NA' avant le calcul
is.na(x)                # tester si les données sont 'NA'

## VALEURS INFINIES ET INDÉTERMINÉES
1/0                    # +infini
-1/0                   # -infini
0/0                    # indétermination
x <- c(65, Inf, NaN, 88) # s'utilisent comme des valeurs
is.finite(x)           # quels sont les nombres réels?
is.nan(x)              # lesquels ne sont «pas un nombre»?

## ATTRIBUTS

## Les objets peuvent être dotés d'un ou plusieurs attributs.
data(cars)             # jeu de données intégré
attributes(cars)        # liste de tous les attributs
attr(cars, "class")     # extraction d'un seul attribut

## Attribut 'class'. Selon la classe d'un objet, certaines
## fonctions (dites «fonctions génériques») vont se comporter
## différemment.
```



```
## La fonction 'vector' sert à initialiser des vecteurs avec
## des valeurs prédéterminées. Elle compte deux arguments: le
## mode du vecteur et sa longueur. Les fonctions 'numeric',
## 'logical', 'complex' et 'character' constituent des
## raccourcis pour des appels à 'vector'.
```

```
vector("numeric", 5)      # vecteur initialisé avec des 0
numeric(5)                # équivalent
numeric                   # en effet, voici la fonction
logical(5)                # initialisé avec FALSE
complex(5)                # initialisé avec 0 + 0i
character(5)              # initialisé avec chaînes vides
```

```
###
```

```
### MATRICES ET TABLEAUX
```

```
###
```

```
## Une matrice est un vecteur avec un attribut 'dim' de
## longueur 2 une classe implicite "matrix". La manière
## naturelle de créer une matrice est avec la fonction
## 'matrix'.
```

```
(x <- matrix(1:12, nrow = 3, ncol = 4)) # créer la matrice
length(x)                             # 'x' est un vecteur...
dim(x)                                 # ... avec un attribut 'dim'...
class(x)                               # ... et classe implicite "matrix"
```

```
## Une manière moins naturelle mais équivalente --- et parfois
## plus pratique --- de créer une matrice consiste à ajouter
## un attribut 'dim' à un vecteur.
```

```
x <- 1:12                             # vecteur simple
dim(x) <- c(3, 4)                     # ajout d'un attribut 'dim'
x; class(x)                           # 'x' est une matrice!
```

```
## Les matrices sont remplies par colonne par défaut. Utiliser
## l'option 'byrow' pour remplir par ligne.
```

```
matrix(1:12, nrow = 3, byrow = TRUE)
```

```
## Indicer la matrice ou le vecteur sous-jacent est
## équivalent. Utiliser l'approche la plus simple selon le
## contexte.
```

```
x[1, 3]                               # l'élément en position (1, 3)...
x[7]                                   # ... est le 7e élément du vecteur
x[1, ]                                # première ligne
x[, 2]                                # deuxième colonne
nrow(x)                               # nombre de lignes
```

```

dim(x)[1]          # idem
ncol(x)            # nombre de colonnes
dim(x)[2]          # idem

## Fusion de matrices et vecteurs.
x <- matrix(1:12, 3, 4)  # 'x' est une matrice 3 x 4
y <- matrix(1:8, 2, 4)   # 'y' est une matrice 2 x 4
z <- matrix(1:6, 3, 2)   # 'z' est une matrice 3 x 2
rbind(x, 1:4)           # ajout d'une ligne à 'x'
rbind(x, y)             # fusion verticale de 'x' et 'y'
cbind(x, 1:3)           # ajout d'une colonne à 'x'
cbind(x, z)             # concaténation de 'x' et 'z'
rbind(x, z)             # dimensions incompatibles
cbind(x, y)             # dimensions incompatibles

## Les vecteurs ligne et colonne sont rarement nécessaires. On
## peut les créer avec les fonctions 'rbind' et 'cbind',
## respectivement.
rbind(1:3)             # un vecteur ligne
cbind(1:3)             # un vecteur colonne

## Un tableau (array) est un vecteur avec un attribut 'dim' de
## longueur supérieure à 2 et une classe implicite "array".
## Quant au reste, la manipulation des tableaux est en tous
## points identique à celle des matrices. Ne pas oublier:
## les tableaux sont remplis de la première dimension à la
## dernière!
x <- array(1:60, 3:5)    # tableau 3 x 4 x 5
length(x)             # 'x' est un vecteur...
dim(x)                # ... avec un attribut 'dim'...
class(x)              # ... une classe implicite "array"
x[1, 3, 2]            # l'élément en position (1, 3, 2)...
x[19]                 # ... est l'élément 19 du vecteur

## Le tableau ci-dessus est un prisme rectangulaire 3 unités
## de haut, 4 de large et 5 de profond. Indicer ce prisme avec
## un seul indice équivaut à en extraire des «tranches», alors
## qu'utiliser deux indices équivaut à en tirer des «carottes»
## (au sens géologique du terme). Il est laissé en exercice de
## généraliser à plus de dimensions...
x                     # les cinq matrices
x[, , 1]             # tranches de haut en bas
x[, 1, ]             # tranches d'avant à l'arrière
x[1, , ]             # tranches de gauche à droite
x[, 1, 1]            # carotte de haut en bas

```

```

x[1, 1, ]           # carotte d'avant à l'arrière
x[1, , 1]           # carotte de gauche à droite

###
### LISTES
###

## La liste est l'objet le plus général en R. C'est un objet
## récursif qui peut contenir des objets de n'importe quel
## mode et longueur.
(x <- list(joueur = c("V", "C", "C", "M", "A"),
           score = c(10, 12, 11, 8, 15),
           expert = c(FALSE, TRUE, FALSE, TRUE, TRUE),
           niveau = 2))
is.vector(x)         # vecteur...
length(x)            # ... de quatre éléments...
mode(x)              # ... de mode "list"
is.recursive(x)      # objet récursif

## Comme tout autre vecteur, une liste peut être concaténée
## avec un autre vecteur avec la fonction 'c'.
y <- list(TRUE, 1:5)  # liste de deux éléments
c(x, y)              # liste de six éléments

## Pour initialiser une liste d'une longueur déterminée, mais
## dont chaque élément est vide, utiliser la fonction
## 'vector'.
vector("list", 5)     # liste de NULL

## Pour extraire un élément d'une liste, il faut utiliser les
## doubles crochets [[ ]]. Les simples crochets [ ]
## fonctionnent aussi, mais retournent une sous liste -- ce
## qui est rarement ce que l'on souhaite.
x[[1]]               # premier élément de la liste...
mode(x[[1]])         # ... un vecteur
x[1]                 # aussi le premier élément...
mode(x[1])           # ... mais une sous liste...
length(x[1])         # ... d'un seul élément
x[[2]][1]            # 1er élément du 2e élément
x[[c(2, 1)]]         # idem, par indiciage récursif

## Les éléments d'une liste étant généralement nommés (c'est
## une bonne habitude à prendre!), il est souvent plus simple
## et sûr d'extraire les éléments d'une liste par leur
## étiquette.

```

```

x$joueur           # équivalent à a[[1]]
x$score[1]         # équivalent à a[[c(2, 1)]]
x[["expert"]]      # aussi valide, mais peu usité
x$level <- 1       # aussi pour l'affectation

## Une liste peut contenir n'importe quoi...
x[[5]] <- matrix(1, 2, 2) # ... une matrice...
x[[6]] <- list(20:25, TRUE) # ... une autre liste...
x[[7]] <- seq           # ... même le code d'une fonction!
x                     # eh ben!
x[[c(6, 1, 3)]]       # de quel élément s'agit-il?

## Pour supprimer un élément d'une liste, lui assigner la
## valeur 'NULL'.
x[[7]] <- NULL; length(x) # suppression du 7e élément

## Il est parfois utile de convertir une liste en un simple
## vecteur. Les éléments de la liste sont alors «déroulés», y
## compris la matrice en position 5 (qui n'est rien d'autre
## qu'un vecteur, on s'en souviendra).
unlist(x)           # remarquer la conversion
unlist(x, recursive = FALSE) # ne pas appliquer aux sous-listes
unlist(x, use.names = FALSE) # éliminer les étiquettes

###
### DATA FRAMES
###

## Un data frame est une liste dont les éléments sont tous de
## même longueur. Il comporte un attribut 'dim', ce qui fait
## qu'il est représenté comme une matrice. Cependant, les
## colonnes peuvent être de modes différents.
(DF <- data.frame(Noms = c("Pierre", "Jean", "Jacques"),
                  Age = c(42, 34, 19),
                  Fumeur = c(TRUE, TRUE, FALSE)))

mode(DF)           # un data frame est une liste...
class(DF)          # ... de classe 'data.frame'
dim(DF)            # dimensions implicites
names(DF)          # titres des colonnes
row.names(DF)      # titres des lignes (implicites)
DF[1, ]           # première ligne
DF[, 1]           # première colonne
DF$Name            # idem, mais plus simple

## Lorsque l'on doit travailler longtemps avec les différentes

```



```
## colonnes d'un data frame, il est pratique de pouvoir y
## accéder directement sans devoir toujours indiquer. La
## fonction 'attach' permet de rendre les colonnes
## individuelles visibles dans l'espace de travail. Une fois
## le travail terminé, 'detach' masque les colonnes.
exists("Noms")          # variable n'existe pas
attach(DF)              # rendre les colonnes visibles
exists("Noms")          # variable existe
Noms                   # colonne accessible
detach(DF)              # masquer les colonnes
exists("Noms")          # variable n'existe plus

###
### INDICAGE
###

## Les opérations suivantes illustrent les différentes
## techniques d'indilage d'un vecteur pour l'extraction et
## l'affectation, c'est-à-dire que l'on utilise à la fois la
## fonction '[' et la fonction '[<-'. Les mêmes techniques
## existent aussi pour les matrices, tableaux et listes.
##
## On crée d'abord un vecteur quelconque formé de vingt
## nombres aléatoires entre 1 et 100 avec répétitions
## possibles.
(x <- sample(1:100, 20, replace = TRUE))

## On ajoute des étiquettes aux éléments du vecteur à partir
## de la variable interne 'letters'.
names(x) <- letters[1:20]

## On génère ensuite cinq nombres aléatoires entre 1 et 20
## (sans répétitions).
(y <- sample(1:20, 5))

## On remplace maintenant les éléments de 'x' correspondant
## aux positions dans le vecteur 'y' par des données
## manquantes.
x[y] <- NA
x

## Les cinq méthodes d'indilage de base.
x[1:10]                # avec des entiers positifs
"["(x, 1:10)           # idem, avec la fonction '['
x[-(1:3)]              # avec des entiers négatifs
```

```

x[x < 10]           # avec un vecteur booléen
x[c("a", "k", "t")] # par étiquettes
x[]                # aucun indice...
x[numeric(0)]      # ... différent d'indice vide

## Il arrive souvent de vouloir indexer spécifiquement les
## données manquantes d'un vecteur (pour les éliminer ou les
## remplacer par une autre valeur, par exemple). Pour ce
## faire, on utilise la fonction 'is.na' et l'indexage par un
## vecteur booléen. (Note: l'opérateur '!' ci-dessous est la
## négation logique.)
is.na(x)            # positions des données manquantes
x[!is.na(x)]        # suppression des données manquantes
x[is.na(x)] <- 0; x  # remplace les NA par des 0
"[<-"(x, is.na(x), 0) # idem, mais très peu usité

## On laisse tomber les étiquettes de l'objet.
names(x) <- NULL

## Quelques cas spéciaux d'indexage.
length(x)           # un rappel
x[1:25]             # allonge le vecteur avec des NA
x[25] <- 10; x      # remplis les trous avec des NA
x[0]                # n'extraie rien
x[0] <- 1; x        # n'affecte rien
x[c(0, 1, 2)]       # le 0 est ignoré
x[c(1, NA, 5)]      # indices NA retourne NA
x[2.6]             # fractions tronquées vers 0

## On laisse tomber les 5 derniers éléments et on convertit le
## vecteur en une matrice 4 x 5.
x <- x[1:20]        # ou x[-(21:25)]
dim(x) <- c(4, 5); x # ajouter un attribut 'dim'

## Dans l'indexage des matrices et tableaux, l'indice de
## chaque dimension obéit aux mêmes règles que ci-dessus. On
## peut aussi indexer une matrice (ou un tableau) avec une
## matrice. Si les exemples ci-dessous ne permettent pas d'en
## comprendre le fonctionnement, consulter la rubrique d'aide
## de la fonction '[' (ou de 'Extract').
x[1, 2]             # élément en position (1, 2)
x[1, -2]            # 1ère rangée sans 2e colonne
x[c(1, 3), ]       # 1ère et 3e rangées
x[-1, ]             # supprimer 1ère rangée
x[, -2]            # supprimer 2e colonne

```

```
x[x[, 1] > 10, ]           # lignes avec 1er élément > 10
x[rbind(c(1, 1), c(2, 2))] # éléments x[1, 1] et x[2, 2]
x[cbind(1:4, 1:4)]        # éléments x[i, i] (diagonale)
diag(x)                   # idem et plus explicite
```

2.10 Exercices

2.1 a) Écrire une expression R pour créer la liste suivante :

```
> x
[[1]]
[1] 1 2 3 4 5

$data
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

[[3]]
[1] 0 0 0

$test
[1] FALSE FALSE FALSE FALSE
```

- b) Extraire les étiquettes de la liste.
- c) Trouver le mode et la longueur du quatrième élément de la liste.
- d) Extraire les dimensions du second élément de la liste.
- e) Extraire les deuxième et troisième éléments du second élément de la liste.
- f) Remplacer le troisième élément de la liste par le vecteur 3:8.

2.2 Soit x un vecteur contenant les valeurs suivantes d'un échantillon :

```
> x
[1]  3 19  8  6  6 15 13 15 10 16  3  7  9 10  3 14
[17] 20 16  1 19
```

Écrire une expression R permettant d'extraire les éléments suivants.

- a) Le deuxième élément de l'échantillon.
- b) Les cinq premiers éléments de l'échantillon.
- c) Les éléments strictement supérieurs à 14.
- d) Tous les éléments sauf les éléments en positions 6, 10 et 12.

2.3 Soit x une matrice 10×7 obtenue aléatoirement avec

```
> x <- matrix(sample(1:100, 70), 7, 10)
```

Écrire des expressions R permettant d'obtenir les éléments de la matrice demandés ci-dessous.

- a) L'élément (4,3).
- b) Le contenu de la sixième ligne.
- c) Les première et quatrième colonnes (simultanément).
- d) Les lignes dont le premier élément est supérieur à 50.

3 Opérateurs et fonctions

Objectifs du chapitre

- ▶ Connaître les règles de l'arithmétique vectorielle caractéristique du langage R.
- ▶ Savoir faire l'appel d'une fonction dans R ; comprendre comment les arguments sont passés à la fonction et le traitement des valeurs par défaut.
- ▶ Connaître et savoir utiliser les opérateurs R les plus courants, notamment pour le traitement des vecteurs, le calcul de sommaires et la manipulation des matrices et tableaux
- ▶ Savoir utiliser la fonction `if` pour l'exécution conditionnelle de commandes R.
- ▶ Distinguer la construction `if () ... else` de la fonction `ifelse`.
- ▶ Savoir faire des boucles en R.
- ▶ Savoir choisir entre les opérateurs `for`, `while` et `repeat` lors de la construction d'une boucle R.

Ce chapitre présente les principaux opérateurs arithmétiques, fonctions mathématiques et structures de contrôle disponibles dans R. La liste est évidemment loin d'être exhaustive, surtout étant donné l'évolution rapide du langage. Un des meilleurs endroits pour découvrir de nouvelles fonctions demeure la section `See Also` des rubriques d'aide, qui offre des hyperliens vers des fonctions apparentées au sujet de la rubrique.

3.1 Opérations arithmétiques

L'unité de base en R est le vecteur.

- ▶ Les opérations sur les vecteurs sont effectuées *élément par élément* :

```
> c(1, 2, 3) + c(4, 5, 6)
```

```
[1] 5 7 9
```

```
> 1:3 * 4:6
```

```
[1] 4 10 18
```

- Si les vecteurs impliqués dans une expression arithmétique ne sont pas de la même longueur, les plus courts sont *recyclés* de façon à correspondre au plus long vecteur. Cette règle est particulièrement apparente avec les vecteurs de longueur 1 :

```
> 1:10 + 2
[1] 3 4 5 6 7 8 9 10 11 12
> 1:10 + rep(2, 10)
[1] 3 4 5 6 7 8 9 10 11 12
```

- Si la longueur du plus long vecteur est un multiple de celle du ou des autres vecteurs, ces derniers sont recyclés un nombre entier de fois :

```
> 1:10 + 1:5 + c(2, 4) # vecteurs recyclés 2 et 5 fois
[1] 4 8 8 12 12 11 11 15 15 19
> 1:10 + rep(1:5, 2) + rep(c(2, 4), 5) # équivalent
[1] 4 8 8 12 12 11 11 15 15 19
```

- Sinon, le plus court vecteur est recyclé un nombre fractionnaire de fois, mais comme ce résultat est rarement souhaité et provient généralement d'une erreur de programmation, un avertissement est affiché :

```
> 1:10 + c(2, 4, 6)
[1] 3 6 9 6 9 12 9 12 15 12
Message d'avis :
In 1:10 + c(2, 4, 6) :
la taille d'un objet plus long n'est pas un multiple de la
taille d'un objet plus court
```

3.2 Opérateurs

Le tableau 3.1 présente les opérateurs mathématiques et logiques les plus fréquemment employés, en ordre décroissant de priorité des opérations. Le tableau contient également les opérateurs d'assignation et d'extraction présentés au chapitre précédent ; il est utile de connaître leur niveau de priorité dans les expressions R.

Les opérateurs de puissance (^) et d'assignation à gauche (<-, <<-) sont évalués de droite à gauche ; tous les autres de gauche à droite. Ainsi, $2 \wedge 2 \wedge 3$ est $2 \wedge 8$, et non $4 \wedge 3$, alors que $1 - 1 - 1$ vaut -1, et non 1.

Opérateur	Fonction
\$	extraction d'une liste
^	puissance
-	changement de signe
:	génération de suites
%% %/%	produit matriciel, modulo, division entière
* /	multiplication, division
+ -	addition, soustraction
< <= == >= > !=	plus petit, plus petit ou égal, égal, plus grand ou égal, plus grand, différent de
!	négation logique
& &&	«et» logique
	«ou» logique
-> ->>	assignation
<- <<-	assignation

TAB. 3.1: Principaux opérateurs du langage R, en ordre décroissant de priorité

3.3 Appels de fonctions

Les opérateurs du tableau 3.1 constituent des raccourcis utiles pour accéder aux fonctions les plus courantes de R. Pour toutes les autres, il faut appeler la fonction directement. Cette section passe en revue les règles d'appels d'une fonction et la façon de spécifier les arguments, qu'il s'agisse d'une fonction interne de R ou d'une fonction personnelle (voir le chapitre ??).

- ▶ Il n'y a pas de limite pratique quant au nombre d'arguments que peut avoir une fonction.
- ▶ Les arguments d'une fonction peuvent être spécifiés selon l'ordre établi dans la définition de la fonction. Cependant, il est beaucoup plus prudent et *fortement recommandé* de spécifier les arguments par leur nom, avec une construction de la forme `nom = valeur`, surtout après les deux ou trois premiers arguments.
- ▶ L'ordre des arguments est important ; il est donc nécessaire de les nommer s'ils ne sont pas appelés dans l'ordre.
- ▶ Certains arguments ont une valeur par défaut qui sera utilisée si l'argument n'est pas spécifié dans l'appel de la fonction.

Par exemple, la définition de la fonction `matrix` est la suivante :

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

- La fonction compte cinq arguments : `data`, `nrow`, `ncol`, `byrow` et `dimnames`.
- Ici, chaque argument a une valeur par défaut (ce n'est pas toujours le cas). Ainsi, un appel à `matrix` sans argument résulte en une matrice 1×1 remplie par colonne (sans importance, ici) de l'objet `NA` et dont les dimensions sont dépourvues d'étiquettes :

```
> matrix()
      [,1]
[1,]    NA
```

- Appel plus élaboré utilisant tous les arguments. Le premier argument est rarement nommé :

```
> matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE,
+       dimnames = list(c("Gauche", "Droit"),
+       c("Rouge", "Vert", "Bleu")))
      Rouge Vert Bleu
Gauche  1    2    3
Droit   4    5    6
```

3.4 Quelques fonctions utiles

Le langage R compte un très grand nombre (quelques milliers !) de fonctions internes. Cette section en présente quelques-unes seulement, les fonctions de base les plus souvent utilisées pour programmer en R et pour manipuler des données.

Pour chaque fonction présentée dans les sections suivantes, on fournit un ou deux exemples d'utilisation. Ces exemples sont souvent loin de couvrir toutes les utilisations possibles d'une fonction. La section 3.7 fournit des exemples additionnels, mais il est recommandé de consulter les diverses rubriques d'aide pour connaître toutes les options des fonctions.

3.4.1 Manipulation de vecteurs

seq génération de suites de nombres

```
> seq(1, 9, by = 2)
[1] 1 3 5 7 9
```

rep répétition de valeurs ou de vecteurs


```
> rep(2, 10)
[1] 2 2 2 2 2 2 2 2 2 2
```

sort tri en ordre croissant ou décroissant

```
> sort(c(4, -1, 2, 6))
[1] -1 2 4 6
```

order ordre d'extraction des éléments d'un vecteur pour les placer en ordre croissant ou décroissant

```
> order(c(4, -1, 2, 6))
[1] 2 3 1 4
```

rank rang des éléments d'un vecteur dans l'ordre croissant ou décroissant

```
> order(c(4, -1, 2, 6))
[1] 2 3 1 4
```

rev renverser un vecteur

```
> rev(1:10)
[1] 10 9 8 7 6 5 4 3 2 1
```

head extraction des n premiers éléments d'un vecteur ($n > 0$) ou suppression des n derniers ($n < 0$)

```
> head(1:10, 3); head(1:10, -3)
[1] 1 2 3
[1] 1 2 3 4 5 6 7
```

tail extraction des n derniers éléments d'un vecteur ($n > 0$) ou suppression des n premiers ($n < 0$)

```
> tail(1:10, 3); tail(1:10, -3)
[1] 8 9 10
[1] 4 5 6 7 8 9 10
```

unique extraction des éléments différents d'un vecteur

```
> unique(c(2, 4, 2, 5, 9, 5, 0))
[1] 2 4 5 9 0
```

3.4.2 Recherche d'éléments dans un vecteur

Les fonctions de cette sous-section sont toutes illustrées avec le vecteur

```
> x
[1] 4 -1 2 -3 6
```

which positions des valeurs TRUE dans un vecteur booléen

```

> which(x < 0)
[1] 2 4

```

which.min position du minimum dans un vecteur

```

> which.min(x)
[1] 4

```

which.max position du maximum dans un vecteur

```

> which.max(x)
[1] 5

```

match position de la première occurrence d'un élément dans un vecteur

```

> match(2, x)
[1] 3

```

%in% appartenance d'une ou plusieurs valeurs à un vecteur

```

> -1:2 %in% x
[1] TRUE FALSE FALSE TRUE

```

3.4.3 Arrondi

Les fonctions de cette sous-section sont toutes illustrées avec le vecteur

```

> x
[1] -3.6800000 -0.6666667  3.1415927  0.3333333
[5]  2.5200000

```

round arrondi à un nombre défini de décimales (par défaut 0)

```

> round(x)
[1] -4 -1  3  0  3
> round(x, 3)
[1] -3.680 -0.667  3.142  0.333  2.520

```

floor plus grand entier inférieur ou égal à l'argument

```

> floor(x)
[1] -4 -1  3  0  2

```

ceiling plus petit entier supérieur ou égal à l'argument

```

> ceiling(x)
[1] -3  0  4  1  3

```

trunc troncature vers zéro de l'argument ; différent de floor pour les nombres négatifs

```

> trunc(x)
[1] -3  0  3  0  2

```

3.4.4 Sommaires et statistiques descriptives

Les fonctions de cette sous-section sont toutes illustrées avec le vecteur

```
> x
[1] 14 17 7 9 3 4 25 21 24 11
```

<code>sum, prod</code>	<p>somme et produit des éléments d'un vecteur</p> <pre>> sum(x); prod(x) [1] 135 [1] 24938020800</pre>
<code>diff</code>	<p>différences entre les éléments d'un vecteur (opérateur mathématique ∇)</p> <pre>> diff(x) [1] 3 -10 2 -6 1 21 -4 3 -13</pre>
<code>mean</code>	<p>moyenne arithmétique (et moyenne tronquée avec l'argument <code>trim</code>)</p> <pre>> mean(x) [1] 13.5</pre>
<code>var, sd</code>	<p>variance et écart type (versions sans biais)</p> <pre>> var(x) [1] 64.5</pre>
<code>min, max</code>	<p>minimum et maximum d'un vecteur</p> <pre>> min(x); max(x) [1] 3 [1] 25</pre>
<code>range</code>	<p>vecteur contenant le minimum et le maximum d'un vecteur</p> <pre>> range(x) [1] 3 25</pre>
<code>median</code>	<p>médiane empirique</p> <pre>> median(x) [1] 12.5</pre>
<code>quantile</code>	<p>quantiles empiriques</p> <pre>> quantile(x) 0% 25% 50% 75% 100% 3.0 7.5 12.5 20.0 25.0</pre>
<code>summary</code>	<p>statistiques descriptives d'un échantillon</p> <pre>> summary(x) Min. 1st Qu. Median Mean 3rd Qu. Max. 3.0 7.5 12.5 13.5 20.0 25.0</pre>

3.4.5 Sommaires cumulatifs et comparaisons élément par élément

Les fonctions de cette sous-section sont toutes illustrées avec le vecteur

```
> x
[1] 14 17 7 9 3
```

`cumsum, cumprod`

somme et produit cumulatif d'un vecteur

```
> cumsum(x); cumprod(x)
[1] 14 31 38 47 50
[1] 14 238 1666 14994 44982
```

`cummin, cummax`

minimum et maximum cumulatif

```
> cummin(x); cummax(x)
[1] 14 14 7 7 3
[1] 14 17 17 17 17
```

`pmin, pmax`

minimum et maximum en parallèle, c'est-à-dire élément par élément entre deux vecteurs ou plus

```
> pmin(x, c(16, 23, 4, 12, 3))
[1] 14 17 4 9 3
> pmax(x, c(16, 23, 4, 12, 3))
[1] 16 23 7 12 3
```

3.4.6 Opérations sur les matrices

Les fonctions de cette sous-section sont toutes illustrées avec la matrice

```
> x
      [,1] [,2]
[1,]    2    4
[2,]    1    3
```

`nrow, ncol`

nombre de lignes et de colonnes d'une matrice

```
> nrow(x); ncol(x)
[1] 2
[1] 2
```

`rowSums, colSums`

sommes par ligne et par colonne, respectivement, des éléments d'une matrice; voir aussi la fonction `apply` à la section ??

	<pre>> rowSums(x) [1] 6 4</pre>
rowMeans, colMeans	<p>moyennes par ligne et par colonne, respectivement, des éléments d'une matrice ; voir aussi la fonction <code>apply</code> à la section ??</p> <pre>> colMeans(x) [1] 1.5 3.5</pre>
t	<p>transposée</p> <pre>> t(x) [,1] [,2] [1,] 2 1 [2,] 4 3</pre>
det	<p>déterminant</p> <pre>> det(x) [1] 2</pre>
solve	<p>avec un seul argument (une matrice carrée) : inverse d'une matrice ; avec deux arguments (une matrice carrée et un vecteur) : solution du système d'équations linéaires $A\mathbf{x} = \mathbf{b}$</p> <pre>> solve(x) [,1] [,2] [1,] 1.5 -2 [2,] -0.5 1 > solve(x, c(1, 2)) [1] -2.5 1.5</pre>
diag	<p>avec une matrice en argument : diagonale de la matrice ; avec un vecteur en argument : matrice diagonale formée avec le vecteur ; avec un scalaire p en argument : matrice identité $p \times p$</p> <pre>> diag(x) [1] 2 3</pre>

3.4.7 Produit extérieur

Le produit extérieur n'est pas la fonction la plus intuitive à utiliser, mais s'avère extrêmement utile pour faire plusieurs opérations en un seul appel de fonction tout en évitant les boucles. La syntaxe de la fonction `outer` est

```
outer(X, Y, FUN)
```

Le résultat est l'application la fonction FUN (prod par défaut) entre chacun des éléments de X et chacun des éléments de Y, autrement dit

$$\text{FUN}(X[i], Y[j])$$

pour toutes les valeurs des indices i et j .

- La dimension du résultat est par conséquent $c(\dim(X), \dim(Y))$.
- Par exemple, le résultat du produit extérieur entre deux vecteurs est une matrice contenant tous les produits entre les éléments des deux vecteurs :

```
> outer(c(1, 2, 5), c(2, 3, 6))
```

```
      [,1] [,2] [,3]
[1,]     2     3     6
[2,]     4     6    12
[3,]    10    15    30
```

- L'opérateur `%o%` est un raccourci de `outer(X, Y, prod)`.

3.5 Structures de contrôle

Les structures de contrôle sont des commandes qui permettent de déterminer le flux d'exécution d'un programme : choix entre des blocs de code, répétition de commandes ou sortie forcée.

On se contente, ici, de mentionner les structures de contrôle disponibles en R. Se reporter à la section 3.7 pour des exemples d'utilisation.

3.5.1 Exécution conditionnelle

```
if (condition) branche.vrai else branche.faux
```

Si *condition* est vraie, *branche.vrai* est exécutée, sinon ce sera *branche.faux*. Dans le cas où l'une ou l'autre de *branche.vrai* ou *branche.faux* comporte plus d'une expression, grouper celles-ci dans des accolades `{ }`.

```
ifelse(condition, expression.vrai, expression.faux)
```

Fonction vectorielle qui retourne un vecteur de la même longueur que *condition* formé ainsi : pour chaque élément TRUE de *condition* on choisit l'élément correspondant de *expression.vrai* et pour chaque élément FALSE on choisit l'élément correspondant de *expression.faux*. L'utilisation n'est pas très intuitive, alors examiner attentivement les exemples de la rubrique d'aide.

```
switch(test, cas.1 = action.1, cas.2 = action.2, ...)
```

Structure utilisée plutôt rarement. Consulter la rubrique d'aide au besoin.

3.5.2 Boucles

Les boucles sont et doivent être utilisées avec parcimonie en R, car elles sont généralement inefficaces. Dans la majeure partie des cas, il est possible de vectoriser les calculs pour éviter les boucles explicites, ou encore de s'en remettre aux fonctions `outer`, `apply`, `lapply`, `sapply` et `mapply` (section ??) pour réaliser les boucles de manière plus efficace.

```
for (variable in suite) expression
```

Exécuter *expression* successivement pour chaque valeur de *variable* contenue dans *suite*. Encore ici, on groupera les expressions dans des accolades `{ }`. À noter que *suite* n'a pas à être composée de nombres consécutifs, ni même de nombres, en fait.

```
while (condition) expression
```

Exécuter *expression* tant que *condition* est vraie. Si *condition* est fausse lors de l'entrée dans la boucle, celle-ci n'est pas exécutée. Une boucle `while` n'est par conséquent pas nécessairement toujours exécutée.

```
repeat expression
```

Répéter *expression*. Cette dernière devra comporter un test d'arrêt qui utilisera la commande `break`. Une boucle `repeat` est toujours exécutée au moins une fois.

```
break
```

Sortie immédiate d'une boucle `for`, `while` ou `repeat`.

```
next
```

Passage immédiat à la prochaine itération d'une boucle `for`, `while` ou `repeat`.

3.6 Fonctions additionnelles

La bibliothèque des fonctions internes de R est divisée en ensembles de fonctions et de jeux de données apparentés nommés *packages* (terme que l'équipe de traduction française de R a choisi de conserver tel quel). On démarre, R charge automatiquement quelques packages de la bibliothèque, ceux contenant les fonctions les plus fréquemment utilisées. On peut voir la liste des packages déjà en mémoire avec

```
> search()
[1] ".GlobalEnv"      "package:stats"
[3] "package:graphics" "package:grDevices"
[5] "package:utils"    "package:datasets"
[7] "package:methods"  "AutoLoads"
[9] "package:base"
```

et le contenu de toute la bibliothèque avec la fonction `library` (résultat non montré ici).

Une des grandes forces de R est la facilité avec laquelle on peut ajouter des fonctionnalités au système par le biais de packages externes. Dès les débuts de R, les développeurs et utilisateurs ont mis sur pied le dépôt central de packages *Comprehensive R Archive Network* (CRAN; <http://cran.r-project.org>). Ce site compte aujourd'hui plusieurs centaines d'extensions et le nombre ne cesse de croître.

R rend simple de télécharger et d'installer de nouveaux packages avec la fonction `install.packages`. L'annexe ?? explique plus en détails comment gérer sa bibliothèque personnelle et installer des packages externes.

3.7 Exemples

```
###
### OPÉRATIONS ARITHMÉTIQUES
###

## L'arithmétique vectorielle caractéristique du langage R
## rend très simple et intuitif de faire des opérations
## mathématiques courantes. Là où plusieurs langages de
## programmation exigent des boucles, R fait le calcul
## directement. En effet, les règles de l'arithmétique en R
## sont globalement les mêmes qu'en algèbre vectorielle et
## matricielle.
5 * c(2, 3, 8, 10)      # multiplication par une constante
c(2, 6, 8) + c(1, 4, 9) # addition de deux vecteurs
c(0, 3, -1, 4)^2        # élévation à une puissance

## Dans les règles de l'arithmétique vectorielle, les
## longueurs des vecteurs doivent toujours concorder. R permet
## plus de flexibilité en recyclant les vecteurs les plus
## courts dans une opération. Il n'y a donc à peu près jamais
## d'erreurs de longueur en R! C'est une arme à deux
```



```

## tranchants: le recyclage des vecteurs facilite le codage,
## mais peut aussi résulter en des réponses complètement
## erronées sans que le système ne détecte d'erreur.
8 + 1:10           # 8 est recyclé 10 fois
c(2, 5) * 1:10     # c(2, 5) est recyclé 5 fois
c(-2, 3, -1, 4)^1:4 # quatre puissances différentes

## On se rappelle que les matrices (et les tableaux) sont des
## vecteurs. Les règles ci-dessus s'appliquent donc aussi aux
## matrices, ce qui résulte en des opérateurs qui ne sont pas
## définis en algèbre linéaire usuelle.
(x <- matrix(1:4, 2)) # matrice 2 x 2
(y <- matrix(3:6, 2)) # autre matrice 2 x 2
5 * x                 # multiplication par une constante
x + y                 # addition matricielle
x * y                 # produit *élément par élément*
x %*% y               # produit matriciel
x / y                 # division *élément par élément*
x * c(2, 3)           # produit par colonne

###
### OPÉRATEURS
###

## Seuls les opérateurs %, %/% et logiques sont illustrés
## ici. Premièrement, l'opérateur modulo retourne le reste
## d'une division.
5 %% 2                # 5/2 = 2 reste 1
5 %% 1:5              # remarquer la périodicité
10 %% 1:15            # x %% y = x si x < y

## Le modulo est pratique dans les boucles, par exemple pour
## afficher un résultat à toutes les n itérations seulement.
for (i in 1:50)
{
  ## Affiche la valeur du compteur toutes les 5 itérations.
  if (0 == i %% 5)
    print(i)
}

## La division entière retourne la partie entière de la
## division d'un nombre par un autre.
5 %/% 1:5
10 %/% 1:15

```

```

## Le ET logique est vrai seulement lorsque les deux
## expressions sont vraies.
c(TRUE, TRUE, FALSE) & c(TRUE, FALSE, FALSE)

## Le OU logique est faux seulement lorsque les deux
## expressions sont fausses.
c(TRUE, TRUE, FALSE) | c(TRUE, FALSE, FALSE)

## La négation logique transforme les vrais en faux et vice
## versa.
! c(TRUE, FALSE, FALSE, TRUE)

## On peut utiliser les opérateurs logiques &, | et !
## directement avec des nombres. Dans ce cas, le nombre zéro
## est traité comme FALSE et tous les autres nombres comme
## TRUE.
0:5 & 5:0
0:5 | 5:0
!0:5

## Ainsi, dans une expression conditionnelle, inutile de
## vérifier si, par exemple, un nombre est égal à zéro. On
## peut utiliser le nombre directement et sauver des
## opérations de comparaison qui peuvent devenir coûteuses en
## temps de calcul.
x <- 1 # valeur quelconque
if (x != 0) x + 1 # TRUE pour tout x != 0
if (x) x + 1 # tout à fait équivalent!

## L'exemple de boucle ci-dessus peut donc être légèrement
## modifié.
for (i in 1:50)
{
  ## Affiche la valeur du compteur toutes les 5 itérations.
  if (!i %% 5)
    print (i)
}

## Dans les calculs numériques, TRUE vaut 1 et FALSE vaut 0.
a <- c("Impair", "Pair")
x <- c(2, 3, 6, 8, 9, 11, 12)
x %% 2
(!x %% 2) + 1
a[(!x %% 2) + 1]

```

```
## Un mot en terminant sur l'opérateur '=='. C'est l'opérateur
## à utiliser pour vérifier si deux valeurs sont égales, et
## non '='. C'est là une erreur commune --- et qui peut être
## difficile à détecter --- lorsque l'on programme en R.
5 = 2                # erreur de syntaxe
5 == 2               # comparaison

###
### APPELS DE FONCTIONS
###

## Les invocations de la fonction 'matrix' ci-dessous sont
## toutes équivalentes. On remarquera, entre autres, comment
## les arguments sont spécifiés (par nom ou par position).
matrix(1:12, 3, 4)
matrix(1:12, ncol = 4, nrow = 3)
matrix(nrow = 3, ncol = 4, data = 1:12)
matrix(nrow = 3, ncol = 4, byrow = FALSE, 1:12)
matrix(nrow = 3, ncol = 4, 1:12, FALSE)

###
### QUELQUES FONCTIONS UTILES
###

## MANIPULATION DE VECTEURS
x <- c(50, 30, 10, 20, 60, 30, 20, 40) # vecteur non ordonné

## Séquences de nombres.
seq(from = 1, to = 10)      # équivalent à 1:10
seq(-10, 10, length = 50)  # incrément automatique
seq(-2, by = 0.5, along = x) # même longueur que 'x'

## Répétition de nombres ou de vecteurs complets.
rep(1, 10)                  # utilisation de base
rep(x, 2)                   # répéter un vecteur
rep(x, times = 2, each = 4) # combinaison des arguments
rep(x, times = 1:8)         # nombre de répétitions différent
                             # pour chaque élément de 'x'

## Classement en ordre croissant ou décroissant.
sort(x)                     # classement en ordre croissant
sort(x, decr = TRUE)        # classement en ordre décroissant
sort(c("abc", "B", "Aunt", "Jemima")) # chaînes de caractères
sort(c(TRUE, FALSE))        # FALSE vient avant TRUE
```

```
## La fonction 'order' retourne la position, dans le vecteur
## donné en argument, du premier élément selon l'ordre
## croissant, puis du deuxième, etc. Autrement dit, on obtient
## l'ordre dans lequel il faut extraire les données du vecteur
## pour les obtenir en ordre croissant.
```

```
order(x)                # regarder dans le blanc des yeux
x[order(x)]              # équivalent à 'sort(x)'
```

```
## Rang des éléments d'un vecteur dans l'ordre croissant.
rank(x)                  # rang des éléments de 'x'
```

```
## Renverser l'ordre d'un vecteur.
rev(x)
```

```
## Extraction ou suppression en tête ou en queue de vecteur.
head(x, 3)               # trois premiers éléments
head(x, -2)              # tous sauf les deux derniers
tail(x, 3)               # trois derniers éléments
tail(x, -2)              # tous sauf les deux premiers
```

```
## Expressions équivalentes sans 'head' et 'tail'
x[1:3]                   # trois premiers éléments
x[1:(length(x) - 2)]     # tous sauf les deux derniers
x[(length(x)-2):length(x)] # trois derniers éléments
rev(rev(x)[1:3])         # avec petits vecteurs seulement
x[c(-1, -2)]             # tous sauf les deux premiers
```

```
## Seulement les éléments différents d'un vecteur.
unique(x)
```

RECHERCHE D'ÉLÉMENTS DANS UN VECTEUR

```
which(x >= 30)           # positions des éléments >= 30
which.min(x)             # position du minimum
which.max(x)             # position du maximum
match(20, x)             # position du premier 20 dans 'x'
match(c(20, 30), x)      # aussi pour plusieurs valeurs
60 %in% x                # 60 appartient à 'x'
70 %in% x                # 70 n'appartient pas à 'x'
```

ARRONDI

```
(x <- c(-21.2, -pi, -1.5, -0.2, 0, 0.2, 1.7823, 315))
round(x)                 # arrondi à l'entier
round(x, 2)              # arrondi à la seconde décimale
round(x, -1)             # arrondi aux dizaines
ceiling(x)               # plus petit entier supérieur
```

```
floor(x)           # plus grand entier inférieur
trunc(x)           # troncature des décimales

## SOMMAIRES ET STATISTIQUES DESCRIPTIVES
sum(x)             # somme des éléments
prod(x)            # produit des éléments
diff(x)            # x[2] - x[1], x[3] - x[2], etc.
mean(x)            # moyenne des éléments
mean(x, trim = 0.125) # moyenne sans minimum et maximum
var(x)             # variance (sans biais)
(length(x) - 1)/length(x) * var(x) # variance biaisée
sd(x)              # écart type
max(x)             # maximum
min(x)             # minimum
range(x)           # c(min(x), max(x))
diff(range(x))     # étendue de 'x'
median(x)          # médiane (50e quantile) empirique
quantile(x)        # quantiles empiriques
quantile(x, 1:10/10) # on peut spécifier les quantiles
summary(x)         # plusieurs des résultats ci-dessus

## SOMMAIRES CUMULATIFS ET COMPARAISONS ÉLÉMENT PAR ÉLÉMENT
(x <- sample(1:20, 6))
(y <- sample(1:20, 6))
cumsum(x)          # somme cumulative de 'x'
cumprod(y)         # produit cumulatif de 'y'
rev(cumprod(rev(y))) # produit cumulatif renversé
cummin(x)          # minimum cumulatif
cummax(y)          # maximum cumulatif
pmin(x, y)         # minimum élément par élément
pmax(x, y)         # maximum élément par élément

## OPÉRATIONS SUR LES MATRICES
(A <- sample(1:10, 16, replace = TRUE)) # avec remise
dim(A) <- c(4, 4)      # conversion en une matrice 4 x 4
b <- c(10, 5, 3, 1)    # un vecteur quelconque
A                     # la matrice 'A'
t(A)                 # sa transposée
solve(A)             # son inverse
solve(A, b)          # la solution de Ax = b
A %*% solve(A, b)     # vérification de la réponse
diag(A)              # extraction de la diagonale de 'A'
diag(b)              # matrice diagonale formée avec 'b'
diag(4)              # matrice identité 4 x 4
(A <- cbind(A, b))    # matrice 4 x 5
```

```

nrow(A)           # nombre de lignes de 'A'
ncol(A)           # nombre de colonnes de 'A'
rowSums(A)        # sommes par ligne
colSums(A)        # sommes par colonne
apply(A, 1, sum)   # équivalent à 'rowSums(A)'
apply(A, 2, sum)   # équivalent à 'colSums(A)'
apply(A, 1, prod)  # produit par ligne avec 'apply'

## PRODUIT EXTÉRIEUR
x <- c(1, 2, 4, 7, 10, 12)
y <- c(2, 3, 6, 7, 9, 11)
outer(x, y)       # produit extérieur
x %0% y           # équivalent plus court
outer(x, y, "+")   # «somme extérieure»
outer(x, y, "<=")   # toutes les comparaisons possibles
outer(x, y, pmax)  # idem

###
### STRUCTURES DE CONTRÔLE
###

## Pour illustrer les structures de contrôle, on a recours à
## un petit exemple tout à fait artificiel: un vecteur est
## rempli des nombres de 1 à 100, à l'exception des multiples
## de 10. Ces derniers sont affichés à l'écran.
##
## À noter qu'il est possible --- et plus efficace --- de
## créer le vecteur sans avoir recours à des boucles.
(1:100)[-((1:10) * 10)]      # sans boucle!
rep(1:9, 10) + rep(0:9*10, each = 9) # une autre façon!

## Bon, l'exemple proprement dit...
x <- numeric(0)             # initialisation du contenant 'x'
j <- 0                      # compteur pour la boucle
for (i in 1:100)
{
  if (i % 10)               # si i n'est pas un multiple de 10
    x[j <- j + 1] <- i      # stocker sa valeur dans 'x'
  else                      # sinon
    print(i)               # afficher la valeur à l'écran
}
x                          # vérification

## Même chose que ci-dessus, mais sans le compteur 'j' et les
## valeurs manquantes aux positions 10, 20, ..., 100 sont

```

éliminées à la sortie de la boucle.

```
x <- numeric(0)
for (i in 1:100)
{
  if (i %% 10)
    x[i] <- i
  else
    print(i)
}
x <- x[!is.na(x)]
x
```

*## On peut refaire l'exemple avec une boucle 'while', mais
cette structure n'est pas naturelle ici puisque l'on sait
d'avance qu'il faudra faire la boucle exactement 100
fois. Le 'while' est plutôt utilisé lorsque le nombre de
répétitions est inconnu. De plus, une boucle 'while' n'est
pas nécessairement exécutée puisque le critère d'arrêt est
évalué dès l'entrée dans la boucle.*

```
x <- numeric(0)
j <- 0
i <- 1                                # pour entrer dans la boucle [*]
while (i <= 100)
{
  if (i %% 10)
    x[j <- j + 1] <- i
  else
    print(i)
  i <- i + 1                          # incrémenter le compteur!
}
x
```

*## La remarque faite au sujet de la boucle 'while' s'applique
aussi à la boucle 'repeat'. Par contre, le critère d'arrêt
de la boucle 'repeat' étant évalué à la toute fin, la
boucle est exécutée au moins une fois. S'il faut faire la
manoeuvre marquée [*] ci-dessus pour s'assurer qu'une
boucle 'while' est exécutée au moins une fois... c'est
qu'il faut utiliser 'repeat'.*

```
x <- numeric(0)
j <- 0
i <- 1
repeat
{
  if (i %% 10)
```

```

        x[j <- j + 1] <- i
    else
        print(i)
    if (100 < (i <- i + 1)) # incrément et critère d'arrêt
        break
}
x

###
### FONCTIONS ADDITIONNELLES
###

## La fonction 'search' retourne la liste des environnements
## dans lesquels R va chercher un objet (en particulier une
## fonction). '.GlobalEnv' est l'environnement de travail.
search()

## Liste de tous les packages installés sur votre système.
library()

## Chargement du package 'MASS', qui contient plusieurs
## fonctions statistiques très utiles.
library("MASS")

```

3.8 Exercices

3.1 À l'aide des fonctions `rep`, `seq` et `c` seulement, générer les séquences suivantes.

- a) 0 6 0 6 0 6
- b) 1 4 7 10
- c) 1 2 3 1 2 3 1 2 3 1 2 3
- d) 1 2 2 3 3 3
- e) 1 1 1 2 2 3
- f) 1 5.5 10
- g) 1 1 1 1 2 2 2 2 3 3 3 3

3.2 Générer les suites de nombres suivantes à l'aide des fonctions : et `rep` seulement, donc sans utiliser la fonction `seq`.

- a) 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2

b) 1 3 5 7 9 11 13 15 17 19

c) -2 -1 0 1 2 -2 -1 0 1 2

d) -2 -2 -1 -1 0 0 1 1 2 2

e) 10 20 30 40 50 60 70 80 90 100

3.3 À l'aide de la commande `apply`, écrire des expressions R qui remplaceraient les fonctions suivantes.

a) `rowSums`

b) `colSums`

c) `rowMeans`

d) `colMeans`

3.4 Sans utiliser les fonctions `factorial`, `lfactorial`, `gamma` ou `lgamma`, générer la séquence $1!, 2!, \dots, 10!$

3.5 Trouver une relation entre x , y , $x \% \% y$ (modulo) et $x \% / \% y$ (division entière), où $y \neq 0$.

3.6 Simuler un échantillon $\mathbf{x} = (x_1, x_2, x_3, \dots, x_{20})$ avec la fonction `sample`. Écrire une expression R permettant d'obtenir ou de calculer chacun des résultats demandés ci-dessous.

a) Les cinq premiers éléments de l'échantillon.

b) La valeur maximale de l'échantillon.

c) La moyenne des cinq premiers éléments de l'échantillon.

d) La moyenne des cinq derniers éléments de l'échantillon.

3.7 a) Trouver une formule pour calculer la position, dans le vecteur sous-jacent, de l'élément (i, j) d'une matrice $I \times J$ remplie par colonne.

b) Répéter la partie a) pour l'élément (i, j, k) d'un tableau $I \times J \times K$.

3.8 Simuler une matrice `mat` 10×7 , puis écrire des expressions R permettant d'effectuer les tâches demandées ci-dessous.

a) Calculer la somme des éléments de chacune des lignes de la matrice.

b) Calculer la moyenne des éléments de chacune des colonnes de la matrice.

c) Calculer la valeur maximale de la sous-matrice formée par les trois premières lignes et les trois premières colonnes.

- d) Extraire toutes les lignes de la matrice dont la moyenne des éléments est supérieure à 7.

3.9 On vous donne la liste et la date des 31 meilleurs temps enregistrés au 100 mètres homme entre 1964 et 2005 :

```
> temps <- c(10.06, 10.03, 10.02, 9.95, 10.04, 10.07, 10.08, 10.05,
+           9.98, 10.09, 10.01, 10.00, 9.97, 9.93, 9.96, 9.99,
+           9.92, 9.94, 9.90, 9.86, 9.88, 9.87, 9.85, 9.91,
+           9.84, 9.89, 9.79, 9.80, 9.82, 9.78, 9.77)
> names(temps) <- c("1964-10-15", "1968-06-20", "1968-10-13",
+                  "1968-10-14", "1968-10-14", "1968-10-14",
+                  "1968-10-14", "1975-08-20", "1977-08-11",
+                  "1978-07-30", "1979-09-04", "1981-05-16",
+                  "1983-05-14", "1983-07-03", "1984-05-05",
+                  "1984-05-06", "1988-09-24", "1989-06-16",
+                  "1991-06-14", "1991-08-25", "1991-08-25",
+                  "1993-08-15", "1994-07-06", "1994-08-23",
+                  "1996-07-27", "1996-07-27", "1999-06-16",
+                  "1999-08-22", "2001-08-05", "2002-09-14",
+                  "2005-06-14")
```

Extraire de ce vecteur les records du monde seulement, c'est-à-dire la première fois que chaque temps a été réalisé.

A GNU Emacs et ESS : la base

Emacs est l'Éditeur de texte des éditeurs de texte. Conçu à l'origine comme un éditeur pour les programmeurs (avec des modes spéciaux pour une multitude de langages différents), Emacs est devenu au fil du temps un environnement logiciel en soi dans lequel on peut réaliser une foule de tâches différentes : rédiger des documents \LaTeX , interagir avec R, SAS ou un logiciel de base de données, consulter son courrier électronique, gérer son calendrier ou même jouer à Tetris !

Cette annexe passe en revue les quelques commandes essentielles à connaître pour commencer à travailler avec GNU Emacs et le mode ESS. L'ouvrage de [Cameron et collab. \(2004\)](#) constitue une excellente référence pour l'apprentissage plus poussé de l'éditeur.

A.1 Mise en contexte

Emacs est le logiciel étendard du projet GNU («*GNU is not Unix*»), dont le principal commanditaire est la *Free Software Foundation* (FSF) à l'origine de tout le mouvement du logiciel libre.

- ▶ Richard M. Stallman, président de la FSF et grand apôtre du libre, a écrit la première version de Emacs et il continue à ce jour à contribuer au projet.
- ▶ Les origines de Emacs remontent au début des années 1980, une époque où les interfaces graphiques n'existaient pas, le parc informatique était beaucoup plus hétérogène qu'aujourd'hui (les claviers n'étaient pas les mêmes d'une marque d'ordinateur à une autre) et les modes de communication entre les ordinateurs demeuraient rudimentaires.
- ▶ L'âge vénérable de Emacs transparait à plusieurs endroits, notamment dans la terminologie inhabituelle, les raccourcis clavier qui ne correspondent pas aux standards d'aujourd'hui ou la manipulation des fenêtres qui ne se fait pas avec une souris.

Emacs s'adapte à différentes tâches par l'entremise de *modes* qui modifient son comportement ou lui ajoutent des fonctionnalités. L'un de ces modes est ESS (*Emacs Speaks Statistics*).

- ▶ ESS permet d'interagir avec des logiciels statistiques (en particulier R, S+ et SAS) directement depuis Emacs.
- ▶ Quelques-uns des développeurs de ESS sont aussi des développeurs de R, dont la grande compatibilité entre les deux logiciels.
- ▶ Lorsque ESS est installé, le mode est activé automatiquement en ouvrant dans Emacs un fichier dont le nom se termine par l'extension `.R`.

A.2 Installation

GNU Emacs et le mode ESS sont normalement livrés d'office avec toutes les distributions Linux. Pour les environnements Windows et Mac OS X, le plus simple consiste à télécharger et installer les distributions préparées par le présent auteur. Consulter le site

<http://vgoulet.act.ulaval.ca/emacs/>

A.3 Description sommaire

Au lancement, Emacs affiche un écran d'information contenant des liens vers différentes ressources. Cet écran disparaît dès que l'on appuie sur une touche. La fenêtre Emacs se divise en quatre zones principales (voir la figure A.1) :

1. tout au haut de la fenêtre (ou de l'écran sous OS X), on trouve l'habituelle barre de menu dont le contenu change selon le mode dans lequel se trouve Emacs ;
2. l'essentiel de la fenêtre sert à afficher un *buffer*, soit le contenu d'un fichier ouvert ou l'invite de commande d'un programme externe ;
3. la ligne de mode est le séparateur horizontal contenant diverses informations sur le fichier ouvert et l'état de Emacs ;
4. le *minibuffer* est la région au bas de la fenêtre où l'on entre des commandes et reçoit de l'information de Emacs.

Il est possible de séparer la fenêtre Emacs en sous-fenêtres pour afficher plusieurs *buffers* à la fois. Il y a alors une ligne de mode pour chaque *buffer*.

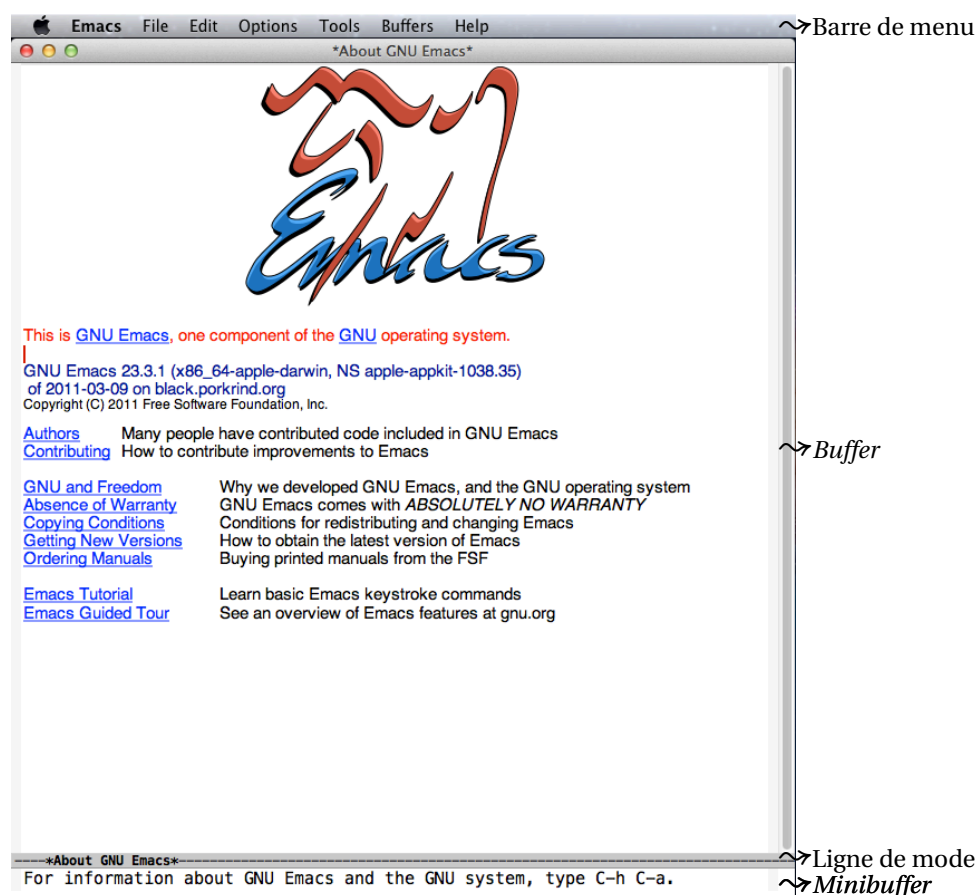


FIG. A.1: Fenêtre GNU Emacs et ses différentes parties au lancement de l'application sous Mac OS X. Sous Windows et Linux, la barre de menu se trouve à l'intérieur de la fenêtre.

A.4 Emacs-ismes et Unix-ismes

Emacs a sa propre terminologie qu'il vaut mieux connaître lorsque l'on consulte la documentation. De plus, l'éditeur fait siennes certaines conventions du monde Unix et qui sont moins usitées sur les plateformes Windows et OS X.

- Dans les définitions de raccourcis claviers :
 - C est la touche Contrôle (^) ;
 - M est la touche Meta, qui correspond à la touche Alt de gauche sur un PC ou la touche Option (⌘) sur un Mac (toutefois, voir l'encadré à la page suivante) ;



Par défaut sous Mac OS X, la touche Meta est assignée à Option (⌥). Sur les claviers français, cela empêche d'accéder à certains caractères spéciaux tels que [,], { ou }.

Une solution consiste à plutôt assigner la touche Meta à Commande (⌘). Cela bloque alors l'accès à certains raccourcis Mac, mais la situation est moins critique ainsi.

Pour assigner la touche Meta à Commande (⌘), il suffit d'insérer les lignes suivantes dans son fichier de configuration `.emacs` (voir la section A.7) :

```
;;; =====
;;; Assigner la touche Meta à Commande
;;; =====
(setq-default ns-command-modifier 'meta) ; Commande est Meta
(setq-default ns-option-modifier 'none) ; Option est Option
```

- ESC est la touche Échap (⌘) et est équivalente à Meta ;
- SPC est la barre d'espace ;
- RET est la touche Entrée (↵).
- ▶ Toutes les fonctionnalités de Emacs correspondent à une commande pouvant être tapée dans le *minibuffer*. M-x démarre l'invite de commande.
- ▶ Le caractère ~ représente le dossier vers lequel pointe la variable d'environnement \$HOME (Linux, OS X) ou %HOME% (Windows). C'est le dossier par défaut de Emacs.
- ▶ La barre oblique (/) est utilisée pour séparer les dossiers dans les chemins d'accès aux fichiers, même sous Windows.
- ▶ En général, il est possible d'appuyer sur TAB dans le *minibuffer* pour compléter les noms de fichiers ou de commandes.

A.5 Commandes de base

Emacs comporte une pléthore de commandes, il serait donc futile de tenter d'en faire une liste exhaustive ici. Nous nous contenterons de mentionner les commandes les plus importantes regroupées par tâche.

Pour débiter, il est utile de suivre le Tour guidé de Emacs¹ et de lire le tutoriel de Emacs, que l'on démarre avec C-h t.

1. <http://www.gnu.org/software/emacs/tour/> ou cliquer sur le lien dans l'écran d'accueil.

A.5.1 Les essentielles

- M-x** démarrer l'invite de commande
- C-g** bouton de panique : annuler, quitter ! Presser plus d'une fois au besoin.

A.5.2 Manipulation de fichiers

Entre parenthèses, le nom de la commande Emacs correspondante. On peut entrer cette commande dans le *minibuffer* au lieu d'utiliser le raccourci clavier.



On remarquera qu'il n'existe pas de commande «nouveau fichier» dans Emacs. Pour créer un nouveau fichier, il suffit d'ouvrir un fichier n'existant pas.

- C-x C-f** ouvrir un fichier (find-file)
- C-x C-s** sauvegarder (save-buffer)
- C-x C-w** sauvegarder sous (write-file)
- C-x k** fermer un fichier (kill-buffer)
-
- C-_** annuler (pratiquement illimité) ; aussi **C-x u** (undo)
-
- C-s** recherche incrémentale avant (isearch-forward)
- C-r** Recherche incrémentale arrière (isearch-backward)
- M-%** rechercher et remplacer (query-replace)

A.5.3 Sélection de texte, copier, coller, couper

- C-SPC** débute la sélection (set-mark-command)
- C-w** couper la sélection (kill-region)
- M-w** copier la sélection (kill-ring-save)
- C-y** coller (yank)
- M-y** remplacer le dernier texte collé par la sélection précédente (yank-pop)
- Il est possible d'utiliser les raccourcis clavier usuels de Windows (**C-c**, **C-x**, **C-v**) et OS X (**⌘C**, **⌘X**, **⌘V**) en activant le mode CUA dans le menu Options.
 - On peut copier-coller directement avec la souris dans Windows en sélectionnant du texte puis en appuyant sur le bouton central (ou la molette) à l'endroit souhaité pour y copier le texte.

A.5.4 Manipulation de fenêtres

<code>C-x b</code>	changer de <i>buffer</i> (<i>switch-buffer</i>)
<code>C-x 2</code>	séparer l'écran en deux fenêtres (<i>split-window-vertically</i>)
<code>C-x 1</code>	conserver uniquement la fenêtre courante (<i>delete-other-windows</i>)
<code>C-x 0</code>	fermer la fenêtre courante (<i>delete-window</i>)
<code>C-x o</code>	aller vers une autre fenêtre lorsqu'il y en a plus d'une (<i>other-window</i>)

A.5.5 Manipulation de fichiers de script dans le mode ESS

<code>C-c C-n</code>	évaluer la ligne sous le curseur dans le processus R, puis déplacer le curseur à la prochaine expression (<i>ess-eval-line-and-step</i>)
<code>C-c C-r</code>	évaluer la région sélectionnée dans le processus R (<i>ess-eval-region</i>)
<code>C-c C-f</code>	évaluer le code de la fonction courante dans le processus R (<i>ess-eval-function</i>)
<code>C-c C-l</code>	évaluer le code du fichier courant en entier dans le processus R (<i>ess-load-file</i>)
<code>C-c C-v</code>	aide sur une commande R (<i>ess-display-help-on-object</i>)

A.5.6 Interaction avec l'invite de commande R

<code>M-p, M-n</code>	navigation dans l'historique des commandes (<i>previous-matching-history-from-input</i> , <i>next-matching-history-from-input</i>)
<code>C-c C-e</code>	remplacer la dernière ligne au bas de la fenêtre (<i>comint-show-maximum-output</i>)
<code>M-h</code>	sélectionner le résultat de la dernière commande (<i>mark-paragraph</i>)
<code>C-c C-o</code>	effacer le résultat de la dernière commande (<i>comint-delete-output</i>)
<code>C-c C-v</code>	aide sur une commande R (<i>ess-display-help-on-object</i>)
<code>C-c C-q</code>	terminer le processus R (<i>ess-quit</i>)

A.5.7 Consultation des rubriques d'aide de R

<code>h</code>	ouvrir une nouvelle rubrique d'aide, par défaut pour le mot se trouvant sous le curseur (<i>ess-display-help-on-object</i>)
<code>n, p</code>	aller à la section suivante (n) ou précédente (p) de la rubrique (<i>ess-skip-to-next-section</i> , <i>ess-skip-to-previous-section</i>)

- l évaluer la ligne sous le curseur ; pratique pour exécuter les exemples
(`ess-eval-line-and-step`)
- r évaluer la région sélectionnée (`ess-eval-region`)
- q retourner au processus ESS en laissant la rubrique d'aide visible
(`ess-switch-to-end-of-ESS`)
- x fermer la rubrique d'aide et retourner au processus ESS
(`ess-kill-buffer-and-go`)

A.6 Anatomie d'une session de travail (bis)

On reprend ici les étapes d'une session de travail type présentées à la section 1.6, mais en expliquant comment compléter chacune dans Emacs avec le mode ESS.

1. Lancer Emacs et ouvrir un fichier de script avec

`C-x C-f`

ou avec le menu

File|Open file...

En spécifiant un nom de fichier qui n'existe pas déjà, on se trouve à créer un nouveau fichier de script. S'assurer de terminer le nom des nouveaux fichiers par `.R` pour que Emacs reconnaisse automatiquement qu'il s'agit de fichiers de script R.

2. Démarrer un processus R à l'intérieur même de Emacs avec

`M-x R ↵`

Emacs demandera alors de spécifier de répertoire de travail (*starting data directory*). Accepter la valeur par défaut, par exemple

`~/ ↵`

ou indiquer un autre dossier. Un éventuel message de Emacs à l'effet que le fichier `.Rhistory` n'a pas été trouvé est sans conséquence et peut être ignoré.

3. Composer le code. Lors de cette étape, on se déplacera souvent du fichier de script à la ligne de commande afin d'essayer diverses expressions. On exécutera également des parties seulement du code se trouvant dans le fichier de script. Les commandes les plus utilisées sont alors

`C-x o` pour se déplacer d'une fenêtre à l'autre ;

`C-c C-n` pour exécuter une ligne du fichier de script ;

`C-c C-e` pour replacer la ligne de commande au bas de la fenêtre.

4. Sauvegarder le fichier de script :

`C-x C-s`

Les quatrième et cinquième caractères de la ligne de mode changent de `**` à `--`.

5. Sauvegarder si désiré l'espace de travail de R avec `save.image()`. On le répète, cela n'est habituellement pas nécessaire à moins que l'espace de travail ne contienne des objets importants ou longs à recréer.
6. Quitter le processus R avec

`C-c C-q`

Cette commande ESS se chargera de fermer tous les fichiers associés au processus R. On peut ensuite quitter Emacs en fermant l'application de la manière usuelle.

A.7 Configuration de l'éditeur

Une des grandes forces de Emacs est qu'à peu près chacune de ses facettes est configurable : couleurs, polices de caractère, raccourcis clavier, etc.

- ▶ Un utilisateur place ses commandes de configuration dans un fichier nommé `.emacs` (le point est important !) que Emacs lit au démarrage.
- ▶ Le fichier `.emacs` doit se trouver dans le dossier `~/`, c'est-à-dire dans le dossier de départ de l'utilisateur sous Linux et OS X, et dans le dossier référencé par la variable d'environnement `%HOME%` sous Windows.

A.8 Aide et documentation

Emacs possède son propre système d'aide très exhaustif, mais dont la navigation est peu intuitive selon les standards d'aujourd'hui. Consulter le menu `Help`.

Autrement, on trouvera les manuels de Emacs et de ESS en divers formats dans les sites respectifs des deux projets :

<http://www.gnu.org/software/emacs>

<http://ess.r-project.org>

Enfin, si le désespoir vous prend au cours d'une séance de codage intensive, vous pouvez toujours consulter le psychothérapeute Emacs. On le trouve, bien entendu, dans le menu `Help` !

B GNU Free Documentation License

Version 1.2, November 2002

Copyright ©2000, 2001, 2002 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

B.1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under

the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF

designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

B.2 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

B.3 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

B.4 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were

any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

B.5 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of

the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

B.6 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

B.7 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

B.8 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

B.9 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

B.10 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliographie

- Abelson, H., G. J. Sussman et J. Sussman. 1996, *Structure and Interpretation of Computer Programs*, 2^e éd., MIT Press, ISBN 0-26201153-0.
- Becker, R. A. 1994, «A brief history of S», cahier de recherche, AT&T Bell Laboratories. URL <http://cm.bell-labs.com/cm/ms/departments/sia/doc/94.11.ps>.
- Becker, R. A. et J. M. Chambers. 1984, *S: An Interactive Environment for Data Analysis and Graphics*, Wadsworth, ISBN 0-53403313-X.
- Becker, R. A., J. M. Chambers et A. R. Wilks. 1988, *The New S Language: A Programming Environment for Data Analysis and Graphics*, Wadsworth & Brooks/Cole, ISBN 0-53409192-X.
- Braun, W. J. et D. J. Murdoch. 2007, *A First Course in Statistical Programming with R*, Cambridge University Press, ISBN 978-0-52169424-7.
- Cameron, D., J. Elliott, M. Loy, E. S. Raymond et B. Rosenblatt. 2004, *Leaning GNU Emacs*, 3^e éd., O'Reilly, Sebastopol, CA, ISBN 0-59600648-9.
- Chambers, J. M. 1998, *Programming with Data: A Guide to the S Language*, Springer, ISBN 0-38798503-4.
- Chambers, J. M. 2000, «Stages in the evolution of S», URL <http://cm.bell-labs.com/cm/ms/departments/sia/S/history.html>.
- Chambers, J. M. 2008, *Software for Data Analysis: Programming with R*, Springer, ISBN 978-0-38775935-7.
- Chambers, J. M. et T. J. Hastie. 1992, *Statistical Models in S*, Wadsworth & Brooks/Cole, ISBN 0-53416765-9.
- Hornik, K. 2011, «The R FAQ», URL <http://cran.r-project.org/doc/FAQ/R-FAQ.html>, ISBN 3-90005108-9.

- Iacus, S. M., S. Urbanek et R. J. Goedman. 2011, «R for Mac OS X FAQ», URL <http://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html>.
- IEEE. 2003, *754-1985 IEEE Standard for Binary Floating-Point Arithmetic*, IEEE, Piscataway, NJ.
- Ihaka, R. et R. Gentleman. 1996, «R: A language for data analysis and graphics», *Journal of Computational and Graphical Statistics*, vol. 5, n° 3, p. 299–314.
- Ligges, U. 2003, «R-winedt», dans *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, édité par K. Hornik, F. Leisch et A. Zeileis, TU Wien, Vienna, Austria, ISSN 1609-395X. URL <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/>.
- Redd, A. 2010, «Introducing NppToR: R interaction for Notepad++», *R Journal*, vol. 2, n° 1, p. 62–63. URL http://journal.r-project.org/archive/2010-1/RJournal_2010-1.pdf.
- Ripley, B. D. et D. J. Murdoch. 2011, «R for Windows FAQ», URL <http://cran.r-project.org/bin/windows/base/rw-FAQ.html>.
- Venables, W. N. et B. D. Ripley. 2000, *S Programming*, Springer, New York, ISBN 0-38798966-8.
- Venables, W. N. et B. D. Ripley. 2002, *Modern Applied Statistics with S*, 4^e éd., Springer, New York, ISBN 0-38795457-0.
- Venables, W. N., D. M. Smith et R Development Core Team. 2011, *An Introduction to R*, R Foundation for Statistical Computing. URL <http://cran.r-project.org/doc/manuals/R-intro.html>.

Index

Les numéros de page en caractères gras indiquent les pages où les concepts sont introduits, définis ou expliqués.

!, **43**
!=, **43**
*, **43**
+, **43**
-, **43**
->, **14**, **43**
-Inf, **19**
/, **43**
:, **43**, **60**
;, **14**
<, **43**
<-, **14**, **42**, **43**
<=, **43**
=, **14**
==, **43**
>, **43**
>=, **43**
[, **27**
[<-, **27**
[[]], **25**, **25**
[], **21**, **23**, **25**, **27**
\$, **26**, **27**, **43**
\$<-, **27**
%*%, **43**
%/%, **43**
%%, **43**
%in%, **46**, **56**
%0%, **50**, **58**
&, **43**
&&, **43**
^, **42**, **43**
{ }, **15**

affectation, **13**
apply, **51**, **58**, **61**
array, **22**, **34**
array (classe), **22**
arrondi, **46**
as.data.frame, **26**
attach, **26**, **37**
attr, **19**, **31**
attribut, **19**
attributes, **19**, **31**, **32**

boucle, **51**
break, **51**, **60**
by, **10**, **55**
byrow, **22**, **44**

c, **20**
cbind, **24**, **26**, **34**, **39**, **57**
ceiling, **46**, **56**
character, **21**, **33**
character (mode), **17**, **21**
class, **32–34**, **36**

- class (attribut), **20**
- colMeans, **49**, **61**
- colSums, **48**, **58**, **61**
- compilé (langage), **2**
- complex, **33**
- complex (mode), **17**
- cos, **28**
- cummax, **48**, **57**
- cummin, **48**, **57**
- cumprod, **48**, **57**
- cumsum, **48**, **57**

- data, **31**, **44**, **55**
- data frame, **26**
- data.frame, **26**
- data.frame (classe), **26**
- density, **10**
- det, **49**
- detach, **26**, **37**
- diag, **39**, **49**, **57**
- diff, **47**, **57**
- différences, **47**
- dim, **32–34**, **36**, **38**, **57**
- dim (attribut), **20**, **21**, **22**
- dimension, **20**, **39**
- dimnames, **32**, **44**
- dimnames (attribut), **20**
- dossier de travail, voir répertoire de travail

- écart type, **47**
- else, **50**, **58–60**
- Emacs, **7**
 - C-_, **67**
 - C-g, **67**
 - C-r, **67**
 - C-s, **67**
 - C-SPC, **67**
 - C-w, **67**
 - C-x 0, **68**
 - C-x 1, **68**
 - C-x 2, **68**
 - C-x b, **68**
 - C-x C-f, **67**
 - C-x C-s, **67**, **70**
 - C-x C-w, **67**
 - C-x k, **67**
 - C-x o, **68**
 - C-x o , **69**
 - C-x u, **67**
 - C-y, **67**
- configuration, **70**
- M-%, **67**
- M-w, **67**
- M-x, **67**
- M-y, **67**
- nouveau fichier, **67**
- rechercher et remplacer, **67**
- sélection, **67**
- sauvegarder, **67**
- sauvegarder sous, **67**

- ESS, **7**
 - C-c C-e, **68**
 - C-c C-e , **69**
 - C-c C-f, **68**
 - C-c C-l, **68**
 - C-c C-n, **68**
 - C-c C-n , **69**
 - C-c C-o, **68**
 - C-c C-q, **68**, **70**
 - C-c C-r, **68**
 - C-c C-v, **68**
 - h, **68**
 - l, **69**
 - M-h, **68**
 - M-n, **68**
 - M-p, **68**
 - n, **68**
 - p, **68**
 - q, **69**

- r, 69
 - x, 69
- étiquette, 20, 39
- exists, 37
- exp, 28
- expression, 13
- expression, 30
- expression (mode), 17
- extraction, voir aussi indiçage
 - derniers éléments, 45
 - éléments différents, 45
 - premiers éléments, 45
- F, voir FALSE
- factorial, 61
- FALSE, 16
- floor, 46, 56
- fonction
 - appel, 43
- for, 51, 53, 54, 58, 59
- function (mode), 17
- gamma, 28, 61
- head, 45, 56
- if, 50, 53, 54, 58–60
- ifelse, 50
- indiçage
 - liste, 25, 39
 - matrice, 23, 26, 40
 - vecteur, 26, 39
- Inf, 19
- install.packages, 52
- interprété (langage), 2
- is.finite, 19
- is.infinite, 19
- is.na, 19, 31, 38, 59
- is.nan, 19
- is.null, 18
- lapply, 51
- length, 10, 17, 30–36, 38, 55–57
- lfactorial, 61
- lgamma, 61
- library, 52, 60
- list, 25, 30, 32, 35, 36
- list (mode), 17, 24
- liste, 24
- logical, 21, 33
- logical (mode), 17, 19, 21
- longueur, 18, 39
- ls, 11, 29
- mapply, 51
- match, 46, 56
- matrice, 61
 - diagonale, 49
 - identité, 49
 - inverse, 49
 - moyennes par colonne, 49
 - moyennes par ligne, 49
 - somme par colonne, 48
 - sommes par ligne, 48
 - transposée, 49
- matrix, 11, 22, 28, 33, 34, 36, 53, 55
- matrix (classe), 21
- max, 10, 11, 47, 57
- maximum
 - cumulatif, 48
 - d'un vecteur, 47
 - parallèle, 48
 - position dans un vecteur, 46
- mean, 19, 31, 47, 57
- median, 47, 57
- médiane, 47
- min, 10, 11, 47, 57
- minimum
 - cumulatif, 48
 - d'un vecteur, 47
 - parallèle, 48

- position dans un vecteur, 46
- mode, 17, 39
- mode, 16, 30, 31, 35, 36
- moyenne
 - arithmétique, 47
 - tronquée, 47
- NA, 19
- na.rm, 19, 31
- names, 32, 36–38
- names (attribut), 20
- NaN, 19
- nchar, 18, 30
- ncol, 11, 33, 34, 44, 48, 55, 58
- next, 51
- noms d'objets
 - conventions, 15
 - réservés, 16
- Notepad++, 8
- nrow, 11, 33, 44, 48, 55, 57
- NULL, 18, 20
- NULL (mode), 18
- numeric, 21, 31, 33, 38, 58, 59
- numeric (mode), 17, 21
- order, 45, 56
- outer, 49, 51, 58
- package, 51
- plot, 10, 32
- pmax, 48, 57, 58
- pmin, 48, 57
- print, 53, 54, 58–60
- prod, 47, 50, 57, 58
- produit, 47
 - cumulatif, 48
 - extérieur, 49
- q, 8
- quantile, 47
- quantile, 47, 57
- Répertoire de travail, 9
- répertoire de travail, 9
- rang, 45
- range, 47, 57
- rank, 45, 56
- rbind, 24, 26, 34, 39
- renverser un vecteur, 45
- rep, 10, 44, 55, 58, 60
- repeat, 51, 59
- répétition de valeurs, 44
- replace, 37, 57
- rev, 45, 56, 57
- rm, 11
- rnorm, 10
- round, 11, 46, 56
- row.names, 36
- rowMeans, 49, 61
- rowSums, 48, 58, 61
- runif, 10, 11
- S, 1, 2
- S+, 1
- S-PLUS, 1
- sample, 32, 37, 57, 61
- sapply, 51
- save.image, 4, 8, 70
- Scheme, 2
- sd, 47, 57
- search, 51, 60
- seq, 10, 30, 36, 44, 55, 60
- sin, 28
- solve, 11, 49, 57
- somme, 47
 - cumulative, 48
- sort, 45, 55
- suite de nombres, 44
- sum, 19, 47, 57, 58
- summary, 47, 57
- switch, 51

T, voir TRUE
t, 11, 49, 57
tableau, 61
tail, 45, 56
tri, 45
TRUE, 16
trunc, 46, 57
typeof, 17

unique, 45, 56
unlist, 26, 36

var, 47, 57
variance, 47
vecteur, 20, 41
vector, 33, 35
vide, voir NULL

which, 45, 56
which.max, 46, 56
which.min, 46, 56
while, 51, 59
WinEdt, 8

ISBN
978-2-98