Introduction à la programmation en S

Introduction à la programmation en S

Vincent Goulet

École d'actuariat, Université Laval

Version 0.99

Version 1.0 :	10 janvier 2006
Version 0.99 :	29 novembre 2005
Version 0.9 :	9 novembre 2005
Version 0.8 :	16 septembre 2005

Le code source LaTeX de ce document est disponible à l'adresse

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

ou en communiquant directement avec l'auteur.

ISBN 2-9809136-0-X
Dépôt légal – Bibliothèque nationale du Québec, 2006
Dépôt légal – Bibliothèque et Archives Canada, 2006

Préface

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. Cet ouvrage se concentre plutôt sur l'apprentissage du langage de programmation sous-jacent aux diverses fonctions statistiques, le S.

Ce texte est la somme de notes et d'exercices de différents cours donnés par l'auteur à l'École d'actuariat de l'Université Laval depuis septembre 2003. 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 viennent plus tard dans le cursus universitaire et c'est alors que les concepts des chapitres 7, 8 et 9 sont étudiés.

Les cours d'introduction au langage S sont donnés à raison d'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/pub/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 moins familier avec ces notions.

Les chapitres 7 (fonctions d'optimisation), 8 (régression linéaire) et 9 (séries chronologiques) sont structurés de manière plus classique, notamment parce que le texte y est en prose. Ces chapitres comportent néanmoins des sections d'exemples.

Le texte prend partie 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. On y traite également des générateurs de nombres aléatoires et de la planification d'une étude de simulation en S.

Dans la mesure du possible, cet ouvrage tâche de présenter les environnements S-Plus et R en parallèle et en soulignant leurs différences, lorsqu'elles surviennent. Les informations propres à S-Plus ou R sont d'ailleurs identifiées 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 2.2.0 (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 d'ailleurs pourquoi on réfère fréquemment au livre de Venables & 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, novembre 2005

Table des matières

1 Présentation du langage S

1.1 Le langage S

Le S est un langage pour «programmer avec des données» développé chez Bell Laboratories (anciennement propriété de AT&T, maintenant de Lucent Technologies).

- Ce n'est pas seulement un «autre» environnement statistique (comme SPSS ou SAS, par exemple), mais bien un langage de programmation complet et autonome.
- Le S est inspiré de plusieurs langages, dont l'APL et le Lisp :
  - interprété (et non compilé) ;
  - sans déclaration obligatoire des variables ;
  - basé sur la notion de vecteur ;
  - particulièrement puissant pour les applications mathématiques et statistiques (et donc actuarielles).

1.2 Les moteurs S

Il existe quelques «moteurs» ou dialectes du langage S.

- Le plus connu est S-Plus, un logiciel commercial de Insightful Corporation. (Bell Labs octroie à Insightful la licence exclusive de leur système S.)
- R, ou GNU S, est une version libre (*Open Source*) «*not unlike S*».

S-Plus et R constituent tous deux des environnements intégrés de manipulation de données, de calcul et de préparation de graphiques.

1.3 Où trouver de la documentation

S-Plus est livré avec quatre livres (disponibles en format PDF depuis le menu `Help` de l'interface graphique), mais aucun ne s'avère vraiment utile pour apprendre le langage S.

Plusieurs livres — en versions papier ou électronique, gratuits ou non — ont été publiés sur S-Plus et/ou R. On trouvera des listes exhaustives dans les sites de Insightful et du projet R :

- `http://www.insightful.com/support/splusbooks.asp`
- `http://www.r-project.org` (dans la section `Documentation`).

De plus, Venables & Ripley (2000, 2002) constituent des références sur le langage S devenues au cours des dernières années des standards *de facto*.

1.4 Interfaces pour S-Plus et R

Provenant du monde Unix, tant S-Plus que R sont d'abord et avant tout des applications en ligne de commande (`sqpe.exe` et `rterm.exe` sous Windows).

- S-Plus possède toutefois une interface graphique élaborée permettant d'utiliser le logiciel sans trop connaître le langage de programmation.
- R dispose également d'une interface graphique rudimentaire sous Windows et Mac OS.
- L'édition sérieuse de code S bénéficie cependant grandement d'un bon éditeur de texte.
- À la question 6.2 de la foire aux questions (FAQ) de R, «Devrais-je utiliser R à l'intérieur de Emacs ?», la réponse est : «Oui, définitivement.» Nous partageons cet avis, aussi ce document supposera-t-il que S-Plus ou R sont utilisés à l'intérieur de GNU Emacs avec le mode ESS.
- Autre option : WinEdt (partagiciel) avec l'ajout R-WinEdt.

1.5 Installation de Emacs avec ESS

Il n'existe pas de procédure d'installation similaire aux autres applications Windows pour Emacs. L'installation n'en demeure pas moins très simple : il suffit de décompresser un ensemble de fichiers au bon endroit.

- Pour une installation simplifiée de Emacs et ESS, consulter le site Internet

  `http://vgoulet.act.ulaval.ca/pub/emacs/`

  On y trouve une version modifiée de GNU Emacs et des instructions d'installation détaillées.
- L'annexe A présente les plus importantes commandes à connaître pour utiliser Emacs et le mode ESS.

1.6 Démarrer et quitter S-Plus ou R

On suppose ici que S-Plus ou R sont utilisés à l'intérieur de Emacs.

R
- Pour démarrer R à l'intérieur de Emacs :

  `M-x R RET`

puis spécifier un dossier de travail (voir la section 1.8). Une console R est ouverte dans un *buffer* nommé `*R*`.

- Pour démarrer S-Plus sous Windows, la procédure est similaire, sauf que la commande à utiliser est S+

  ```
  M-x Sqpe RET
  ```

  Consulter l'annexe B pour de plus amples détails.
- Pour quitter, deux options sont disponibles :
  1. Taper `q()` à la ligne de commande.
  2. Dans Emacs, faire `C-c C-q`. ESS va alors s'occuper de fermer le processus S ainsi que tous les *buffers* associés à ce processus.

1.7 Stratégies de travail

Il existe principalement deux façons de travailler avec S-Plus et R.

1. Le code est virtuel et les objets sont réels. C'est l'approche qu'encouragent les interfaces graphiques, mais aussi la moins pratique à long terme. On entre des expressions directement à la ligne de commande pour les évaluer immédiatement.

   ```
   > 2 + 3
   [1] 5
   > -2 * 7
   [1] -14
   > exp(1)
   [1] 2.718282
   > log(exp(1))
   [1] 1
   ```

   Les objets créés au cours d'une session de travail sont sauvegardés. Par contre, le code utilisé pour créer ces objets est perdu lorsque l'on quitte S-Plus ou R, à moins de sauvegarder celui-ci dans des fichiers.
2. Le code est réel et les objets sont virtuels. C'est l'approche que nous favorisons. Le travail se fait essentiellement dans des fichiers de script (de simples fichiers de texte) dans lesquels sont sauvegardées les expressions (parfois complexes !) et le code des fonctions personnelles. Les objets sont créés au besoin en exécutant le code. Emacs permet ici de passer efficacement des fichiers de script à l'exécution du code :
   i) Démarrer un processus S-Plus (`M-x Sqpe`) ou R (`M-x R`) et spécifier le dossier de travail.

ii) Ouvrir un fichier de script avec `C-x C-f`. Pour créer un nouveau fichier, ouvrir un fichier n'existant pas.

iii) Positionner le curseur sur une expression et faire `C-c C-n` pour l'évaluer.

iv) Le résultat apparaît dans le *buffer* `*S+6*` ou `*R*`.

1.8 Gestion des projets ou environnements de travail

S-Plus et R ont une manière différente, mais tout aussi particulière de sauvegarder les objets créés au cours d'une session de travail.

- Tous deux doivent travailler dans un dossier et non avec des fichiers individuels.

S+
- Dans S-Plus, tout objet créé au cours d'une session de travail est sauvegardé de façon permanente sur le disque dur dans le sous-dossier `__Data` du dossier de travail.

R
- Dans R, les objets créés sont conservés en mémoire jusqu'à ce que l'on quitte l'application ou que l'on enregistre le travail avec la commande `save.image()`. L'environnement de travail (*workspace*) est alors sauvegardé dans le fichier `.RData` dans le dossier de travail.

Le dossier de travail est déterminé au lancement de l'application.

- Avec Emacs et ESS on doit spécifier le dossier de travail à chaque fois que l'on démarre un processus S-Plus ou R.
- Les interfaces graphiques permettent également de spécifier le dossier de travail.

S+
  - Dans l'interface graphique de S-Plus, choisir `General Settings` dans le menu `Options`, puis l'onglet `Startup`. Cocher la case `Prompt for project folder`. Consulter également le chapitre 13 du guide de l'utilisateur de S-Plus.

R
  - Dans l'interface graphique de R, le plus simple consiste à changer le dossier de travail à partir du menu `Fichier|Changer le répertoire courant...`. Consulter aussi la *R for Windows FAQ*.

1.9 Consulter l'aide en ligne

Les rubriques d'aide des diverses fonctions disponibles dans S-Plus et 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
```

- Dans Emacs, `C-c C-v foo RET` ouvrira la rubrique d'aide de la fonction `foo` dans un nouveau *buffer*.
- Il existe plusieurs touches de raccourcis utiles pour la consultation des rubriques d'aide (voir la carte de référence ESS).
- Entre autres, la touche `l` permet d'exécuter ligne par ligne les exemples se trouvant à la fin de chaque rubrique d'aide.

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

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

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

1.11 Exercices

1.1 Démarrer un processus S-Plus ou R à l'intérieur de Emacs.

1.2 Exécuter un à un les exemples de la section précédente. Une version électronique du code de cette section est disponible dans le site mentionné dans la préface.

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 lignes d'exemples.

1.4 Lire le chapitre 1 de Venables & Ripley (2002) et exécuter les commandes de l'exemple de session de travail de la section 1.3. Bien que davantage orienté vers les application statistiques que vers la programmation, cet exemple démontre quelques unes des possibilités du langage S.

2 Bases du langage S

Ce chapitre présente les bases du langage S, soit les notions d'expression et d'affectation, la description d'un objet S et les manières de créer les objets les plus usuels lorsque le S est utilisé comme langage de programmation.

2.1 Commandes S

Toute commande S est soit une *affectation*, soit une *expression*.

- 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 `<-` (ou `->`).

```
> a <- 5
> a

[1] 5

> b <- a
> b

[1] 5
```

- Éviter d'utiliser l'opérateur `=` pour affecter une valeur à une variable, puisqu'il ne fonctionne que dans certaines situations seulement.

- Dans S-Plus (mais plus dans R depuis la version 1.8.0), on peut également affecter avec le caractère «_», mais cet emploi est fortement découragé puisqu'il rend le code difficile à lire. Dans le mode ESS de Emacs, taper ce caractère génère carrément ␣<-␣.

Astuce. Il arrive fréquemment que l'on souhaite affecter le résultat d'un calcul dans un objet et en même temps voir ce résultat. Pour ce faire, placer l'affectation entre parenthèses (l'opération d'affectation devient alors une nouvelle expression) :

```
> (a <- 2 + 3)

[1] 5
```

2.2 Conventions pour les noms d'objets

Les caractères permis pour les noms d'objets sont les lettres a–z, A–Z, les
R chiffres 0–9 et le point «.». Le caractère «_» est maintenant permis dans R, mais son utilisation est découragée.

- Les noms d'objets ne peuvent commencer par un chiffre.
- Le S est sensible à la casse, ce qui signifie que `foo`, `Foo` et `FOO` 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 :

  `Inf`, `NA`, `NaN`, `NULL`

  `break`, `else`, `for`, `function`, `if`, `in`, `next`, `repeat`, `return`, `while`.

S+ - Dans S-Plus 6.1 et plus, `T` et `TRUE` (vrai), ainsi que `F` et `FALSE` (faux) sont également des noms réservés.

R - Dans R, les noms `TRUE` et `FALSE` sont également réservés. Les variables `T` et `F` prennent par défaut les valeurs `TRUE` et `FALSE`, respectivement, mais peuvent être réaffectées.

```
> T

[1] TRUE

> TRUE <- 3

Erreur dans TRUE <- 3 : membre gauche de
l'assignation (do_set) incorrect

> (T <- 3)

[1] 3
```

`numeric`	nombres réels
`complex`	nombres complexes
`logical`	valeurs booléennes (vrai/faux)
`character`	chaînes de caractères
`function`	fonction
`list`	données quelconques

TAB. 2.1: Modes disponibles et contenus correspondants

2.3 Les objets S

Tout dans le langage S est un objet, même les fonctions et les opérateurs. Les objets possèdent au minimum un *mode* et une *longueur*.

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

```
> 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
```

- Certains objets sont également dotés d'un ou plusieurs *attributs*.

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 modes disponibles en S. À chacun de ces modes correspond une fonction du même nom servant à créer un objet de ce mode.

2.3.2 Longueur

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

- La longueur 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.

```
> v <- "actuariat"
> length(v)
```

```
[1] 1
```

`class`	affecte le comportement d'un objet
`dim`	dimensions des matrices et tableaux
`dimnames`	étiquettes des dimensions des matrices et tableaux
`names`	étiquettes des éléments d'un objet

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

```
> v <- c("a", "c", "t", "u", "a", "r", "i",
+     "a", "t")
> length(v)

[1] 9
```

- Un objet peut être de longueur 0 et doit alors être interprété comme un contenant vide.

```
> v <- numeric(0)
> length(v)

[1] 0
```

2.3.3 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.

2.3.4 L'objet spécial `NA`

`NA` est fréquemment utilisé pour représenter les données manquantes.

- Son mode est `logical`.
- 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 fonction `is.na` permet de tester si les éléments d'un objet sont `NA` ou non.

2.3.5 L'objet spécial `NULL`

`NULL` représente «rien», ou le vide.

- Son mode est `NULL`.
- Sa longueur est 0.
- 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.

2.4 Vecteurs

En S, à peu de choses près, *tout* est un vecteur. (Il n'y a pas de notion de scalaire.)

- Dans un vecteur simple, tous les éléments doivent être du même mode.
- 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
```

- 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`).
- L'indiçage 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'indiçage de vecteurs et matrices.

2.5 Matrices et tableaux

Une matrice ou, de façon plus générale, un tableau (*array*) n'est rien d'autre qu'un vecteur doté d'un attribut `dim`. À l'interne, une matrice est donc stockée sous forme de vecteur.

- La fonction de base pour créer des matrices est `matrix`.
- La fonction de base pour créer des tableaux est `array`.
- ⚠ *Important* : 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.

```
> 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
```

- On extrait les éléments d'une matrice en précisant leurs positions sous la forme (ligne, colonne) dans la matrice, ou encore leurs positions dans le vecteur sous-jacent.

```
> (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

> m[3]

[1] 45
```

- 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

Une liste est un type de vecteur spécial dont les éléments peuvent être de n'importe quel mode, y compris le mode `list` (ce qui permet d'emboîter des listes).

- La fonction de base pour créer des listes est `list`.
- Il est généralement préférable de nommer les éléments d'une liste. Il est en effet plus simple et sûr d'extraire les éléments par leur étiquette.
- L'extraction des éléments d'une liste peut se faire de deux façons :
  1. avec des doubles crochets `[[ ]]`;
  2. par leur étiquette avec `nom.liste$etiquette.element`.
- La fonction `unlist` convertit une liste en un vecteur simple. Attention, cette fonction peut être destructrice si la structure de la liste est importante.

2.7 *Data frames*

Les vecteurs, matrices, tableaux (*arrays*) et listes sont les types d'objets les plus fréquemment utilisés en S pour la programmation de fonctions personnelles ou la simulation. L'analyse de données — la régression linéaire, par exemple — repose toutefois davantage sur les *data frames*.

- Un *data frame* est une liste de classe `data.frame` dont tous les éléments sont de la même longueur.
- Généralement représenté sous forme d'un tableau à deux dimensions (visuellement similaire à une matrice). Chaque élément de la liste sous-jacente correspond à une colonne.

- On peut donc obtenir les étiquettes des colonnes avec la fonction `names` (ou `colnames` dans R). Les étiquettes des lignes sont quant à elles obtenues avec `row.names` (ou `rownames` dans R). R
- Plus général qu'une matrice puisque les colonnes peuvent être de modes différents (`numeric`, `complex`, `character` ou `logical`).
- Peut être indicé à la fois comme une liste et comme une matrice.
- Créé avec la fonction `data.frame` ou `as.data.frame` (pour convertir une matrice en *data frame*, par exemple).
- Les fonctions `rbind` et `cbind` peuvent être utilisées pour ajouter des lignes ou des colonnes, respectivement.
- 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`.

Ce type d'objet est moins important lors de l'apprentissage du langage de programmation.

2.8 Indiçage

L'indiçage des vecteurs et matrices a déjà été brièvement présenté aux sections 2.4 et 2.5. Cette section contient plus de détails sur cette procédure des plus communes lors de l'utilisation du langage S. On se concentre toutefois sur le traitement des vecteurs. Se référer également à Venables & Ripley (2002, section 2.3) pour de plus amples détails.

Il existe quatre façons d'indicer un vecteur dans le langage S. Dans tous les cas, l'indiçage 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.

```
> letters[c(1:3, 22, 5)]

[1] "a" "b" "c" "v" "e"
```

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

```
> letters[c(-(1:3), -5, -22)]

 [1] "d" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
[13] "q" "r" "s" "t" "u" "w" "x" "y" "z"
```

3. Avec un vecteur booléen. Le vecteur d'indiçage 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.

```
> letters > "f" & letters < "q"

 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
 [9]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[17] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE

> letters[letters > "f" & letters < "q"]

 [1] "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
```

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

```
> x <- c(Rouge = 2, Bleu = 4, Vert = 9, Jaune = -5)
> x[c("Bleu", "Jaune")]

 Bleu Jaune
    4    -5
```

2.9 Exemples

```
###
### LES OBJETS S
###

## LONGUEUR

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

## La longueur d'une chaîne de caractères est 1...
( a <- "foobar" )
length(a)

## ... à moins que l'objet ne compte plusieurs chaînes de
## caractères.
( a <- c("f", "o", "o", "b", "a", "r") )
length(a)

## La longueur peut être 0, auquel cas on a un objet vide,
## mais qui existe.
( a <- numeric(0) )
length(a)                  # l'objet 'a' existe...
a[1] <- 1                  # on peut donc affecter sa première
                           # valeur
b[1] <- 1                  # opération impossible, l'objet 'b'
                           # n'existe pas
```

```
## ATTRIBUTS

## Attribut 'class'. Selon la classe d'un objet, certaines
## fonctions (dites «fonctions génériques») vont se comporter
## différemment.
x <- sample(1:100, 10)     # échantillon aléatoire de 10
                           # nombres entre 1 et 100
class(x)                   # classe de l'objet
plot(x)                    # graphique pour cette classe
class(x) <- "ts"           # 'x' est maintenant une série
                           # chronologique
plot(x)                    # graphique pour les séries
                           # chronologiques

## Attribut 'dim'. Si l'attribut 'dim' compte deux valeurs,
## l'objet est traité comme une matrice. S'il en compte plus
## de deux, l'objet est traité comme un tableau (array).
a <- matrix(1:12, nrow=3, ncol=4) # matrice 3 x 4
dim(a)                     # vecteur de deux éléments
length(dim(a))             # nombre de dimensions de 'a'
class(a)                   # objet considéré comme une matrice
length(a)                  # à l'interne, 'a' est un vecteur

a <- array(1:24, c(2, 3, 4))  # tableau 2 x 3 x 4
dim(a)                     # vecteur de 3 éléments
length(dim(a))             # nombre de dimensions de 'a'
class(a)                   # objet considéré comme un tableau
length(a)                  # à l'interne, 'a' est un vecteur

## Attribut 'dimnames'. Permet d'assigner des étiquettes (ou
## noms) aux dimensions d'une matrice ou d'un tableau.
( a <- matrix(1:12, nrow=3) ) # matrice 3 x 4
dimnames(a)                # pas d'étiquettes par défaut
letters                    # objet prédéfini
LETTERS                    # idem
dimnames(a) <- list(letters[1:3], LETTERS[1:4])
                           # 'dimnames' est une liste de
                           # deux éléments
a                          # joli
dimnames(a)                # noms stockés dans une liste

## Attributs 'names'. Similaire à 'dimnames', mais pour les
## éléments d'un vecteur ou d'une liste.
( a <- 1:4 )               # vecteur de quatre éléments
names(a)                   # pas d'étiquettes par défaut
names(a) <- c("Rouge", "Vert", "Bleu", "Jaune")
                           # attribution d'étiquettes
a                          # joli
names(a)                   # extraction des étiquettes
```

```
( a <- c("Rouge"=1, "Vert"=2, "Bleu"=3, "Jaune"=4) )
                          # autre façon de faire
names(a)                  # même résultat

## L'OBJET SPÉCIAL 'NA'
a <- c(65, NA, 72, 88)    # traité comme une valeur
mean(a)                   # tout calcul donne NA...
mean(a, na.rm=TRUE)       # ... à moins d'éliminer les NA
                          # avant de faire le calcul

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

###
### VECTEURS
###
a <- c(-1, 2, 8, 10)      # création d'un vecteur
names(a) <- letters[1:length(a)] # attribution d'étiquettes
a[1]                      # extraction par position
a["c"]                    # extraction par étiquette
a[-2]                     # élimination d'un élément

## Les fonctions 'numeric', 'logical' et 'character'
## consistuent la manière «officielle» d'initialiser des
## contenants vides.
( a <- numeric(10) )      # vecteur initialisé avec des 0
( a <- logical(10) )      # vecteur initialisé avec des FALSE
( a <- character(10) )    # vecteur initialisé avec ""

## 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(5, TRUE, FALSE)         # conversion au mode 'numeric'
c(5, "z")                 # conversion au mode 'character'
c(TRUE, "z")              # conversion au mode 'character'
c(5, TRUE, "z")           # conversion au mode 'character'

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

## Deux façons de créer des matrices: à l'aide de la fonction
## 'matrix', ou en ajoutant un attribut 'dim' à un vecteur.
( a <- matrix(1:12, nrow=3, ncol=4) ) # avec 'matrix'
class(a); length(a); dim(a)# vecteur à deux dimensions

a <- 1:12                 # vecteur simple
```

```
dim(a) <- c(3, 4)          # ajout d'un attribut 'dim'
class(a); length(a); dim(a)# même résultat!

a[1, 3]                    # l'élément en position (1, 3)...
a[7]                       # ... est le 7e élément du vecteur
a[1,]                      # première ligne
a[,2]                      # deuxième colonne

matrix(1:12, nrow=3, byrow=TRUE) # remplir par ligne

## On procède exactement de la même façons avec les tableaux,
## sauf que le nombre de dimensions est plus élevé. Attention:
## les tableaux sont remplis de la première à la dernière
## dimension, dans l'ordre.
( a <- array(1:60, 3:5) )  # tableau 3 x 4 x 5
class(a); length(a); dim(a)# vecteur à trois dimensions
a[1, 3, 2]                 # l'élément (1, 3, 2)...
a[19]                      # ... est le 19e élément du vecteur

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

###
### LISTES
###

## La liste est l'objet le plus général en S puisqu'il peut
## contenir des objets de n'importe quel mode et longueur.
( a <- list(joueur=c("V", "C", "C", "M", "A"),
            score=c(10, 12, 11, 8, 15),
            expert=c(FALSE, TRUE, FALSE, TRUE, TRUE),
            bidon=2) )
mode(a)                    # mode 'list'
length(a)                  # quatre éléments

## 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.
a[[1]]                      # premier élément de la liste...
mode(a[[1]])                # ... un vecteur
a[1]                        # aussi le premier élément...
mode(a[1])                  # ... mais une sous liste...
length(a[1])                # ... d'un seul élément
a[[2]][1]                   # 1er élément du 2e élément

## Les éléments d'une liste étant généralement nommés (c'est
## une bonne habitude à prendre!), il est généralement plus
## simple et sûr d'extraire les éléments d'une liste par leur
## étiquette.
a$joueur                    # équivalent à a[[1]]
a$score[1]                  # équivalent à a[[2]][1]
a[["expert"]]               # aussi valide, mais peu usité

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

## 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).
a <- a[1:6]                 # éliminer la fonction
unlist(a)                   # remarquer la conversion
unlist(a, 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.
( dframe <- data.frame(Noms=c("Pierre", "Jean", "Jacques"),
                       Age=c(42, 34, 19),
                       Fumeur=c(TRUE, TRUE, FALSE)) )
mode(dframe)                # un data frame est une liste...
dim(dframe)                 # ... avec un attribut 'dim'
class(dframe)               # ... et de classe 'data.frame'

## 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
```

```
## indicer. La fonction 'attach' permet de rendre les
## colonnes individuelles visibles.  Une fois terminé,
## 'detach' masque les colonnes.
exists("Noms")
attach(dframe)
exists("Noms")
Noms
detach(dframe)
exists("Noms")

###
### INDIÇAGE
###

## Les opérations suivantes illustrent les différentes
## techniques d'indiçage d'un vecteur. 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) )

## Toutes les techniques d'indiçage peuvent aussi servir à
## affecter de nouvelles valeurs à une partie d'un
## vecteur. Ici, les éléments de 'x' correspondant aux
## positions dans le vecteur 'y' sont remplacés par des
## données manquantes.
x[y] <- NA
x

## La fonction 'is.na' permet de tester si une valeur est NA
## ou non.
is.na(x)

## Élimination des données manquantes.
( x <- x[!is.na(x)] )

## Tout le vecteur 'x' sauf les trois premiers éléments.
x[-(1:3)]

## Extraction par chaîne de caractères.
x[c("a", "k", "t")]
```

2.10 Exercices

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

```
[[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 `obs` un vecteur contenant les valeurs suivantes :

```
> obs

 [1]  3 19 13  3 13 11  9  8 15  5 20 15  9 11 18  4
[17]  1 12  9 12
```

Écrire une expression S 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 `mat` une matrice 10×7 obtenue aléatoirement avec

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

Écrire une expression S permettant d'obtenir les éléments demandés ci-dessous.

(a) L'élément $(4, 3)$ de la matrice.

(b) Le contenu de la sixième ligne de la matrice.

(c) Les première et quatrième colonnes de la matrice (simultanément).

(d) Les lignes de la matrice dont le premier élément est supérieur à 50.

3 Opérateurs et fonctions

Ce chapitre présente les principaux opérateurs arithmétiques, fonctions mathématiques et structures de contrôles offertes par le S. La liste est évidemment loin d'être exhaustive, surtout étant donné l'évolution rapide du langage. Un des meilleurs endroits pour connaître 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 S 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)

 [1]  4  8  8 12 12 11 11 15 15 19
```

^ ou **	puissance
-	changement de signe
* /	multiplication, division
+ -	addition, soustraction
%*% %% %/%	produit matriciel, modulo, division entière
< <= == >= > !=	plus petit, plus petit ou égal, égal, plus grand ou égal, plus grand, différent de
!	négation logique
& |	«et» logique, «ou» logique

TAB. 3.1: Principaux opérateurs mathématiques, en ordre décroissant de priorité des opérations

```
> 1:10 + rep(1:5, 2) + rep(c(2, 4), 5)

 [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 cela 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 :
la longueur de l'objet le plus long n'est pas un
multiple de la longueur de l'objet le plus court in:
1:10 + c(2, 4, 6)
```

3.2 Opérateurs

On trouvera dans le tableau 3.1 les opérateurs mathématiques et logiques les plus fréquemment employés, en ordre décroissant de priorité des opérations. Le tableau 3.1 de Venables & Ripley (2002) contient une liste plus complète.

3.3 Appels de fonctions

Ou comment spécifier les arguments d'une fonction interne ou personnelle.

- 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 dans 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, surtout après les deux ou trois premiers arguments.
- L'ordre des arguments étant important, il est 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.

3.3.1 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 la «valeur» `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
```

La section 3.6 de Venables & Ripley (2002) contient de plus amples détails.

3.4 Quelques fonctions utiles

Le langage S compte un très grand nombre de fonctions internes. La terminologie du système de classement de ces fonctions et la façon de les charger en mémoire diffèrent quelque peu entre S-Plus et R.

Dans S-Plus, les fonctions sont classées dans des *sections* d'une bibliothèque (*library*). La bibliothèque principale se trouve dans le dossier `library` du S+

dossier d'installation de S-Plus. Au démarrage, plusieurs sections de la bibliothèque de base (dont, entre autres, `main`, `splus` et `stat`) sont immédiatement chargées en mémoire, avec comme conséquence qu'un très grand nombre de fonctions sont immédiatement disponibles.

R Dans R, un ensemble de fonctions est appelé un package (terme non traduit). Par défaut, R charge en mémoire quelques packages de la bibliothèque seulement, ce qui économise l'espace mémoire et accélère le démarrage. En revanche, on a plus souvent recours à la fonction `library` pour charger de nouveaux packages.

Nous utiliserons dorénavant la terminologie de R pour référer à un élément de la bibliothèque.

Cette section présente quelques unes seulement des nombreuses fonctions disponibles dans S-Plus et R. On s'y concentre sur les fonctions de base les plus souvent utilisées pour programmer en S et pour manipuler des données.

3.4.1 Manipulation de vecteurs

`seq`	génération de suites de nombres
`rep`	répétition de valeurs ou de vecteurs
`sort`	tri en ordre croissant ou décroissant
`order`	positions dans un vecteur des valeurs en ordre croissant ou décroissant
`rank`	rang des éléments d'un vecteur en ordre croissant ou décroissant
`rev`	renverser un vecteur
`head`	extraction des n premières valeurs (R seulement)
`tail`	extraction des n dernières valeurs (R seulement)
`unique`	extraction des éléments différents d'un vecteur

3.4.2 Recherche d'éléments dans un vecteur

`which`	positions des valeurs `TRUE` dans un vecteur booléen
`which.min`	position du minimum dans un vecteur
`which.max`	position du maximum dans un vecteur
`match`	position de la première occurrence d'un élément dans un vecteur
`%in%`	appartenance d'une ou plusieurs valeurs à un vecteur

3.4.3 Arrondi

`round`	arrondi à un nombre défini de décimales
`floor`	plus grand entier inférieur ou égal à l'argument
`ceiling`	plus petit entier supérieur ou égal à l'argument

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

3.4.4 Sommaires et statistiques descriptives

`sum`, `prod`	somme et produit des éléments d'un vecteur
`diff`	différences entre les éléments d'un vecteur
`mean`	moyenne arithmétique et moyenne tronquée
`var`, `sd`	variance et écart type (versions sans biais)
`min`, `max`	minimum et maximum d'un vecteur
`range`	vecteur contenant le minimum et le maximum d'un vecteur
`median`	médiane empirique
`quantile`	quantiles empiriques
`summary`	statistiques descriptives d'un échantillon

3.4.5 Sommaires cumulatifs et comparaisons élément par élément

`cumsum`, `cumprod`	somme et produit cumulatif d'un vecteur
`cummin`, `cummax`	minimum et maximum cumulatif
`pmin`, `pmax`	minimum et maximum en parallèle, c'est-à-dire élément par élément entre deux vecteurs ou plus

3.4.6 Opérations sur les matrices

`t`	transposée
`solve`	avec un seul argument (une matrice carrée) : inverse d'une matrice ; avec deux arguments (une matrice carrée et un vecteur) : solution du système d'équation $\mathbf{Ax} = \mathbf{b}$
`diag`	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 $p \times p$
`nrow`, `ncol`	nombre de lignes et de colonnes d'une matrice
`rowSums`, `colSums`	sommes par ligne et par colonne, respectivement, des éléments d'une matrice ; voir aussi la fonction `apply` à la section 6.2
`rowMeans`, `colMeans`	moyennes par ligne et par colonne, respectivement, des éléments d'une matrice ; voir aussi la fonction `apply` à la section 6.2
`rowVars`, `colVars`	variance par ligne et par colonne des éléments d'une matrice (S-Plus seulement)

3.4.7 Produit extérieur

La fonction `outer`, dont la syntaxe est

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

applique la fonction `FUN` (`prod` par défaut) entre chacun des éléments de `X` et chacun des éléments de `Y`.

- 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

On se contente, ici, de mentionner les structures de contrôle disponibles en S. Se reporter à Venables & Ripley (2002, section 3.8) pour plus de détails sur leur utilisation.

3.5.1 Exécution conditionnelle

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

Si *`condition`* est vraie, *`branche.vrai`* est exécutée, et *`branche.faux`* sinon. 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 vectorisée qui remplace chaque élément `TRUE` du vecteur *`condition`* par l'élément correspondant de *`expression.vrai`* et chaque élément `FALSE` par 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`*`, ...)`

Utilisé plutôt rarement.

3.5.2 Boucles

Les boucles sont et doivent être utilisées avec parcimonie en S, car elles sont généralement inefficaces (particulièrement avec S-Plus). Dans la majeure partie des cas, il est possible de vectoriser les calcul pour éviter les boucles explicites, ou encore de s'en remettre aux fonctions `apply`, `lapply` et `sapply` (section 6.2) pour faire 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 par ailleurs de nombres.

`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 Exemples

```
###
### OPÉRATEURS
###

## Seuls les opérateurs %%, %/% et logiques sont illustrés
## ici. Premièrement, l'opérateur modulo retourne le reste
## d'une division.
5 %% 1:5
10 %% 1:15

## 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

## Dans les opérations logiques impliquant les opérateurs &, |
## et !, le nombre zéro est traité comme FALSE et tous les
## autres nombres comme TRUE.
0:5 & 5:0
0:5 | 5:0
!0:5

## 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]

###
### 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
```

```
a <- 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 déterminé automatiquement
seq(-2, by=0.5, along=a) # même longueur que 'a'

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

## Classement en ordre croissant ou décroissant.
sort(a)                  # classement en ordre croissant
sort(a, 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 dans 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(a)                  # regarder dans le blanc des yeux
a[order(a)]               # équivalent à 'sort(a)'

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

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

## --- R ---
head(a, 3)                # trois premiers éléments de 'a'
tail(a, 3)                # trois derniers éléments de 'a'
## ---------

## Équivalents S-Plus
a[1:3]                    # trois premiers éléments de 'a'
a[(length(a)-2):length(a)] # trois derniers éléments de 'a'
rev(rev(a)[1:3])          # avec petits vecteurs seulement

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

## RECHERCHE D'ÉLÉMENTS DANS UN VECTEUR
which(a >= 30)            # positions des éléments >= 30
which.min(a)              # position du minimum
```

```
which.max(a)               # position du maximum
match(20, a)               # position du premier 20 dans 'a'
match(c(20, 30), a)        # aussi pour plusieurs valeurs
60 %in% a                  # 60 appartient à 'a'
70 %in% a                  # 70 n'appartient pas à 'a'

## ARRONDI
( a <- c(-21.2, -pi, -1.5, -0.2, 0, 0.2, 1.7823, 315) )
round(a)                   # arrondi à l'entier
round(a, 2)                # arrondi à la seconde décimale
round(a, -1)               # arrondi aux dizaines
ceiling(a)                 # plus petit entier supérieur
floor(a)                   # plus grand entier inférieur
trunc(a)                   # troncature des décimales

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

## SOMMAIRES CUMULATIFS ET COMPARAISONS ÉLÉMENTS PAR ÉLÉMENT
( a <- sample(1:20, 6) )
( b <- sample(1:20, 6) )
cumsum(a)                  # somme cumulative de 'a'
cumprod(b)                 # produit cumulatif de 'b'
rev(cumprod(rev(b)))       # produit cumulatif renversé
cummin(a)                  # minimum cumulatif
cummax(b)                  # maximum cumulatif
pmin(a, b)                 # minimum élément par élément
pmax(a, b)                 # 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 ligne par ligne
colSums(A)                  # sommes colonne 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
a <- c(1, 2, 4, 7, 10, 12)
b <- c(2, 3, 6, 7, 9, 11)
outer(a, b)                 # produit extérieur
a %o% b                     # équivalent plus court
outer(a, b, "+")            # «somme extérieure»
outer(a, b, "<=")           # toutes les comparaisons possibles
outer(a, b, pmax)           # idem

###
### STRUCTURES DE CONTRÔLE
###

## Pour illustrer les structures de contrôle, on fait un petit
## exemple tout à fait artificiel: un vecteur est rempli des
## nombres de 1 à 100, sauf les 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 que nous devrons 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 fin de la boucle,
## la boucle est exécutée au moins une fois. S'il faut faire
## un tour de passe passe 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
```

3.7 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 S 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 $1!, 2!, ..., 10!$

3.5 Trouver une relation entre `x`, `y`, `x %% y` et `x %/% y`, où `y != 0`.

3.6 Simuler un échantillon $\mathbf{x} = (x_1, x_2, x_3, ..., x_{20})$ avec la fonction `sample`. Écrire une expression S 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 S permettant d'effectuer les tâches demandées ci-dessous.

(a) Calculer la somme des éléments de chacunes des lignes de la matrice.

(b) Calculer la moyenne des éléments de chacunes des colonnes de la matrice.

(c) Calculer la valeur maximale prise par les éléments 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, 9.97, 9.93, 9.96, 9.99, 9.92, 9.94,
+     9.9, 9.86, 9.88, 9.87, 9.85, 9.91, 9.84,
+     9.89, 9.79, 9.8, 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 est survenu.

4 Exemples résolus

Ce chapitre propose de faire le point sur les concepts étudiés jusqu'à maintenant par le biais de quelques exemples résolus. On y met particulièrement en évidence les avantages de l'approche vectorielle du langage S.

La compréhension du contexte de ces exemples requiert quelques connaissances de base en mathématiques financières et en théorie des probabilités.

4.1 Calcul de valeurs présentes

Un prêt est remboursé par une série de cinq paiements, le premier dans un an. Trouver le montant du prêt pour chacune des hypothèses ci-dessous.

(a) Paiement annuel de 1 000, taux d'intérêt de 6 % effectif annuellement.

(b) Paiements annuels de 500, 800, 900, 750 et 1 000, taux d'intérêt de 6 % effectif annuellement.

(c) Paiements annuels de 500, 800, 900, 750 et 1 000, taux d'intérêt de 5 %, 6 %, 5,5 %, 6,5 % et 7 % effectifs annuellement.

Solution. De manière générale, la valeur présente d'une série de paiements $P_1, P_2, \ldots, P_n$ à la fin des années $1, 2, \ldots, n$ est

$$\sum_{j=1}^{n} \prod_{k=1}^{j} (1 + i_k)^{-1} P_j, \tag{4.1}$$

où i_k est le taux d'intérêt effectif annuellement durant l'année k. Lorsque le taux d'intérêt est constant au cours des n années, cette formule se simplifie en

$$\sum_{j=1}^{n} (1 + i)^{-j} P_j. \tag{4.2}$$

(a) Un seul paiement annuel, un seul taux d'intérêt. On utilise la formule (4.2) avec $P_j = P = 1\,000$.

```
> 1000 * sum((1 + 0.06)^(-(1:5)))

[1] 4212.364
```

(b) Différents paiements annuels, un seul taux d'intérêt : la formule (4.2) s'applique directement.

```
> sum(c(500, 800, 900, 750, 1000) * (1 +
+     0.06)^(-(1:5)))
```

```
[1] 3280.681
```

(c) Avec différents paiements annuels et différents taux d'intérêt, il faut employer la formule (4.1). Le produit des taux d'intérêts est obtenu avec la fonction `cumprod`.

```
> sum(c(500, 800, 900, 750, 1000)/cumprod(1 +
+     c(0.05, 0.06, 0.055, 0.065, 0.07)))
```

```
[1] 3308.521
```

□

4.2 Fonctions de probabilité

Calculer toutes ou la majeure partie des probabilités des deux lois de probabilité ci-dessous. Vérifier que la somme des probabilités est bien égale à 1.

(a) La distribution binomiale, dont la fonction de masse de probabilité est

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, \ldots, n.$$

(b) La distribution de Poisson, dont la fonction de masse de probabilité est

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, \ldots,$$

où $x! = x(x-1)\cdots 2 \cdot 1$.

Solution. Cet exemple est quelque peu artificiel dans la mesure où il existe, dans S-Plus et R, des fonctions internes pour calculer les principales caractéristiques des lois de probabilité les plus usuelles (voir l'annexe C).

(a) Solution pour le cas $n = 10$ et $p = 0{,}8$. Les coefficients binomiaux sont calculés avec la fonction `choose`.

```
> n <- 10
> p <- 0.8
> x <- 0:n
> choose(n, x) * p^x * (1 - p)^rev(x)
```

```
 [1] 0.0000001024 0.0000040960 0.0000737280
 [4] 0.0007864320 0.0055050240 0.0264241152
 [7] 0.0880803840 0.2013265920 0.3019898880
[10] 0.2684354560 0.1073741824
```

On vérifie les réponses obtenues avec la fonction interne `dbinom`.

```
> dbinom(x, n, prob = 0.8)
```

```
 [1] 0.0000001024 0.0000040960 0.0000737280
 [4] 0.0007864320 0.0055050240 0.0264241152
 [7] 0.0880803840 0.2013265920 0.3019898880
[10] 0.2684354560 0.1073741824
```

On vérifie enfin que les probabilités somment à 1.

```
> sum(choose(n, x) * p^x * (1 - p)^rev(x))
```

```
[1] 1
```

(b) La loi de Poisson ayant un support infini, on calcule les probabilités en $x = 0, 1, \ldots, 10$ seulement avec $\lambda = 5$. On calcule les factorielles avec la fonction `factorial`. On notera au passage que `factorial(x) == gamma(x + 1)`, où `gamma` calcule les valeurs de la fonction gamma

$$\Gamma(n) = \int_0^\infty x^{n-1} e^{-x}\, dx = (n-1)\Gamma(n-1),$$

avec $\Gamma(0) = 1$. Pour n entier, on a donc $\Gamma(n) = (n-1)!$.

```
> lambda <- 5
> x <- 0:10
> exp(-lambda) * (lambda^x/factorial(x))
```

```
 [1] 0.006737947 0.033689735 0.084224337
 [4] 0.140373896 0.175467370 0.175467370
 [7] 0.146222808 0.104444863 0.065278039
[10] 0.036265577 0.018132789
```

Vérification avec la fonction interne `dpois` :

```
> dpois(x, lambda)
```

```
 [1] 0.006737947 0.033689735 0.084224337
 [4] 0.140373896 0.175467370 0.175467370
 [7] 0.146222808 0.104444863 0.065278039
[10] 0.036265577 0.018132789
```

Pour vérifier que les probabilités somment à 1, il faudra d'abord tronquer le support infini de la Poisson à une «grande» valeur. Ici, 200 est suffisamment éloigné de la moyenne de la distribution, 5. Remarquer que le produit par $e^{-\lambda}$ est placé à l'extérieur de la somme pour ainsi faire un seul produit plutôt que 201.

```
> x <- 0:200
> exp(-lambda) * sum((lambda^x/factorial(x)))
```

```
[1] 1
```

□

4.3 Fonction de répartition de la loi gamma

La loi gamma est fréquemment utilisée pour la modélisation d'événements ne pouvant prendre que des valeurs positives et pour lesquels les petites valeurs sont plus fréquentes que les grandes. Par exemple, la loi gamma est parfois utilisée en sciences actuarielles pour la modélisation des montants de sinistres. Nous utiliserons la paramétrisation où la fonction de densité de probabilité est

$$f(x) = \frac{\lambda^\alpha}{\Gamma(\alpha)}\, x^{\alpha-1} e^{-\lambda x}, \quad x > 0, \tag{4.3}$$

où $\Gamma(\cdot)$ est la fonction gamma définie dans la solution de l'exemple précédent.

Il n'existe pas de formule explicite de la fonction de répartition de la loi gamma. Néanmoins, la valeur de la fonction de répartition d'une loi gamma de paramètre α entier et $\lambda = 1$ peut être obtenue à partir de la formule

$$F(x;\alpha,1) = 1 - e^{-x} \sum_{j=0}^{\alpha-1} \frac{x^j}{j!}. \tag{4.4}$$

(a) Évaluer $F(4;5,1)$.

(b) Évaluer $F(x;5,1)$ pour $x = 2,3,\ldots,10$ en une seule expression.

Solution. Le premier exercice est plutôt simple, alors que le second est plus compliqué qu'il n'y paraît au premier abord.

(a) Calcul de la fonction de répartition en une seule valeur de x et avec un paramètre α fixe.

```
> alpha <- 5
> x <- 4
> 1 - exp(-x) * sum(x^(0:(alpha - 1))/gamma(1:alpha))
[1] 0.3711631
```

Vérification avec la fonction interne `pgamma` :

```
> pgamma(x, alpha)
[1] 0.3711631
```

On peut aussi éviter de générer essentiellement la même suite de nombres à deux reprises en ayant recours à une variable intermédiaire. Au risque de rendre le code un peu moins lisible (mais plus compact !), l'affectation et le calcul final peuvent même se faire dans une seule expression.

```
> 1 - exp(-x) * sum(x^(-1 + (j <- 1:alpha))/gamma(j))
[1] 0.3711631
```

(b) Ici, la valeur de α demeure fixe, mais on doit calculer la valeur de la fonction de répartition en plusieurs points en une seule expression. C'est un travail pour la fonction `outer`.

```
> x <- 2:10
> 1 - exp(-x) * colSums(t(outer(x, 0:(alpha -
+     1), "^"))/gamma(1:alpha))
```

```
[1] 0.05265302 0.18473676 0.37116306 0.55950671
[5] 0.71494350 0.82700839 0.90036760 0.94503636
[9] 0.97074731
```

Vérification avec la fonction interne `pgamma` :

```
> pgamma(x, alpha)
```

```
[1] 0.05265302 0.18473676 0.37116306 0.55950671
[5] 0.71494350 0.82700839 0.90036760 0.94503636
[9] 0.97074731
```

□

4.4 Algorithme du point fixe

Trouver la racine d'une fonction g — c'est-à-dire le point x où $g(x) = 0$ — est un problème classique en mathématiques. Très souvent, il est possible de reformuler le problème de façon à plutôt chercher le point x où $f(x) = x$. La solution d'un tel problème est appelée *point fixe*.

L'algorithme du calcul numérique du point fixe d'une fonction $f(x)$ est très simple :

1. choisir une valeur de départ x_0 ;
2. calculer $x_n = f(x_{n-1})$;
3. répéter l'étape 2 jusqu'à ce que $|x_n - x_{n-1}| < \epsilon$ ou $|x_n - x_{n-1}|/|x_{n-1}| < \epsilon$.

Trouver, à l'aide de la méthode du point fixe, la valeur de i telle que

$$a_{\overline{10}|} = \frac{1 - (1+i)^{-10}}{i} = 8{,}21.$$

Solution. Puisque, d'une part, nous ignorons combien de fois la procédure itérative devra être répétée et que, d'autre part, il faut exécuter la procédure au moins une fois, le choix logique pour la structure de contrôle à utiliser dans cette procédure itérative est `repeat`. Nous verrons au chapitre 5 comment faire une fonction à partir de ce code.

```
> i <- 0.05
> repeat {
+     it <- i
+     i <- (1 - (1 + it)^(-10))/8.21
+     if (abs(i - it)/it < 1e-10)
+         break
+ }
> i
```

```
[1] 0.03756777
```

Vérification :

```
> (1 - (1 + i)^(-10))/i
[1] 8.21
```

□

4.5 Exercices

Dans chacun des exercices ci-dessous, écrire une expression S pour faire le calcul demandé. Parce qu'elles ne sont pas nécessaires, il est interdit d'utiliser des boucles.

4.1 Calculer la valeur présente d'une série de paiements fournie dans un vecteur `P` en utilisant les taux d'intérêt annuels d'un vecteur `i`.

4.2 Étant donné un vecteur d'observations $\mathbf{x} = (x_1, \ldots, x_n)$ et un vecteur de poids correspondants $\mathbf{w} = (w_1, \ldots, w_n)$, calculer la moyenne pondérée des observations,

$$\sum_{i=1}^{n} \frac{w_i}{w_\Sigma} x_i,$$

où $w_\Sigma = \sum_{i=1}^{n} w_i$. Tester l'expression avec les vecteurs de données

$$\mathbf{x} = (7, 13, 3, 8, 12, 12, 20, 11)$$

et

$$\mathbf{w} = (0{,}15,\ 0{,}04,\ 0{,}05,\ 0{,}06,\ 0{,}17,\ 0{,}16,\ 0{,}11,\ 0{,}09).$$

4.3 Soit un vecteur d'observations $\mathbf{x} = (x_1, \ldots, x_n)$. Calculer la moyenne harmonique de ce vecteur, définie comme

$$\frac{n}{\dfrac{1}{x_1} + \cdots + \dfrac{1}{x_n}}.$$

Tester l'expression avec les valeurs de l'exercice 4.2.

4.4 Calculer la fonction de répartition en $x = 5$ d'une loi de Poisson avec paramètre $\lambda = 2$, qui est donnée par

$$\sum_{k=0}^{5} \frac{2^k e^{-2}}{k!},$$

où $k! = 1 \cdot 2 \cdots k$.

4.5 (a) Calculer l'espérance d'une variable aléatoire X dont le support est $x = 1, 10, 100, ..., 1\,000\,000$ et les probabilités correspondant à chacun de ces points $\frac{1}{28}, \frac{2}{28}, ..., \frac{7}{28}$, respectivement.

(b) Calculer la variance de la variable aléatoire X définie en (a).

4.6 Calculer le taux d'intérêt nominal composé quatre fois par année, $i^{(4)}$, équivalent à un taux de $i = 6$ % effectif annuellement.

4.7 La valeur présente d'une série de n paiements de fin d'année à un taux d'intérêt i effectif annuellement est

$$a_{\overline{n}|} = v + v^2 + \cdots + v^n = \frac{1 - v^n}{i},$$

où $v = (1 + i)^{-1}$. Calculer en une seule expression, toujours sans boucle, un tableau des valeurs présentes de séries de $n = 1, 2, \ldots, 10$ paiements à chacun des taux d'intérêt effectifs annuellement $i = 0{,}05, 0{,}06, \ldots, 0{,}10$.

4.8 Calculer la valeur présente d'une annuité croissante de 1 $ payable annuellement en début d'année pendant 10 ans si le taux d'actualisation est de 6 %. Cette valeur présente est donnée par

$$I\ddot{a}_{\overline{10}|} = \sum_{k=1}^{10} kv^{k-1},$$

toujours avec $v = (1 + i)^{-1}$.

4.9 Calculer la valeur présente de la séquence de paiements 1, 2, 2, 3, 3, 3, 4, 4, 4, 4 si les paiements sont effectués en fin d'année et que le taux d'actualisation est de 7 %.

4.10 Calculer la valeur présente de la séquence de paiements définie à l'exercice 4.9 en supposant que le taux d'intérêt d'actualisation alterne successivement entre 5 % et 8 % à chaque année, c'est-à-dire que le taux d'intérêt est 5 %, 8 %, 5 %, 8 %, etc.

5 Fonctions définies par l'usager

La possibilité pour l'usager de définir facilement et rapidement de nouvelles fonctions — et donc des extensions au langage — est une des grandes forces du S.

5.1 Définition d'une fonction

On définit une nouvelle fonction de la manière suivante :

```
fun <- function(arguments) expression
```

où

- `fun` est le nom de la fonction (les règles pour les noms de fonctions étant les mêmes que pour tout autre objet) ;
- *`arguments`* est la liste des arguments, séparés par des virgules ;
- *`expression`* constitue le corps de la fonction, soit une liste d'expressions groupées entre accolades (nécessaires s'il y a plus d'une expression seulement).

5.2 Retourner des résultats

La plupart des fonctions sont écrites dans le but de retourner un résultat.

- Une fonction retourne tout simplement le résultat de la *dernière expression* du corps de la fonction.
- On évitera donc que la dernière expression soit une affectation, car la fonction ne retournera alors rien.
- On peut également utiliser explicitement la fonction `return` pour retourner un résultat, mais cela est rarement nécessaire.
- Lorsqu'une fonction doit retourner plusieurs résultats, il est en général préférable de le faire dans une liste nommée.

5.3 Variables locales et globales

Comme dans la majorité des langages de programmation, les concepts de variable locale et de variable globale existent en S.

- Toute variable définie dans une fonction est locale à cette fonction, c'est-à-dire
  - qu'elle n'apparaît pas dans l'espace de travail ;
  - qu'elle n'écrase pas une variable du même nom dans l'espace de travail.
- Il est possible de définir une variable dans l'espace de travail depuis une fonction avec l'opérateur d'affectation `<<-`. Il est très rare — et généralement non recommandé — de devoir recourir à de telles variables globales.
- On peut définir une fonction à l'intérieur d'une autre fonction. Cette fonction sera locale à la fonction dans laquelle elle est définie.

5.4 Exemple de fonction

Le code développé pour l'exemple de point fixe de la section 4.4 peut être intégré dans une fonction tel que montré à la figure 5.1.

- Le nom de la fonction est `fp`.
- La fonction compte cinq arguments : `k`, `n`, `start` et `TOL`.
- Les deux derniers arguments ont des valeurs par défaut de 0,05 et 10^{-10}, respectivement.
- La fonction retourne la valeur de la variable `i`.
- Avec Emacs et le mode ESS, positionner le curseur à l'intérieur de la fonction et soumettre le code d'une fonction à un processus S-Plus ou `R` avec `C-c C-f`.

5.5 Fonctions anonymes

Il est parfois utile de définir une fonction sans lui attribuer un nom — d'où la notion de *fonction anonyme*. Il s'agira en général de fonctions courtes utilisées dans une autre fonction. Par exemple, pour calculer la valeur de xy^2 pour toutes les combinaisons de x et y stockées dans des vecteurs du même nom, on pourrait utiliser la fonction `outer` ainsi :

```
> x <- 1:3
> y <- 4:6
> f <- function(x, y) x * y^2
> outer(x, y, f)
```

```
fp <- function(k, n, start=0.05, TOL=1E-10)
{
    ## Fonction pour trouver par la méthode du point
    ## fixe le taux d'intérêt auquel une série de 'n'
    ## paiements vaut 'k'.
    ##
    ## ARGUMENTS
    ##
    ##     k: la valeur présente des paiements
    ##     n: le nombre de paiements
    ## start: point de départ des itérations
    ##   TOL: niveau de précision souhaité
    ##
    ## RETOURNE
    ##
    ## Le taux d'intérêt

    i <- start
    repeat
    {
        it <- i
        i <- (1 - (1 + it)^(-n))/k
        if (abs(i - it)/it < TOL)
            break
    }
    i  # ou return(i)
}
```

FIG. 5.1: Exemple de fonction de point fixe

```
     [,1] [,2] [,3]
[1,]   16   25   36
[2,]   32   50   72
[3,]   48   75  108
```

Cependant, si la fonction `f` ne sert à rien ultérieurement, on peut simplement utiliser une fonction anonyme à l'intérieur de `outer` :

```
> outer(x, y, function(x, y) x * y^2)

     [,1] [,2] [,3]
[1,]   16   25   36
[2,]   32   50   72
[3,]   48   75  108
```

5.6 Débogage de fonctions

Nous n'abordons ici que les techniques les plus simples et naïves.

- Les simples erreurs de syntaxe sont les plus fréquentes (en particulier l'oubli de virgules). Lors de la définition d'une fonction, une vérification de la syntaxe est effectuée.
- Lorsqu'une fonction ne retourne pas le résultat attendu, placer des commandes `print` à l'intérieur de la fonction, de façon à pouvoir suivre les valeurs prises par les différentes variables.

  Par exemple, la modification suivante à la boucle de la fonction `fp` permet d'afficher les valeurs successives de la variable `i` et de détecter une procédure itérative divergente :

```
repeat
{
    it <- i
    i <- (1 - (1 + it)^(-n))/k
    print(i)
    if (abs((i - it)/it < TOL))
        break
}
```

- Avec Emacs et le mode ESS, la principale technique de débogage consiste à s'assurer que toutes les variables passées en arguments à une fonction existent dans l'espace de travail, puis à exécuter successivement les lignes de la fonction avec `C-c C-n`. Les interfaces graphiques de S-Plus et `R` ne permettent pas une telle procédure puisque la fenêtre d'édition de fonctions bloque l'accès à l'interface de commande.

5.7 Styles de codage

Si tous s'entendent que l'adoption qu'un style propre et uniforme favorise le développement et la lecture de code, il existe plusieurs chapelles dans le monde des programmeurs quant à la «bonne façon» de présenter et, surtout, d'indenter du code informatique.

Par exemple, Emacs reconnaît et supporte les styles de codage suivants, entre autres :

C++/Stroustrup

```
for (i in 1:10)
{
    expression
}
```

K&R (1TBS)

```
for (i in 1:10){
     expression
}
```

Whitesmith
```
for (i in 1:10)
     {
     expression
     }
```

GNU
```
for (i in 1:10)
  {
    expression
  }
```

- Pour des raisons générales de lisibilité et de popularité, le style C++, avec les accolades sur leurs propres lignes et une indentation de quatre (4) espaces est considéré standard en programmation en S.
- Pour utiliser ce style dans Emacs, faire

  ```
  M-x ess-set-style RET C++ RET
  ```

  une fois qu'un fichier de script est ouvert.
- Pour éviter de devoir répéter cette commande à chaque session de travail, créer ou éditer le fichier de configuration `.emacs` dans le dossier vers lequel pointe la variable d'environnement `HOME` et y placer les lignes suivantes :

  ```
  (add-hook 'ess-mode-hook
            (lambda () (ess-set-style 'C++)))
  ```

5.8 Exemples

```
### Premier exemple de fonction: la mise en oeuvre de
### l'algorithme du point fixe pour trouver le taux d'intérêt
### tel que a_angle{n} = k pour 'n' et 'k' donnés. Cette mise
### en oeuvre est peu générale puisqu'il faudrait modifier la
### fonction à chaque fois que l'on change la fonction f(x)
### dont on cherche le point fixe.
fp1 <- function(k, n, start=0.05, TOL=1E-10)
{
    i <- start
    repeat
    {
        it <- i
        i <- (1 - (1 + it)^(-n))/k
        if (abs(i - it)/it < TOL)
            break
    }
    i
}

fp1(7.2, 10)              # valeur de départ par défaut
fp1(7.2, 10, 0.06)        # valeur de départ spécifiée
```

```
i                           # les variables n'existent pas
start                       # dans l'espace de travail

### Second exemple: généralisation de la fonction 'fp1' où la
### fonction f(x) dont on cherche le point fixe (c'est-à-dire
### la valeur de 'x' tel que f(x) = x) est passée en
### argument. On peut faire ça? Bien sûr, puisqu'une fonction
### est un objet comme un autre en S. On ajoute également à
### la fonction un argument 'echo' qui, lorsque TRUE, fera
### en sorte d'afficher à l'écran les valeurs successives
### de 'x'.
fp2 <- function(FUN, start, echo=FALSE, TOL=1E-10)
{
    x <- start
    repeat
    {
        xt <- x

        if (echo)  # inutile de faire 'if (echo == TRUE)'
            print(xt)

        x <- FUN(xt)

        if (abs(x - xt)/xt < TOL)
            break
    }
    x
}

f <- function(i) (1 - (1+i)^(-10))/7.2 # définition de f(x)
fp2(f, 0.05)               # solution
fp2(f, 0.05, echo=TRUE)    # avec résultats intermédiaires
fp2(function(x) 3^(-x), start=0.5) # avec une fonction anonyme

### Troisième exemple: amélioration mineure à la fonction
### 'fp2'. Puisque la valeur de 'echo' ne change pas pendant
### l'exécution de la fonction, on peut éviter de refaire le
### test à chaque itération de la boucle. Une solution
### élégante consiste à utiliser un outil avancé du langage S:
### les expressions. On se contentera d'une illustration ici,
### sans entrer dans les détails.
fp3 <- function(FUN, start, echo=FALSE, TOL=1E-10)
{
    x <- start

    if (echo)
        expr <- expression(print(xt <- x))
    else
        expr <- expression(xt <- x)
```

```
    repeat
    {
        eval(expr)

        x <- FUN(xt)

        if (abs(x - xt)/xt < TOL)
            break
    }
    x
}

fp3(f, 0.05, echo=TRUE)    # avec résultats intermédiaires
fp3(function(x) 3^(-x), start=0.5) # avec une fonction anonyme

### La suite de Fibonacci (et son lien avec le nombre d'or) a
### été remise au goût du jour par le best seller «Code Da
### Vinci». Les valeurs de la suite de Fibonacci sont données
### par la fonction suivante:
###
###      f(0) = 0
###      f(1) = 1
###      f(n) = f(n - 1) + f(n - 2), n >= 2.
###
### Voici deux exemples de fonctions calculant la suite de
### Fibonacci. La première calcule les 'n' premières valeurs
### de la série.
fib1 <- function(n)
{
    res <- c(0, 1)
    for (i in 3:n)
        res[i] <- res[i - 1] + res[i - 2]
    res
}
fib1(10)
fib1(20)

### La fonction 'fib1' a un gros défaut: la taille de l'objet
### 'res' est constamment augmentée pour stocker une nouvelle
### valeur de la série. Cela coûte très cher en S et doit
### absolument être évité lorsque c'est possible (et ce l'est
### la plupart du temps). Quand on sait quelle sera la
### longueur d'un objet (comme c'est le cas ici), il vaut
### mieux créer un contenant vide de la bonne longueur et le
### remplir par la suite.
fib2 <- function(n)
{
    res <- numeric(n)      # contenant créé
    res[2] <- 1            # res[1] vaut déjà 0
    for (i in 3:n)
```

```
        res[i] <- res[i - 1] + res[i - 2]
    res
}
fib2(10)
fib2(20)

### A-t-on vraiment gagné quelque chose? Comparons le temps
### requis pour générer une longue suite de Fibonacci avec les
### deux fonctions. (En fait, le gain est beaucoup plus
### important avec R qu'avec S-Plus.)
sys.time(fib1(10000))      # S-Plus seulement
sys.time(fib2(10000))      # S-Plus seulement
system.time(fib1(10000))   # R seulement
system.time(fib2(10000))   # R seulement

### Second exemple basé sur le suite de Fibonacci: une
### fonction pour calculer non pas les 'n' premières valeurs
### de la suite, mais uniquement la 'n'ieme valeur.
###
### Mais il y a un mais: la fonction 'fib3' est truffée
### d'erreurs (de syntaxe, d'algorithmique, de conception). À
### vous de trouver les bogues. (Afin de préserver cet
### exemple, copier le code erroné plus bas ou dans un autre
### fichier avant d'y faire les corrections.)
fib3 <- function(nb)
{
    x <- 0
    x1 _ 0
    x2 <- 1
    while (n > 0))
x <- x1 + x2
x2 <- x1
x1 <- x
n <- n - 1
}
fib3(1)                    # devrait donner 0
fib3(2)                    # devrait donner 1
fib3(5)                    # devrait donner 3
fib3(10)                   # devrait donner 34
fib3(20)                   # devrait donner 4181
```

5.9 Exercices

5.1 La fonctions `var` calcule l'estimateur sans biais de la variance d'une population à partir de l'échantillon donné en argument. Écrire une fonction `variance` qui calculera l'estimateur biaisé ou sans biais selon que l'argument `biased` sera `TRUE` ou `FALSE`, respectivement. Le comportement par défaut de `variance` devrait être le même que celui de `var`. L'estimateur

sans biais de la variance à partir d'un échantillon $X_1, \ldots, X_n$ est

$$S_{n-1}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2,$$

alors que l'estimateur biaisé est

$$S_n^2 = \frac{1}{n} \sum_{i=1}^{n} (X_i - \bar{X})^2,$$

où $\bar{X} = n^{-1}(X_1 + \cdots + X_n)$.

5.2 Écrire une fonction `matrix2` qui, contrairement à la fonction `matrix`, remplira par défaut la matrice par ligne. La fonction *ne doit pas* utiliser `matrix`. Les arguments de la fonction `matrix2` seront les mêmes que ceux de `matrix`, sauf que l'argument `byrow` sera remplacé par `bycol`.

5.3 Écrire une fonction `phi` servant à calculer la fonction de densité de probabilité d'une loi normale centrée réduite, soit

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \quad -\infty < x < \infty.$$

La fonction devrait prendre en argument un vecteur de valeurs de x. Comparer les résultats avec ceux de la fonction `dnorm`.

5.4 Écrire une fonction `Phi` servant à calculer la fonction de répartition d'une loi normale centrée réduite, soit

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-y^2/2}\, dy, \quad -\infty < x < \infty.$$

Supposer, pour le moment, que $x \geqslant 0$. L'évaluation numérique de l'intégrale ci-dessus peut se faire avec l'identité

$$\Phi(x) = \frac{1}{2} + \phi(x) \sum_{n=0}^{\infty} \frac{x^{2n+1}}{1 \cdot 3 \cdot 5 \cdots (2n+1)}, \quad x \geqslant 0.$$

Utiliser la fonction `phi` de l'exercice 5.3 et tronquer la somme infinie à une grande valeur, 50 par exemple. La fonction ne doit pas utiliser de boucles, mais peut ne prendre qu'une seule valeur de x à la fois. Comparer les résultats avec ceux de la fonction `pnorm`.

5.5 Modifier la fonction `Phi` de l'exercice 5.4 afin qu'elle admette des valeurs de x négatives. Lorsque $x < 0$, $\Phi(x) = 1 - \Phi(-x)$. La solution simple consiste à utiliser une structure de contrôle `if ... else`, mais les curieux chercheront à s'en passer. Les plus ambitieux regarderont même du côté de la fonction `Recall` (Venables & Ripley 2000, page 49).

5.6 Généraliser maintenant la fonction de l'exercice 5.5 pour qu'elle prenne en argument un vecteur de valeurs de x. Ne pas utiliser de boucle. Comparer les résultats avec ceux de la fonction `pnorm`.

5.7 Sans utiliser l'opérateur `%*%`, écrire une fonction `prod.mat` qui effectuera le produit matriciel de deux matrices seulement si les dimensions de celles-ci le permettent. Cette fonction aura deux arguments (`mat1` et `mat2`) et devra tout d'abord vérifier si le produit matriciel est possible. Si celui-ci est impossible, la fonction retourne un message d'erreur.

(a) Utiliser une structure de contrôle `if ... else` et deux boucles.

(b) Utiliser une structure de contrôle `if ... else` et une seule boucle.

Dans chaque cas, comparer le résultat avec l'opérateur `%*%`.

5.8 Vous devez calculer la note finale d'un groupe d'étudiants à partir de deux informations : (1) une matrice contenant la note sur 100 de chacun des étudiants à chacune des évaluations, et (2) un vecteur contenant la pondération de chacune des évaluations. Un collègue a composé la fonction `notes.finales` ci-dessous afin de calculer la note finale de chacun des étudiants. Votre collègue vous mentionne toutefois que sa fonction est plutôt lente et inefficace pour de grands groupes d'étudiants. Vous décidez donc de modifier la fonction afin d'en réduire le nombre d'opérations et qu'elle n'utilise aucune boucle.

```
notes.finales <- function(notes, p)
{
    netud <- nrow(notes)
    neval <- ncol(notes)
    final <- (1:netud) * 0
    for(i in 1:netud)
    {
        for(j in 1:neval)
        {
            final[i] <- final[i] + notes[i, j] * p[j]
        }
    }
    final
}
```

5.9 Trouver les erreurs qui empêchent la définition de la fonction ci-dessous.

```
AnnuiteFinPeriode <- function(n, i)
{{
    v <- 1/1 + i)
    ValPresChaquePmt <- v^(1:n)
    sum(ValPresChaquepmt)
}
```

5.10 La fonction ci-dessous calcule la valeur des paramètres d'une loi normale, gamma ou Pareto à partir de la moyenne et de la variance, qui sont connues par l'utilisateur.

```
param <- function(moyenne, variance, loi)
{
    loi <- tolower(loi)
    if (loi == "normale")
        param1 <- moyenne
        param2 <- sqrt(variance)
        return(list(mean=param1, sd=param2))
    if (loi == "gamma")
        param2 <- moyenne/variance
        param1 <- moyenne * param2
        return(list(shape=param1, scale=param2))
    if (loi == "pareto")
        cte <- variance/moyenne^2
        param1 <- 2 * cte/(cte-1)
        param2 <- moyenne * (param1 - 1)
        return(list(alpha=param1, lambda=param2))
    stop("La loi doit etre une de \"normale\",
\"gamma\" ou \"pareto\"")
}
```

L'utilisation de la fonction pour diverses lois donne les résultats suivants :

```
> param(2, 4, "normale")

$mean
[1] 2

$sd
[1] 2

> param(50, 7500, "gamma")

Erreur dans param(50, 7500, "gamma") : Objet "param1"
non trouvé

> param(50, 7500, "pareto")

Erreur dans param(50, 7500, "pareto") : Objet "param1"
non trouvé
```

(a) Expliquer pour quelle raison la fonction se comporte ainsi.

(b) Appliquer les corrections nécessaires à la fonction pour que celle-ci puisse calculer les bonnes valeurs. (Les erreurs ne sont pas contenues dans les mathématiques de la fonction.) *Astuce* : tirer profit du moteur d'indentation de Emacs.

6 Concepts avancés

Ce chapitre traite de divers concepts et fonctions un peu plus avancés du langage S. Le lecteur intéressé à approfondir ses connaissances de ce langage pourra consulter Venables & Ripley (2000), en particulier les chapitre 3 et 4.

6.1 L'argument '...'

La mention '...' apparaît dans la définition de plusieurs fonction en S. Il ne faut pas voir là de la paresse de la part des rédacteurs des rubriques d'aide, mais bel et bien un argument formel dont '...' est le nom.

- Cet argument signifie qu'une fonction peut accepter un ou plusieurs autres arguments autres que ceux faisant partie de sa définition.
- Le contenu de l'argument '...' n'est ni pris en compte, ni modifié par la fonction.
- Il est généralement simplement passé tel quel à une autre fonction.
- Voir les définitions des fonctions `apply`, `lapply` et `sapply`, ci-dessous, pour des exemples.

6.2 Fonction `apply`

La fonction `apply` sert à appliquer une fonction quelconque sur une partie d'une matrice ou, plus généralement, d'un tableau. La syntaxe de la fonction est la suivante :

```
apply(X, MARGIN, FUN, ...),
```

où

- `X` est une matrice ou un tableau (*array*) ;
- `MARGIN` est un vecteur d'entiers contenant la ou les dimensions de la matrice ou du tableau sur lesquelles la fonction doit s'appliquer ;
- `FUN` est la fonction à appliquer ;
- '...' est un ensemble d'arguments supplémentaires, séparés par des virgules, à passer à la fonction `FUN`.

Lorsque `X` est une matrice, `apply` sert principalement à calculer des sommaires par ligne (dimension 1) ou par colonne (dimension 2) autres que la somme ou la moyenne (puisque les fonctions `rowSums`, `colSums`, `rowMeans` et `colMeans` existent pour ce faire).

- Utiliser la fonction `apply` plutôt que des boucles puisque celle-ci est plus efficace.
- Considérer les exemples suivants.

```
> (m <- matrix(sample(1:100, 20, rep = TRUE),
+     5, 4))

     [,1] [,2] [,3] [,4]
[1,]   54   33   30   17
[2,]    3   46   95   83
[3,]   47    6   56   58
[4,]   18   22   50   36
[5,]   41   41   77   31

> apply(m, 1, var)

[1]  235.0000 1718.9167  590.9167  211.6667
[5]  409.0000

> apply(m, 2, min)

[1]  3  6 30 17

> apply(m, 1, mean, trim = 0.2)

[1] 33.50 56.75 41.75 31.50 47.50
```

Puisqu'il n'existe pas de fonctions internes pour effectuer des sommaires sur des tableaux, il faut toujours utiliser la fonction `apply`. Si `X` est un tableau de plus de deux dimensions, alors l'argument passé à `FUN` peut être une matrice ou un tableau.

- Déterminants des cinq sous-matrices 4×4 d'un tableau $4 \times 4 \times 5$:

```
> arr <- array(sample(1:100, 80, rep = TRUE),
+     c(4, 4, 5))
> apply(arr, 3, det)

[1]  1178800 16153716 14298240 20093933  6934743
```

6.3 Fonctions `lapply` et `sapply`

Les fonctions `lapply` et `sapply` sont similaires à la fonction `apply` en ce qu'elles permettent d'appliquer une fonction aux éléments d'une structure — le vecteur ou la liste en l'occurence. Leur syntaxe est similaire :

```
lapply(X, FUN, ...)
sapply(X, FUN, ...)
```

- La fonction `lapply` applique une fonction `FUN` à tous les éléments d'un vecteur ou d'une liste `X` et retourne le résultat sous forme de liste.

```
> (v <- lapply(5:8, sample, x = 1:100))

[[1]]
[1] 18 58 96 22 10

[[2]]
[1] 42  2 75 24 81 93

[[3]]
[1]  32   2 100  64  80  15  84

[[4]]
[1] 97 25 32 52 11 34 74 46

> lapply(v, mean)

[[1]]
[1] 40.8

[[2]]
[1] 52.83333

[[3]]
[1] 53.85714

[[4]]
[1] 46.375
```

- La fonction `sapply` est similaire à `lapply`, sauf que le résultat est retourné sous forme de vecteur, si possible.

```
> sapply(v, mean)

[1] 40.80000 52.83333 53.85714 46.37500
```

- Si le résultat de chaque application de la fonction est un vecteur, alors `sapply` retourne une matrice, remplie comme toujours par colonne.

```
> (v <- lapply(rep(5, 3), sample, x = 1:100))

[[1]]
[1] 20 18 66  6 97

[[2]]
```

```
[1]  5 87 68  7 18

[[3]]
[1] 11  2 84 70 89
```

```
> sapply(v, sort)
```

```
     [,1] [,2] [,3]
[1,]    6    5    2
[2,]   18    7   11
[3,]   20   18   70
[4,]   66   68   84
[5,]   97   87   89
```

- Dans un grand nombre de cas, il est possible de remplacer les boucles `for` par l'utilisation de `lapply` ou `sapply`. On ne saurait donc trop insister sur l'importance de ces fonctions.

6.4 Fonction `mapply`

La fonction `mapply` est une version multidimensionnelle de `sapply`. Sa syntaxe est, essentiellement,

```
mapply(FUN, ...)
```

- Le résultat de `mapply` est l'application de la fonction `FUN` aux premiers éléments de tous les arguments contenus dans '`...`', puis à tous les seconds éléments, et ainsi de suite.
- Ainsi, si `v` et `w` sont des vecteurs, `mapply(FUN, v, w)` retourne sous forme de liste, de vecteur ou de matrice, selon le cas, `FUN(v[1], w[1])`, `FUN(v[2], w[2])`, etc.

```
> mapply(rep, 1:4, 4:1)
```

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

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

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

- Les éléments de '`...`' sont recyclés au besoin.

```
> mapply(seq, 1:6, 6:8)
```

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

[[2]]
[1] 2 3 4 5 6 7

[[3]]
[1] 3 4 5 6 7 8

[[4]]
[1] 4 5 6

[[5]]
[1] 5 6 7

[[6]]
[1] 6 7 8
```

6.5 Fonction `replicate`

La fonction `replicate`, propre à R, est une fonction enveloppante de `sapply` simplifiant la syntaxe pour l'exécution répétée d'une expression. R

- Son usage est particulièrement indiqué pour les simulations. Ainsi, on peut construire une fonction `fun` qui fait tous les calculs d'une simulation, puis obtenir les résultats pour, disons, 10 000 simulations avec

  ```
  > replicate(10000, fun(...))
  ```

- L'annexe D présente en détail différentes stratégies — dont l'utilisation de `replicate` — pour la réalisation d'études de simulation en S.

6.6 Classes et fonctions génériques

Tous les objets dans le langage S ont une classe. La classe est parfois implicite ou dérivée du mode de l'objet (consulter la rubrique d'aide de `class` pour de plus amples détails).

- Certaines fonctions, dites fonctions *génériques*, se comportent différemment selon la classe de l'objet donné en argument. Les fonctions génériques les plus fréquemment employées sont `print`, `plot` et `summary`.
- Une fonction générique possède une *méthode* correspondant à chaque classe qu'elle reconnaît et, généralement, une méthode `default` pour les autres objets. La liste des méthodes existant pour une fonction générique s'obtient avec `methods` :

  ```
  > methods(plot)
  ```

```
 [1] plot.acf*           plot.data.frame*
 [3] plot.Date*          plot.decomposed.ts*
 [5] plot.default        plot.dendrogram*
 [7] plot.density        plot.ecdf
 [9] plot.factor*        plot.formula*
[11] plot.hclust*        plot.histogram*
[13] plot.HoltWinters*   plot.isoreg*
[15] plot.lm             plot.medpolish*
[17] plot.mlm            plot.POSIXct*
[19] plot.POSIXlt*       plot.ppr*
[21] plot.prcomp*        plot.princomp*
[23] plot.profile.nls*   plot.spec
[25] plot.spec.coherency plot.spec.phase
[27] plot.stepfun        plot.stl*
[29] plot.table*         plot.ts
[31] plot.tskernel*      plot.TukeyHSD

   Non-visible functions are asterisked
```

- À chaque méthode `methode` d'une fonction générique `fun` correspond une fonction `fun.methode`. C'est donc la rubrique d'aide de cette dernière fonction qu'il faut consulter au besoin, et non celle de la fonction générique, qui contient en général peu d'informations.
- Il est intéressant de savoir que lorsque l'on tape le nom d'un objet à la ligne de commande pour voir son contenu, c'est la fonction générique `print` qui est appelée. On peut donc complètement modifier la représentation à l'écran du contenu d'un objet est créant une nouvelle classe et une nouvelle méthode pour la fonction `print`.

6.7 Exemples

```
###
### FONCTION 'apply'
###

## Création d'une matrice et d'un tableau à trois dimensions
## pour les exemples.
m <- matrix(sample(1:100, 20), nrow=4, ncol=5)
a <- array(sample(1:100, 60), dim=3:5)

## Les fonctions 'rowSums', 'colSums', 'rowMeans' et
## 'colMeans' sont des raccourcis pour des utilisations
## fréquentes de 'apply'.
rowSums(m)
apply(m, 1, sum)
colMeans(m)
```

```
apply(m, 2, mean)

## Puisqu'il n'existe pas de fonctions comme 'rowMax' ou
## 'colProds', il faut utiliser 'apply'.
apply(m, 1, max)           # maximum par ligne
apply(m, 2, prod)          # produit par colonne

## L'argument '...' de 'apply' permet de passer des arguments
## à la fonction FUN.
m[sample(1:20, 5)] <- NA   # ajout de données manquantes
apply(m, 1, var, na.rm=TRUE) # variance par ligne sans NA

## Lorsque 'apply' est utilisée sur un tableau, son résultat
## est de dimensions dim(X)[MARGIN].
apply(a, c(2, 3), sum)     # le résultat est une matrice
apply(a, 1, prod)          # le résultat est un vecteur

###
### FONCTIONS 'lapply' ET 'sapply'
###

## La fonction 'lapply' applique une fonction à tous les
## éléments d'une liste et retourne une liste, peu importe les
## dimensions des résultats. La fonction 'sapply' retourne un
## vecteur ou une matrice, si possible.
##
## Somme «interne» des éléments d'une liste.
( liste <- list(1:10,
                c(-2, 5, 6),
                matrix(3, 4, 5)) )
sum(liste)                 # erreur
lapply(liste, sum)         # sommes internes (liste)
sapply(liste, sum)         # sommes internes (vecteur)

## Création de la suite 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, ..., 1,
## 2, ..., 9, 10.
lapply(1:10, seq)          # le résultat est une liste
unlist(lapply(1:10, seq))  # le résultat est un vecteur

## Soit une fonction calculant la moyenne pondérée d'un
## vecteur. Cette fonction prend en argument une liste de deux
## éléments: 'donnees' et 'poids'.
fun <- function(liste)
    sum(liste$donnees * liste$poids)/sum(liste$poids)

## On peut maintenant calculer la moyenne pondérée de
## plusieurs ensembles de données réunis dans une liste
## itérée.
( a <- list(list(donnees=1:7,
                 poids=(5:11)/56),
```

```
            list(donnees=sample(1:100, 12),
                 poids=1:12),
            list(donnees=c(1, 4, 0, 2, 2),
                 poids=c(12, 3, 17, 6, 2))) )
sapply(a, fun)

###
### FONCTION 'mapply'
###

## Création de quatre échantillons aléatoires de taille 12.
a <- lapply(rep(12, 4), sample, x=1:100)

## Moyennes tronquées à 0, 10, 20 et 30%, respectivement, de
## ces quatre échantillons aléatoires.
mapply(mean, a, 0:3/10)

###
### FONCTION 'replicate'
###

## La fonction 'replicate' va répéter un certain nombre de
## fois une expression quelconque. Le principal avantage de
## 'replicate' par rapport à 'sapply' est qu'on n'a pas à se
## soucier des arguments à passer à une fonction.
##
## Par exemple, on veut simuler dix échantillons aléatoires
## indépendants de longueur 12. On peut utiliser 'sapply',
## mais la syntaxe n'est ni élégante, ni facile à lire
## (l'argument 'i' ne sert à rien).
sapply(rep(1, 10), function(i) sample(1:100, 12))

## En utilisant 'replicate', on sait tout de suite de quoi il
## s'agit. À noter que les échantillons se trouvent dans les
## colonnes de la matrice résultante.
replicate(10, sample(1:100, 12))

## Vérification que la moyenne arithmétique (bar{X}) est un
## estimateur sans biais de la moyenne de la loi normale. On
## doit calculer la moyenne de plusieurs échantillons
## aléatoires, puis la moyenne de toutes ces moyennes.
##
## On définit d'abord une fonction pour faire une simulation.
fun <- function(n, mean, sd)
    mean(rnorm(n, mean=mean, sd=sd))

## Avec 'replicate', on fait un grand nombre de simulations.
res <- replicate(10000, fun(100, 0, 1)) # 10000 simulations
hist(res)                  # distribution de bar{X}
mean(res)                  # moyenne de bar{X}
```

```
###
### CLASSES ET FONCTIONS GÉNÉRIQUES
###

## Afin d'illustrer l'utilisation des classes et des fonctions
## génériques, nous allons créer une classe 'toto' et une
## méthode de la fonction générique 'print' pour cette classe.
##
## Si la fonction 'print' est appelée avec un objet de mode
## 'numeric' et de classe 'toto', c'est le résultat de la
## fonction 'diag' qui est retourné. (Ne pas chercher un sens
## caché à tout ça, il n'y en a pas.)
##
## Définition de la nouvelle méthode.
print.toto <- function(x)
{
    if (mode(x) == "numeric")
    {
        cat("\n  Resultat de 'diag':\n")
        print(diag(x))
    }
    else
        print.default(x)
}

## Vérification que la méthode est disponible.
methods(print)

## Essai de la nouvelle méthode sur un scalaire.
x <- 4
class(x)                  # classe par défaut
x                         # méthode par défault
class(x) <- "toto"        # objet de classe 'toto'
x                         # méthode pour cette classe

## Essai de la nouvelle méthode sur un vecteur.
x <- 1:5
class(x)                  # classe par défaut
x                         # méthode par défault
class(x) <- "toto"        # objet de classe 'toto'
x                         # méthode pour cette classe

## Essai de la nouvelle méthode sur une matrice. Les matrices
## ont une classe "matrix" implicite.
x <- matrix(1:9, 3, 3)
class(x)                  # classe implicite
x                         # méthode par défaut
class(x) <- "toto"        # objet de classe 'toto'
x                         # méthode pour cette classe
```

```
## La nouvelle méthode ne fait rien de spécial pour les objets
## d'un mode autre que 'numeric'.
x <- letters
mode(x)
class(x) <- "toto"
x
```

6.8 Exercices

6.1 À l'exercice 2 du chapitre 4, on a calculé la moyenne pondérée d'un vecteur d'observations

$$X_w = \sum_{i=1}^{n} \frac{w_i}{w_\Sigma} X_i,$$

où $w_\Sigma = \sum_{i=1}^{n} w_i$. Si l'on a plutôt une matrice $n \times p$ d'observations X_{ij}, on peut définir les moyennes pondérées

$$X_{iw} = \sum_{j=1}^{p} \frac{w_{ij}}{w_{i\Sigma}} X_{ij}, \quad w_{i\Sigma} = \sum_{j=1}^{p} w_{ij}$$

$$X_{wj} = \sum_{i=1}^{n} \frac{w_{ij}}{w_{\Sigma j}} X_{ij}, \quad w_{\Sigma j} = \sum_{i=1}^{n} w_{ij}$$

et

$$X_{ww} = \sum_{i=1}^{n} \sum_{j=1}^{p} \frac{w_{ij}}{w_{\Sigma\Sigma}} X_{ij}, \quad w_{\Sigma\Sigma} = \sum_{i=1}^{n} \sum_{j=1}^{p} w_{ij}.$$

De même, on peut définir des moyennes pondérées calculées à partir d'un tableau de données X_{ijk} de dimensions $n \times p \times r$ dont la notation suit la même logique que ci-dessus. Écrire des expressions S pour calculer, sans boucle, les moyennes pondérées suivantes.

(a) X_{iw} en supposant une matrice de données $n \times p$.

(b) X_{wj} en supposant une matrice de données $n \times p$.

(c) X_{ww} en supposant une matrice de données $n \times p$.

(d) X_{ijw} en supposant un tableau de données $n \times p \times r$.

(e) X_{iww} en supposant un tableau de données $n \times p \times r$.

(f) X_{wjw} en supposant un tableau de données $n \times p \times r$.

(g) X_{www} en supposant un tableau de données $n \times p \times r$.

6.2 Générer les suites de nombres suivantes à l'aide d'une expression S. (Évidemment, il faut trouver un moyen de générer les suites sans simplement concaténer les différentes sous suites.)

(a) $0, 0, 1, 0, 1, 2, \ldots, 0, 1, 2, 3, \ldots, 10$.

(b) $10, 9, 8, \ldots, 2, 1, 10, 9, 8, \ldots 3, 2, \ldots, 10, 9, 10$

(c) $10, 9, 8, \ldots, 2, 1, 9, 8, \ldots, 2, 1, \ldots, 2, 1, 1$

6.3 La fonction de densité de probabilité et la fonction de répartition de la loi de Pareto de paramètres α et λ sont, respectivement,

$$f(x) = \frac{\alpha\lambda^\alpha}{(x+\lambda)^{\alpha+1}}$$

et

$$F(x) = 1 - \left(\frac{\lambda}{x+\lambda}\right)^\alpha.$$

La fonction suivante simule un échantillon aléatoire de taille n issu d'une distribution de Pareto de paramètres α et λ :

```
rpareto <- function(n, alpha, lambda)
    lambda * (runif(n)^(-1/alpha) - 1)
```

(a) Écrire une expression S permettant de simuler, en utilisant la fonction `rpareto` ci-dessus, cinq échantillons aléatoires de tailles 100, 150, 200, 250 et 300 d'une loi de Pareto avec $\alpha = 2$ et $\lambda = 5\,000$. Les échantillons aléatoires devraient être stockés dans une liste.

(b) On vous donne l'exemple suivant d'utilisation de la fonction `paste` :

```
> paste("a", 1:5, sep = "")
[1] "a1" "a2" "a3" "a4" "a5"
```

Nommer les éléments de la liste créée en (a) `echantillon1`, ..., `echantillon5`.

(c) Calculer la moyenne de chacun des échantillons aléatoires obtenus en (a). Retourner le résultat dans un vecteur.

(d) Évaluer la fonction de répartition de la loi de Pareto$(2, 5\,000)$ en chacune des valeurs de chacun des échantillons aléatoires obtenus en (a). Retourner les valeurs de la fonction de répartition en ordre croissant.

(e) Faire l'histogramme des données du cinquième échantillon aléatoire à l'aide de la fonction `hist`.

(f) Ajouter 1 000 à toutes les valeurs de tous les échantillons simulés en (a), ceci afin d'obtenir des observations d'une distribution de Pareto *translatée*.

6.4 Une base de données contenant toutes les informations sur les assurés est stockée dans une liste de la façon suivante :

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

```
$num.police
[1] 1001

$franchise
[1] 500

$nb.acc
[1] 1 2 1 2 0 1

$montants
[1]  1233.7867   754.0062  5341.2330  1638.7506
[5] 14444.6491  2016.8539  7176.3796
```

```
> x[[2]]

$num.police
[1] 1002

$franchise
[1] 250

$nb.acc
[1] 2 0 3 1 6 3 0

$montants
 [1]  5449.3392  2850.1634  4710.4497  7307.5198
 [5]  3049.5136  1773.3971 12556.5939  2141.2826
 [9]   823.0804  5384.6596  8123.8746  1814.9773
[13]  2133.6178 24039.2302 18158.8161
```

Ainsi, `x[[i]]` contient les informations relatives à l'assuré i. Sans utiliser de boucles, écrire une expression ou une fonction S qui permettra de calculer les quantités suivantes.

(a) La franchise moyenne dans le portefeuille.

(b) Le nombre annuel moyen de réclamations par assuré.

(c) Le nombre total de réclamations dans le portefeuille.

(d) Le montant moyen par accident dans le portefeuille.

(e) Le nombre d'assurés n'ayant eu aucune réclamation.

(f) Le nombre d'assurés ayant eu une seule réclamation dans leur première année.

(g) La variance du nombre total de sinistres.

(h) La variance du nombre de sinistres pour chaque assuré.

(i) La probabilité empirique qu'une réclamation soit inférieure à x (un scalaire) dans le portefeuille.

(j) La probabilité empirique qu'une réclamation soit inférieure à $\mathbf{x}$ (un vecteur) dans le portefeuille.

7 Fonctions d'optimisation

Les méthodes de bissection, du point fixe, de Newton–Raphson et consorts permettent de résoudre des équations à une variable de la forme $f(x) = 0$ ou $g(x) = x$. Il existe également des versions de ces méthodes pour les systèmes à plusieurs variables de la forme

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 0 \\ f_2(x_1, x_2, x_3) &= 0 \\ f_3(x_1, x_2, x_3) &= 0. \end{aligned}$$

De tels systèmes d'équations surviennent plus souvent qu'autrement lors de l'optimisation d'une fonction. Par exemple, en recherchant le maximum ou le minimum d'une fonction $f(x, y)$, on souhaitera résoudre le système d'équations

$$\begin{aligned} \frac{\partial}{\partial x} f(x, y) &= 0 \\ \frac{\partial}{\partial y} f(x, y) &= 0. \end{aligned}$$

En statistique, les fonctions d'optimisation sont fréquemment employées pour calculer numériquement des estimateurs du maximum de vraisemblance.

La grande majorité des suites logicielles de calcul comportent des outils d'optimisation de fonctions. Ce chapitre passe en revue les fonctions disponibles dans S-Plus et R.

7.1 Le package `MASS`

L'offre en fonctions d'optimisation est un des domaines où S-Plus et R diffèrent passablement. Il existe toutefois une option commune avec le package `MASS`.

Le package `MASS` (Venables & Ripley 2002) contient plusieurs fonctions utiles et de grande qualité. Les auteurs de ces fonctions contribuent activement au développement de R et de S-Plus et, tel que mentionné au chapitre 1, leurs livres sur le langage S (Venables & Ripley 2000, 2002) constituent des références de choix.

Le package MASS est distribué autant avec S-Plus (depuis au moins la version 6.1) que R. On peut aussi le télécharger gratuitement depuis l'URL

`http://www.stats.ox.ac.uk/pub/MASS4/Software.html`

Pour accéder aux fonctions du package, il suffit de le charger en mémoire avec la commande

```
> library(MASS)
```

7.2 Fonctions d'optimisation disponibles

Les fonctions d'optimisation disponibles dans S-Plus et R sont les suivantes.

7.2.1 uniroot

La fonction `uniroot` recherche la racine d'une fonction `f` entre les points `lower` et `upper`. C'est la fonction de base pour trouver la solution (unique) de l'équation $f(x) = 0$.

Exemple 7.1. *Trouver la racine de la fonction* $f(x) = x - 2^{-x}$ *dans l'intervalle* $[0, 1]$.

Solution.

```
> uniroot(function(x) x - 2^(-x), lower = 0,
+     upper = 1)

$root
[1] 0.6411922

$f.root
[1] 9.310346e-06

$iter
[1] 3

$estim.prec
[1] 6.103516e-05
```

□

7.2.2 polyroot

La fonction `polyroot` calcule toutes les racines (complexes) du polynôme $\sum_{i=0}^{n} a_i x^i$. Le premier argument est le vecteur des coefficients $a_0, a_1, \ldots, a_n$, dans cet ordre.

Exemple 7.2. *Trouver les racines du polynôme* $x^3 + 4x^2 - 10$.

Solution.

```
> polyroot(c(-10, 0, 4, 1))

[1]  1.365230-0.000000i -2.682615+0.358259i
[3] -2.682615-0.358259i
```

□

7.2.3 `optimize`

La fonction `optimize` recherche le maximum ou minimum local d'une fonction `f` entre les points `lower` et `upper`.

Exemple 7.3. *Trouver l'extremum de la fonction de densité de la loi bêta de paramètres* $\alpha = 3$ *et* $\beta = 2$.

Solution. On sait que l'extremum se trouve dans l'intervalle $[0, 1]$.

```
> f <- function(x) dbeta(x, 3, 2)
> optimize(f, lower = 0, upper = 1, maximum = TRUE)

$maximum
[1] 0.6666795

$objective
[1] 1.777778
```

□

7.2.4 `ms`

La fonction `ms`, minimise une somme. C'est une des principales fonction d'optimisation de S-Plus. Elles est utile, par exemple, pour minimiser la valeur négative d'une fonction de log-vraisemblance, $-l(\theta) = -\sum_{i=1}^{n} \ln f(x_i; \theta)$. Son utilisation est toutefois compliquée par l'usage de formules (voir la section 8.2) et de *data frames* (section 2.7). S+

Exemple 7.4. *Calculer les estimateurs du maximum de vraisemblance des paramètres* α *et* λ *de la distribution gamma dont la densité est donnée à l'équation* (4.3) *à la page 40 à partir de l'échantillon aléatoire*

```
> x

 [1] 2.2557923 2.6291918 2.1579953 5.2925777
 [5] 0.8625360 0.6744605 1.5091443 1.0829637
 [9] 2.5340812 1.9135480
```

Solution. On cherche à minimiser $-l(\alpha, \lambda) = -\sum_{i=1}^{n} \ln f(x_i; \alpha, \lambda)$, donc l'argument de `ms` doit être $-\ln f(x_i; \alpha, \lambda)$.

```
> x <- rgamma(10, shape=5, rate=2)
> ms(~-log(dgamma(x, a, l)),
           data=as.data.frame(x),
           start=list(a=1, l=1))

value: 14.60445
parameters:
        a        l
 3.217898 1.538759
formula:  ~   - log(dgamma(x, a, l))
100 observations
call: ms(formula =  ~  - log(dgamma(x, a, l)), data =
as.data.frame(x), start = list(         a = 1, l = 1))
```

□

7.2.5 `nlmin`

S+ La fonction `nlmin`, propre à S-Plus, minimise une fonction non linéaire. La fonction que `nlmin` minimisera ne peut avoir qu'un seul argument, soit le vecteur des paramètres à trouver.

Exemple 7.5. *Répéter l'exemple 7.4 à l'aide de* `nlmin` *dans S-Plus.*

Solution. Il faut cette fois passer en argument la fonction $-l(\alpha, \lambda)$. Le second argument, `c(1, 1)`, contient des valeurs de départ.

```
> f <- function(p) -sum(log(dgamma(x, p[1], p[2])))
> nlmin(f, c(1, 1))

$x:
[1] 3.217898 1.538759

$converged:
[1] T

$conv.type:
[1] "relative function convergence"
```

□

7.2.6 `nlminb`

S+ Minimisation d'une fonction non linéaire avec des bornes inférieure et/ou supérieure pour les paramètres (S-Plus seulement).

7.2.7 `nlm`

La fonction `nlm`, propre à R, minimise aussi une fonction non linéaire. La principale différence entre la fonction `nlmin` de S-Plus et `nlm` est que cette dernière peut passer des arguments à la fonction à minimiser, ce qui en facilite l'utilisation. R

Exemple 7.6. *Répéter l'exemple 7.4 à l'aide de `nlm` dans R.*

Solution. Remarquer comment on peut passer le vecteur de données à la fonction de log-vraisemblance à optimiser.

```
> f <- function(p, x) -sum(dgamma(x, p[1],
+     p[2], log = TRUE))
> nlm(f, c(1, 1), x = x)

$minimum
[1] 14.60445

$estimate
[1] 3.217881 1.538752

$gradient
[1] -1.057352e-05  2.737923e-05

$code
[1] 2

$iterations
[1] 15
```

□

7.2.8 `optim`

La fonction `optim` est un outil d'optimisation tout usage, souvent utilisée par d'autres fonctions. Elle permet, selon l'algorithme utilisé, de fixer des seuils minimum et/ou maximum aux paramètres à optimiser. Dans S-Plus, il faut charger la section `MASS` de la bibliothèque.

Exemple 7.7. *Répéter l'exemple 7.4 à l'aide de `optim`.*

Solution. En réutilisant la fonction `f` définie dans la solution de l'exemple 7.6 :

```
> optim(c(1, 1), f, x = x)

$par
[1] 3.217098 1.538413
```

```
$value
[1] 14.60445

$counts
function gradient
      65       NA

$convergence
[1] 0

$message
NULL
```

□

R *Remarque.* L'option `log = TRUE` de la fonction `dgamma` (et de toutes les autres fonctions de densité) permet de calculer plus précisément le logarithme de la densité. Cette option n'est disponible que dans R.

Remarque. L'estimation par le maximum de vraisemblance est beaucoup simplifiée par l'utilisation de la fonction `fitdistr` du package `MASS`.

7.3 Exemples

```
### On répète simplement les exemples présentés dans le
### chapitre.

###
### FONCTION 'uniroot'
###

## Solution de l'équation x - 2^(-x) = 0 dans l'intervalle
## [0, 1]
uniroot(function(x) x - 2^(-x), lower=0, upper=1)

###
### FONCTION 'polyroot'
###

## Racines du polynôme x^3 + 4 x^2 - 10. Les réponses sont
## données sous forme de nombre complexe. Utiliser les
## fonctions 'Re' et 'Im' pour extraire les parties réelles et
## imaginaires des nombres, respectivement.
polyroot(c(-10, 0, 4, 1))  # racines
Re(polyroot(c(-10, 0, 4, 1)))  # parties réelles
Im(polyroot(c(-10, 0, 4, 1)))  # parties imaginaires

###
```

```
### FONCTION 'optimize'
###

## Maximum local de la densité d'une loi bêta dans
## l'intervalle [0, 1].
f <- function(x) dbeta(x, 3, 2)
optimize(f, lower=0, upper=1, maximum=TRUE)

###
### FONCTION 'ms'
###

## Fonction de minimisation d'une somme. La somme à minimiser
## doit être spécifiée sous forme de formule et les données se
## trouver dans un data frame. Utile pour minimiser une
## fonction de log-vraisemblance.
x <- rgamma(10, shape=5, rate=2)
ms(~-log(dgamma(x, a, l)), data=as.data.frame(x),
   start=list(a=1, l=1))   # S-Plus seulement

###
### FONCTION 'nlmin'
###

## La fonction 'nlmin' cherche le minimum (global) d'une
## fonction non linéaire quelconque. On peut donc trouver des
## estimateurs du maximum de vraisemblance en tentant de
## minimiser moins la fonction de log-vraisemblance. Il faut
## spéficier des valeurs de départ.
f <- function(p) -sum(log(dgamma(x, p[1], p[2])))
nlmin(f, c(1, 1))          # S-Plus seulement

###
### FONCTION 'nlm'
###

## Équivalent dans R de la fonction 'nlmin' de S-Plus.
nlm(f, c(1, 1), x=x)       # R seulement

###
### FONCTION 'optim'
###

## La fonction 'optim' est très puissante, mais requiert aussi
## une bonne dose de prudence pour bien l'utiliser. Dans
## S-Plus, il faut charger la section MASS de la bibliothèque.
library(MASS)              # S-Plus seulement
optim(c(1, 1), f, x=x)     # même exemple que ci-dessus
```

7.4 Exercices

7.1 Trouver la solution des équations suivantes à l'aide des fonctions S appropriées.

(a) $x^3 - 2x^2 - 5 = 0$ pour $1 \leqslant x \leqslant 4$

(b) $x^3 + 3x^2 - 1 = 0$ pour $-4 \leqslant x \leqslant 0$

(c) $x - 2^{-x} = 0$ pour $0 \leqslant x \leqslant 1$

(d) $e^x + 2^{-x} + 2\cos x - 6 = 0$ pour $1 \leqslant x \leqslant 2$

(e) $e^x - x^2 + 3x - 2 = 0$ pour $0 \leqslant x \leqslant 1$

7.2 En théorie de la crédibilité, l'estimateur d'un paramètre a est donné sous forme de point fixe

$$\hat{a} = \frac{1}{n-1} \sum_{i=1}^{n} z_i (X_i - \bar{X}_z)^2,$$

où

$$z_i = \frac{\hat{a} w_i}{\hat{a} w_i + s^2}$$

$$\bar{X}_z = \sum_{i=1}^{n} \frac{z_i}{z_\Sigma} X_i$$

et $X_1, \ldots, X_n$, $w_1, \ldots, w_n$ et s^2 sont des données. Calculer la valeur de $\hat{a}$ si $s^2 = 140\,000\,000$ et que les valeurs de X_i et w_i sont telles que données dans le tableau ci-dessous.

i	1	2	3	4	5
X_i	2 061	1 511	1 806	1 353	1 600
w_i	100 155	19 895	13 735	4 152	36 110

7.3 Les fonctions de densité de probabilité et de répartition de la distribution de Pareto sont données à l'exercice 6.3. Calculer les estimateurs du maximum de vraisemblance des paramètres de la Pareto à partir d'un échantillon aléatoire obtenu par simulation avec la commande

```
> x <- lambda * (runif(100)^(-1/alpha) - 1)
```

pour des valeurs de `alpha` et `lambda` choisies.

8 Le S et la régression linéaire

Comme tous les grands logiciels statistiques — et même plusieurs calculatrices scientifiques — S-Plus et R comportent des fonctions pour calculer les coefficients d'une régression simple ou multiple. Les outils disponibles vont toutefois bien au-delà de ce calcul relativement simple. Par l'entremise de quelques fonctions génériques simples à utiliser, il est ainsi possible de générer différents graphiques relatifs à la régression, de calculer le tableau ANOVA de celle-ci et d'en extraire les informations principales, de calculer des prévisions ainsi que des intervalles de confiance. Bref, l'analyse complète d'un ensemble de données tient en quelques lignes de code ; il suffit de connaître les fonctions à utiliser.

Le but de ce chapitre consiste à présenter les principales fonctions — dont la liste se trouve au tableau 8.1 — utiles lors de l'analyse de données et la modélisation par régression. Il n'a cependant aucune prétention d'exhaustivité. Consulter l'aide en ligne de S-Plus ou R, ainsi que Venables & Ripley (2002) pour de plus amples détails.

À noter que l'on souligne les menues différences entre S-Plus et R, mais que les exemples ont été exécutés en R.

8.1 Importation de données

La modélisation statistique en S — comme, par exemple, l'analyse de régression — repose souvent sur l'utilisation de *data frames* pour le stockage des données. On se référera à la section 2.7 pour une présentation générale de ce type d'objet.

La principale fonction utilisée pour importer des données dans S-Plus ou R en vue d'une analyse de régression est `read.table`. Celle-ci retourne un *data frame*. Les arguments de `read.table` les plus souvent utilisés sont :

`file`	le nom ou l'URL (R seulement) du fichier de données à importer ;	R
`header`	`TRUE` si la première ligne du fichier à être lue contient les étiquettes des colonnes ;	
`comment.char`	le caractère (# par défaut) représentant le début d'un commentaire dans le fichier (R seulement) ;	R

Phase de l'analyse	**Fonctions**

Création et manipulation de *data frames*	`data.frame`
	`as.data.frame`
	`read.table`
	`cbind`
	`rbind`
	`names`, `colnames`[1]
	`row.names`, `rownames`[1]
	`attach`
	`detach`

Modélisation	`lm` `add1`, `addterm`[2] `drop1`, `dropterm`[2] `step`, `stepAIC`[2]

Analyse des résultats et diagnostics	`summary`
	`anova`
	`coef`, `coefficients`
	`confint`[1]
	`residuals`
	`fitted`
	`deviance`
	`df.residual`[1]

Mise à jour et prévision	`update` `predict`

Graphiques	`plot`
	`abline`
	`matplot`
	`matlines`

[1] R seulement.
[2] Dans le package `MASS`.

TAB. 8.1: Principales fonctions S-Plus et R pour la régression linéaire

Modèle mathématique	Formule S
$y_i = \alpha + \beta x_i + \varepsilon_i$	`y ~ x` `y ~ 1 + x`
$y_i = \beta x_i + \varepsilon_i$	`y ~ -1 + x` `y ~ x - 1`
$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \varepsilon_i$	`y ~ x1 + x2` `y ~ x` où `x <- cbind(x1, x2)`

TAB. 8.2: Modèles linéaires simples et leur formulation en S

`skip` le nombre de lignes à sauter au début du fichier (principalement utilisé pour sauter des lignes de commentaires dans S-Plus).

8.2 Formules

Lorsque l'on fait une régression, il faut informer S-Plus ou R des variables que l'on entend inclure dans celle-ci et de quelle façon. La convention utilisée dans le langage S est celle dite des «formules». Le tableau 8.2 présente quelques exemples de formulation de modèles linéaires simples en S.

Pour une utilisation de base des fonctions de régression, la connaissance les règles suivantes suffit.

1. Les opérateurs `+` et `-` prennent une nouvelle signification dans les formules : `+` signifie «inclusion» et `-`, «exclusion».
2. Le terme constant d'une régression est inclus implicitement. Pour l'exclure explicitement (régression passant par l'origine), il faut donc ajouter un terme `-1` du côté droit de la formule.
3. Dans une régression multiple, on peut soit lister toutes les variables à inclure du côté droit de la formule, soit ne spécifier qu'une matrice contenant ces variables (dans les colonnes).

Consulter les sections 6.2 de Venables & Ripley (2002) et 11.1 de Venables et al. (2005) pour plus de détails.

8.3 Modélisation des données

Supposons que l'on souhaite étudier la relation entre la variable indépendante `x1` et la variable dépendante (ou réponse) `y1` du jeu de données `anscombe`. La première étape de la modélisation des données en régression linéaire simple consiste habituellement à représenter celles-ci graphiquement.

La fonction `plot` est une fonction générique comportant des méthodes pour un grand nombre de classes d'objets différentes. Puisqu'une méthode

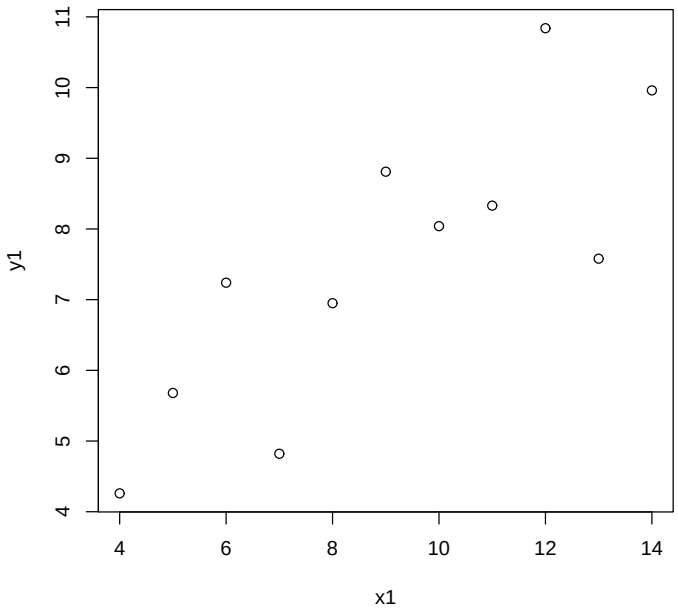

FIG. 8.1: Relation entre `y1` et `x1` des données `anscombe`

existe pour les objets de classe `formula`, on peut tracer un graphique de `y1` en fonction de `x1` avec

```
> plot(y1 ~ x1, data=anscombe)
```

ou, si les colonnes du *data frame* `anscombe` sont visibles, simplement avec

```
> plot(y1 ~ x1)
```

Le résultat de ces commandes se trouve à la figure 8.1.

Le graphique nous montre qu'il est raisonnable de postuler une relation linéaire entre les éléments de `y1` et `x1`. On pose donc le modèle

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

où y_i et x_i, $i = 1, \ldots, 11$ sont les éléments des vecteurs `y1` et `x1`, respectivement, et ε_i est le terme d'erreur.

C'est avec la fonction `lm` (pour *linear model*) que l'on calcule les estimateurs des coefficients de la régression β_0 et β_1. De façon simplifiée, cette fonction prend en arguments une formule et un *data frame* dans lequel se trouvent les données relatives aux termes de celle-ci. La fonction `lm` retourne un objet de classe `lm` pour laquelle il existe de nombreuses méthodes.

```
> (fit <- lm(y1 ~ x1, data = anscombe))

Call:
lm(formula = y1 ~ x1, data = anscombe)

Coefficients:
(Intercept)           x1
     3.0001       0.5001

> class(fit)

[1] "lm"
```

8.4 Analyse des résultats

Le résultat de la fonction `lm` est une liste dont on peut extraire manuellement les différents éléments (consulter la rubrique d'aide). Grâce à quelques fonctions génériques disposant d'une méthode pour les objets de classe `lm`, il est toutefois facile et intuitif d'extraire les principaux résultats d'une régression :

1. `coef` ou `coefficients` extraient les coefficients $\hat{\beta}_0$ et $\hat{\beta}_1$ de la régression ;
2. `fitted` extrait les valeurs ajustées $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$;
3. `residuals` extrait les résidus $y_i - \hat{y}_i$;
4. `deviance` retourne la somme des carrés des résidus SSR $= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$;
5. `df.residual` extrait le nombre de degrés de liberté de la somme des carrés des résidus (R seulement). R

La fonction générique `summary` présente les informations ci-dessus de manière facile à consulter. Plus précisément, le sommaire de la régression contient, outre le modèle utilisé et les estimateurs des coefficients de la régression : les résultats des tests t, la valeur du coefficient de détermination

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

et, dans R seulement, du coefficient de détermination ajusté R

$$R_a^2 = 1 - (1 - R^2) \frac{n-1}{n-p-1},$$

ainsi que le résultat du test F global.

La fonction `confint` calcule les intervalles de confiance des paramètres de la régression (R seulement). R

D'autre part, le tableau d'analyse de variance (séquentiel, en régression multiple) est calculé avec la fonction générique `anova`.

Pour ajouter la droite de régression au graphique créé au début de l'analyse, utiliser la fonction `abline`, qui dispose elle aussi d'une méthode pour les objets de classe `lm`.

8.5 Diagnostics

Les statistiques servant à mesurer la qualité d'un modèle de régression (R^2, R^2 ajusté, statistiques t et F) sont calculées par les fonctions `summary` et `anova`.

La méthode de la fonction `plot` pour les objets de classe `lm` produit une série de six graphiques (quatre dans R avant la version 2.2.0) permettant de juger de la qualité d'une régression. Consulter la rubrique d'aide de la fonction `plot.lm` pour plus de détails.

8.6 Mise à jour des résultats et prévision

Il peut arriver que, une fois la modélisation d'un ensemble de données effectuée, il soit nécessaire d'ajouter ou de modifier une ou plusieurs données ou variables. Plutôt que de reprendre toute la modélisation avec la fonction `lm`, il peut alors s'avérer plus simple et élégant d'utiliser la fonction `update`.

```
> update(fit, ~. + x4)

Call:
lm(formula = y1 ~ x1 + x4, data = anscombe)

Coefficients:
(Intercept)           x1           x4
    4.33291      0.45073     -0.09873
```

Le calcul de prévisions et d'intervalles de confiance et de prévision se fait avec la fonction générique `predict` et sa méthode pour les objets de classe `lm`. Par défaut, `predict` calculera les prévisions pour les valeurs x_i, $i = 1, \ldots, n$. Par conséquent, le résultat de `predict` sera le même que celui de `fitted` :

```
> all.equal(predict(fit), fitted(fit))

[1] TRUE
```

Comme on souhaite généralement prévoir la réponse pour d'autres valeurs de la variable indépendante, on spécifiera celles-ci par le biais d'un *data frame* passé à `predict` avec l'option `newdata`.

S+ Le calcul des intervalles de confiance et de prévision diffère entre S-Plus et R. Dans S-Plus, les intervalles de confiance en chaque point seront calculés par `predict` avec l'option `ci.fit = T`, alors que les intervalles de prévision le sont avec `pi.fit = T`. Il est par conséquent possible de calculer en un

seul appel à `predict` les deux types d'intervalles. Le niveau de confiance est spécifié avec l'option `conf.level` (0,95 par défaut).

Si un ou plusieurs intervalles de confiance sont calculés, le résultat de `predict` est une liste nommée dont les éléments sont `$fit`, `$ci.fit` et `$pi.fit` (et ce peu importe le nom de l'objet contenant le modèle linéaire).

Dans R, il n'est pas possible de calculer les deux types d'intervalles en un seul appel à `predict`. Pour calculer les intervalles de confiance, on utilisera l'option `interval="confidence"`, alors que pour les intervalles de prévision on utilise `interval="prediction"`. Le niveau de confiance est déterminé avec l'option `level`. Le résultat est une matrice de trois colonnes dont la première contient les prévisions et les deux autres les bornes inférieures (`lwr`) et supérieures (`upr`) des intervalles de confiance. R

Les limites des intervalles de confiance peuvent être ajoutées au graphique des données avec les fonctions `matlines` ou `matplot` (R seulement). Consulter les rubriques d'aide et les exemples pour de plus amples détails. R

8.7 Exemples

```
###
### IMPORTATION DE DONNÉES
###

## On importe les données du fichier anscombe.dat se trouvant
## à l'adresse http://vgoulet.act.ulaval.ca/pub/data/. Si le
## fichier est sauvegardé dans l'espace de travail, il n'est
## pas nécessaire de spécifier le chemin d'accès complet. Il y
## a deux lignes de commentaires au début du fichier.
anscombe <- read.table("anscombe.dat", skip=2)

## --- R ---
## Avec R, on peut lire le fichier directement sans le
## sauvegarder localement. De plus, les lignes débutant par #
## sont automatiquement reconnues comme des lignes de
## commentaires (argument 'skip' pas nécessaire, donc).
anscombe <- read.table(
  "http://vgoulet.act.ulaval.ca/pub/data/anscombe.dat")
## ---------

## --- R ---
## Ce jeu de données se trouve en fait déjà dans R et il est
## chargé en mémoire avec 'data'.
data(anscombe)
## ---------

## Le résultat est un data frame.
mode(anscombe)             # une liste...
class(anscombe)            # ... de classe "data.frame"
```

```
## Extraction des étiquettes des colonnes et des lignes.
names(anscombe)            # étiquettes des colonnes
colnames(anscombe)         # idem, R seulement
row.names(anscombe)        # étiquettes des lignes
rownames(anscombe)         # idem, R seulement

###
### MODÉLISATION DES DONNÉES
###

## Relation graphique entre les variables Y et X7 des données
## anscombe.
plot(y1 ~ x1, data=anscombe)

## On peut aussi rendre les colonnes du data frame visibles
## dans l'espace de travail et référer ensuite à celles-ci
## directement.
attach(anscombe)
plot(y1 ~ x1)

## Estimation des coefficients de la régression. Il est
## souhaitable de sauvegarder les résultats dans un objet (de
## classe "lm") puisqu'il existe de multiples méthodes pour de
## tels objets.
( fit <- lm(y1 ~ x1, data=anscombe) )
class(fit)

###
### ANALYSE DES RÉSULTATS
###

## Le sommaire de la régression contient, outre le modèle
## utilisé, les résultats des tests t, la valeur du
## coefficient de détermination (et du coefficient de
## détermination ajusté, dans R), ainsi que le résultat du
## test F global.
summary(fit)

## Calcul du coefficient de détermination à la main.
attach(anscombe)
1 - sum(residuals(fit)^2)/sum((y1 - mean(y1))^2)
1 - deviance(fit)/sum((y1 - mean(y1))^2)
detach(anscombe)

## Intervalles de confiance pour les paramètres de la
## régression.
confint(fit)               # R seulement

## Le tableau d'analyse de variance (séquentiel, en régression
```

```
## multiple) est calculé avec la fonction générique 'anova'.
anova(fit)

## Pour ajouter la droite de régression au graphique créé
## précédemment, utiliser la fonction générique
## 'abline'. L'ordonnée à l'origine et la pente sont extraites
## de l'objet 'fit'.
abline(fit)

###
### MISE À JOUR DES RÉSULTATS ET PRÉVISION
###

## La fonction 'update' est utilisé pour modifier une ou
## plusieurs données dans le modèle ou pour enlever ou ajouter
## une ou plusieurs variables dans le modèle.
anscombe$x1[11] <- 6       # modification d'une donnée
update(fit)                # modèle mis à jour
update(fit, . ~ . + x4)    # ajout de la variable "x4"

## Retour au modèle d'origine
fit <- lm(y1 ~ x1, data=anscombe)

## Prévisions du modèle pour des valeurs de la variables "x1"
## de 3 et 15:
predict(fit, newdata=data.frame(x1=c(3, 15)))

## Calcul des intervalles de confiance et de prévision pour
## les prévisions ci-dessus avec un niveau de confiance de
## 90%.
##
## --- S-Plus ---
predict(fit, newdata=data.frame(x1=c(3, 15)),
        ci.fit=T, pi.fit=T, conf.level=0.90)
## --------------
##
## --- R ---
predict(fit, newdata=data.frame(x1=c(3, 15)),
        interval="confidence", level=0.90)
predict(fit, newdata=data.frame(x1=c(3, 15)),
        interval="prediction", level=0.90)
## ---------

## Ajout des limites supérieures et inférieures des
## intervalles de confiance au graphique des données. On
## utilise la fonction 'matplot' qui prend en argument deux
## matrices 'x' et 'y' et produit un graphique des coordonnées
## de la première colonne de 'x' avec la première colonne de
## 'y', la seconde de 'x' avec la seconde de 'y', etc.
##
```

```
## Afin d'obtenir un beau graphique, il faut s'assurer de
## mettre les valeurs de 'x' en ordre croissant et de classer
## celles de 'y' en conséquence.
##
## En fait, on utilise la fonction 'matlines' qui ajoute à un
## graphique existant. La fonction 'matplot' créerait un
## nouveau graphique. (Note: il est possible de combiner les
## deux commandes matlines() ci-dessous en une seule.)
##
## Rendre les colonnes visibles.
attach(anscombe)

## Calcul des prévisions et des intervalles pour toutes les
## valeurs de "x1".
pred <- predict(fit, , ci.fit=TRUE, pi.fit=TRUE) # S-Plus
pred.ci <- predict(fit, interval="confidence")   # R
pred.pi <- predict(fit, interval="prediction")   # R

## --- S-Plus ---
matlines(sort(x1), pred$ci.fit[order(x1),],
         lty=2, col=2)
matlines(sort(x1), pred$pi.fit[order(x1),],
         lty=2, col=2)
## --------------
##
## --- R ---
matlines(sort(x1), pred.ci[order(x1), -1],
         lty=2, col="red")
matlines(sort(x1), pred.pi[order(x1), -1],
         lty=2, col="green")
## ---------

## Pour éviter que des lignes ne dépassent à extérieur du
## graphique, il faut trouver, avant de faire le graphique,
## les limites inférieure et supérieure des ordonnées.
##
## --- S-Plus ---
y <- cbind(y1, pred$fit, pred$ci.fit, pred$pi.fit)
plot(y1 ~ x1, pch=19, xlim=range(x1), ylim=range(y))
matlines(sort(x1), y[order(x1), -1],
         lty=c(1, 2, 2, 2, 2), col=c(4, 2, 2, 3, 3))
## --------------
##
## --- R ---
## La version R de 'matplot' peut combiner des lignes et des
## points, ce qui permet de faire tout le graphique avec une
## seule commande.
y <- cbind(y1, pred.ci, pred.pi[, -1])
matplot(sort(x1), y[order(x1),],
        pch=19, type=c("p", rep("l", 5)),
```

```
        lty=c(0, 1, rep(2, 4)),
        col=c("black", "blue", "red", "red", "green", "green"))
## ---------
```

8.8 Exercices

8.1 Importer dans S-Plus ou R l'ensemble de données `steam.dat` se trouvant dans le site Internet

`http://vgoulet.act.ulaval.ca/pub/data/`

à l'aide de la fonction `read.table`. Les trois première lignes du fichier sont des lignes de commentaires débutant par le caractère #. La quatrième ligne contient les étiquettes des colonnes.

8.2 Rendre les colonnes individuelles de l'ensemble de données `steam` visibles dans l'espace de travail.

8.3 Faire (même à l'aveuglette) l'analyse de régression de la variable `Y` en fonction de la variable `X7` des données `steam`.

(a) Évaluer visuellement le type de relation pouvant exister entre `Y` et `X7`.

(b) Évaluer les coefficients d'une régression linéaire entre `Y` et `X7` et ajouter la droite de régression ainsi obtenue au graphique créé en (a).

(c) Répéter la partie (b) en forçant la droite de régression à passer par l'origine $(0, 0)$. Quel modèle semble le plus approprié ?

(d) Le coefficient de détermination R^2 mesure la qualité de l'ajustement d'une droite de régression aux données. Calculer le R^2 pour les modèles en (b) et (c). Obtient-on les mêmes résultats que ceux donnés par `summary` ? Semble-t-il y avoir une anomalie ?

(e) Calculer les prévisions de chaque modèle pour quelques valeurs choisies de la variable indépendante.

(f) Calculer les intervalles de confiance et de prévision pour tous les points de `X7` (bref, ne pas utiliser `newdata`). Ajouter les limites inférieures et supérieures des intervalles au graphique créé précédemment. Utiliser des types de lignes (option `lty`) et des couleurs (option `col`) différents pour chaque ensemble de limites.

8.4 Répéter l'exercice précédent en ajoutant la variable `X5` à l'analyse, transformant ainsi le modèle de régression linéaire simple en un modèle de régression multiple.

9 Le S et les séries chronologiques

S-Plus et R offrent toutes les fonctions nécessaires pour faire l'analyse complète de séries chronologiques : création et manipulation d'objets de classe «série chronologique», identification, modélisation, prévision et simulation de séries.

À certains égards, cependant, les fonctions dans la distribution S-Plus de base sont mal intégrées au langage. Nous utiliserons donc les fonctions du package `MASS` de Venables & Ripley (2002), auquel on a déjà fait référence à la section 7.1. Rappelons que pour accéder aux fonctions du package, il suffit de le charger en mémoire avec la commande S+

```
> library(MASS)
```

À noter que le package `MASS` est aussi distribué avec R, mais qu'il n'est pas nécessaire de le charger, les fonctions du package `stats` pour les séries chronologiques étant essentiellement les mêmes que celles de `MASS`. R

La liste des principales fonctions utilisées pour l'analyse de séries chronologiques se trouve au tableau 9.1. La fonction `arima` et la méthode de `predict` pour les objets de classe `Arima`, toutes deux fort utiles, ne se trouvent pas dans la distribution S-Plus de base. Quelques autres fonctions sont disponibles, principalement pour le traitement des séries multivariées ; voir Venables & Ripley (2002, chapitre 14).

9.1 Importation des données

Les séries chronologiques sont typiquement créées à partir de vecteurs simples. Or, la fonction `scan` lit justement l'intégralité des données du fichier dont le nom est donné en premier argument, puis retourne un vecteur. Elle constitue donc le meilleur choix pour importer des séries chronologiques dans S-Plus ou R.

Contrairement à `read.table`, la fonction `scan` ne reconnaît pas les commentaires par défaut. Dans S-Plus, il faut utiliser une combinaison des arguments `skip`, `what` et `flush`. Dans R, il suffit de spécifier le caractère représentant le début d'un commentaire avec l'argument `comment.char`. S+ R

Phase de l'analyse	Fonctions

Création et manipulation de séries	`ts`, `rts`, `cts`, `its`
	`time`
	`start`
	`end`
	`frequency`
	`cycle`
	`window`
	`diff`
	`filter`
	`stl`

Identification	`ts.plot`, `plot`[1] `acf` `pacf`[1]

Modélisation	`ar`
	`arima`
	`arima.mle`[2]
	`ARMAacf`
	`ARMAtoMA`

Diagnostics	`tsdiag`
Prévision	`predict` `arima.forecast`[2]
Simulation	`arima.sim`

[1] R seulement.
[2] S-Plus de base seulement.

TAB. 9.1: Principales fonctions S-Plus (avec package MASS) et R pour l'analyse de séries chronologiques

9.2 Création et manipulation de séries

La façon la plus simple de créer des séries chronologiques est avec la fonction `ts`. Les fonctions `rts` (séries régulières), `cts` (séries avec dates) et `its` (séries irrégulières) sont plus récentes et parfois nécessaires. S-Plus propose
S+ également les classes `timeSeries` et `signalSeries` (et les fonctions du même nom pour créer les objets), mais celles-ci n'ajoutent pas de fonctionnalité nouvelle.

La fonction `window` permet d'extraire un sous-ensemble d'une série chronologique en spécifiant des dates de début et de fin plutôt que des positions dans le vecteur des observations.

9.3 Identification

La première chose à faire dans l'analyse d'une série chronologique consiste à tracer le graphique de la série et son corrélogramme. Le premier graphique est obtenu avec `ts.plot` ou plus simplement avec `plot` (R seulement). R

La fonction `acf` peut calculer et tracer les fonctions (échantillonnales) d'autocovariance $\hat{\gamma}_X(h)$, d'autocorrélation $\hat{\rho}_X(h)$ ou d'autocorrélation partielle $\hat{\phi}_{hh}$ selon la valeur de son argument `type` (spécifier `covariance`, `correlation` et `partial`, respectivement). Par défaut, `acf` trace le corrélogramme de la série. Si l'on souhaite obtenir les valeurs de la fonction d'autocorrélation sans un graphique, ajouter l'option `plot = FALSE` dans l'appel de la fonction.

Dans R, la fonction d'autocorrélation partielle s'obtient plus directement avec la fonction `pacf`. R

9.4 Modélisation

Un processus ARMA d'ordre (p, q) est définit comme la solution $\{X_t\}$ des équations

$$\phi(B)X_t = \theta(B)Z_t, \quad t = 0, \pm 1, \pm 2, \ldots$$

où

$$\begin{aligned}\phi(z) &= 1 - \phi_1 z - \cdots - \phi_p z^p \\ \theta(z) &= 1 + \theta_1 z + \cdots + \theta_q z^q,\end{aligned}$$

$BX_t = X_{t-1}$ et $\{Z_t\} \sim \mathrm{WN}(0, \sigma^2)$. C'est là la paramétrisation retenue dans les fonctions du package `MASS` (`arima`, entre autres) ainsi que dans R, mais pas dans S-Plus.

Remarque. Le signe des paramètres $\theta_1, \ldots, \theta_q$ est inversé dans les fonctions `arima.mle` et `arima.sim` de S-Plus. ⚠

Un processus ARIMA est un processus non stationnaire qui, une fois la d^{e} différence appliquée sur la série, est un processus ARMA. Autrement dit, $\{X_t\} \sim \mathrm{ARIMA}(p, d, q)$ si $\{\nabla^d X_t\} \sim \mathrm{ARMA}(p, q)$ et donc $\{X_t\}$ est la solution stationnaire de

$$\phi(B)(1 - B)^d X_t = \theta(B)Z_t.$$

L'étape de la modélisation consiste donc à ajuster un modèle ARIMA aux observations d'une série chronologique en estimant les paramètres $\phi_1, \ldots, \phi_p$, $\theta_1, \ldots, \theta_q$ et σ^2. C'est le rôle des fonctions `ar` et `arima`.

La fonction `ar` est très pratique pour une première estimation : elle ajuste un modèle $\mathrm{AR}(p)$ aux données pour plusieurs valeurs de p à l'aide des équations de Yule–Walker (par défaut) et retourne le modèle avec la plus faible statistique AIC. Cette statistique est égale à moins deux fois la fonction de log-vraisemblance pénalisée par le nombre de paramètres dans le modèle.

D'autre part, la fonction `arima` estime les paramètres d'un modèle ARIMA d'ordre (p, d, q) par la technique du maximum de vraisemblance (par défaut).

Contrairement à `ar`, la fonction `arima` ne fait pas un choix parmi plusieurs modèles — il y en aurait beaucoup trop. Il faut donc spécifier les valeurs de p, d et q à l'aide de l'argument `order` (un vecteur de trois éléments). À noter que la fonction `arima` inclut une moyenne μ dans le modèle lorsque $d = 0$.

Finalement, les séries comportant de la saisonnalité sont modélisées à l'aide des très généraux processus SARIMA. Le processus SARIMA d'ordre $(p, d, q) \times (P, D, Q)_s$ est défini comme la solution stationnaire $\{X_t\}$ des équations

$$\phi(B)\Phi(B^s)W_t = \theta(B)\Theta(B^s)Z_t, \quad W_t = \nabla^d \nabla_s^D X_t,$$

où

$$\begin{aligned}
\phi(z) &= 1 - \phi_1 z - \cdots - \phi_p z^p \\
\Phi(z) &= 1 - \Phi_1 z - \cdots - \Phi_P z^P \\
\theta(z) &= 1 + \theta_1 z + \cdots + \theta z^q \\
\Theta(z) &= 1 + \Theta_1 z + \cdots + \Theta_Q z^Q
\end{aligned}$$

et $\{Z_t\} \sim \mathrm{WN}(0, \sigma^2)$.

Les paramètres d'un modèle SARIMA sont toujours estimés à l'aide de la fonction `arima` en spécifiant les valeurs de P, D, Q et s par l'argument `seasonal`.

La fonction `ARMAacf` permet de calculer la fonction d'autocorrélation ou d'autocorrélation partielle théorique d'un processus ARMA quelconque. La fonction `ARMAtoMA`, comme son nom l'indique, permet quant à elle d'inverser un processus ARMA quelconque. Toutes deux peuvent s'avérer utiles pour vérifier ses calculs.

9.5 Diagnostics

La fonction `tsdiag` du package `MASS` permet de juger rapidement de la qualité d'ajustement d'un modèle. La fonction crée trois graphiques : la série des résidus $\{Z_t\}$, le corrélogramme de cette même série et un graphique de la valeur p de la statistique de Ljung–Box pour des valeurs de $H = 1, 2, \ldots$. La statistique de Ljung–Box est simplement une version améliorée de la statistique du test portmanteau :

$$Q_{LB} = n(n+2) \sum_{h=1}^{H} \frac{\hat{\rho}^2(h)}{n-h}.$$

Si l'ajustement du modèle est bon, les résidus forment un bruit blanc. Le corrélogramme généré par `tsdiag` devrait donc ressembler à celui d'un bruit blanc et les valeurs p devraient être grandes (on ne rejette pas l'hypothèse de bruit blanc).

9.6 Prévisions

La prévision de modèles ARIMA repose sur la fonction `arima.forecast` dans la distribution de base de S-Plus . S+

De manière plus élégante, le package `MASS` fournit une nouvelle méthode à la fonction générique `predict` pour les objets de classe `Arima` (créés par la fonction `arima`). Les prévisions sont donc calculées exactement comme en régression, outre que l'argument principal de `predict` devient le nombre de périodes pour lesquelles l'on veut une prévision, et non les valeurs d'une ou plusieurs variables indépendantes. L'écart type de chaque prévision est également calculé par `predict`, ce qui permet de calculer des bornes d'intervalles de prévision.

Remarque. La fonction `predict` ne fonctionne pas avec les séries de classe `ts` dans S-Plus. Il faut donc s'assurer de créer la série avec `rts` ou `cts`.

9.7 Simulation

La simulation de séries chronologiques ARIMA est très simple avec la fonction `arima.sim`. Il suffit de savoir comment spécifier le modèle à simuler. L'argument `model` de la fonction `arima.sim` est une liste comportant un ou plusieurs des éléments `ar`, `ma` et `order`. Le premier de ces éléments est le vecteur des paramètres $\phi_1, \ldots, \phi_p$, le second le vecteur des paramètres $\theta_1, \ldots, \theta_q$ et le troisième le vecteur (p, d, q) — utilisé seulement si $d > 0$.

Par défaut, le bruit blanc est généré avec une loi normale centrée réduite. On peut changer la distribution à utiliser avec l'argument `rand.gen` ou passer des arguments différents à la fonction de simulation du bruit blanc directement dans l'appel de `arima.sim`. Voir les exemples à la section 9.8.

Remarque. Ne pas oublier d'inverser les signes des paramètres $\theta_1, \ldots, \theta_q$ dans la fonction `arima.sim` de S-Plus ! ⚠

Remarque. Dans S-Plus, la fonction `arima.sim` retourne simplement un vecteur d'observations. Utiliser l'une des fonctions `ts`, `rts` ou `cts` pour convertir ce vecteur en une série chronologique.

9.8 Exemples

```
###
### IMPORTATION DE DONNÉES
###

## On utilise la fonction 'scan' pour importer des données
## sous forme de vecteur. Les fichiers 'deaths.dat' et
## 'strikes.dat' comptent chacun trois lignes de commentaires
## en début de fichier.
##
```

```
## --- S-Plus ---
## Dans S-Plus, on utilisera l'argument 'skip' pour sauter les
## lignes de commentaires.
deaths <- scan("deaths.dat", skip=3)
strikes <- scan("strikes.dat", skip=3)
## --------------
##
## --- R ---
## Dans R, on spécifie simplement le caractère délimitant les
## commentaires avec l'argument 'comment.char'. De plus, on
## peut lire les fichiers directement depuis Internet.
deaths <- scan(
  "http://vgoulet.act.ulaval.ca/pub/data/deaths.dat",
  comment.char="#")
strikes <- scan(
  "http://vgoulet.act.ulaval.ca/pub/data/strikes.dat",
  comment.char="#")
## ---------

###
### CRÉATION ET MANIPULATION DE SÉRIES
###

## Le fichier deaths.dat contient le nombre mensuel de morts
## accidentelles, 1973-1978. À l'aide de la fonction 'ts', on
## transforme l'objet 'deaths' en une série chronologique aux
## propriétés correspondantes.
( deaths <- ts(deaths, start=1973, frequency=12) )

## Le résultat est une série chronologique.
mode(deaths)              # un vecteur...
class(deaths)             # ... de classe "ts"

## Même chose avec l'objet 'strikes', qui contient le nombre
## de grèves aux États-Unis entre 1951-1980. L'argument
## 'frequency' n'est pas nécessaire; les séries sont annuelles
## par défaut.
( strikes <- ts(strikes, start=1951) )

## La fonction 'window' est la façon élégante d'extraire les
## observations de la série 'deaths' du mois de février 1974
## au mois d'octobre 1974, inclusivement,
window(deaths, start=c(1974, 2), end=c(1974, 10))

###
### IDENTIFICATION
###

## Graphiques des séries 'deaths' et 'strikes'.
ts.plot(deaths)           # S-Plus ou R
```

```
plot(deaths)               # R seulement
ts.plot(strikes)           # S-Plus et R
plot(strikes)              # R seulement

## Corrélogramme de la série 'deaths'. Par défaut, 'acf'
## trace le corrélogramme.
acf(deaths)

## Pour obtenir les valeurs numériques de la fonction
## d'autocorrélation empirique, utiliser l'argument
## 'plot=FALSE'.
acf(deaths, plot=FALSE)

###
### MODÉLISATION
###

## On ajuste d'abord un modèle autorégressif pur aux données
## 'strikes' avec la fonction 'ar'.
( modele <- ar(strikes) )  # modèle AR(2) choisi

## On peut comparer les statistiques AIC des divers
## modèles. La statistique AIC du modèle AR(2) ne vaut pas
## vraiment 0; les statistiques sont simplement mise à
## l'échelle avec cette valeur comme référence.
modele$aic

## Ajustement d'un modèle ARIMA(1, 2, 1) aux données
## 'strikes'.
( fit.strikes <- arima(strikes, order=c(1, 2, 1)) )

## Ajustement d'un modèle SARIMA(0, 1, 1) x (0, 1, 1)_{12} aux
## données 'deaths'. Par défaut, la fréquence de la série (s =
## 12) est supposée identique à celle spécifiée dans
## l'objet. Il n'est donc pas nécessaire de préciser la valeur
## de s dans l'appel de 'arima', ici, puisque la série a été
## correctement définie dès le départ.
( fit.deaths <- arima(deaths, order=c(0, 1, 1),
                      seasonal=c(0, 1, 1)) )

## Cinq premières valeurs de la fonction d'autocorrélation
## théorique d'un processus ARMA(1, 1) avec phi = 0,6 et
## theta = -0,4.
ARMAacf(ar=0.6, ma=-0.4, lag.max=5)

## Cinq premiers coefficients de la représentation MA(infini)
## d'un processus AR(1) avec phi = 0,8.
ARMAtoMA(ar=0.8, lag.max=3)

###
```

```
### DIAGNOSTICS
###

## Vérification graphique de la qualité de l'ajustement du
## modèle ARIMA(1, 2, 1) aux données 'strikes' à l'aide de la
## fonction 'tsdiag'.
tsdiag(fit.strikes)

## Idem pour le modèle des données 'deaths'.
tsdiag(fit.deaths)

###
### PRÉVISIONS
###

## Prévision des six prochaines valeurs de la série 'deaths' à
## partir du modèle SARIMA.
##
## --- S-Plus ---
## Convertir d'abord l'objet 'deaths' en une série de
## classe "rts".
deaths <- as.rts(deaths)
## --------------
( pred <- predict(fit.deaths, n.ahead=6) )

## Graphique présentant la série originale, les prévisions des
## six prochaines années et les intervalles de prévision.
ts.plot(deaths,
        pred$pred,
        pred$pred + 1.96 * pred$se,
        pred$pred - 1.96 * pred$se,
        col=c(1, 2, 4, 4), lty=c(1, 3, 2, 2))

###
### SIMULATION
###

## Simulation de 10 observations d'un modèle ARMA(1, 1) avec
## phi = 0,8, theta = 0,5 et sigma^2 = 1.
arima.sim(10, model=list(ar=0.8, ma=-0.5))

## Simulation de 10 observations d'un modèle ARIMA(2, 1, 1)
## avec phi_1 = 0,6, phi_2 = 0,3, theta = -0,2 et
## sigma^2 = 25.
arima.sim(10, model=list(ar=c(0.6, 0.3), ma=0.2,
              order=c(2, 1, 1), sd=5))
```

9.9 Exercices

Avant de faire les exercices ci-dessous, importer dans S-Plus ou R les ensembles de données `deaths`, `strikes`, `uspop` et `wine` disponibles à l'URL

`http://vgoulet.act.ulaval.ca/pub/data/`

Utiliser pour ce faire les commandes suivantes (R seulement) : R

```
> deaths <- ts(scan("deaths.dat", comment.char = "#"),
+     start = 1973, frequency = 12)
> strikes <- ts(scan("strikes.dat", comment.char = "#"),
+     start = 1951)
> uspop <- ts(scan("uspop.dat", comment.char = "#"),
+     start = 1790, deltat = 10)
> wine <- ts(scan("wine.dat", comment.char = "#"),
+     start = 1980, frequency = 12)
```

Le package `MASS` contient également de nombreux ensembles de données. Pour obtenir la liste des fonctions et des données du package, faire

```
> library(help = MASS)
```

Il est possible d'afficher plus d'un graphique à la fois sur un périphérique graphique en le subdivisant à l'aide des options `mfrow` (remplissage par ligne) et `mfcol` (remplissage par colonne) de la fonction `par`. Par exemple,

```
> par(mfrow = c(2, 1))
```

divisera la «page» en deux lignes et une colonne. Les deux prochains graphiques se retrouveront donc l'un au-dessus de l'autre.

9.1 Exécuter chacune des commandes `par` ci-dessous. Après chacune, exécuter les commandes suivantes pour constater l'effet de `par` sur le périphérique graphique :

```
> plot(deaths)
> plot(strikes)
> plot(uspop)
> acf(wine)
```

(a) `par(mfrow = c(2, 1))`

(b) `par(mfrow = c(1, 2))`

(c) `par(mfrow = c(2, 2))`

(d) `par(mfcol = c(2, 2))`

9.2 Simuler 100 observations des processus suivants. Pour chacun, tracer sur un seul périphérique graphique le graphique de la série simulée ainsi que son corrélogramme (l'un au-dessus de l'autre). Comparer le corrélogramme à la fonction d'autocorrélation théorique.

(a) $\{Z_t\} \sim \text{WN}(0, 2)$ où chaque Z_t est une variable aléatoire normale de moyenne 0 et variance 2.

(b) $\{X_t\} \sim \text{MA}(1)$ avec $\theta = 0{,}8$ et $\sigma^2 = 1$.

(c) $\{X_t\} \sim \text{MA}(1)$ avec $\theta = -0{,}6$ et $\sigma^2 = 100$.

(d) $\{X_t\} \sim \text{MA}(2)$ avec $\theta_1 = 0{,}5$, $\theta_2 = 0{,}4$ et $\sigma^2 = 1$.

(e) $\{X_t\} \sim \text{AR}(1)$ avec $\phi = 0{,}8$ et $\sigma^2 = 1$.

(f) $\{X_t\} \sim \text{AR}(1)$ avec $\phi = -0{,}9$ et $\sigma^2 = 100$.

(g) $\{X_t\} \sim \text{AR}(2)$ avec $\phi = 0{,}7$, $\phi_2 = -0{,}1$ et $\sigma^2 = 1$.

9.3 Ajuster un modèle autorégressif pur aux données `lh` du package `MASS` à l'aide de la fonction `ar`.

9.4 L'exercice suivant, bien qu'un peu artificiel, illustre la procédure d'analyse d'une série chronologique.

(a) Simuler 100 valeurs d'un processus $\text{ARMA}(1, 1)$ avec $\phi = 0{,}7$, $\theta = 0{,}5$ et $\sigma^2 = 1$. Dans S-Plus, s'assurer que l'objet contenant la série est de classe `rts`.

(b) Tracer les graphiques suivants sur un même périphérique : la série, le corrélogramme et la fonction d'autocorrélation partielle empirique.

(c) Ajuster un modèle $\text{ARMA}(1, 1)$ aux données simulées en (a) en estimant les paramètres à l'aide de la fonction `arima`. Les estimateurs devraient être près des valeurs utilisées lors de la simulation.

(d) Vérifier la qualité de l'ajustement du modèle en (c) à l'aide de la fonction `tsdiag`.

(e) Prévoir les 12 prochaines valeurs du processus. Tracer un graphique de la série originale et des prévisions en fournissant les deux séries en argument à la fonction `ts.plot`.

A GNU Emacs et ESS : la base

Emacs est l'Éditeur de texte des éditeurs de texte. Bien que d'abord et avant tout un éditeur pour programmeurs (avec des modes spéciaux pour une multitude de langages différents), c'est également un environnement idéal pour travailler sur des documents LaTeX, interagir avec R, S-Plus, SAS ou SQL, ou même pour lire son courrier électronique.

Le présent auteur distribue une version simple à installer et augmentée de quelques ajouts de la plus récente version de GNU Emacs pour Windows. Consulter le site Internet

`http://vgoulet.act.ulaval.ca/ressources/#Emacs`

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 al. (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*.

- Distribué sous la GNU *General Public License* (GPL), donc gratuit, ou «libre».
- Le nom provient de «*Editing MACroS*».
- La première version de Emacs a été écrite par Richard M. Stallman, président de la FSF.

A.2 Configuration de l'éditeur

Une des grandes forces de Emacs est d'être configurable à l'envi.

- Depuis la version 21, le menu `Customize` rend la configuration aisée.
- Une grande part de la configuration provient du fichier `.emacs` :
  - nommé `.emacs` sous Linux et Unix, Windows 2000 et Windows XP ;
  - sous Windows 95/98/Me, utiliser plutôt `_emacs`.

A.3 *Emacs-ismes* et *Unix-ismes*

- Un *buffer* contient un fichier ouvert («*visited*»). Équivalent à une fenêtre dans Windows.
- Le *minibuffer* est la région au bas de l'écran Emacs où l'on entre des commandes et reçoit de l'information de Emacs.
- La ligne de mode («*mode line*») est le séparateur horizontal contenant diverses informations sur le fichier ouvert et l'état de Emacs.
- Toutes les fonctionnalités de Emacs correspondent à une commande pouvant être tapée dans le *minibuffer*. `M-x` démarre l'interpréteur (ou invite) de commandes.
- Dans les définitions de raccourcis claviers :
  - `C` est la touche `Ctrl` (`Control`) ;
  - `M` est la touche `Meta`, qui correspond à la touche `Alt` de gauche sur un PC ;
  - `ESC` est la touche `Échap` (`Esc`) et est équivalente à `Meta` ;
  - `SPC` est la barre d'espacement ;
  - `RET` est la touche Entrée.
- Le caractère ~ représente le dossier vers lequel pointe la variable d'environnement `$HOME` (Unix) ou `%HOME%` (Windows).
- 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.4 Commandes d'édition de base

Il n'est pas vain de lire le tutoriel de Emacs, que l'on démarre avec

```
C-h t
```

Pour une liste plus exhaustive des commandes Emacs les plus importantes, consulter la *GNU Emacs Reference Card*, dans le fichier

```
.../emacs-21.x/etc/refcard.ps
```

- Pour créer un nouveau fichier, ouvrir un fichier n'existant pas.
- Principales commandes d'édition avec, entre parenthèses, le nom de la commande correspondant au raccourci clavier :

  `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-x C-c` quitter Emacs (`save-buffers-kill-emacs`)

`C-g`	bouton de panique : quitter ! (`keyboard-quit`)
`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`)
`C-x b`	changer de *buffer* (`switch-buffer`)
`C-x 2`	séparer l'écran en deux fenêtres (`split-window-vertically`)
`C-x 1`	conserver uniquement la fenêtre courante (`delete-other-windows`)
`C-x 0`	fermer la fenêtre courante (`delete-window`)
`C-x o`	aller vers une autre fenêtre lorsqu'il y en a plus d'une (`other-window`)

A.5 Sélection de texte

La sélection de texte fonctionne différemment du standard Windows.

- Les raccourcis clavier standards sous Emacs sont :

`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 existe quelques extensions de Emacs permettant d'utiliser les raccourcis clavier usuels de Windows (`C-c`, `C-x`, `C-v`) ; voir `http://www.emacswiki.org/cgi-bin/wiki/CuaMode`.

A.6 Mode ESS

Le mode ESS (*Emacs Speaks Statistics*) est un mode pour interagir avec des logiciels statistiques (S-Plus, R, SAS, etc.) depuis Emacs. Ce mode est installé dans la version modifiée de GNU Emacs distribuée dans le site Internet `http://vgoulet.act.ulaval.ca/pub/emacs/`

- Voir le fichier

  `.../emacs-21.x/site-lisp/ess/doc/html/index.html`

  pour la documentation complète.

- Deux modes mineurs : `ESS` pour les fichiers de script (code source) et `iESS` pour l'invite de commande.
- Une fois installé, le mode mineur `ESS` s'active automatiquement en éditant des fichiers avec l'extension `.S` ou `.R`.
- Commandes les plus fréquemment employées lors de l'édition d'un fichier de script (mode `ESS`) :

  `C-c C-n` évalue la ligne sous le curseur dans le processus S (`ess-eval-line-and-step`)

  `C-c C-r` évalue la région sélectionnée dans le processus S (`ess-eval-region`)

  `C-c C-f` évalue le code de la fonction courante dans le processus S (`ess-eval-function`)

  `C-c C-l` évalue le code du fichier courant dans le processus S (`ess-load-file`)

  `C-c C-v` aide sur une commande S (`ess-display-help-on-object`)

  `C-c C-s` changer de processus (utile si l'on a plus d'un processus S actif)

- Pour démarrer un processus S et activer le mode mineur `iESS`, entrer l'une des commandes `S`, `Sqpe` ou `R` dans l'invite de commande de Emacs (voir aussi l'annexe B). Par exemple, pour démarrer un processus `R` à l'intérieur même de Emacs, on fera

  ```
  M-x R RET
  ```

- Commandes le plus fréquemment employées à la ligne de commande (mode `iESS`) :

  `C-c C-e` replacer la dernière ligne au bas de la fenêtre (`comint-show-maximum-output`)

  `M-h` sélectionner le résultat de la dernière commande (`mark-paragraph`)

  `C-c C-o` effacer le résultat de la dernière commande (`comint-delete-output`)

  `C-c C-v` aide sur une commande S (`ess-display-help-on-object`)

  `C-c C-q` terminer le processus S (`ess-quit`)

B Utilisation de ESS et S-Plus sous Windows

L'utilisation de R et S-Plus avec ESS dans Emacs est virtuellement identique sous Unix. Sous Windows, la procédure est exactement la même que sous Unix pour R, mais l'interface avec S-Plus est légèrement plus compliquée.

Avant toute chose, il faut s'assurer d'avoir une installation de Emacs, ESS et S-Plus fonctionnelle. L'installation de la version modifiée de Emacs distribuées dans le site Internet

```
http://vgoulet.act.ulaval.ca/pub/emacs/
```

devrait permettre de satisfaire cette exigence rapidement.

Il y a deux façons de travailler avec S-Plus depuis Emacs sous Windows : tout dans Emacs ou une combinaison de Emacs et de l'interface graphique de S-Plus.

B.1 Tout dans Emacs

Cette approche est similaire à celle favorisée sous Unix ainsi qu'avec R. Un processus S-Plus est démarré à l'intérieur même de Emacs, un fichier de script (habituellement avec une extension `.S`) est ouvert dans Emacs et les lignes de ce fichier sont exécutées dans le processus S-Plus. La fenêtre Emacs est alors scindée en deux. C'est l'approche prônée à la section 1.7.

Le truc consiste ici à utiliser non pas l'exécutable `splus.exe` (qui est l'interface graphique), mais plutôt l'interface en ligne de commande, plus simple et rapide. L'exécutable est `sqpe.exe`. Pour démarrer une session S-Plus dans Emacs, on fera donc

```
M-x Sqpe RET
```

Lorsque demandé, on spécifie le dossier de travail. Une fois l'invite de commande S-Plus obtenue, on pourra exécuter des lignes du fichier de script dans le processus S-Plus avec `C-c C-n`, `C-c C-f`, etc.

Il y a toutefois un os avec cette approche : aucun périphérique graphique n'est disponible. Sauf depuis la version 6.1 de S-Plus : on peut utiliser un périphérique graphique Java. Afin de pouvoir l'utiliser, il faut exécuter les deux lignes suivantes *avant* de créer un graphique :

```
> library(winjava)
> java.graph()
```

Il est possible d'automatiser ce processus en sauvegardant ces deux lignes dans un fichier nommé `S.init` dans le dossier de travail. Le contenu de ce fichier sera exécuté à chaque fois que S-Plus sera démarré dans ce dossier.

B.2 Combinaison Emacs et interface graphique de S-Plus

Cette option est moins élégante que la précédente, mais certains pourraient lui voir comme avantage d'utiliser l'interface graphique (GUI) de S-Plus. En fin de compte, la procédure ci-dessous revient à remplacer par Emacs la fenêtre d'édition de script incluse dans S-Plus.

En faisant

```
M-x S RET
```

à l'intérieur de Emacs, une nouvelle session graphique de S-Plus sera démarrée (il faut être patient, les négociations entre les deux logiciels peuvent prendre du temps). On se retrouve donc avec deux fenêtres : une pour Emacs et une pour S-Plus.

Ouvrir un fichier de script dans Emacs et exécuter les lignes de comme ci-dessus. Les lignes de code seront exécutées dans l'interface graphique. En d'autres mots, le code source se trouve dans une fenêtre (Emacs) et les résultats de ce code source dans une autre (S-Plus). Il faut bien disposer les fenêtres côte à côte pour que cette stratégie se révèle minimalement efficace.

L'information ci-dessus se trouve dans la documentation de ESS.

C Générateurs de nombres aléatoires

Avant d'utiliser pour quelque tâche de simulation moindrement importante un générateur de nombres aléatoires inclus dans un logiciel, il importe de s'assurer de la qualité de celui-ci. On trouvera en général relativement facilement de l'information dans Internet.

On présente ici, sans entrer dans les détails, les générateurs de nombres uniformes utilisés dans S-Plus et R ainsi que la liste des différentes fonctions de simulation de variables aléatoires.

C.1 Générateurs de nombres aléatoires

On obtient des nombres uniformes sur un intervalle quelconque (par défaut $[0, 1]$) avec la fonction `runif` dans S-Plus et R. L'amorce du générateur aléatoire est déterminée avec la fonction `set.seed`.

Dans S-Plus, le générateur utilisé est une version modifiée de *Super Duper*. Sa période est $2^{30} \times 4\,292\,868\,097 \approx 4{,}6 \times 10^{18}$. S+

Dans R, on a la possibilité de choisir entre six générateurs de nombres aléatoires différents, ou encore de spécifier son propre générateur. Par défaut, R utilise le générateur Marsenne–Twister, considéré comme le plus avancé au moment d'écrire ces lignes. La période de ce générateur est $2^{19\,937} - 1$, rien de moins ! R

Consulter les rubriques d'aide des fonctions `.Random.seed` et `set.seed` pour de plus amples détails.

C.2 Fonctions de simulation de variables aléatoires

Les caractéristiques de plusieurs lois de probabilité sont directement accessibles dans S-Plus et R par un large éventail de fonctions. La logique règne dans les noms de fonctions : pour chaque racine *`loi`*, il existe quatre fonctions différentes :

1. `d`*`loi`* calcule la fonction de densité de probabilité (lois continues) ou la fonction de masse de probabilité (lois discrètes) ;
2. `p`*`loi`* calcule la fonction de répartition ;
3. `q`*`loi`* calcule la fonction de quantile ;

Loi de probabilité	Racine dans S	Noms des paramètres
Bêta	`beta`	`shape1`, `shape2`
Binomiale	`binom`	`size`, `prob`
Binomiale négative	`nbinom`	`size`, `prob` ou `mu`
Cauchy	`cauchy`	`location`, `scale`
Exponentielle	`exp`	`rate`
F (Fisher)	`f`	`df1`, `df2`
Gamma	`gamma`	`shape`, `rate` ou `scale`
Géométrique	`geom`	`prob`
Hypergéométrique	`hyper`	`m`, `n`, `k`
Khi carré	`chisq`	`df`
Logistique	`logis`	`location`, `scale`
Log-normale	`lnorm`	`meanlog`, `sdlog`
Normale	`norm`	`mean`, `sd`
Poisson	`pois`	`lambda`
t (Student)	`t`	`df`
Uniforme	`unif`	`min`, `max`
Weibull	`weibull`	`shape`, `scale`
Wilcoxon	`wilcox`	`m`, `n`

TAB. C.1: Lois de probabilité pour lesquelles existent des fonctions dans S-Plus et R

4. `rloi` simule des observations de cette loi.

Les différentes lois de probabilité disponibles dans S-Plus et R, leur racine et le nom de leurs paramètres sont rassemblées au tableau C.1

Toutes les fonctions du tableau C.1 sont vectorielles, c'est-à-dire qu'elles acceptent en argument un vecteur de points où la fonction (de densité, de répartition ou de quantile) doit être évaluée et même un vecteur de paramètres. Par exemple,

```
> dpois(c(3, 0, 8), lambda = c(1, 4, 10))
```

```
[1] 0.06131324 0.01831564 0.11259903
```

retourne la probabilité que des lois de Poisson de paramètre 1, 4, et 10 prennent les valeurs 3, 0 et 8, respectivement.

Le premier argument des fonctions de simulation est la quantité de nombres aléatoires désirée. Ainsi,

```
> rpois(3, lambda = c(1, 4, 10))
```

```
[1] 2 4 7
```

retourne trois nombres aléatoires issus de distributions de Poisson de paramètre 1, 4 et 10, respectivement. Évidemment, passer un vecteur comme premier argument n'a pas tellement de sens, mais, si c'est fait, S retournera une

quantité de nombres aléatoires égale à la *longueur* du vecteur (sans égard aux valeurs contenues dans le vecteur).

La fonction `sample` permet de simuler des nombres d'une distribution discrète quelconque. Sa syntaxe est

```
sample(x, size, replace = FALSE, prob = NULL),
```

où `x` est un vecteur des valeurs possibles de l'échantillon à simuler (le support de la distribution), `size` est la quantité de nombres à simuler et `prob` est un vecteur de probabilités associées à chaque valeur de `x` (`1/length(x)` par défaut). Enfin, si `replace` est `TRUE`, l'échantillonnage se fait avec remise.

C.3 Exercices

C.1 La loi log-normale est obtenue par transformation de la loi normale : si la distribution de la variable alatoire X est une normale de paramètres μ et σ^2, alors la distribution de e^X est une log-normale. Simuler 1 000 observations d'une loi log-normale de paramètres $\mu = \ln 5000 - \frac{1}{2}$ et $\sigma^2 = 1$, puis tracer l'histogramme de l'échantillon aléatoire obtenu.

C.2 Simuler 10 000 observations d'un mélange continu Poisson/gamma où les paramètres de la loi gamma sont $\alpha = 5$ et $\lambda = 4$, puis tracer la distribution de fréquence de l'échantillon aléatoire obtenu à l'aide des fonctions `plot` et `table`. Superposer à ce graphique la fonction de probabilité d'une binomiale négative de paramètres $r = 5$ et $\theta = 0{,}8$.

C.3 Simuler 10 000 observations d'un mélange discret de deux distributions log-normales, l'une de paramètres $(\mu = 3{,}5, \sigma^2 = 0{,}6)$ et l'autre de paramètres $(\mu = 4{,}6, \sigma^2 = 0{,}3)$. Utiliser un paramètre de mélange $p = 0{,}55$. Tracer ensuite l'histogramme de l'échantillon aléatoire obtenu.

D Planification d'une simulation en S

D.1 Introduction

La simulation est de plus en plus utilisée pour résoudre des problèmes complexes. Il existe de multiples façons de réaliser la mise en œuvre informatique d'une simulation, mais certaines sont plus efficaces que d'autres. Ce document passe en revue diverses façons de faire des simulations avec S-Plus et R. On procédera à l'aide d'un exemple simple de nature statistique.

Soit $X_1, \ldots, X_n$ un échantillon aléatoire tiré d'une population distribuée selon une loi uniforme sur l'intervalle $(\theta - \frac{1}{2}, \theta + \frac{1}{2})$. On considère les trois estimateurs suivants du paramètre inconnu θ :

1. la moyenne arithmétique

$$\hat{\theta}_1 = \frac{1}{n} \sum_{i=1}^{n} X_i \,;$$

2. la médiane empirique

$$\hat{\theta}_2 = \begin{cases} X_{(\frac{n+1}{2})}, & n \text{ impair} \\ \frac{1}{2}(X_{(\frac{n}{2})} + X_{(\frac{n}{2}+1)}), & n \text{ pair,} \end{cases}$$

   où $X_{(k)}$ est la k^{e} statistique d'ordre de l'échantillon aléatoire ;

3. la mi-étendue

$$\hat{\theta}_3 = \frac{X_{(1)} + X_{(n)}}{2}.$$

On veut vérifier par simulation deux choses : que les trois estimateurs sont bel et bien sans biais, et lequel a la plus faible variance. Pour ce faire, on doit d'abord simuler un grand nombre N d'échantillons aléatoires de taille n d'une distribution $U(\theta - \frac{1}{2}, \theta + \frac{1}{2})$ pour une valeur de θ choisie. Pour chaque échantillon, on calculera ensuite les trois estimateurs ci-dessus, puis la moyenne et la variance, par type d'estimateur, de tous les estimateurs obtenus. Si la moyenne des N estimateurs $\hat{\theta}_i, i = 1, 2, 3$ est près de θ, alors on pourra conclure que $\hat{\theta}_i$ est sans biais. De même, on déterminera lequel des trois estimateurs a la plus faible variance selon le classement des variances empiriques.

D.2 Première approche : avec une boucle

La façon la plus intuitive de mettre en œuvre cette étude de simulation en S consiste à utiliser une boucle `for`. Avec cette approche, il est nécessaire d'initialiser une matrice de 3 lignes et N colonnes (ou l'inverse) dans laquelle seront stockées les valeurs des trois estimateurs pour chaque simulation. Une fois la matrice remplie dans la boucle, il ne reste plus qu'à calculer la moyenne et la variance par ligne pour obtenir les résultats souhaités.

La figure D.1 présente un exemple de code adéquat pour réaliser la simulation à l'aide d'une boucle.

Si l'on souhaite pouvoir exécuter le code de la figure D.1 facilement à l'aide d'une seule expression, il suffit de placer l'ensemble du code dans une fonction. La fonction `simul1` de la figure D.2 reprend le code de la figure D.1, sans les commentaires. On a alors :

```
> simul1(10000, 100, 0)

$biais
     Moyenne      Mediane   Mi-etendue
1.544109e-04 8.259181e-05 9.461955e-05

$variances
     Moyenne      Mediane   Mi-etendue
8.274645e-04 2.420766e-03 4.799883e-05
```

D.3 Seconde approche : avec `sapply`

On le sait, les boucles sont inefficaces en S — tout particulièrement dans S-Plus. Il est en général plus efficace de déléguer les boucles aux fonctions `lapply` et `sapply` (section 6.3), dont la syntaxe est

`lapply(x, FUN, ...)` et `sapply(x, FUN, ...)`.

Celles-ci appliquent la fonction `FUN` à tous les éléments de la liste ou du vecteur `x` et retournent les résultats sous forme de liste (`lapply`) ou, lorsque c'est possible, de vecteur ou de matrice (`sapply`). Il est important de noter que les valeurs successives de `x` seront passées comme *premier* argument à la fonction `FUN`. Les autres arguments de `FUN`, s'il y a lieu, sont spécifiés dans le champ '`...`'.

Pour pouvoir utiliser ces fonctions dans le cadre d'une simulation telle que celle dont il est question ici, il s'agit de définir une fonction qui fera tous les calculs pour une simulation, puis de la passer à `sapply` pour obtenir les résultats de N simulations. La figure D.3 présente une première version d'une telle fonction. On remarquera que l'argument `i` ne joue aucun rôle dans la fonction. Voici un exemple d'utilisation pour un petit nombre (4) de simulations :

```
### Bonne habitude à prendre: stocker les constantes dans
### des variables faciles à modifier au lieu de les écrire
### explicitement dans le code.
size <- 100                # taille de chaque échantillon
nsimul <- 10000            # nombre de simulations
theta <- 0                 # la valeur du paramètre

### Les lignes ci-dessous éviteront de faire deux additions
### 'nsimul' fois.
a <- theta - 0.5           # borne inférieure de l'uniforme
b <- theta + 0.5           # borne supérieure de l'uniforme

### Initialisation de la matrice dans laquelle seront
### stockées les valeurs des estimateurs. On donne également
### des noms aux lignes de la matrice afin de facilement
### identifier les estimateurs.
x <- matrix(0, nrow=3, ncol=nsimul)
rownames(x) <- c("Moyenne", "Mediane", "Mi-etendue")

### Simulation comme tel.
for (i in 1:nsimul)
{
    u <- runif(size, a, b)
    x[1, i] <- mean(u)     # moyenne
    x[2, i] <- median(u)   # médiane
    x[3, i] <- mean(range(u)) # mi-étendue
}

### On peut maintenant calculer la moyenne et la variance
### par ligne.
rowMeans(x) - theta        # vérification du biais
apply(x, 1, var)           # comparaison des variances
```

FIG. D.1: Code pour la simulation utilisant une boucle `for`

```
simul1 <- function(nsimul, size, theta)
{
    a <- theta - 0.5
    b <- theta + 0.5

    x <- matrix(0, nrow=3, ncol=nsimul)
    rownames(x) <- c("Moyenne","Mediane","Mi-etendue")

    for (i in 1:nsimul)
    {
        u <- runif(size, a, b)
        x[1, i] <- mean(u)
        x[2, i] <- median(u)
        x[3, i] <- mean(range(u))
    }

    list(biais=rowMeans(x) - theta,
         variances=apply(x, 1, var))
}
```

FIG. D.2: Définition de la fonction `simul1`

```
fun1 <- function(i, size, a, b)
{
    u <- runif(size, a, b)
    c(Moyenne=mean(u),
      Mediane=median(u),
      "Mi-etendue"=mean(range(u)))
}
```

FIG. D.3: Définition de la fonction `fun1`

```
simul2 <- function(nsimul, size, theta)
{
    a <- theta - 0.5
    b <- theta + 0.5

    x <- sapply(1:nsimul, fun1, size, a, b)

    list(biais=rowMeans(x) - theta,
         variances=apply(x, 1, var))
}
```

FIG. D.4: Définition de la fonction `simul2`

```
> sapply(1:4, fun1, size = 10, a = -0.5, b = 0.5)
                    [,1]        [,2]         [,3]
Moyenne    -0.009050612 -0.18045795 -0.005573472
Mediane    -0.023214453 -0.29474525 -0.015951460
Mi-etendue -0.027885151 -0.03705596 -0.002689286
                   [,4]
Moyenne    -0.08354421
Mediane    -0.28099295
Mi-etendue  0.01804147
```

On remarque donc que les résultats de chaque simulation se trouvent dans les colonnes de la matrice obtenue avec `sapply`.

Pour compléter l'analyse, on englobe le tout dans une fonction `simul2`, dont le code se trouve à la figure D.4 :

```
> simul2(10000, 100, 0)

$biais
     Moyenne      Mediane   Mi-etendue
1.777698e-04 1.461156e-05 1.013551e-04

$variances
     Moyenne      Mediane   Mi-etendue
8.373507e-04 2.445200e-03 4.783119e-05
```

Il est généralement plus facile de déboguer le code avec cette approche.

D.4 Variante de la seconde approche

Une chose manque d'élégance dans la seconde approche : le fait de devoir inclure un argument factice dans la fonction `fun1`. La fonction `replicate` R

```
fun2 <- function(size, a, b)
{
    u <- runif(size, a, b)
    c(Moyenne=mean(u),
      Mediane=median(u),
      "Mi-etendue"=mean(range(u)))
}
```

FIG. D.5: Définition de la fonction `fun2`

```
simul3 <- function(nsimul, size, theta)
{
    a <- theta - 0.5
    b <- theta + 0.5

    x <- replicate(nsimul, fun2(size, a, b))

    list(biais=rowMeans(x) - theta,
         variances=apply(x, 1, var))
}
```

FIG. D.6: Définition de la fonction `simul3`

(section 6.5), disponible dans R seulement, permet d'éviter cela : cette fonction exécutera un nombre donné de fois une expression quelconque. Les fonctions `fun2` et `simul3` des figures D.5 et D.6, respectivement, sont des versions légèrement modifiées de `fun1` et `simul2` pour utilisation avec `replicate`.

On a alors

```
> simul3(10000, 100, 0)

$biais
      Moyenne       Mediane    Mi-etendue
-8.022200e-05 -4.698536e-04  7.155081e-05

$variances
     Moyenne      Mediane   Mi-etendue
8.244527e-04 2.407576e-03 4.880371e-05
```

D.5 Comparaison des temps de calcul

A-t-on gagné quoi que ce soit en termes de temps de calcul d'une approche à l'autre ? La fonction `system.time` de R (ou `sys.time` de S-Plus) permet de mesurer le temps requis pour l'exécution d'une expression. Le premier résultat de `system.time` est le temps CPU utilisé et le troisième, le temps total écoulé. Sous Windows, les quatrième et cinquième résultats sont `NA`.

```
> system.time(simul1(10000, 100, 0))

[1] 8.16 0.00 8.30 0.00 0.00

> system.time(simul2(10000, 100, 0))

[1] 7.85 0.05 8.04 0.00 0.00

> system.time(simul3(10000, 100, 0))

[1] 7.54 0.02 7.66 0.00 0.00
```

Les différences, petites ici, peuvent être plus importantes lors de grosses simulations et davantage favoriser l'utilisation de la fonction `replicate`.

D.6 Gestion des fichiers

Pour un petit projet comme celui utilisé en exemple ici, il est simple et pratique de placer tout le code informatique dans un seul fichier de script. Pour un plus gros projet, cependant, il vaut souvent mieux avoir recours à plusieurs fichiers différents. Le présent auteur utilise pour sa part un fichier par fonction.

Pour des fins d'illustrations, supposons que l'on utilise l'approche de la section D.4 avec la fonction `replicate` en R et que le code des fonctions `fun2` et `simul3` est sauvegardé dans des fichiers `fun2.R` et `simul3.R`, respectivement. Si l'on crée un autre fichier, `go.R`, ne contenant que des expressions `source` pour lire les autres fichiers, il est alors possible de démarrer des simulations en exécutant ce seul fichier. Dans notre exemple, le fichier `go.R` contiendrait les lignes suivantes :

```
source("fun2.R")
source("simul3.R")
simul3(10000, 100, 0)
```

Une simple commande

```
> source("go.R")
```

exécutera alors une simulation complète.

D.7 Exécution en lot

On peut accélérer le traitement d'une simulation en l'exécutant en lot — ou mode *batch* — et ce, avec S-Plus comme avec R. Dans ce mode, aucune interface graphique n'est démarrée et tous les résultats sont redirigés vers un fichier pour consultation ultérieure. Pour les simulations demandant un long temps de calcul, c'est très pratique.

Pour exécuter S-Plus ou R en lot sous Windows, ouvrir une invite de commande (dans le menu Accessoires du menu Démarrer) puis se déplacer (à l'aide de la commande `cd`) dans le dossier où sont sauvegardés les fichiers de script. Avec S-Plus, il faut par la suite exécuter la commande suivante :

```
C:\> Splus /BATCH go.S go.Sout
```

Le troisième élément de cette commande est le nom du fichier de script contenant les expressions à exécuter et le quatrième, le nom du fichier dans lequel seront sauvegardés les résultats. Ils peuvent évidemment être différents de ceux ci-dessus.

Avec R, la syntaxe est plutôt

```
C:\> R CMD BATCH go.R
```

et les résultats se trouveront par défaut dans le fichier `go.Rout`. Si l'exécutable de R n'est pas trouvé par Windows, il faut spécifier le chemin d'accès complet, comme par exemple :

```
C:\> "c:\program files\R\R-2.2.0\bin\R" CMD BATCH go.R
```

Depuis la version 6.2, S-Plus sous Windows contient un outil BATCH dans le dossier S-Plus du menu Démarrer facilitant l'utilisation en lot. Il suffit de remplir les champs appropriés dans la boîte de dialogue.

D.8 Quelques remarques

S+ 1. La fonction `rownames` utilisée dans la figure D.1 existe seulement dans R. Dans S-Plus, on utilisera plutôt `row.names` ou `dimnames`.

S+ 2. Dans S-Plus, on peut calculer la variance par ligne ou par colonne d'une matrice avec les fonctions `rowVars` et `colVars`.

3. Le nombre de simulations, N, et la taille de l'échantillon, n, ont tous deux un impact sur la qualité des résultats, mais de manière différente. Quand n augmente, la précision des estimateurs augmente. Ainsi, dans l'exemple ci-dessus, le biais et la variance des estimateurs de θ seront plus faibles. D'autre part, l'augmentation du nombre de simulations diminue l'impact des échantillons aléatoires individuels et, de ce fait, améliore la fiabilité des conclusions de l'étude.

4. Conclusion de l'étude de simulation sur le biais et la variance des trois estimateurs de la moyenne d'une loi uniforme : les trois estimateurs sont sans biais et la mi-étendue a la plus faible variance. On peut d'ailleurs prouver que, pour n impair,

$$\begin{aligned}
\mathrm{Var}[\hat{\theta}_1] &= \frac{1}{12n} \\
\mathrm{Var}[\hat{\theta}_2] &= \frac{1}{4n+2} \\
\mathrm{Var}[\hat{\theta}_3] &= \frac{1}{2(n+1)(n+2)}
\end{aligned}$$

et donc

$$\mathrm{Var}[\hat{\theta}_3] \leqslant \mathrm{Var}[\hat{\theta}_1] \leqslant \mathrm{Var}[\hat{\theta}_2]$$

pour tout $n \geqslant 2$.

E GNU Free Documentation License

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.

E.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 re-

quires 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.

E.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.

E.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.

E.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.

E.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".

E.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.

E.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.

E.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.

E.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.

E.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.

Réponses des exercices

Chapitre 2

2.1 Soit x le nom de la liste.

(a)
```
> list(1:5, data = matrix(1:6, 2, 3), numeric(3),
+     test = logical(4))
```

(b)
```
> names(x)
```

(c)
```
> mode(x$test)
> length(x$test)
```

(d)
```
> dim(x$data)
```

(e)
```
> x[[2]][c(2, 3)]
```

(f)
```
> x[[3]] <- 3:8
```

2.2 (a)
```
> obs[2]
```

(b)
```
> obs[1:5]
```

(c)
```
> obs[obs > 14]
```

(d)
```
> obs[-c(6, 10, 12)]
```

2.3 (a)
```
> mat[4, 3]
```

(b)
```
> mat[6, ]
```

(c)
```
> mat[, c(1, 4)]
```

(d)
```
> mat[mat[, 1] > 50, ]
```

Chapitre 3

3.1 (a)
```
> rep(c(0, 6), 3)
```

(b)
```
> seq(1, 10, by = 3)
```

(c)
```
> rep(1:3, 4)
```

(d)
```
> rep(1:3, 1:3)
```

(e)
```
> rep(1:3, 3:1)
```

(f)
```
> seq(1, 10, length = 3)
```

(g) `> rep(1:3, rep(4, 3))`

3.2 (a) `> 11:20/10`

(b) `> 2 * 0:9 + 1`

(c) `> rep(-2:2, 2)`

(d) `> rep(-2:2, each = 2)`

(e) `> 10 * 1:10`

3.3 Soit `mat` une matrice.

(a) `> apply(mat, 1, sum)`

(b) `> apply(mat, 2, sum)`

(c) `> apply(mat, 1, mean)`

(d) `> apply(mat, 2, mean)`

3.4 `> cumprod(1:10)`

3.5 `x == (x %% y) + y * ( x %/% y )`

3.6 (a)
```
> x[1:5]
> head(x, 5)
```

(b) `> max(x)`

(c)
```
> mean(x[1:5])
> mean(head(x, 5))
```

(d)
```
> mean(x[16:20])
> mean(x[(length(x) - 4):length(x)])
> mean(tail(x, 5))
```

3.7 (a) `(j - 1)*I + i`

(b) `((k - 1)*J + j - 1)*I + i`

3.8 (a) `> rowSums(mat)`

(b) `> colMeans(mat)`

(c) `> max(mat[1:3, 1:3])`

(d) `> mat[rowMeans(mat) > 7, ]`

3.9 `> temps[match(unique(cummin(temps)), temps)]`

Chapitre 4

4.1 `> sum(P/cumprod(1 + i))`

4.2
```
> x <- c(7, 13, 3, 8, 12, 12, 20, 11)
> w <- c(0.15, 0.04, 0.05, 0.06, 0.17, 0.16,
+     0.11, 0.09)
> sum(x * w)/sum(w)
```

4.3
```
> 1/mean(1/x)
```

4.4
```
> lambda <- 2
> x <- 5
> exp(-lambda) * sum(lambda^(0:x)/gamma(1 +
+     0:x))
```

4.5 (a)
```
> x <- 10^(0:6)
> probs <- (1:7)/28
```

(b)
```
> sum(x^2 * probs) - (sum(x * probs))^2
```

4.6
```
> i <- 0.06
> 4 * ((1 + i)^0.25 - 1)
```

4.7
```
> n <- 1:10
> i <- seq(0.05, 0.1, by = 0.01)
> (1 - outer((1 + i), -n, "^"))/i
```

ou

```
> n <- 1:10
> i <- (5:10)/100
> apply(outer(1/(1 + i), n, "^"), 1, cumsum)
```

4.8
```
> v <- 1/1.06
> k <- 1:10
> sum(k * v^(k - 1))
```

4.9
```
> pmts <- rep(1:4, 1:4)
> v <- 1/1.07
> k <- 1:10
> sum(pmts * v^k)
```

4.10
```
> v <- cumprod(1/(1 + rep(c(0.05, 0.08),
+     5)))
> pmts <- rep(1:4, 1:4)
> sum(pmts * v)
```

Chapitre 5

5.1
```
variance <- function(x, biased=FALSE)
{
    if (biased)
    {
        n <- length(x)
        (n - 1)/n * var(x)
    }
    else
        var(x)
}
```

5.2 Une première solution utilise la transposée. La première expression de la fonction s'assure que la longueur de `data` est compatible avec le nombre de lignes et de colonnes de la matrice demandée.

```
matrix2 <- function(data=NA, nrow=1, ncol=1,
                    bycol=FALSE, dimnames=NULL)
{
    data <- rep(data, length=nrow * ncol)

    if (bycol)
        dim(data) <- c(nrow, ncol)
    else
    {
        dim(data) <- c(ncol, nrow)
        data <- t(data)
    }

    dimnames(data) <- dimnames
    data
}
```

La seconde solution n'a pas recours à la transposée. Si l'on doit remplir la matrice par ligne, l'idée consiste à réordonner les éléments du vecteur `data` en utilisant la formule obtenue à l'exercice 3.7.

```
matrix2 <- function(data=NA, nrow=1, ncol=1,
                    bycol=FALSE, dimnames=NULL)
{
    data <- rep(data, length=nrow * ncol)

    if (!bycol)
    {
        i <- 1:nrow
        j <- rep(1:ncol, each=nrow)
        data <- data[(i - 1)*ncol + j]
    }
    dim(data) <- c(nrow, ncol)
    dimnames(data) <- dimnames
    data
}
```

5.3
```
phi <- function(x)
{
    exp(-x^2/2) / sqrt(2 * pi)
}
```

5.4
```
Phi <- function(x)
{
    n <- 1 + 2 * 0:50
```

```
    0.5 + phi(x) * sum(x^n / cumprod(n))
}
```

5.5 Première solution utilisant une fonction interne et une structure de contrôle `if ... else`.

```
Phi <- function(x)
{
    fun <- function(x)
    {
        n <- 1 + 2 * 0:50
        0.5 + phi(x) * sum(x^n / cumprod(n))
    }

    if (x < 0)
        1 - fun(-x)
    else
        fun(x)
}
```

Seconde solution récursive, c'est-à-dire que si $x < 0$, la fonction s'appelle elle-même avec un argument positif.

```
Phi <- function(x)
{
    if (x < 0)
        1 - Recall(-x)
    else
    {
        n <- 1 + 2 * 0:50
        0.5 + phi(x) * sum(x^n / cumprod(n))
    }
}
```

Troisième solution sans structure de contrôle `if ... else`. Rappelons que dans des calculs algébriques, FALSE vaut 0 et TRUE vaut 1.

```
Phi <- function(x)
{
    n <- 1 + 2 * 0:50
    neg <- x < 0
    x <- abs(x)
    neg + (-1)^neg * (0.5 + phi(x) *
                      sum(x^n / cumprod(n)))
}
```

5.6

```
Phi <- function(x)
{
    n <- 1 + 2 * 0:30
    0.5 + phi(x) * colSums(t(outer(x, n, "^")) /
```

```
                                cumprod(n))
    }
```

5.7 (a)

```
prod.mat <- function(mat1, mat2)
    {
        if (ncol(mat1) == nrow(mat2))
        {
            res <- matrix(0, nrow=nrow(mat1),
                          ncol=ncol(mat2))
            for (i in 1:nrow(mat1))
            {
                for (j in 1:ncol(mat2))
                {
                    res[i, j] <- sum(mat1[i,] * mat2[,j])
                }
            }
            res
        }
        else
            stop("Les dimensions des matrices ne
    permettent pas le produit matriciel.")
    }
```

(b)

```
prod.mat<-function(mat1, mat2)
    {
        if (ncol(mat1) == nrow(mat2))
        {
            res <- matrix(0, nrow=nrow(mat1),
                          ncol=ncol(mat2))
            for (i in 1:nrow(mat1))
                res[i,] <- colSums(mat1[i,] * mat2)
            res
        }
        else
            stop("Les dimensions des matrices ne
    permettent pas le produit matriciel.")
    }
```

Solutions bonus : deux façons de faire équivalentes qui cachent la boucle dans un `sapply`.

```
prod.mat<-function(mat1, mat2)
{
    if (ncol(mat1) == nrow(mat2))
        t(sapply(1:nrow(mat1),
                 function(i) colSums(mat1[i,] * mat2)))
    else
        stop("Les dimensions des matrices ne permettent
pas le produit matriciel.")
```

```
}

prod.mat<-function(mat1, mat2)
{
    if (ncol(mat1) == nrow(mat2))
        t(sapply(1:ncol(mat2),
                 function(j) colSums(t(mat1) * mat2[,j])))
    else
        stop("Les dimensions des matrices ne permettent
pas le produit matriciel.")
}
```

5.8 `notes.finales <- function(notes, p) notes %*% p`

5.10
```
param <- function (moyenne, variance, loi)
 {
     loi <- tolower(loi)
     if (loi == "normale")
     {
         param1 <- moyenne
         param2 <- sqrt(variance)
         return(list(mean=param1, sd=param2))
     }
     if (loi == "gamma")
     {
         param2 <- moyenne/variance
         param1 <- moyenne * param2
         return(list(shape=param1, scale=param2))
     }
     if (loi == "pareto")
     {
         cte <- variance/moyenne^2
         param1 <- 2 * cte/(cte-1)
         param2 <- moyenne * (param1 - 1)
         return(list(alpha=param1, lambda=param2))
     }
     stop("La loi doit etre une de \"normale\",
 \"gamma\" ou \"pareto\"")
 }
```

Chapitre 6

6.1 Soit `Xij` et `wij` des matrices, et `Xijk` et `wijk` des tableaux à trois dimensions.

(a) `> rowSums(Xij * wij)/rowSums(wij)`

(b) `> colSums(Xij * wij)/colSums(wij)`

(c)
```
> sum(Xij * wij)/sum(wij)
```

(d)
```
> apply(Xijk * wijk, c(1, 2), sum)/apply(wijk,
+     c(1, 2), sum)
```

(e)
```
> apply(Xijk * wijk, 1, sum)/apply(wijk, 1,
+     sum)
```

(f)
```
> apply(Xijk * wijk, 2, sum)/apply(wijk, 2,
+     sum)
```

(g)
```
> sum(Xijk * wijk)/sum(wijk)
```

6.2 (a)
```
> unlist(lapply(0:10, seq, from = 0))
```

(b)
```
> unlist(lapply(1:10, seq, from = 10))
```

(c)
```
> unlist(lapply(10:1, seq, to = 1))
```

6.3 (a)
```
> ea <- lapply(seq(100, 300, by = 50), rpareto,
+     alpha = 2, lambda = 5000)
```

(b)
```
> names(ea) <- paste("echantillon", 1:5, sep = "")
```

(c)
```
> sapply(ea, mean)
```

(d)
```
> lapply(ea, function(x) sort(ppareto(x, 2,
+     5000)))
> lapply(lapply(ea, sort), ppareto, alpha = 2,
+     lambda = 5000)
```

(e)
```
> hist(ea$echantillon5)
```

(f)
```
> lapply(ea, "+", 1000)
```

6.4 (a)
```
> mean(sapply(x, function(liste) liste$franchise))
```

Les crochets utilisés pour l'indiçage constituent en fait un opérateur dont le «nom» est `[[`. On peut donc utiliser cet opérateur dans la fonction `sapply` :

```
> mean(sapply(x, "[[", "franchise"))
```

(b)
```
> sapply(x, function(x) mean(x$nb.acc))
```

(c)
```
> sum(sapply(x, function(x) sum(x$nb.acc)))
```

ou

```
> sum(unlist(sapply(x, "[[", "nb.acc")))
```

(d)
```
> mean(unlist(lapply(x, "[[", "montants")))
```

(e)
```
> sum(sapply(x, function(x) sum(x$nb.acc) ==
+     0))
```

(f)
```
> sum(sapply(x, function(x) x$nb.acc[1] ==
+     1))
```

(g)
```
> var(unlist(lapply(x, function(x) sum(x$nb.acc))))
```

(h)
```
> sapply(x, function(x) var(x$nb.acc))
```

(i)
```
> y <- unlist(lapply(x, "[[", "montants"))
> sum(y <= x)/length(y)
```

La fonction `ecdf` retourne une fonction permettant de calculer la fonction de répartition empirique en tout point :

```
> ecdf(unlist(lapply(x, "[[", "montants")))(x)
```

(j)
```
> y <- unlist(lapply(x, "[[", "montants"))
> colSums(outer(y, x, "<="))/length(y)
```

La fonction retournée par `ecdf` accepte un vecteur de points en argument :

```
> ecdf(unlist(lapply(x, "[[", "montants")))(x)
```

Chapitre 7

7.1 (a)
```
> f <- function(x) x^3 - 2 * x^2 - 5
> uniroot(f, lower = 1, upper = 4)
```

(b)
```
> f <- function(x) x^3 + 3 * x^2 - 1
> uniroot(f, lower = -4, upper = -1)
```

(c)
```
> f <- function(x) x - 2^(-x)
> uniroot(f, lower = 0, upper = 1)
```

(d)
```
> f <- function(x) exp(x) + 2^(-x) + 2 * cos(x) -
+     6
> uniroot(f, lower = 1, upper = 2)
```

(e)
```
> f <- function(x) exp(x) - x^2 + 3 * x -
+     2
> uniroot(f, lower = 0, upper = 1)
```

7.2
```
> X <- c(2061, 1511, 1806, 1353, 1600)
> w <- c(100155, 19895, 13735, 4152, 36110)
> g <- function(a, X, w, s2) {
+     z <- 1/(1 + s2/(a * w))
+     Xz <- sum(z * X)/sum(z)
+     sum(z * (X - Xz)^2)/(length(X) - 1)
+ }
> uniroot(function(x) g(x, X, w, 1.4e+08) -
+     x, c(50000, 80000))
```

7.3
```
> dpareto <- function(x, alpha, lambda) {
+     (alpha * lambda^alpha)/(x + lambda)^(alpha +
+         1)
+ }
> f <- function(par, x) -sum(log(dpareto(x,
+     par[1], par[2])))
> optim(c(1, 1000), f, x = x)
```

ou

```
> dpareto <- function(x, logAlpha, logLambda) {
+     alpha <- exp(logAlpha)
+     lambda <- exp(logLambda)
+     (alpha * lambda^alpha)/(x + lambda)^(alpha +
+         1)
+ }
> optim(c(log(2), log(1000)), f, x = x)
> exp(optim(c(log(2), log(1000)), f, x = x)$par)
```

Chapitre 9

9.2 (a) `> arima.sim(100, model = list(sd = sqrt(2)))`

(b) `> arima.sim(100, model = list(ma = 0.8))`

(c) `> arima.sim(100, model = list(ma = -0.6, sd = 10))`

(d) `> arima.sim(100, model = list(ma = c(0.5, 0.4)))`

(e) `> arima.sim(100, model = list(ar = 0.8))`

(f) `> arima.sim(100, model = list(ar = -0.9, sd = 10))`

(g) `> arima.sim(100, model = list(ar = c(0.7, -0.1)))`

Annexe C

C.1
```
> x <- rlnorm(1000, meanlog = log(5000) -
+     0.5, sdlog = 1)
> hist(x)
```

C.2
```
> x <- rpois(10000, lambda = rgamma(10000,
+     shape = 5, rate = 4))
> px <- table(x)
> plot(px/sum(px))
> points(0:10, dnbinom(0:10, size = 5, prob = 0.8))
```

C.3
```
> w <- rbinom(1, 10000, 0.55)
> x <- c(rlnorm(w, 3.5, 0.6), rlnorm(10000 -
+     w, 4.6, 0.3))
> hist(x)
```

Bibliographie

Cameron, D., Elliott, J., Loy, M., Raymond, E. S. & Rosenblatt, B. (2004), *Leaning GNU Emacs*, third edn, O'Reilly.

Venables, W. N. & Ripley, B. D. (2000), *S programming*, Springer, New York.

Venables, W. N. & Ripley, B. D. (2002), *Modern applied statistics with S*, fourth edn, Springer, New York.

Venables, W. N., Smith, D. M. & the R Development Core Team (2005), *An introduction to R : A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria.

ISBN 2-9809136-0-X