Le logiciel R

Maîtriser le langage
Effectuer des analyses
(bio)statistiques

Pierre Lafaye de Micheaux
Rémy Drouilhet
Benoit Liquet

Springer

À Dominique, à Luka et à Mathias
À mes parents

À tous ceux qui ont contribué, contribuent et contribueront
à éveiller nos consciences

À Pierre et à sa persévérance

Avant-propos

Cet ouvrage est fondé sur les notes d'un cours dispensé pendant quelques années à l'Institut universitaire de technologie de Grenoble 2, au sein du département Statistique et informatique décisionnelle (STID). Il a donc été « digéré » pour la première fois, dans une version très imparfaite, par les étudiants de ce département que nous remercions ici. Sans l'intérêt témoigné par ces derniers, cet ouvrage n'aurait probablement pas vu le jour. Nous voulons également vivement remercier notre collègue et ami Michel Lejeune, qui a réussi à nous convaincre de travailler à la rédaction d'un manuscrit à soumettre aux éditions Springer. Nous souhaitons aussi souligner l'importance du hasard qui a permis que les trajectoires des trois auteurs de ce livre se croisent dans un même lieu, pendant quelques années. L'expérience humaine et scientifique qui a résulté de cette rencontre a été très enrichissante, et chacun des auteurs a pu apporter des compétences complémentaires ayant permis de venir à bout du travail considérable qu'a nécessité la rédaction de cet ouvrage. Nous tenons enfin à remercier ici très chaleureusement Matthieu Dubois, un collègue et ami, chercheur en psychologie expérimentale et féru de R et de l'environnement Macintosh qui a été le premier à lire ce livre dans sa version quasi finalisée et nous a conseillé de nombreuses améliorations.

L'information contenue dans ce livre a été choisie et organisée de la meilleure façon possible afin d'être **exhaustive** tout en étant également **assimilable** par le lecteur. Cet ouvrage peut ainsi servir comme support d'un cours sur le logiciel R à un niveau de débutant à avancé. Une emphase particulière a été mise sur la forme du livre, ce qui, à notre sens, permet d'en faciliter la compréhension. Il devrait aussi pouvoir être utilisé comme un support d'auto-apprentissage par tout autodidacte. Notons que la présentation de l'ouvrage sera majoritairement indépendante de tout système d'exploitation. Toutefois, quelques chapitres seront destinés principalement à des utilisateurs du système d'exploitation Microsoft Windows. Nous pensons également utile de donner, par endroits, des compléments pour les utilisateurs de Linux ou de Macintosh.

Les chapitres du livre sont tous structurés de la même manière. Chaque chapitre débute par un petit encart indiquant les pré-requis nécessaires à la lecture dudit chapitre ainsi qu'un descriptif succinct du contenu du chapitre.

Les notions théoriques sont agrémentées de nombreux exemples et également parsemées de pauses invitant à pratiquer directement sur l'ordinateur ce qui a été vu. Chaque chapitre se termine enfin par une partie de contrôle de l'acquisition des connaissances sous la forme d'un encadré de termes à retenir, suivie d'une section d'exercices théoriques à faire sur feuille, et pouvant servir de questions à un examen sur table. Une fiche de travaux pratiques est également fournie en fin de chapitre. Celle-ci permet de vérifier que les compétences pratiques ont bien été assimilées. Notez que les exercices et les travaux pratiques doivent être traités uniquement avec les notions apprises dans les chapitres précédents.

La trame séquentielle du livre se déroule comme suit. Après une brève introduction destinée à mettre le lecteur en appétit, et la présentation de quelques jeux de données qui seront exploités tout au long de l'ouvrage pour illustrer l'utilisation de R, la première partie du livre est ensuite dédiée à l'apprentissage des concepts principaux du logiciel R : organisation des données, importation et exportation, manipulations diverses, accès à la documentation, représentations graphiques, programmation et maintenance. Cette partie consiste donc à « faire ses gammes » sur R.

La seconde partie du livre est consacrée à l'utilisation du logiciel R dans quelques contextes mathématiques et statistiques. Cette partie devrait être lue après les chapitres de la première partie, mais elle devrait tout de même se révéler accessible aux utilisateurs possédant déjà quelques notions de R. Elle contient les instructions R nécessaires pour quelques-uns des principaux cours de statistique et de mathématiques jusqu'à la licence (couvrant par exemple le programme en IUT de statistique et informatique décisionnelle en France) : calcul matriciel, intégration, optimisation, statistiques descriptives, simulations, intervalles de confiance et tests d'hypothèses, régression linéaire simple et multiple, analyse de la variance.

Notons enfin que chaque chapitre de statistique dans la seconde partie s'appuie sur un ou plusieurs jeux de données réelles, gracieusement mis à disposition par l'ISPED (Institut de santé publique, d'épidémiologie et de développement de Bordeaux) et présentés en début d'ouvrage, qui en rendent ainsi l'apprentissage plus concret et plus attractif. Nous en profitons pour remercier particulièrement toute l'équipe pédagogique du master de santé publique de l'ISPED. Ces données, ainsi que plusieurs fonctions développées spécialement pour le livre, et qui y sont présentées ou utilisées, sont disponibles dans un *package* R associé à l'ouvrage qui s'appelle `LeLogicielR`. Nous remercions également Mohamed El Methni et Taghi Barumandzadeh pour le matériel qu'ils nous ont fourni dans la rédaction du chapitre sur l'ANOVA.

Deuxième édition

Nous tenons à remercier Hubert Raymondaud pour nous avoir donné la motivation nécessaire pour écrire cette seconde édition, qui s'accroît de près de 200 pages. Plusieurs erreurs mineures ont été corrigées, certaines notions clarifiées et de nombreuses astuces ou renvois vers d'autres ressources ont été ajoutés au fil du texte.

La section A.4, intitulée « L'interface graphique de **R** (GUI) », a été tronquée et une nouvelle section A.5 intitulée « Mes premiers pas en **R** » a été ajoutée. Dans cette dernière, nous décrivons l'utilisation de l'outil `RCommander`, un package permettant l'utilisation de **R** via des menus, puis expliquons comment utiliser au mieux **R** via sa console.

Dans le Chapitre 2, un nouvelle section 2.4, intitulée « Lecture/écriture dans les bases de données », a été ajoutée.

Dans le Chapitre 3, la section 3.4 a été déplacée après la section 3.7. Elle devient donc la nouvelle section 3.7. Une nouvelle section 3.8, intitulée « Création de fonctions », a été ajoutée après cette section suivie d'une nouvelle section 3.9, intitulée « Représentation des nombres à virgule fixe, flottante », expliquant les problèmes numériques pouvant survenir du fait des limites de représentation des nombres sur un ordinateur. De plus, un TP sur la création de fonctions (le F-) a été ajouté à la toute fin du TP du Chapitre 3.

Dans le Chapitre 6, une nouvelle section 6.5, intitulée « Interfacer **R** et `C/C++` ou `Fortran` », fait son apparition juste avant l'ancienne section 6.5 « Gestion de son activité de développement » qui s'intitule désormais « Débogage de fonctions » et porte le numéro 6.6. Le contenu de toute cette section a été modifié et largement augmenté. L'ancienne sous-section 6.5.1 « Débogage de fonctions » de la version 1 devient la sous-section 6.6.1 « Débogage de fonctions en **R** pur ». Nous avons aussi rajouté une section 6.7 intitulée « Calcul parallèle et calculs sur cartes graphiques ».

Le titre du Chapitre 10 a été changé en « Variables aléatoires, lois et simulations : une meilleure compréhension grâce aux spécificités de **R** » pour être plus représentatif de son contenu.

Pour finir, la correction de tous les exercices et de tous les TPs a été intégrée dans l'ouvrage, ce qui en fait très probablement le manuel le plus complet à ce jour sur le logiciel **R**. Celui-ci pourra être utilisé pour former les lycéens français dans le cadre du nouveau programme national, ainsi que les étudiants des classes préparatoires et de l'université. Il permet toujours de mener ses lecteurs à un stade avancé de maîtrise du logiciel.

Parcours différenciés

Nous avons mentionné explicitement, à l'aide du symbole †, les sections plus délicates ou moins fondamentales pouvant être écartées lors d'une première lecture de l'ouvrage, sans pour autant nuire à la compréhension et à la maîtrise du logiciel R.

Notez que ce livre a d'abord été pensé pour être lu par des étudiants issus de formations mathématiques ou statistiques. Toutefois, nous proposons ci-dessous, pour les étudiants ou les chercheurs ayant suivi un parcours plus « appliqué », d'adopter un parcours différencié pour le cœur de l'ouvrage. La lecture des sections délicates sera également omise.

PARTIE I : LES BASES DU LOGICIEL

a) Les concepts de base, l'organisation des données (chapitre 1).
b) Importation-exportation et production de données (chapitre 2).
c) Manipulation de données (chapitre 3).
d) R et sa documentation (chapitre 4).
e) Techniques pour tracer des courbes et des graphiques (chapitre 5).
f) Maintenance des sessions (chapitre 7).

PARTIE II : STATISTIQUES ÉLÉMENTAIRES

a) Variables aléatoires, lois et simulations (chapitre 10).
b) Statistique descriptive (chapitre 9).
c) Intervalles de confiance et tests d'hypothèses (chapitre 11).
d) Régression linéaire simple et multiple (chapitre 12).
e) Analyse de variance élémentaire (chapitre 13).

PARTIE III : CONCEPTS AVANCÉS

a) Mathématiques de base : calcul matriciel, intégration, optimisation (chapitre 8).
b) Programmation en R (chapitre 6).

Mises en relief

Nous avons souhaité soigner le mode de présentation de l'ouvrage (la forme) pour que l'information (le contenu) soit digeste. Par conséquent, des encadrés qui permettent la mise en relief de certaines informations importantes afin de faciliter la compréhension des notions abordées sont disposés à plusieurs endroits stratégiques du livre. Ces encadrés se distinguent par des icônes apparaissant dans la marge.

Astuce

Information supplémentaire relative au sujet traité.

Attention

Souligne un point important à ne pas négliger.

Remarque

Propose conseils et trucs pratiques.

Renvoi

Fait référence à un autre chapitre ou à un site internet.

Expert

Éléments avancés dont la lecture peut être omise en premier lieu.

Linux

Information réservée aux utilisateurs Linux.

Mac

Information réservée aux utilisateurs Macintosh.

Solutions des exercices et des travaux pratiques

Les corrigés des exercices et des séances de travaux pratiques sont fournis sur le site internet associé au livre (`http://www.biostatisticien.eu/springeR`).

Par ailleurs, quelques projets plus ambitieux que les travaux pratiques seront rendus accessibles sur ce site.

Conventions de police

- La lettre R désigne le logiciel R.
- Nous utiliserons l'écriture *italique* pour désigner des termes empruntés à la langue anglaise comme *data.frame* ou *package* ou bien des termes latins comme *versus* ou *a priori*.
- Nous utiliserons une police de caractères à `chasse fixe` (environnement `Verbatim`) pour noter des instructions R.
- Nous utiliserons une police de caractères en Petites capitales pour désigner un jeu de données et une police avec des caractères sans empattement pour désigner le nom du fichier physique contenant ce jeu de données. Cette dernière police de caractères sera utilisée pour indiquer n'importe quel fichier ou dossier mentionné dans cet ouvrage.

Sommaire

Liste des figures

Liste des tableaux

Notations mathématiques

$:=$	Symbole indiquant des notations différentes pour un même objet
$\cup$	Fusion de tables
$a \in A$	a appartient à l'ensemble A
$A \subset B$	A inclus dans B
$A \supset B$	A contient B
$A \cap B$	Intersection des ensembles A et B
$A \cup B$	Réunion des ensembles A et B
$A \backslash B$	Complémentaire de l'ensemble B dans l'ensemble A
$(A \cup B) \backslash (A \cap B)$	Différence symétrique des ensembles A et B
f_i	Fréquence d'une modalité
$\lvert x \rvert$	Valeur absolue du nombre x
$x!$	Factorielle du nombre x
$\binom{n}{p}$	Nombre de combinaisons de p éléments pris parmi n, coefficients du binôme
$\Gamma(\cdot)$	Fonction gamma
γ	Constante d'Euler
$\psi(\cdot)$	Fonction digamma
π	Nombre π
λ	Nombre scalaire
$\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{C}}$, etc.	Matrices
$\boldsymbol{\mathcal{I}}$	Matrice identité
$n \times p$	Pour indiquer la taille d'une matrice
$\boldsymbol{\mathcal{A}}^\mathsf{T}$	Transposée de la matrice $\boldsymbol{\mathcal{A}}$
$\boldsymbol{\mathcal{B}}^{-1}$	Inverse de la matrice $\boldsymbol{\mathcal{B}}$
$\overline{\boldsymbol{\mathcal{C}}}$	Conjuguée de la matrice complexe $\boldsymbol{\mathcal{C}}$
$\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$	Vecteur d'éléments en colonne
$\boldsymbol{x}^\mathsf{T}$	Transposée du vecteur $\boldsymbol{x}$
$\boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}}$	Produit de Kronecker de la matrice $\boldsymbol{\mathcal{A}}$ par la matrice $\boldsymbol{\mathcal{B}}$
$vec(\boldsymbol{\mathcal{A}})$	Vecteur de l'empilement des colonnes de la matrice $\boldsymbol{\mathcal{A}}$
$vech(\boldsymbol{\mathcal{A}})$	Vecteur de l'empilement des colonnes de la matrice $\boldsymbol{\mathcal{A}}$, mais en excluant les éléments au-dessus de la diagonale
$\boldsymbol{\mathcal{M}}^*$	Matrice adjointe (transposée conjuguée) de la matrice $\boldsymbol{\mathcal{M}}$
$\star$	Produit usuel

$\boldsymbol{\mathcal{M}}^{1/2}$	Racine carrée de la matrice $\boldsymbol{\mathcal{M}}$
$\mathbf{1}_{[A]}(x)$	Vaut 1 si $x \in A$ et 0 sinon
$[a, b]$	Intervalle des valeurs comprises entre a et b
$\det(\boldsymbol{\mathcal{A}})$	Déterminant de la matrice $\boldsymbol{\mathcal{A}}$
$\Phi(\cdot)$	Fonction de répartition d'une variable aléatoire de loi normale standard $\mathcal{N}(0, 1)$
$\dot{\boldsymbol{\mathcal{X}}}$	Matrice obtenue en centrant les colonnes de la matrice $\boldsymbol{\mathcal{X}}$
$\mathbf{1}_n$	Vecteur $(1, \ldots, 1)^\mathsf{T}$ de longueur n
X, Y	Variables non aléatoires (statistique descriptive)
N	Taille de la population
n	Taille échantillonnale
$m_e := q_{1/2}$	Médiane
$PFC_X(\cdot)$	Valeur du polygone des fréquences cumulées de X
μ_X, μ	Espérance de la variable aléatoire X ou moyenne de la population en statistique descriptive
q_p ou x_p	Fractile (quantile) d'ordre p d'une variable
$q_{1/4}, q_{3/4}$	Premier et troisième quartile (aussi notés q_1 et q_3)
$\sigma^2_{Pop}(\boldsymbol{x})$	Variance de la population (statistique descriptive)
$\sigma_{Pop}(\boldsymbol{x})$	Écart type de la population (statistique descriptive)
c_v	Coefficient de variation de la population (statistique descriptive)
γ_1	Coefficient d'asymétrie (*skewness*)
β_2	Coefficient d'aplatissement (*kurtosis*)
μ_3	Moment centré d'ordre 3
μ_4	Moment centré d'ordre 4
χ^2	Statistique du χ^2 de Pearson
Φ^2, V^2	Φ^2 et V^2 de Cramér
τ, τ_b	τ et τ_b de Kendall
ρ	Coefficient de corrélation théorique de Pearson
$\eta^2_{Y\|X}$	Rapport de corrélation
X, Y, ϵ	Variables aléatoires
x_i, y_i, ϵ_i	Réalisations des variables aléatoires X, Y, ϵ
$\mathbf{X}$, $\mathbf{Y}$, $\boldsymbol{\epsilon}$	Vecteurs aléatoires
$\mathbf{X}_n$	Échantillon (aléatoire)
$\boldsymbol{x}_n$	Échantillon (observé)
$\mathbf{X}$	Matrice aléatoire
$\mathcal{L}$	Loi (générique) d'une variable aléatoire
$\mathcal{N}(0, 1)$	Loi gaussienne standard
$\mathcal{N}(\mu, \sigma^2)$	Loi gaussienne (normale) d'espérance μ et de variance σ^2
$\mathcal{U}(a, b)$	Loi uniforme sur l'intervalle $[a, b]$
$\mathcal{B}in(n, p)$	Loi binomiale de paramètres n et p

$\mathcal{E}(\lambda)$	Loi exponentielle de paramètre λ
$\mathcal{P}(\lambda)$	Loi de Poisson de paramètre λ
$\mathcal{T}(n)$	Loi de Student à n degrés de liberté
$\chi^2(n)$ ou χ^2_n	Loi du χ^2 à n degrés de liberté
$\mathcal{F}(n,m)$	Loi de Fisher à n et m degrés de liberté
$f_{\mathrm{X}}(\cdot)$	Fonction de densité de la variable aléatoire X
$F_{\mathrm{X}}(\cdot)$	Fonction de répartition de la variable aléatoire X
$F_{\mathrm{X}}^{-1}(\cdot)$	Fonction de répartition réciproque de la variable aléatoire X
σ^2	Variance d'une variable aléatoire
$\mathbb{E}(\mathrm{Y})$	Espérance théorique de la variable aléatoire Y
$\mathbb{V}\mathrm{ar}(\mathrm{Y})$	Variance théorique de la variable aléatoire Y
$\overline{\mathrm{X}}_n$	Moyenne empirique $\frac{1}{n}\sum_{i=1}^{n}\mathrm{X}_i$ de l'échantillon $\mathbf{X}_n = (\mathrm{X}_1,\ldots,\mathrm{X}_n)^\intercal$, estimateur de μ_{X}
$\overline{x}_n$	Réalisation de la moyenne empirique $\frac{1}{n}\sum_{i=1}^{n}\mathrm{X}_i$ de l'échantillon $\mathbf{X}_n = (\mathrm{X}_1,\ldots,\mathrm{X}_n)^\intercal$, estimation de μ_{X}
$\xrightarrow{P}$	Symbole de convergence en probabilité
$\hat{\mathrm{F}}_n(\cdot) := \hat{\mathrm{F}}_{\mathbf{X}_n}(\cdot)$	Fonction de répartition empirique de l'échantillon $\mathbf{X}_n$
θ	Paramètre inconnu (parfois on notera $\theta^\bullet$ la vraie valeur inconnue du paramètre)
$\hat{\theta}(\mathrm{X}_1,\ldots,\mathrm{X}_n)$ ou $\hat{\theta}$	Estimateur du paramètre inconnu θ basé sur l'échantillon $\mathbf{X}_n = (\mathrm{X}_1,\ldots,\mathrm{X}_n)^\intercal$
$\hat{\theta}(x_1,\ldots,x_n)$ ou $\hat{\theta}$	Estimation du paramètre inconnu θ basé sur l'échantillon observé $\boldsymbol{x}_n = (x_1,\ldots,x_n)^\intercal$
$\mathbb{B}(\hat{\theta}(\mathrm{X}_1,\ldots,\mathrm{X}_n);\theta)$	Biais de l'estimateur $\hat{\theta}(\mathrm{X}_1,\ldots,\mathrm{X}_n)$ pour estimer le paramètre inconnu θ
$P[A]$	Probabilité de l'ensemble A
$\mathcal{V}(\theta;\mathrm{X}_1,\ldots,\mathrm{X}_n)$	Fonction de vraisemblance de l'échantillon $\mathbf{X}_n$ évaluée en θ
$\boldsymbol{x}^* = (x_1^*,\ldots,x_n^*)^\intercal$	Échantillon *bootstrap* généré à partir de l'échantillon observé $\boldsymbol{x}_n = (x_1,\ldots,x_n)^\intercal$
$\hat{\sigma}$	Estimateur de σ
$\hat{\sigma}$	Estimation de σ
p	Proportion théorique
$\hat{\mathrm{p}}$	Estimateur d'une proportion (ou d'une probabilité)
$\hat{p}$	Estimation d'une proportion (ou d'une probabilité)
$\widehat{\mathrm{m}_e}$	Estimateur d'une médiane
$\widehat{m_e}$	Estimation d'une médiane
M	Nombre de boucles (d'échantillons générés) dans une simulation de Monte-Carlo
B	Nombre d'échantillons *bootstrap* générés
$B(\cdot,\cdot)$	Fonction bêta

$I'_x(\cdot,\cdot)$	Dérivée de la fonction bêta incomplète
$I(\cdot)$	Fonction de Bessel modifiée
$I_\alpha(\cdot)$	Fonctions de Bessel modifiées
u_p	Quantile d'ordre p d'une $\mathcal{N}(0,1)$
t_p^n	Quantile d'ordre p d'une $\mathcal{T}(n)$
q_p^n	Quantile d'ordre p d'une $\chi^2(n)$
$f_p^{n,m}$	Quantile d'ordre p d'une $\mathcal{F}(n,m)$
$\mathrm{IC}_{1-\alpha}(\theta)$	Intervalle de confiance (aléatoire) de niveau de confiance $1-\alpha$ pour θ
$ic_{1-\alpha}(\theta)$	Intervalle de confiance (réalisé) de niveau de confiance $1-\alpha$ pour θ
$1-\alpha$	Niveau de confiance d'un intervalle de confiance
$(x_{(1)},\ldots,x_{(n)})$	Échantillon (observé) ordonné par valeurs croissantes
$\mathcal{H}_1$	Assertion d'intérêt dans les tests d'hypothèses
$\mathcal{H}_0$	Hypothèse dite nulle, contraire de $\mathcal{H}_1$
α	Niveau de signification ou risque de première espèce dans les tests d'hypothèses
R	Coefficient de corrélation empirique aléatoire de Pearson
r	Coefficient de corrélation empirique réalisé de Pearson
β_0, β_1	Coefficients inconnus d'un modèle de régression linéaire simple
$\hat{\beta}_0$, $\hat{\beta}_1$	Estimations des coefficients inconnus d'un modèle de régression linéaire simple
$\hat{\epsilon}_i$	Résidus observés d'un modèle de régression linéaire simple
$\hat{y}_i$	Valeurs ajustées observées d'un modèle de régression linéaire simple
R^2	Coefficient de détermination aléatoire en régression
r^2	Coefficient de détermination réalisé en régression
R_a^2	Coefficient de détermination ajusté aléatoire en régression
r_a^2	Coefficient de détermination ajusté réalisé en régression
$\hat{\mathrm{Y}}^p$	Préviseur de la variable aléatoire Y pour une nouvelle valeur de la variable explicative X en régression
$\mathrm{IP}_{1-\alpha}(\mathrm{Y}_0, x_0)$	Intervalle de prévision de niveau $1-\alpha$ pour la variable aléatoire Y_0 associée à une nouvelle valeur x_0 de la variable explicative
$\boldsymbol{\beta} = (\beta_0,\ldots,\beta_p)^\mathsf{T}$	Vecteur des $p+1$ coefficients inconnus d'un modèle de régression linéaire multiple à p variables explicatives

$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\mathbf{Y}$	Estimateur du vecteur des paramètres inconnus $\boldsymbol{\beta}$ pour la matrice observée des variables explicatives $\boldsymbol{X}$ et pour le vecteur des valeurs à expliquer dans un modèle de régression linéaire multiple
$\hat{\boldsymbol{\beta}}$	Estimation de $\boldsymbol{\beta}$
VIF	Facteur d'inflation de la variance, *variance inflation factor*
AIC	*an information criterion*
BIC	*bayesian information criterion*
h_{ii}	Levier de la i-ème observation en régression
t_i	Résidus standardisés
t_i^*	Résidus studentisés
$\hat{\sigma}_{(-i)}$	Estimation de σ sans utiliser la i-ème observation
C_i	Distances de Cook
$\hat{y}_j^{(-i)}$	Prédiction de y_j sans utiliser la i-ème observation
$\hat{\beta}_j^{(-i)}$	Estimation de β_j sans utiliser la i-ème observation
I, J	Nombre de niveaux d'un facteur en ANOVA
$\mu_{\bullet\bullet}$	Effet moyen général en ANOVA
$\mu_{i\bullet}$	Effet du niveau i d'un facteur en ANOVA
$\mu_{\bullet j}$	Effet du niveau j d'un facteur en ANOVA

Chapitre A

Présentation du logiciel R

Pré-requis et objectif

- La lecture du chapitre sur l'installation de R dans les Annexes peut se révéler utile.
- Ce chapitre présente les origines, l'objectif et les spécificités du logiciel R.

SECTION A.1

Présentation du logiciel

A.1.1 Origines

Le logiciel R est un logiciel de statistique créé par Ross Ihaka & Robert Gentleman [28]. Il est à la fois un langage informatique et un environnement de travail : les commandes sont exécutées grâce à des instructions codées dans un langage relativement simple, les résultats sont affichés sous forme de texte et les graphiques sont visualisés directement dans une fenêtre qui leur est propre. C'est un clone du logiciel S-plus qui est fondé sur le langage de programmation orienté objet S, développé par AT&T Bell Laboratories en 1988 [6]. Ce logiciel sert à manipuler des données, à tracer des graphiques et à faire des analyses statistiques sur ces données.

A.1.2 Pourquoi utiliser R ?

Tout d'abord R est un logiciel **gratuit** et à **code source ouvert** (*open-source*). Il fonctionne sous UNIX (et Linux), Microsoft Windows et Mac OS. C'est donc un logiciel **multi-plates-formes**. Il est développé dans la mouvance

des logiciels libres par une communauté sans cesse plus vaste de bénévoles motivés. Tout le monde peut d'ailleurs contribuer à son amélioration en y intégrant de nouvelles fonctionnalités ou méthodes d'analyse non encore implémentées. Cela en fait donc un logiciel en rapide et constante évolution.

C'est aussi un outil très puissant et très complet, particulièrement bien adapté pour la mise en œuvre informatique de **méthodes statistiques**. Il est plus difficile d'accès que certains autres logiciels du marché (comme SPSS ou Minitab par exemple), car il n'est pas conçu pour être utilisé à l'aide de « clics » de souris dans des menus. L'avantage en est toutefois double :

- l'approche est **pédagogique** puisqu'il faut maîtriser les méthodes statistiques pour parvenir à les mettre en œuvre ;
- l'outil est très **efficace** lorsque l'on domine le langage R puisque l'on devient alors capable de créer ses propres outils, ce qui permet ainsi d'opérer des analyses très sophistiquées sur les données.

Attention

R est plus difficile d'accès que certains autres logiciels du marché. Il faut passer du temps à en apprendre la syntaxe et les commandes.

Le logiciel R est particulièrement performant pour la manipulation de données, le calcul et l'affichage de graphiques. Il possède, entre autres choses :

- un système de documentation intégré très bien conçu (en anglais) ;
- des procédures efficaces de traitement des données et des capacités de stockage de ces données ;
- une suite d'opérateurs pour des calculs sur des tableaux et en particulier sur des matrices ;
- une vaste et cohérente collection de procédures statistiques pour l'analyse de données ;
- des capacités graphiques évoluées ;
- un langage de programmation simple et efficace intégrant les conditions, les boucles, la récursivité, et des possibilités d'entrée-sortie.

Remarque

Pour les lecteurs déjà habitués à SAS, SPSS ou Stata, nous conseillons, en parallèle du présent livre, de consulter les ouvrages (en anglais) [39] et [40], ainsi que les deux sites internet suivants :

- `http://rforsasandspssusers.com`
- `http://www.statmethods.net`

Notons également qu'il est possible d'appeler des fonctions de Matlab directement depuis R au moyen du *package* `R.matlab` et des fonctions R

depuis Excel au moyen du *package* `RExcelInstaller`. La lecture de [27] (en anglais) pourra s'avérer utile dans ce contexte. Enfin, un équivalent pour OpenOffice, sous la forme d'une extension nommée `ROOo` existe ; voir le site internet `http://rcom.univie.ac.at`.

Renvoi

Les enseignants du secondaire pourront trouver quelques exemples d'utilisation de R dans certains articles de la revue collaboratrice en ligne MathémaTICE. Ainsi on peut par exemple y trouver des exemples mettant en œuvre certains outils du programme de probabilité, de statistique et d'analyse du lycée :

- « Quelques activités avec R, logiciel professionnel polyvalent de statistique. Illustrations en analyse, probabilité et statistique(s) au lycée » : `http://revue.sesamath.net/spip.php?article501`
- « Regards croisés sur l'algorithmique et la programmation » : `http://revue.sesamath.net/spip.php?article552`

Nous mentionnons également « Le logiciel R comme outil d'initiation à la statistique descriptive [au lycée] : enquête sur les dépenses des ménages », paru dans la revue de la Société Française de Statistique : `http://publications-sfds.math.cnrs.fr/index.php/StatEns/article/view/95`

SECTION A.2

R et les statistiques

R est un logiciel dans lequel de nombreuses techniques statistiques modernes et classiques ont été implémentées. Les méthodes les plus courantes permettant de réaliser une analyse statistique telles que :

- statistique descriptive ;
- tests d'hypothèses ;
- analyse de la variance ;
- méthodes de régression linéaire (simple et multiple) ;
- etc.

sont enchâssées directement dans le cœur du système. Notez également que la plupart des méthodes avancées de statistique sont aussi disponibles au travers de modules externes appelés *packages*. Ceux-ci sont faciles à installer directement à partir d'un menu du logiciel. Ils sont tous regroupés sur le site internet du *Comprehensive R Archive Network* (CRAN) (`http://cran.r-project.org`) sur lequel vous pouvez les consulter. Ce site fournit aussi, pour certains grands domaines d'étude, une liste commentée des *packages* associés à ces thèmes (appelée *Task View*), ce qui facilite ainsi la recherche d'une méthode

statistique particulière. Par ailleurs, une documentation détaillée en anglais de chaque *package* est disponible sur le CRAN.

Il est par ailleurs utile de noter que les méthodes statistiques les plus récentes y sont régulièrement ajoutées par la communauté statistique elle-même.

Renvoi

Le lecteur pourra consulter avec profit la section C.2, page 574, détaillant les procédures à mettre en œuvre afin d'installer de nouveaux *packages*.

SECTION A.3

R et les graphiques

Une des grandes forces de R réside dans ses capacités, bien supérieures à celles des autres logiciels courants du marché, à combiner un langage de programmation avec la possibilité de réaliser des graphiques de qualité. Les graphiques usuels s'obtiennent aisément au moyen de fonctions prédéfinies. Ces dernières possèdent de très nombreux paramètres permettant par exemple d'ajouter des titres, des légendes, des couleurs, etc. Mais il est également possible d'effectuer des graphiques plus sophistiqués permettant de représenter des données complexes telles que des courbes de surface ou de niveau, des volumes affichés avec un effet 3D, des courbes de densité, et bien d'autres choses encore. Il vous est également possible d'y ajouter des formules mathématiques. Vous pouvez aussi agencer ou superposer plusieurs graphiques sur une même fenêtre, et utiliser de nombreuses palettes de couleur.

Vous pouvez obtenir une démonstration des possibilités graphiques de R en tapant successivement les commandes suivantes :

```
demo(image)
example(contour)
demo(graphics)
demo(persp)
demo(plotmath)
demo(Hershey)
require("lattice") # Charge le package que vous devez avoir
                   # préalablement installé en passant par le
                   # menu Packages/Installer le(s) package(s).
demo(lattice)
example(wireframe)
require("rgl")     # Même remarque que ci-dessus.
demo(rgl)          # Possibilité d'interaction avec la souris.
example(persp3d)
```

La figure ci-dessous présente quelques-uns de ces graphiques.

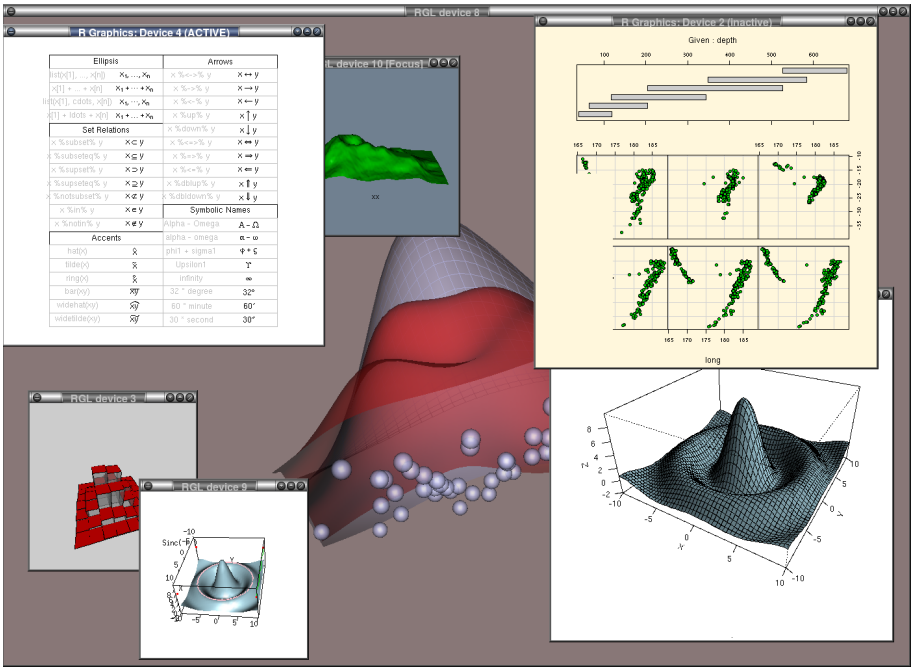


FIGURE A.1 – Quelques possibilités graphiques offertes par R.

SECTION A.4

L'interface graphique de R (GUI)

L'interface graphique de R (c'est-à-dire l'ensemble de ses menus) est très limitée, voire inexistante sur certaines plates-formes, en comparaison des autres logiciels standards. Cette minimalité peut dérouter les novices. Toutefois, nous pouvons nuancer cet inconvénient au travers des points suivants :

- cela offre l'avantage pédagogique d'inciter l'utilisateur à bien maîtriser la procédure statistique qu'il compte appliquer ;
- il existe des outils additionnels qui permettent d'étendre l'interface graphique.

Nous présentons dans la prochaine section le *package* `Rcmdr`, à installer via le menu `Packages`, qui permet d'effectuer des analyses graphiques et statistiques usuelles au moyen d'une interface plus conviviale comprenant des menus déroulants. Par ailleurs, les instructions R permettant de réaliser l'analyse choisie dans les menus de `RCommander` s'affichent dans une portion de fenêtre dédiée. Cela peut par exemple être utile si l'on ne connaît pas (ou que l'on a oublié) une instruction R nécessaire à la réalisation d'une tâche particulière.

Astuce

Notez qu'il est possible, lorsque vous maîtriserez le R, de développer vous-même des outils similaires à `Rcmdr`, à destination d'un utilisateur final qui ne souhaite pas apprendre le logiciel R mais seulement utiliser, de la façon la plus ergonomique possible, une procédure que vous aurez élaborée pour lui. Vous pourriez pour cela faire appel au *package* `tcltk`.

Attention

Notons qu'en utilisant `RCommander`, nous nous éloignons de ce qui fait la force et la flexibilité de R. Nous déconseillons donc l'emploi d'un tel outil si vous désirez devenir un utilisateur avancé.

SECTION A.5

Mes premiers pas en R

A.5.1 Utilisation de `RCommander`

Nous offrons dans cette section un bref aperçu de l'utilisation du *package* `Rcmdr`. Nous présentons ensuite quelques fonctionnalités offertes par cette interface pour effectuer quelques manipulations statistiques. Nous finissons en indiquant comment rajouter des fonctionnalités à l'interface de `RCommander`.

A.5.1.1 Lancement de `RCommander`

Veuillez suivre les étapes suivantes afin de lancer `RCommander`.

- Double-cliquez sur l'icône R présente sur votre bureau.
- Dans la console, tapez `install.packages("Rcmdr")`. Choisissez un miroir proche de vous.
- Dans la console, tapez `require("Rcmdr")`. Répondre `Oui` à toute question posée. L'interface graphique de `RCommander` s'ouvre alors. Une autre solution consiste à cliquer sur le menu `Packages`, puis *Chargez le package...*, puis à choisir `Rcmdr`.
- Dans la portion de fenêtre `Messages`, vous devriez voir apparaître `AVIS : La version Windows de R Commander fonctionne mieux sous RGui avec l'interface de document unique (SDI)`.
- Pour remédier à ce problème, fermez `RCommander`.
- Dans RGui, allez dans `Édition`, puis *Préférences*. Cochez *SDI* puis cliquez sur *Save...* et puis sur `Enregistrer`. Vous pouvez aussi en profiter pour personnaliser R.
- Fermez R et sauvez une image de la session.
- Relancez R, puis `RCommander` en tapant `require("Rcmdr")` dans la console de R.

Renvoi

Le lecteur pourra consulter avec profit la section C.2 détaillant les procédures à mettre en œuvre afin d'installer le *package* `Rcmdr`.

Mac

Les utilisateurs du système d'exploitation MacOS, pourront consulter avec profit les instructions disponibles à l'adresse `http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html`, après avoir installé le *package* `tcltk` disponible sur le CRAN.

L'interface graphique de `RCommander` se décompose en quatre parties comme cela est illustré sur la figure A.2 :

A. Des menus déroulants permettant d'effectuer des tâches spécifiques.
B. Une `Fenêtre de script` présentant le code lancé via un clic-souris dans l'un des menus déroulants.
C. Une `Fenêtre de Sortie` présentant les sorties correspondant au code exécuté.
D. Une fenêtre `Messages` pésentant un Avis sur la derniere tâche effectuée.

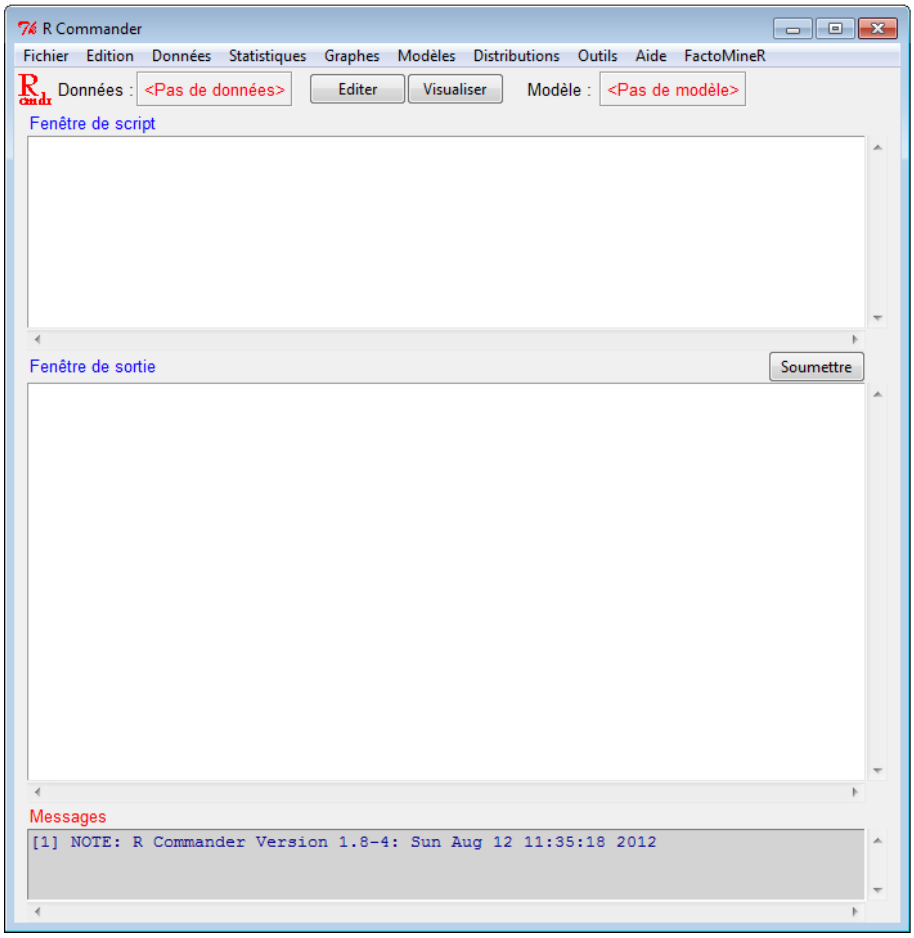


FIGURE A.2 – L'interface graphique de `RCommander`.

A.5.1.2 Manipulation de données avec `RCommander`

Avant de faire des statistiques, il vous faut des données.

- **Entrer des données à la main**

Veuillez suivre les étapes suivantes afin d'entrer des données à la main.

- Dans le menu `Données`, choisissez *Nouveau jeu de données...*
- Dans la fenêtre `Nouveau tableau de données`, choisissez un nom pour votre jeu de données, par exemple `Data1`.

- Un éditeur de données apparaît. Cliquez sur `var1` et remplacez par `Nom`. Entrez quelques noms pour cette variable : Pierre, Rémy, Benoit (voir figure A.3).
- Créez une variable `Taille` de type `numeric` avec les valeurs suivantes : 182, 184, 190.
- Cliquez sur la croix en haut à droite de la fenêtre active afin de fermer l'éditeur de données.
- Vous pouvez visualiser votre jeu données en cliquant sur `Visualiser`

Il nous est maintenant possible de calculer quelques statistiques élémentaires.

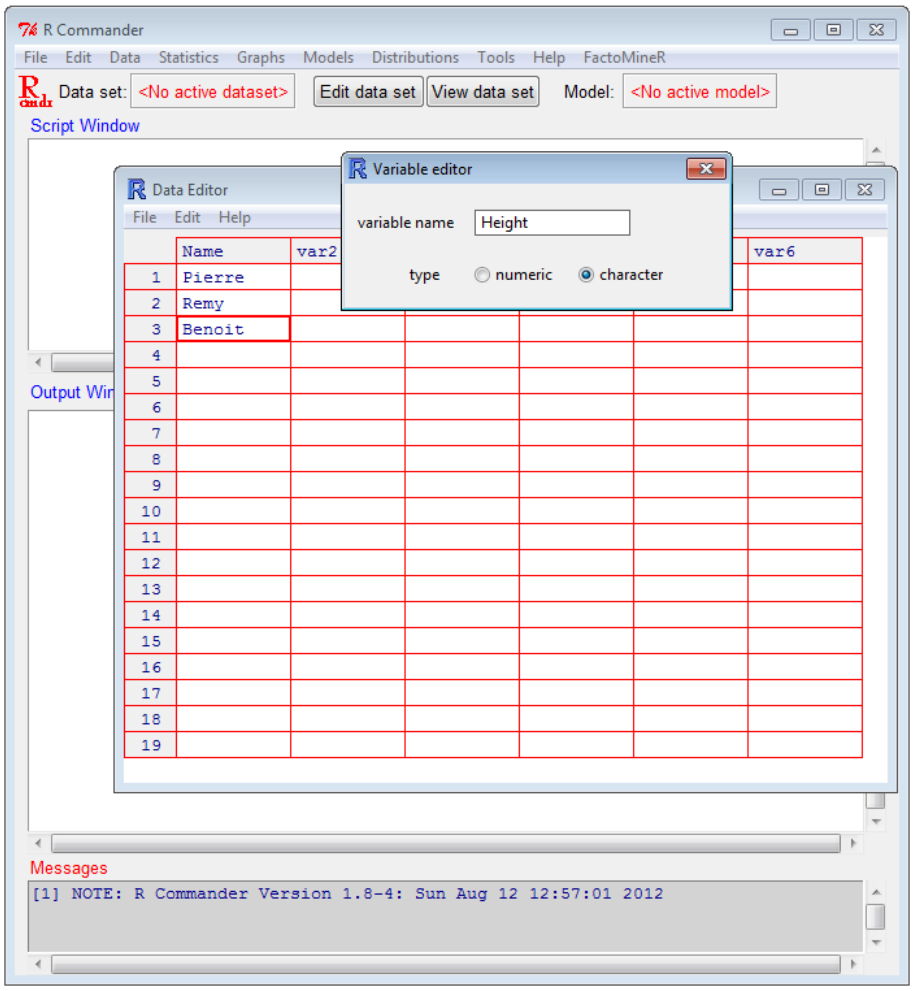


FIGURE A.3 – Entrer des données via l'interface graphique de `RCommander`.

• **Statistiques élémentaires**

Veuillez suivre les étapes suivantes afin d'obtenir quelques résumés statistiques sur votre jeu de données.

- ▶ Dans le menu `Statistiques`, choisissez l'entrée *Résumés*, puis *Statistiques Descriptives....*
- ▶ Une fenêtre nommée `Statistiques générales` apparaît ; la seule variable de type `numeric` disponible dans votre jeu de données est la variable `Taille`.
- ▶ Choisissez comme statistiques `Moyenne`, `Ecart-type` et `Quantiles`, et cliquez sur `OK`.
- ▶ Le résultat apparaît dans la `Fenêtre de sortie`. Notez que vous pouvez consulter l'instruction R ayant permis d'effectuer cette tâche dans la `Fenêtre de script` (voir la figure A.4).

Notez qu'il est également possible d'écrire directement une instruction dans la `Fenêtre de script` sans passer par un menu. Voyons ceci sur un exemple.

- ▶ Tapez dans la `Fenêtre de script` :

```
numSummary(Data1[,"Taille"], statistics=c("mean","sd"))
```

- ▶ Cliquez avec votre souris sur cette ligne de commande afin de voir apparaître le curseur sur cette ligne, puis cliquez sur `Soumettre`.
- ▶ Vous venez de calculer la moyenne et l'écart-type de la variable `Taille` contenant 3 observations. Le résultat apparaît dans la `Fenêtre de sortie` :

```
mean       sd % n
 184 5.291503 0 3
```

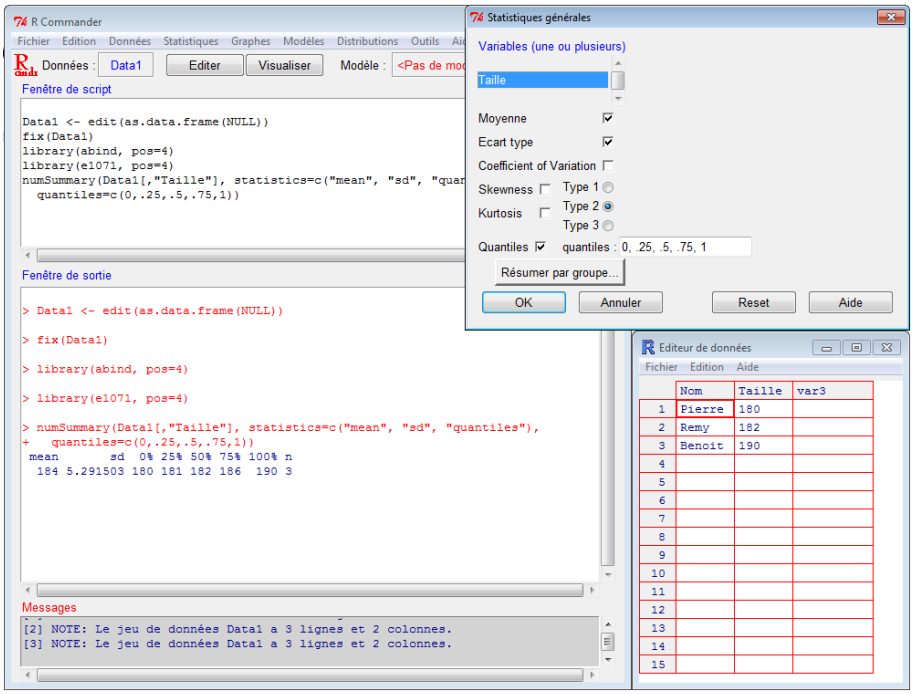


FIGURE A.4 – Statistiques élémentaires avec `RCommander`.

- **Manipulation du jeu de données**

Supposons que dans notre exemple jouet, nous ayons recueilli le poids comme information supplémentaire et que nous désirions calculer les indices de masse corporelle : $IMC = Poids/Taille^2$ (taille en mètres).

- ▶ Cliquez sur `Éditer` (en dessous des menus de `RCommander`).
- ▶ L'éditeur de données s'ouvre et vous pouvez ajouter la variable numérique `Poids`, avec les entrées suivantes : 70, 72 et 75. Fermez ensuite l'éditeur de données.
- ▶ Dans le menu `Données`, chosisissez *Gérer les variables dans le jeu de données actif*, puis `Calculer une nouvelle variable...`. Une fenêtre apparaît.
- ▶ Dans la partie `Nom de la nouvelle variable`, veuillez indiquer `IMC` et dans la partie `Expression à calculer` : `Poids/((Taille/100)**2)` (voir la figure A.5). Cliquez sur `OK` pour terminer le calcul.
- ▶ Cliquez sur `Visualiser` afin de voir apparaître le résultat pour votre jeu de données.

Vous commencez à être fatigué et vous avez sûrement besoin d'une petite pause café ! Mais avant cela, veuillez effectuer les étapes suivantes afin de sauvegarder votre jeu de données.

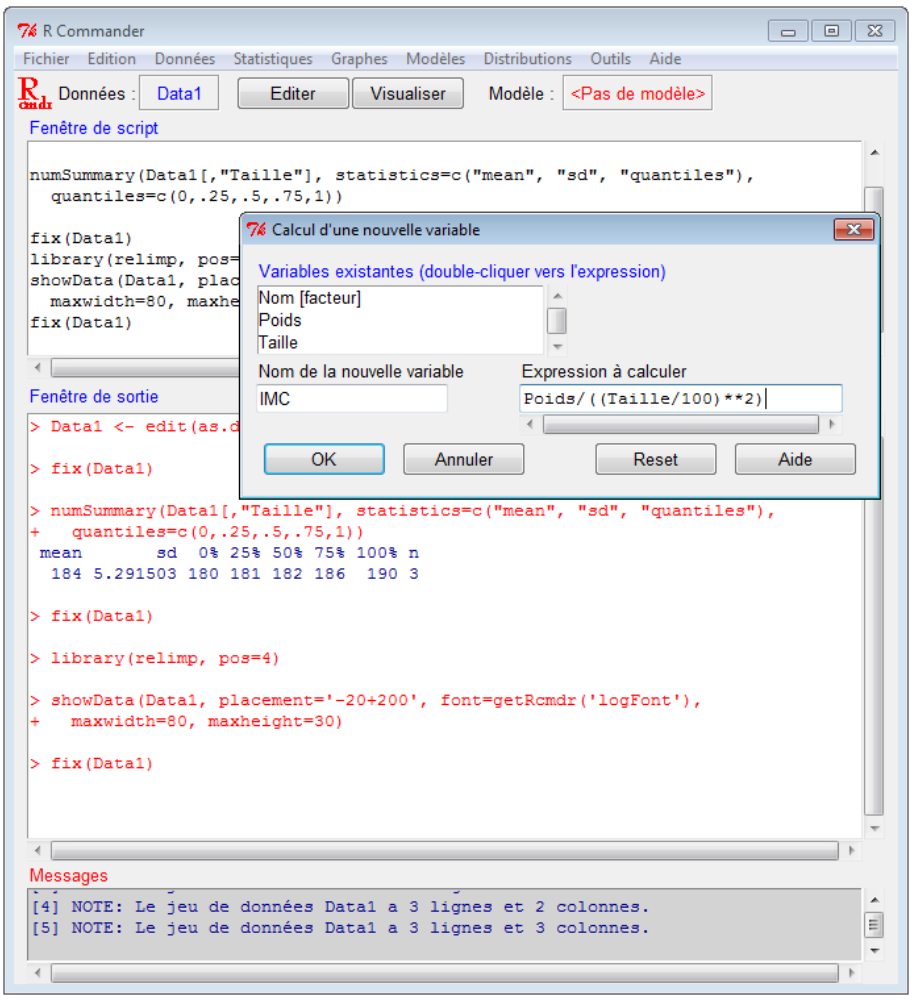


FIGURE A.5 – Manipulation d'un jeu de données avec `RCommander`.

- Dans le menu `Données`, choisissez `Jeu de données actif`, puis `Sauver le jeu de données actif...`.
- Une fenêtre s'ouvre. Vous pouvez choisir un emplacement où enregistrer

votre jeu de données. Nous le nommerons `IMC` et il aura par défaut l'extension `.RData`.
- Fermez `RCommander` en cliquant sur la croix en haut à droite et répondez `OK` à la question `Voulez-vous sortir?`, `Non` à la question `Sauver le fichier script?` et aussi `Non` à la question `Sauver le fichier de sortie?`.
- Vous pouvez maintenant fermer le logiciel R en cliquant sur la croix en haut à droite, et répondre `Non` à la question `Sauver une image de la session?`.

De retour après une pause bien méritée, vous désirez rajouter d'autres données dans votre fichier `IMC.RData`.
- Ouvrez une session R. Tapez `library(Rcmdr)`.
- Dans le menu `Données`, choisissez `Charger un jeu de données...`
- Une fenêtre s'ouvre. Explorez l'arborescence afin de trouver et d'ouvrir le fichier `IMC.RData`.
- Cliquez sur `Visualiser` afin d'afficher votre jeu de données.
- Rajoutez les informations d'une personne supplémentaire (`"Dominique"`, `Taille`=160, `Poids`=49) en cliquant sur le bouton `Éditer`.
- Après avoir fermé l'éditeur, vous pouvez visualiser les changements en cliquant sur `Visualiser`. Vous voyez alors apparaître la valeur `NA` (*Non Available*, i.e. non disponible) pour la valeur de l'IMC de Dominique.
- Si vous voulez obtenir l'IMC de Dominique, il vous faut répéter les étapes présentées à la partie *Manipulation du jeu de données*. Nous verrons dans la suite du livre comment créer une fonction qui calcule l'IMC de façon plus conviviale.

Vous voulez maintenant transmettre votre jeu de données à un collègue qui n'utilise pas encore le logiciel R.

- Dans le menu `Données`, choisissez `Jeu de données actif`, puis `Exporter le jeu de données actif...`
- Une première fenêtre s'ouvre. Décochez l'entrée `Écrire le nom des individus (lignes)` puisque nous ne les avons pas définis. Choisissez `Espaces` pour les séparateurs de champs.
- Après avoir cliqué sur `OK`, une deuxième fenêtre s'ouvre. Vous pouvez choisir un emplacement où enregistrer votre jeu de données. Nous le nommerons IMC et il aura par défaut l'extension `.txt`.
- Vous pouvez maintenant envoyer votre jeu de données `IMC.txt` à votre collègue en lui indiquant au passage l'existence du merveilleux logiciel R, doté d'une interface plutôt conviviale pour manipuler des jeux de données.

A.5.1.3 Quelques manipulations statistiques avec `RCommander`

Nous présentons dans cette partie un bref aperçu de l'utilisation de `RCommander` afin d'effectuer quelques manipulations statistiques. Nous commençons

par un test de comparaison de moyennes et un test du khi-2 d'indépendance. Nous montrons ensuite comment utiliser `RCommander` afin de visualiser les distributions les plus fréquemment rencontrées (binomiale, poisson, Normale, Gamma, etc.). Nous terminons enfin par l'ajustement d'un modèle linéaire.

- **Test de comparaison de moyennes**

Tout d'abord, nous proposons d'utiliser des données déjà disponibles dans le logiciel R. Veuillez suivre les étapes suivantes afin de récupérer un jeu de données.

- Dans le menu `Données`, choisissez *Données dans les packages*, puis *Lire des données depuis un package attaché...*
- Une fenêtre s'ouvre. Double-cliquez sur `datasets` dans la section `Package`, puis double-cliquez sur `sleep` dans la colonne de droite (utilisez l'ascenseur).
- `sleep` apparaît dans `Entrez le nom d'un jeu de données` (voir la figure A.6).
- Vous pouvez cliquer sur `Aide sur le jeu de données sélectionné` afin d'avoir quelques informations sur celui-ci.
- Cliquez sur `OK` pour fermer la fenêtre précédente, puis visualisez le jeu de données en cliquant sur `Visualiser`.

Ces données permettent de comparer l'effet d'un médicament soporifique sur le sommeil par rapport à un groupe contrôle. Nous allons d'abord visualiser la distribution du gain de sommeil dans les deux groupes, puis effectuer une comparaison de moyennes afin de savoir s'il existe une différence significative entre le médicament et le contrôle.

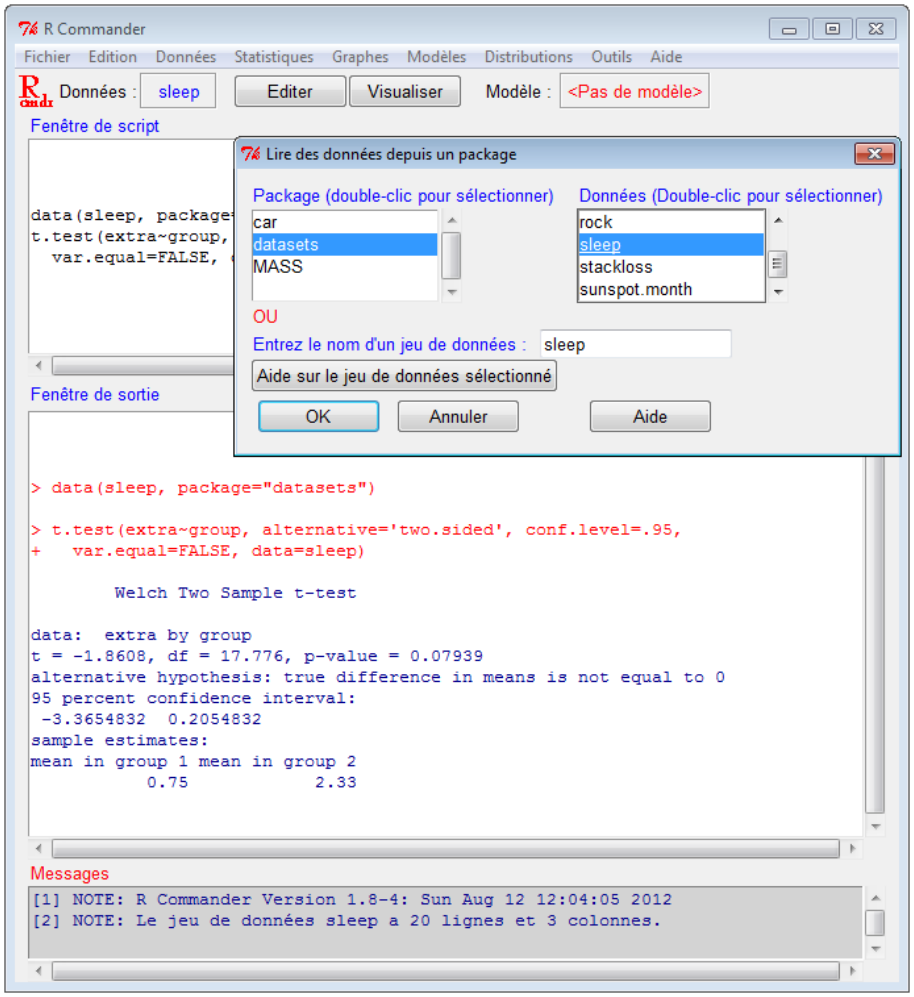


FIGURE A.6 – Test de moyennes avec `RCommander`.

- Dans le menu `Graphes`, choisissez *Boîte de dispersion...*
- Une fenêtre s'ouvre. Cliquez sur `Graphe par groupe...`, puis sur la variable `group`, et enfin sur `OK` deux fois.
- Vous visualisez alors deux boîtes à moustaches représentant la distribution du gain de sommeil dans les deux groupes.
- Il est alors possible de sauver ce graphique en cliquant sur `Fichier`, puis *sauver sous*. Plusieurs formats sont alors proposés pour enregister ce graphique.

Il est aussi possible d'améliorer ce graphique en rajoutant par exemple des couleurs. Tapez dans la fenêtre de script :

```
boxplot(extra~group,ylab="extra",xlab="group",data=sleep,
        col=c("red","blue"))
```

puis cliquez sur `Soumettre`.

Renvoi

Nous reviendrons au chapitre 5 sur les possibilités graphiques du logiciel R.

Effectuons maintenant un test de comparaisons de moyennes.

- Dans le menu `Statistiques`, choisissez *Moyennes*, puis *t-test indépendant...*
- Cliquez sur `group` dans la section `Groupes(un)`. Il est maintenant spécifié la différence `1-2` (groupe 1 *versus* groupe 2).
- Cliquez sur `OK` pour voir apparaître le résultat dans la `Fenêtre de sortie` (voir également la figure A.6).

La valeur-p du test (supérieure à 5 %) ne permet pas de conclure à une différence significative entre le gain moyen de sommeil entre le médicament et le contrôle.

• Test sur une table à double entrée

Dans un essai thérapeutique, la question sous-jacente est de savoir si un traitement sur des mères ayant un statut VIH positif a un effet sur le statut VIH de l'enfant. Si ce n'est pas le cas, alors le statut VIH de l'enfant est indépendant du traitement suivi par la mère. Dans cet essai, sur les 391 enfants considérés, 100 ont un statut VIH négatif, 193 ont une mère ayant reçu le traitement, et 41 ont un statut VIH positif avec une mère ayant reçu le traitement. Afin de savoir si le traitement a un effet, veuillez suivre les étapes suivantes.

- Dans le menu `Statistiques`, choisissez `Tables de contingence`, puis `Remplir et analyser un tableau à double entrée...`
- Une fenêtre s'ouvre. Remplissez le tableau comme indiqué dans la figure A.7. Choisissez `Pourcentages du total` et `Imprimer les fréquences attendues`.
- Cliquez sur `OK` pour voir apparaître le résultat dans la `Fenêtre de sortie`.

Il n'est pas possible de montrer, au risque 5 %, que le traitement a un effet sur le statut VIH de l'enfant.

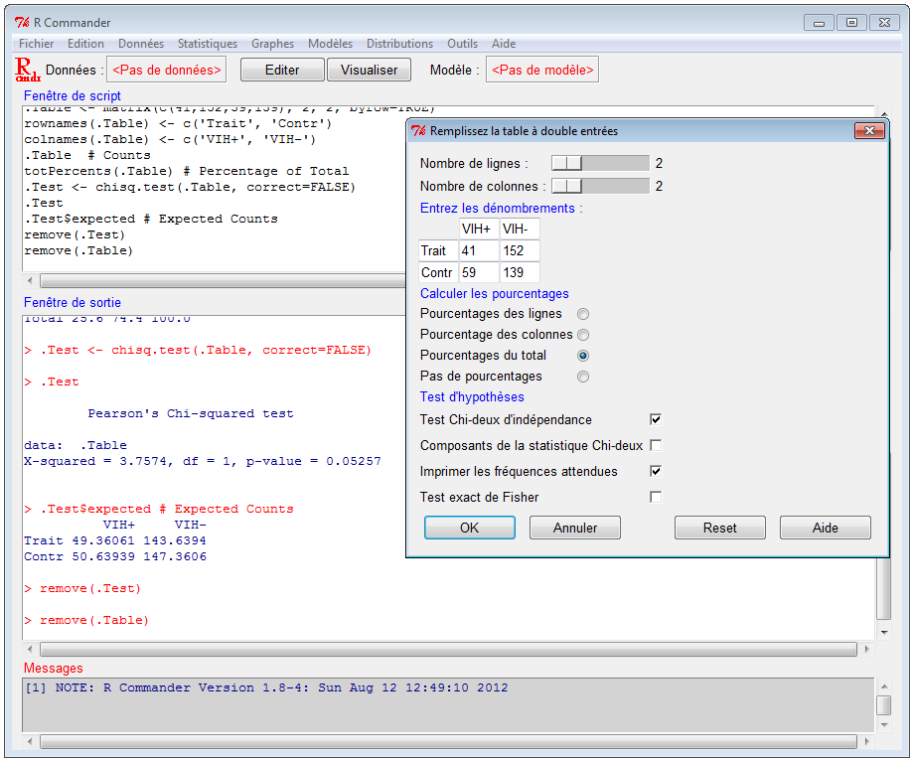


FIGURE A.7 – Test d'indépendance avec `RCommander`.

- **Explorer des distributions**

`RCommander` permet de visualiser facilement les distributions les plus fréquemment rencontrées.

- ▶ Dans le menu `Distributions`, chosissez `Distributions continues`, puis `Distribution Normale`, puis enfin `Graphe de la distribution Normale...`.
- ▶ Une fenêtre s'ouvre. Spécifiez par exemple une moyenne de 4 et un écart-type de 2. Cliquez sur `OK`.
- ▶ Le graphique de la densité d'une loi normale centrée en 4 et d'écart-type 2 apparaît dans une fenêtre graphique.

Vous pouvez effectuer la même procédure pour d'autres lois de probabilité.

• **Ajustement d'un modele linéaire**

`RCommander` permet d'ajuster aisément des modèles de régression classiques. Nous illustrons ici l'utilisation du modèle linéaire. Nous allons tout d'abord importer un jeu de données par le biais d'une adresse internet (URL). Celui-ci contient les mesures, pour 80 patients souffrant d'une maladie invalidante, des variables `SEXE` (1 = Homme, 2 = Femme), `POIDS` (en kg), `TAILLE` (en cm), `SOUFFRANCE` (variable ordinale : a = le moins souffrant), `DISTANCE` (nombre de mètres parcourus), `MOBILITE` (auto-évaluation de sa mobilité ; 1 = le plus mobile), `ESCALIER` (nombre de marches d'escalier gravies).

- ▶ Dans le menu `Données`, choisissez *Importer des données*, puis `depuis un fichier texte, le presse-papier ou un lien URL...`
- ▶ Une fenêtre s'ouvre. Nommez `Maladie` le nom du tableau de données. Cochez l'entrée `Lien internet(URL)` dans `Fichier de données`, et l'entrée `Tabulations` pour le `Séparateur de champs`. Cliquez sur `OK`.
- ▶ Remplissez le champ de la boîte de dialogue `Lien internet (URL)` par `http://biostatisticien.eu/springeR/maladie.txt`.
- ▶ Cliquez sur `OK` pour voir apparaître dans la fenêtre de `Messages` : `Le jeu de données maladie a 80 lignes et 8 colonnes`.

Nous allons maintenant effectuer une régression multiple. Veuillez suivre les étapes suivantes.

- ▶ Dans le menu `Statistiques`, choisissez *Ajustement de modèles*, puis `Régression linéaire...`
- ▶ Choisissez par exemple `Modele.1` comme nom de modèle dans le champ `Entrez un nom pour le modèle`.
- ▶ Choisissez la variable `DISTANCE` comme variable réponse, puis les variables `POIDS` et `TAILLE` pour les variables explicatives (maintenez la touche `CTRL` enfoncée).
- ▶ Après avoir cliqué sur `OK`, vous voyez apparaître le résultat de votre modèle linéaire dans la `Fenêtre de sortie`. Ce résultat correspond aux instructions :

```
Modele.1 <- lm(DISTANCE~POIDS+TAILLE, data=Maladie)
summary(Modele.1)
```

que l'on peut voir dans la `Fenêtre de script`.

Renvoi

Le chapitre 12 présente le modèle linéaire plus en détail.

Nous allons visualiser le plan des moindres carrés correspondant au modèle ajusté.

- ▶ Dans le menu `Graphes`, choisissez *Graphe 3D*, puis `Nuage de points en 3D...`.

- Choisissez la variable `DISTANCE` comme variable réponse, puis les variables `POIDS` et `TAILLE` pour les variables explicatives (utilisez la touche `CTRL`).
- Choisissez `Moindres carrés linéaires` comme surface à ajuster. Cliquez sur `OK`.

Vous pouvez ainsi visualiser le nuage de points en 3D (présenté en figure A.8), ainsi que le plan des moindres carrés. Il est possible de faire bouger l'image obtenue à l'aide de la souris.

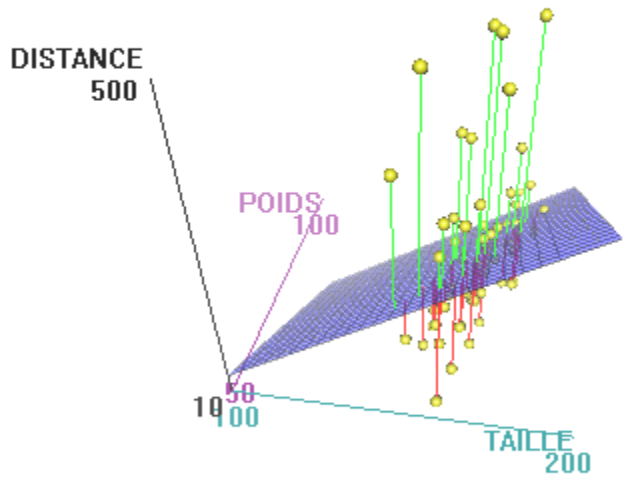


FIGURE A.8 – Plan des moindres carrés.

A.5.1.4 Rajouter des fonctionnalités à l'interface de `RCommander`

Certains *packages* disponibles sur le site officiel de R ont la particularité de pouvoir être intégrés dans les menus de `RCommander`. Ces *packages* sont faciles à identifier puisque leurs noms commencent par `RcmdrPlugin`. Nous illustrons ci-après l'utilisation de ce type de *package*.

Renvoi

Vous pouvez consulter l'article [24] qui présente comment construire un *package* pouvant être intégré dans `RCommander`.

- **Le *package* `TeachingDemos`**

Le *package* `RcmdrPlugin.TeachingDemos` peut être utilisé pour illustrer certains concepts statistiques.

- Tapez, dans la `Fenêtre de script`, `install.packages("RcmdrPlugin.TeachingDemos")`. Cliquez sur `Soumettre` et choisissez un site miroir proche de vous. Après l'installation, fermez puis relancez `RCommander` via l'instruction `Commander()`.
- Dans le menu `Outils`, choisissez *Chargez des plug-ins Rcmdr...*, cliquez sur `OK` puis répondre `Oui` à la question `Redémarrer maintenant?`.
- Vous voyez apparaître le nouveau menu `Demos`. Vous pouvez ainsi choisir dans ce menu, le sous-menu *Simple Correlation* et vous amuser à explorer la notion de corrélation.

Ce *plug-in* ajoute aussi des sous-menus à des menus déjà existants. Par exemple, dans le menu `Distributions`, vous pouvez maintenant choisir `Visualize distributions`, puis *t distributions*. En cochant `Show Normal Distribution`, et en jouant avec le curseur de `d.f.` (*degree of freedom*, i.e. degré de liberté), vous pouvez explorer la proximité de la loi de Student et de la loi normale.

- **Le *package* `sos`**

Le *package* `RcmdrPlugin.sos` peut être utilisé pour faciliter la recherche de l'aide sur un concept ou une fonction donnée.

Suivez les mêmes étapes que précédemment afin d'installer ce *plug-in*.
Un nouveau sous-menu nommé *Search R Help ... (sos)* apparaît dans le menu `Aide`. Explorez cette nouvelle fonctionnalité de `Rcommander` en tapant par exemple `linear model`.

Renvoi

Le chapitre 4 est dédié à la recherche d'informations sur R.

A.5.2 Utiliser R via la console

Nous avons vu dans la sous-section précédente qu'il était possible d'utiliser le logiciel R à l'aide de menus. En fait, cette façon de procéder est loin d'être op-

timale puisqu'elle limite considérablement les possibilités offertes par R. Ainsi, de très nombreuses analyses, plus poussées et/ou plus récentes et innovantes, ne sont pas disponibles dans les menus de `RCommander`. Il s'avère alors très utile de sortir de l'approche « clics-boutons » pour maîtriser le langage de programmation R. Vous pourrez ainsi facilement effectuer des simulations ou encore programmer des tâches répétitives. Nous avons déjà vu quelques instructions R lors de l'utilisation de `RCommander`, outil d'ailleurs lui-même programmé en langage R. Nous offrons maintenant un survol rapide de quelques éléments de syntaxe du langage R, d'abord au travers d'une analyse de données complexes issues d'une expérience menée en imagerie par résonance magnétique (IRM) fonctionnelle, puis en laissant le soin au lecteur de taper quelques commandes R et de réfléchir aux sorties produites.

A.5.2.1 La force de R illustrée sur un exemple

Certains neuroscientifiques cherchent à découvrir quelle est la zone du cerveau où est traitée spécifiquement l'information visuelle de couleur. Pour cela, un stimulus visuel, consistant en une alternance de motifs colorés et de motifs non colorés en mouvement, a été présenté à un sujet. Pendant la présentation de ce stimulus, des images volumiques du cerveau ont été acquises aux temps $t = 1, \ldots, T$ à l'aide d'un scanner IRM. Chacune de ces images 3D est en fait un gros (*rubik's*) cube constitué de nombreux voxels, équivalents 3D des pixels d'une image 2D. Chaque voxel contient, aux temps $t = 1, \ldots, T$, une valeur d'intensité magnétique $x(t)$. On peut ainsi considérer que l'on a observé, dans chacun de ces voxels, une série temporelle $\{x(t); t = 1, \ldots, T\}$ représentant des variations d'intensité électromagnétique. Les données acquises (contenues dans le fichier Mond4D.nii, produit au cours d'une expérience, dite de Mondrian, réalisée par M. Dojat et J. Huppé au Grenoble Institut des Neurosciences) consistent ainsi en un tableau quadridimensionnel, concaténation de plusieurs images volumiques cérébrales mesurées au cours du temps.

Le logiciel R a été utilisé afin de découvrir, dans une tranche donnée du cerveau, quel était le voxel dont les variations temporelles observées étaient les plus corrélées avec le signal du stimulus. Le code présenté ci-dessous peut-être téléchargé à l'adresse `http://biostatisticien.eu/springeR/code-cerveau.R`, et ouvert via le sous-menu *Ouvrir un script...* du menu `Fichier` de R. La combinaison de touches `CTRL+R` permet ensuite d'exécuter une à une les instructions de ce script. Le lecteur pourra essayer de lancer ces instructions pas à pas afin de visualiser les résultats. Cela lui permettra de se familiariser avec quelques-unes des possibilités offertes par R.

On commence par télécharger les fichiers de données dont nous aurons besoin.

```
> recupere <- function(fichier)
+  download.file(paste("http://biostatisticien.eu/springeR/",
```

```
+  fichier,sep=""),paste(getwd(),"/",fichier,sep=""),mode="wb")
> recupere("Mond4D.nii")
> recupere("Mondanat.hdr")
> recupere("Mondanat.img")
```

La paire de fichiers Mondanat.img/Mondanat.hdr contient une image anatomique du cerveau du sujet. Puis, on installe le *package* permettant la lecture de ces données.

```
> install.packages("AnalyzeFMRI") # Choisissez un miroir.

> # Noms des fichiers.
> file.func <- paste(getwd(),"/data/","Mond4D.nii",sep="")
> file.anat <- paste(getwd(),"/data/","Mondanat.img",sep="")

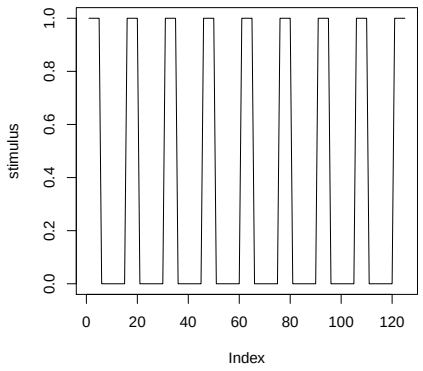
> slice <- 10 # Numéro de la coupe cérébrale.
```

Les instructions ci-dessous permettent la lecture des données (une tranche, *slice* en anglais, du cerveau en fait).

```
> anat.slice <- f.read.nifti.slice(file.anat,slice,1)
> class(anat.slice)
[1] "matrix"
> dim(anat.slice)
[1] 128 128
> func.slice <- f.read.nifti.slice.at.all.timepoints(
+                                        file.func,slice)
> class(func.slice)
[1] "array"
> dim(func.slice)
[1] 128 128 125
```

Nous créons à présent le codage du signal du stimulus visuel (1 = couleur, 0 = absence de couleur).

```
> stimulus <- c(rep(c(1,1,1,1,1,0,0,0,0,0,0,0,0,0,0),8),1,1,1,1,1)
> plot(stimulus,type="l")
```



Nous calculons les corrélations entre le signal temporel observé dans chaque voxel et le signal du stimulus.

```
> corMat <- matrix(NA,nrow=128,ncol=128)
> for (i in 1:128) {
+   for (j in 1:128) {
+     corMat[i,j] <- cor(func.slice[i,j,],stimulus)
+   }
+ }
```

Nous pouvons maintenant calculer les coordonnées du voxel le plus corrélé au stimulus :

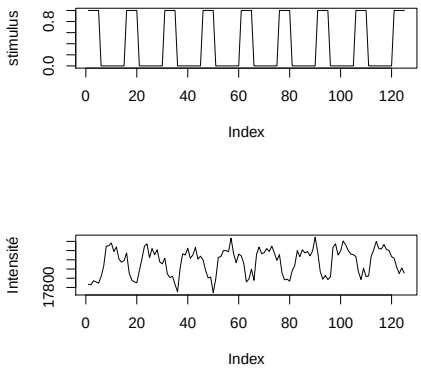
```
> which(abs(corMat)==max(abs(corMat),na.rm=TRUE),arr.ind=TRUE)
     row col
[1,]  67 117
```

ainsi que la valeur de corrélation obtenue pour ce voxel :

```
> corMat[67,117]
[1] -0.6675017
```

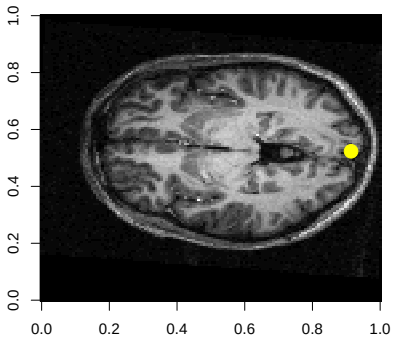
Nous pouvons alors tracer le signal temporel observé dans ce voxel.

```
> par(mfrow=c(2,1))
> plot(stimulus,type="l")
> plot(func.slice[67,117,],type="l",ylab="Intensité")
```



Nous pouvons maintenant identifier sur l'image anatomique du cerveau le voxel le plus actif en ce qui concerne le stimulus visuel.

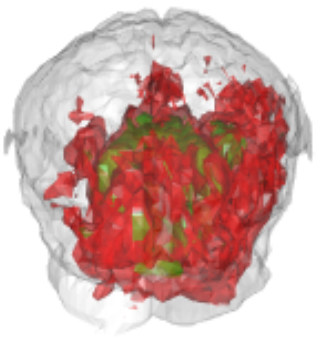
```
> image(as.matrix(rev(as.data.frame(t(anat.slice)))),
+       col = gray((0:32)/32))
> points(117/128,67/128,col="yellow",cex=2,pch=19)
```



Notons qu'il est également possible de visualiser ces données en 3D. Ainsi, les instructions ci-dessous, qui sont tirées du fichier d'aide de la fonction `contour3d()` du *package* `misc3d`, permettent-elles de voir de façon interactive l'image d'un cerveau en 3D.

```
> install.packages("misc3d")

> require("misc3d")
> a <- f.read.analyze.volume(system.file("example.img",
+                  package="AnalyzeFMRI"))
> a <- a[,,,1]
> contour3d(a,1:64,1:64,1.5*(1:21),lev=c(3000, 8000, 10000),
+         alpha=c(0.2,0.5,1),color=c("white","red","green"))
```



Vous pouvez essayer de faire bouger l'image obtenue à l'aide de votre souris.

A.5.2.2 Un survol de la syntaxe de R via des commandes à taper

• Opérations de base

Nous incitons le lecteur à jouer avec les commandes ci-dessous et à essayer d'en comprendre le fonctionnement.

```
> 1*2*3*4
[1] 24
> factorial(4)
[1] 24
> cos(pi)
[1] -1
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> exp(x)
 [1]     2.718282     7.389056    20.085537    54.598150
 [5]   148.413159   403.428793  1096.633158  2980.957987
 [9]  8103.083928 22026.465795
> x^2
 [1]   1   4   9  16  25  36  49  64  81 100
> chaine <- "R est fantastique!"
> chaine
[1] "R est fantastique!"
> nchar(chaine)
[1] 18
> ?nchar
> M <- matrix(x,ncol=5,nrow=2)
> M
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> M[2,3]
[1] 6
> L <- list(matrice=M,vecteur=x,chaine=chaine)
> L[[3]]
[1] "R est fantastique!"
> while(TRUE) {
+   adeviner <- sample(1:3,1)
+   {cat("Choisis un nombre parmi 1, 2, 3: ") ; value <- readline()}
+    if (value == adeviner) {print("Bien joué!") ; break()}
+    else print("Essaye encore.")
+ }

Choisis un nombre parmi 1, 2, 3: 1
[1] "Essaye encore."
Choisis un nombre parmi 1, 2, 3: 3
[1] "Essaye encore."
```

```
Choisis un nombre parmi 1, 2, 3: 1
[1] "Bien joué!"
> ls()
[1] "adeviner" "chaine"   "L"        "M"        "value"
[6] "x"
> rm(chaine)
```

Les commandes ci-dessous permettent d'effectuer certaines opérations sur des matrices.

```
> A <- matrix(runif(9),nrow=3)
> 1/A
         [,1]     [,2]     [,3]
[1,] 3.642582 2.327359 8.809875
[2,] 3.672352 1.534553 1.678063
[3,] 1.623827 1.761377 2.792906
> A * (1/A)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
> B <- matrix(1:12,nrow=3)
> A * B # Renvoie une erreur.
Error in A * B : tableaux de tailles inadéquates
> A %*% B
         [,1]     [,2]      [,3]     [,4]
[1,] 1.474401 3.927534  6.380667  8.83380
[2,] 3.363392 7.923050 12.482709 17.04237
[3,] 2.825455 7.450306 12.075157 16.70001
> (invA <- solve(A)) # Notez l'ajout de parenthèses et son effet!
          [,1]       [,2]      [,3]
[1,] -1.780899 -1.5162594  3.088187
[2,]  4.570620  0.4815619 -2.250472
[3,] -4.184287  1.8443127  1.049802
> A %*% invA
     [,1]         [,2]          [,3]
[1,]    1 5.551115e-17  0.000000e+00
[2,]    0 1.000000e+00 -1.110223e-16
[3,]    0 0.000000e+00  1.000000e+00
> det(A)
[1] 0.05896125
> eigen(A)
$values
[1]  1.3287063+0.0000000i -0.0222351+0.2094768i
[3] -0.0222351-0.2094768i
$vectors
             [,1]                   [,2]                   [,3]
[1,] 0.3506486+0i  0.0314831+0.5389506i  0.0314831-0.5389506i
[2,] 0.6942529+0i -0.4475357-0.3568936i -0.4475357+0.3568936i
[3,] 0.6285368+0i  0.6171547+0.0000000i  0.6171547+0.0000000i
```

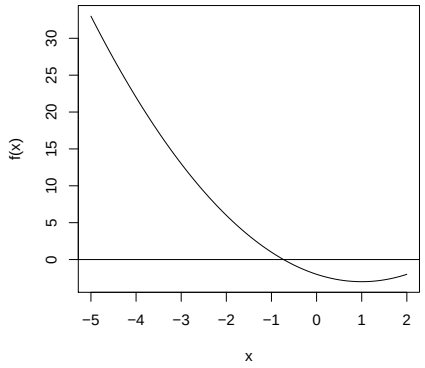
• **Statistiques**

Voilà quelques calculs de nature statistique.

```
> poids <- c(70,75,74)
> mean(poids)
[1] 73
> taille <- c(182,190,184)
> mat <- cbind(poids,taille)
> mat
     poids taille
[1,]    70    182
[2,]    75    190
[3,]    74    184
> apply(mat,MARGIN=2,FUN=mean)
   poids   taille
 73.0000 185.3333
> ?apply
> colMeans(mat)
   poids   taille
 73.0000 185.3333
> noms <- c("Pierre","Benoit","Rémy")
> donnees <- data.frame(Noms=noms,taille,poids)
> summary(donnees)
     Noms       taille          poids
 Benoit:1   Min.   :182.0   Min.   :70.0
 Pierre:1   1st Qu.:183.0   1st Qu.:72.0
 Rémy  :1   Median :184.0   Median :74.0
            Mean   :185.3   Mean   :73.0
            3rd Qu.:187.0   3rd Qu.:74.5
            Max.   :190.0   Max.   :75.0
```

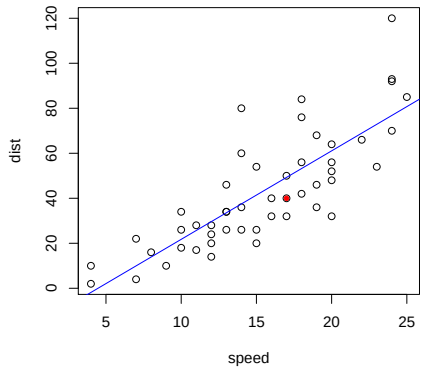
• **Quelques possibilités graphiques**

```
> f <- function(x) x^2-2*x-2
> curve(f,xlim=c(-5,2));abline(h=0)
> locator(1) # Cliquez sur le point au croisement des deux
             # courbes.
```

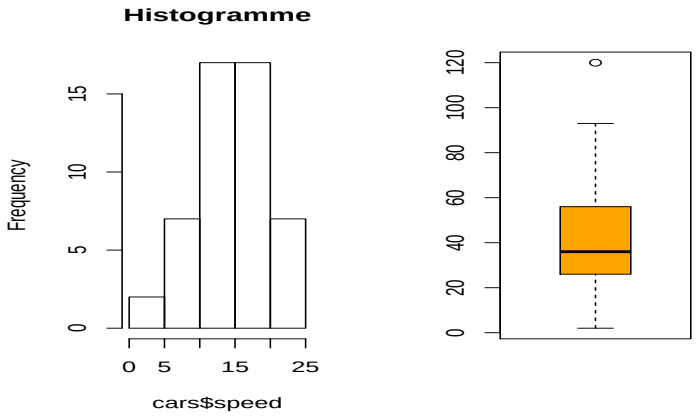



```
> uniroot(f,c(-5,2))
$root
[1] -0.7320503
$f.root
[1] -1.87445e-06
$iter
[1] 8
$init.it
[1] NA
$estim.prec
[1] 6.103516e-05
```

```
> plot(cars)
> abline(lm(dist~speed,data=cars),col="blue")
> points(cars[30,],col="red",pch=20)
```



```
> par(mfrow=c(1,2))
> hist(cars$speed,main="Histogramme")
> boxplot(cars$dist,col="orange")
```



Renvoi

Les liens suivants pointent vers un pense-bête des fonctions les plus utiles en R :

- http://cran.r-project.org/doc/contrib/Kauffmann_aide_memoire_R.pdf
- http://biostatisticien.eu/springeR/Commandes-R.pdf

Chapitre B

Quelques jeux de données et problématiques

Objectif

Ce chapitre présente quelques jeux de données provenant d'études épidémiologiques traitées par différentes équipes de l'Institut de santé publique d'épidémiologie et de développement (ISPED) de Bordeaux. Chacun de ces jeux de données s'accompagne d'une courte problématique qui permettra de mieux comprendre le contexte de l'étude. Ils serviront de fil conducteur tout au long de l'ouvrage. Nous montrerons comment il est possible d'utiliser les différentes fonctionnalités du logiciel R afin d'importer, de manipuler et d'effectuer les analyses statistiques adéquates sur ces jeux de données. Pour chaque jeu de données présenté, un tableau comprenant descriptif, variables et codage est fourni. Le lecteur devra se référer à ce chapitre lorsque ces études seront évoquées. Un tableau qui indique les chapitres dans lesquels chaque jeu de données est utilisé est fourni en fin de chapitre. Notez enfin que ces données sont accessibles sur le site internet associé au livre : http://www.biostatisticien.eu/springeR.

SECTION B.1

Indice de masse corporelle (IMC) chez des enfants

Présentation :

Un échantillon de dossiers d'enfants a été saisi. Ce sont des enfants vus lors d'une visite en 1re section de maternelle en 1996-1997 dans des écoles de Bordeaux (Gironde, France). L'échantillon est constitué de 152 enfants âgés de 3 ou 4 ans.

Variables et codage :

Description	**Unité** ou **Codage**	**Variable**
Sexe	`F` pour fille ; `G` pour garçon	`SEXE`
École située en zone d'éducation prioritaire	`O` pour oui ; `N` pour non	`zep`
Poids	kg (arrondi à 100 g près)	`poids`
Âge à la date de la visite	Années	`an`
Âge à la date de la visite	Mois	`mois`
Taille	cm (arrondi à 0,5 cm près)	`taille`

Jeu de données : IMC.ENFANT **Fichier :** imcenfant.xls

SECTION B.2

Poids de naissance

Présentation :

Il s'agit d'une enquête concernant les facteurs de risque associés au faible poids de naissance de nourrissons (données collectées au centre médical de Baystate dans le Massachusetts pendant l'année 1986). Le faible poids de naissance est un événement qui intéresse les médecins depuis plusieurs années en raison du taux de mortalité infantile et du taux d'anomalies infantiles très élevés chez les nourrissons de faible poids. Le comportement d'une femme pendant la grossesse (régime alimentaire, habitudes tabagiques ...) peut altérer de façon importante les chances de mener la grossesse à terme, et, par conséquent, de donner naissance à un enfant de poids normal. Le fichier de données contient les informations sur 189 femmes (numéro d'identification : `ID`) venant consulter dans le centre médical. On considère qu'un enfant a un faible poids de naissance si celui-ci est inférieur à 2 500 g.

Variables et codage :

Description	**Unité** ou **Codage**	**Variable**
Âge de la mère	Années	`AGE`
Poids de la mère lors du dernier cycle menstruel	Livres	`LWT`
Race de la mère	1 = Blanche ; 2 = Noire ; 3 = Autre	`RACE`
Tabagisme durant la grossesse	Oui = 1 ; Non = 0	`SMOKE`
Nombre d'antécédents de prématurité	0 = Non ; 1 = Un ; 2 = Deux ; etc.	`PTL`
Antécédents d'hypertension	Oui = 1 ; Non = 0	`HT`
Présence d'irritabilité utérine	Oui = 1 ; Non = 0	`UI`
Nombre de visites à un médecin durant le premier trimestre de la grossesse	0 = Aucune ; 1 = Une ; etc.	`FVT`
Poids de naissance	Grammes	`BWT`
Poids de naissance inférieur ou égal à 2 500 g	Oui = 1 ; Non = 0	`LOW`

Jeu de données : POIDS.NAISSANCE **Fichier :** Poids_naissance.xls

SECTION B.3

Épaisseur de l'intima-média

Présentation :

L'athérosclérose est la principale cause de décès chez l'homme après 35 ans et chez la femme après 45 ans dans la plupart des pays développés. C'est un épaississement et une perte d'élasticité des parois internes des artères, dont une des conséquences est l'infarctus du myocarde. La paroi artérielle est constituée de trois couches qui sont respectivement à partir de la lumière artérielle : l'intima, la média et l'adventice. L'épaisseur de l'intima-média est un marqueur reconnu d'athérosclérose. Elle a été mesurée par échographie sur un échantillon de 110 sujets en 1999 dans les CHU de Bordeaux. Des informations sur les principaux facteurs de risque ont aussi été recueillies.

Variables et codage :

Description	**Unité** ou **Codage**	**Variable**
Sexe	1 = Homme ; 2 = Femme	`SEXE`
Âge le jour de la visite	Années	`AGE`
Taille	cm	`taille`
Poids	kg	`poids`
Statut tabagique	0 = Ne fume pas 1 = A arrêté de fumer 2 = Fume	`tabac`
Estimation de consommation pour les fumeurs et ex-fumeurs	Nombre de paquets/année	`paqan`
Activité physique	0 = Non ; 1 = Oui	`SPORT`
Mesure de l'intima-média	mm	`mesure`
Consommation d'alcool	0 = Ne boit pas 1 = Boit occasionnellement 2 = Boit régulièrement	`alcool`

Jeu de données : INTIMA.MEDIA **Fichier :** Intima_Media.xls

SECTION B.4

Alimentation chez des personnes âgées

Présentation :

Un échantillon de personnes âgées résidant à Bordeaux (Gironde, France) a été interrogé en 2000 dans le cadre d'une enquête nutritionnelle. L'échantillon est constitué de 226 sujets.

Variables et codage :

Description	**Unité** ou **Codage**	**Variable**
Sexe	2 = Femme ; 1 = Homme	`sexe`
Situation familiale	1 = Vit seul 2 = Vit en couple 3 = Vit dans sa famille 4 = Autre type de cohabitation	`situation`
Consommation journalière de thé	Nombre de tasses	`the`
Consommation journalière de café	Nombre de tasses	`cafe`
Taille	cm	`taille`
Poids	kg	`poids`
Âge le jour de l'entretien	Années	`age`
Consommation de viande	0 = Jamais 1 = Moins d'une fois par semaine 2 = Une fois par semaine 3 = 2/3 fois par semaine 4 = 4/6 fois par semaine 5 = Tous les jours	`viande`
Consommation de poisson	Idem	`poisson`
Consommation de fruits crus	Idem	`fruit_crus`
Consommation de fruits et légumes cuits	Idem	`fruit_legume_cuits`
Consommation de chocolat	Idem	`chocol`
Matière grasse préférentiellement utilisée pour la cuisson	1 = Beurre 2 = Margarine 3 = Huile d'arachide 4 = Huile de tournesol 5 = Huile d'olive 6 = Mélange d'huile (type Isio4) 7 = Huile de colza 8 = Graisse de canard ou d'oie	`matgras`

Jeu de données : NUTRIAGE **Fichier :** nutriage.xls

SECTION B.5

Étude cas témoins sur l'infarctus du myocarde

Présentation :

Les données suivantes sont issues d'une enquête cas témoins dont le but était d'évaluer l'existence d'un risque plus élevé de survenue d'un infarctus du myocarde chez les femmes qui utilisent ou ont utilisé des contraceptifs oraux. L'étude a été menée auprès de 149 femmes ayant eu un infarctus du myocarde (cas) et 300 femmes n'en n'ayant pas eu (témoins). Le facteur d'exposition principal est la prise de contraceptifs oraux, les autres facteurs recueillis sont : l'âge, le poids, la taille, la consommation de tabac, l'hypertension artérielle, les antécédents familiaux de maladies cardio-vasculaires.

Variables et codage :

Description	**Unité** ou **Codage**	**Variable**
Infarctus du myocarde	0 = Témoins ; 1 = Cas	`infarct`
Prise de contraceptifs oraux	0 = Jamais ; 1 = Oui	`co`
Consommation de tabac	0 = Non 1 = Fumeuse actuelle 2 = Ancienne fumeuse	`tabac`
Âge	Années	`age`
Poids	kg	`poids`
Taille	cm	`taille`
Antécédents familiaux de maladie cardio-vasculaire	0 = Non ; 1 = Oui	`atcd`
Hypertension artérielle	0 = Non ; 1 = Oui	`hta`

Jeu de données : INFARCTUS **Fichier :** Infarct.xls

SECTION B.6

Tableau résumant l'utilisation des jeux de données

		Méthodes					
		Import-Export	**Manipulation**	**Statistique descriptive**	**Tests**	**ANOVA**	**Régression**
Jeux de données	IMC. ENFANT	×	×		×		
	POIDS. NAISSANCE	×	×				×
	INTIMA. MEDIA	×	×		×	×	×
	NUTRIAGE	×	×	×	×		
	INFARCTUS	×		×			

Première partie

Les bases du logiciel R

Chapitre 1

Les concepts de base, l'organisation des données

Objectif

Ce chapitre présente les concepts de base du logiciel **R** (mode calculatrice, opérateur d'affectation, variables, utilisation de fonctions, paramètres) ainsi que les différents types et structures de données que **R** peut manipuler.

SECTION 1.1

Votre première session

Après avoir lancé le logiciel **R** en double-cliquant sur son icône sur le bureau de Windows (ou bien à partir du Menu Démarrer), vous voyez apparaître, à la fin de l'affichage qui se déroule dans la console de **R** (appelée *R Console*), le **caractère d'invite de commande** > vous invitant à taper votre première instruction en langage **R**.

```
R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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

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

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

>
```

Il s'agit du symbole d'incitation à donner une instruction (*prompt symbol*). Tapez par exemple `"R est mon ami"` puis validez en tapant la touche `ENTRÉE` (ou `RETURN`). Vous obtenez alors :

```
> "R est mon ami"
[1] "R est mon ami"
```

Comme vous pouvez le constater, R est bien éduqué et répond gentiment à votre requête. Ce sera généralement le cas, peut-être pour se faire pardonner son manque de convivialité. Nous expliquerons plus tard pourquoi la réponse de R est précédée de `[1]`.

1.1.1 R est une calculatrice

R, comme beaucoup d'autres langages de ce type, remplace aisément les fonctionnalités d'une calculatrice (très sophistiquée !). Il permet aussi, et c'est une grande force, de faire des calculs sur des vecteurs. Voici déjà quelques exemples très simples.

```
> x <- 5*(-3.2)          # Attention, le séparateur décimal doit
                         # être un point (.)
> 5*(-3,2)               # sinon, l'erreur suivante est générée:

Erreur : ',' inattendu(e) dans "5*(-3,"

> 5^2                    # Identique à 5**2.
[1] 25
> sin(2*pi/3)
[1] 0.8660254
> sqrt(4)                # Racine carrée de 4.
[1] 2
> log(1)                 # Logarithme népérien de 1.
[1] 0
> c(1,2,3,4,5)           # Crée un vecteur contenant les cinq
                         # premiers entiers.
[1] 1 2 3 4 5
> z <- c(1,2,3,4,5)*2    # Calcul des cinq premiers nombres pairs.
```

Astuce

Tout code R qui suit le caractère « # » est considéré par R comme un commentaire. En fait, il n'est pas interprété par R.

Vous pouvez maintenant quitter le logiciel R en tapant l'instruction suivante : `q()`.

Il vous est alors proposé de sauver une image de la session. En répondant oui, les commandes tapées précédemment seront de nouveau accessibles lors d'une prochaine réouverture de R, au moyen des flèches « haut » et « bas » du clavier.

1.1.2 Affichage des résultats et redirection dans des variables

Comme vous l'avez sans doute remarqué, R répond à vos requêtes en affichant le résultat obtenu après évaluation. **Ce résultat est affiché puis perdu**. Dans une première utilisation, cela peut paraître agréable, mais dans une utilisation plus poussée il apparaît plus intéressant de rediriger la sortie R de votre requête en la stockant dans une variable : cette opération s'appelle aussi **affectation du résultat dans une variable**. Une affectation évalue ainsi une expression, mais n'affiche pas le résultat qui est en fait stocké dans un objet. Pour afficher ce résultat, il vous suffira de taper le nom de cet objet, suivi de la touche `ENTRÉE`.

Pour réaliser cette opération, on utilise la **flèche d'affectation** `<-`. La flèche `<-` s'obtient en tapant le signe inférieur (`<`) suivi du signe moins (`-`).

Pour créer un objet dans R, on utilise donc la syntaxe suivante :
`Nom.objet.a.creer <- instructions`

Par exemple :

```
> x <- 1        # Affectation.
> x             # Affichage.
[1] 1
```

On dit alors que `x` vaut `1`, ou que l'on affecte `1` à `x`, ou encore que l'on met dans `x` la valeur `1`. Notez que l'on peut aussi utiliser l'opération d'affectation dans l'autre sens `->` de la façon suivante :

```
> 2 -> x
> x
[1] 2
```

Attention

On peut aussi utiliser le signe `=`, mais son utilisation est moins générale et donc déconseillée. En effet, l'égalité en mathématique est une relation symétrique ayant un sens bien précis, très différent de celui de l'affectation.

Par ailleurs, il y a des cas où l'utilisation du signe = ne fonctionne pas du tout.

Astuce

Notez qu'il est possible, en utilisant une paire de parenthèses, d'affecter une valeur à une variable tout en affichant le résultat de l'évaluation :

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

Enfin, si une commande n'est pas complète à la fin d'une ligne, R affichera un signe d'invite différent, par défaut le signe plus (+), sur la deuxième ligne ainsi que sur les lignes subséquentes. R continuera d'attendre des instructions jusqu'à ce que la commande soit syntaxiquement complète.

```
> 2*8*10+exp(1)
[1] 162.7183
> 2*8*
+ 10+exp(
+ 1)
[1] 162.7183
```

Attention

Voici les **règles pour choisir un nom de variable** dans R : tout nom de variable ne peut être constitué que de caractères alphanumériques ainsi que du point (.) ; les noms de variables sont *case sensitive*, signifiant que R fait la distinction entre minuscules et majuscules ; un nom de variable ne peut pas contenir des espaces ou commencer par un chiffre, sauf s'il est encadré de guillemets "".

1.1.3 Stratégie de travail

- Prenez l'habitude de stocker vos fichiers dans un dossier réservé à cet usage (nommé par exemple `TravauxR`). En outre, nous vous conseillons de taper toutes vos instructions **R** dans une fenêtre de script appelée *script* ou *R Editor*, accessible depuis le menu « Fichier/Nouveau script ». Ouvrez une nouvelle fenêtre de script, cliquez dans le menu « Fenêtres/-Juxtaposées » puis copiez le script suivant :

```
x <- 5*(-3.2)
5^2
sin(2*pi/3)
sqrt(4)
c(1,2,3,4,5)
z <- c(1,2,3,4,5)*2
```

Mac

Pour les Mac, le menu est « Fichier/Nouveau Document », et il n'est pas possible de juxtaposer les fenêtres.

À la fin de votre session, vous pourrez sauver ce script, dans le dossier `TravauxR`, sous le nom `monscript.R` par exemple, et le rouvrir lors d'une session ultérieure depuis le menu « Fichier/Ouvrir un script » (ou bien encore « Fichier/Ouvrir Document » pour les Mac).

- Ensuite, vous pouvez taper successivement les combinaisons de touches `CTRL+A` (`COMMAND+A` pour les Mac) pour sélectionner l'ensemble de ces instructions, puis `CTRL+R` (`COMMAND+ENTER` pour les Mac) pour les coller et les exécuter en une seule étape dans la console de **R**. Vous pouvez aussi exécuter une seule ligne d'instructions **R** du script en tapant `CTRL+R` lorsque le curseur clignotant se trouve sur la ligne en question dans la fenêtre de script.

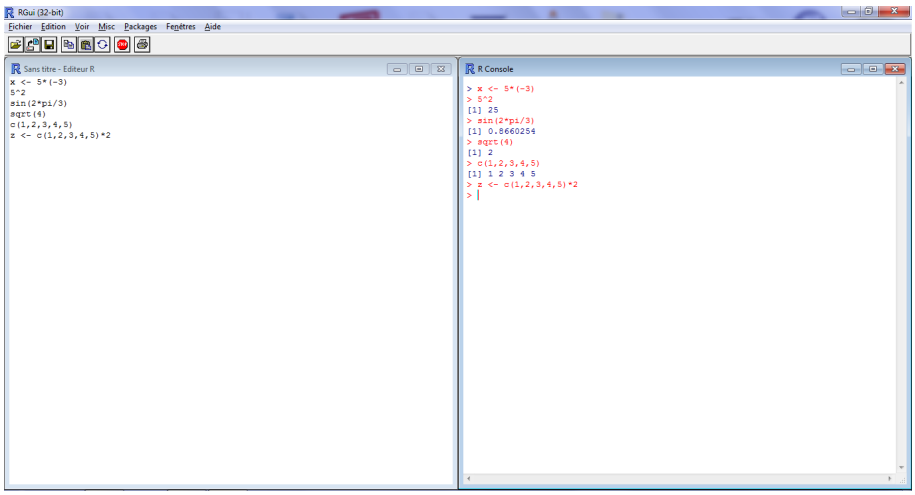


FIGURE 1.1 – Vue de la fenêtre de script et de la console de commandes.

Astuce

Notez sur la figure 1.1 la présence du bouton rouge STOP qui permet d'interrompre un calcul qui s'éterniserait.

Vous pouvez également utiliser la fonction `source()` depuis la console de **R** pour aller lire et exécuter le contenu de votre fichier. Cela évitera de surcharger inutilement votre console, comme nous le verrons dans une pratique ultérieure. Pour cela, il faudrait procéder ainsi :

a) cliquer une fois dans la fenêtre *R Console* ;

b) aller dans le menu « Fichier/Changer le répertoire courant » (ou bien « Divers/Changer de répertoire de travail » pour les Mac) ;

c) explorer votre système de fichiers pour sélectionner le dossier `TravauxR` ;

d) taper dans la console : `source("monscript.R")`. Notez que pour l'exemple ci-dessus, l'utilisation de cette instruction ne produira aucun affichage. La Prise en main qui suit éclaircira ce point.

Prise en main

Commencez par créer un dossier nommé TravauxR dans votre compte. Puis, tapez et sauvez dans un script R les instructions précédentes. Le fichier contenant le script R sera appelé monscript.R et sera placé dans TravauxR. Maintenant, fermez puis réouvrez R. Changez votre répertoire de travail courant pour TravauxR comme expliqué en page précédente. Puis, tapez les instructions suivantes dans la console de R :

```
rm(list=ls()) # Supprime tous les objets existants.
ls()          # Liste les objects existants.
source("myscript.R")
ls()
x
z
```

Notez que la fonction `source()` a permis d'exécuter les instructions précédentes. Vous aurez aussi peut-être noté que les calculs qui n'ont pas été redirigés dans des variables n'ont pas été affichés. Ainsi leur résultat est perdu. Changez votre script et ajoutez-y à la fin les instructions suivantes :

```
print(2*3)
print(x)
```

Sauvez-le, puis sourcez-le. Que s'est-il passé ?

- Prenez le plus possible l'habitude d'utiliser le système d'aide en ligne de R. Cette aide très complète, en anglais, est accessible au moyen de la fonction `help()`. Vous pouvez par exemple taper `help(source)` pour obtenir de l'aide sur la fonction `source()`.

Renvoi

Toutes ces notions seront détaillées dans les chapitres 4 et 7.

Astuce

Pour les utilisateurs travaillant sous l'environnement Microsoft, nous conseillons l'utilisation de l'éditeur de code RStudio (multi-plateformes) ou de l'éditeur Tinn-R (Microsoft Windows seulement), disponibles respectivement à http://www.rstudio.com et http://www.sciviews.org/Tinn-R. Ils offrent en effet une meilleure interaction entre le code d'un script et son exécution. Ils permettent aussi de colorer syntaxiquement le code.

Linux

Notez l'existence des éditeurs JGR et Emacs/ESS sous Linux.

Renvoi

Vous pouvez consulter la liste des éditeurs de **R** sur la page suivante : `http://www.sciviews.org/_rgui/projects/Editors.html`.

Prise en main

L'indice de masse corporelle (IMC) permet de déterminer la corpulence d'une personne. Il se calcule au moyen de la formule suivante :

$$\text{IMC} = \frac{\text{Poids (kg)}}{\text{Taille}^2 \text{ (m)}}.$$

Calculons notre IMC. Pour cela, il suffit de taper dans votre fenêtre de script les lignes suivantes :

```
# Il est possible d'écrire 2 instructions
# sur la même ligne grâce au signe ;
Mon.Poids <- 75 ; Ma.Taille <- 1.90
Mon.IMC <- Mon.Poids/Ma.Taille^2
Mon.IMC
```

Lancez ce script en utilisant la stratégie de travail vue précédemment. Vous pouvez ensuite modifier ce script pour calculer votre propre IMC.

Nous proposons une fonction permettant de visualiser votre type de corpulence. Pour cela, veuillez lancer maintenant les deux instructions suivantes :

```
source("http://www.biostatisticien.eu/springeR/IMC.R",
encoding="utf8")
affiche.IMC(Mon.IMC)
```

Vous apprendrez comment programmer ce genre de résultat au cours des chapitres ultérieurs.

1.1.4 Utilisation de fonctions

Nous avons vu quelques utilisations des fonctions `sin()`, `sqrt()`, `exp()` et `log()`. Le logiciel R contient de nombreuses autres fonctions dans sa version de base, et on peut en ajouter des milliers d'autres (en installant des *packages*, ou même en les réalisant soi-même).

On notera que toute fonction dans R est définie par son **nom** et par la liste de ses **paramètres**. La plupart des fonctions renvoient une **valeur**, qui peut être un nombre, un vecteur, une matrice...

Utiliser une fonction (on dit aussi **appeler** ou **exécuter**) se fait en tapant le nom de celle-ci, suivi, entre une paire de parenthèses, de la liste des paramètres (formels) que l'on veut utiliser. Les paramètres sont séparés par des virgules. Chacun des paramètres peut être suivi du signe = et de la valeur que l'on veut donner au paramètre. Cette valeur du paramètre formel sera appelée paramètre effectif, paramètre d'appel ou parfois paramètre d'entrée.

Attention

Prenez garde toutefois au fait que le R utilise le terme anglais *argument* pour désigner ce que nous appelons paramètre. Nous avons choisi de ne pas utiliser l'anglicisme *argument* dans cet ouvrage.

Nous utiliserons donc l'instruction

```
nomfonction(par1=valeur1,par2=valeur2,par3=valeur3)
```

où `par1, par2, ...` sont appelés les paramètres de la fonction tandis que `valeur1` est la valeur que l'on donne au paramètre `par1`, etc. On peut toutefois noter qu'il n'est pas forcément nécessaire d'indiquer les paramètres mais seulement leurs valeurs, pour autant que l'on respecte leur ordre.
Pour toute fonction présente dans R, certains paramètres sont obligatoires et d'autres sont facultatifs (car une valeur par défaut est déjà fournie dans le code de la fonction).

Attention

Ne pas oublier de saisir la paire de parenthèses lors de l'appel d'une fonction. En effet, une erreur courante en première utilisation consiste à les oublier :

```
> factorial
function (x)
gamma(x + 1)
```

```
<bytecode: 0x472cd30>
<environment: namespace:base>
> factorial(6)
[1] 720
```

La sortie de la première instruction fournit le code (c'est-à-dire la recette de fabrication) de la fonction considérée tandis que la deuxième instruction l'exécute. Cela est également valable pour les fonctions définies sans paramètre comme le montre l'exemple suivant :

```
> date()
[1] "Wed Jun 25 21:40:40 2014"
> date
function ()
.Internal(date())
<bytecode: 0x35fe768>
<environment: namespace:base>
```

Bien évidemment, il n'est pas question ici de commenter le code des fonctions précédentes.

Afin de mieux comprendre la façon dont les paramètres peuvent être utilisés, prenons l'exemple de la fonction `log(x,base=exp(1))`. On peut remarquer qu'elle admet deux paramètres : `x` et `base`.

Le paramètre `x` est obligatoire, c'est le nombre dont on veut calculer le logarithme. Le paramètre `base` est un paramètre optionnel puisqu'il est suivi du signe `=` et de la valeur par défaut `exp(1)`.

Astuce

Un paramètre obligatoire est un paramètre qui n'est pas suivi du signe `=`. Un paramètre est optionnel s'il est suivi du signe `=`.

Ainsi, dans l'appel suivant, le calcul effectué sera celui du logarithme népérien du nombre `1` puisque la valeur de base n'est pas renseignée :

```
> log(1)
[1] 0
```

Remarque

Certaines fonctions ne possèdent aucun paramètre obligatoire, comme la fonction `matrix` que l'on verra plus tard.

Un dernier point important à noter est que l'**on peut appeler une fonction en jouant sur les paramètres de plusieurs façons différentes**. Cela est un atout important de R en termes de simplicité d'utilisation, et il convient de bien comprendre ce principe de fonctionnement. Ainsi, pour calculer le logarithme népérien de 3, on peut utiliser n'importe laquelle des expressions suivantes.

```
log(3)                          log(3,base=exp(1))
log(x=3)                        log(3,exp(1))
log(x=3,base=exp(1))            log(base=exp(1),3)
log(x=3,exp(1))                 log(base=exp(1),x=3)
```

Attention

Il faut prendre garde au fait que l'appel suivant

```
log(exp(1),3)
```

revient à calculer le logarithme de `exp(1)` en base 3.

Pour terminer, il est bon de noter que nous avons pu voir précédemment le code de la fonction `factorial()` :

```
> factorial
function (x)
gamma(x + 1)
<bytecode: 0x472cd30>
<environment: namespace:base>
```

Cette fonction a été définie par les développeurs de R au moyen des instructions suivantes :

```
> factorial <- function(x) gamma(x+1)
```

Il est donc très facile en R de programmer soi-même une nouvelle fonction en utilisant la fonction `function()`. Par exemple, voici comment programmer la fonction des deux variables n et p qui calcule le coefficient du binôme de Newton $\binom{n}{p} = \frac{n!}{p!(n-p)!}$:

```
> binome <- function(n,p) factorial(n)/(factorial(p)*
+                         factorial(n-p))
```

Il est ensuite possible d'utiliser votre nouvelle fonction comme n'importe quelle autre fonction R :

```
> binome(4,3)
[1] 4
```

Nous verrons de façon beaucoup plus détaillée comment créer des fonctions plus élaborées dans la section 3.8 et dans le chapitre 6.

Remarque

En fait, il existe déjà une fonction R permettant de calculer le coefficient du binome de Newton. Il s'agit de la fonction `choose()` qui fonctionne de façon plus efficace, notamment pour les grands nombres.

SECTION 1.2

Les données dans R

Comme la plupart des langages informatiques, R dispose des types de données classiques. Selon la forme des données saisies, R sait d'ailleurs reconnaître automatiquement le type de ces données. Une des grandes forces de R réside aussi dans la possibilité d'organiser les données de façon structurée. **Cela sera très utile lors de l'utilisation de nombreuses procédures statistiques qui seront détaillées ultérieurement.**

1.2.1 Nature (ou type, ou mode) des données

Les fonctions `mode()` et `typeof()`, renvoyant des valeurs identiques à de rares subtilités près non détaillées ici, permettent de gérer le « type » des données.

Remarque

La fonction `class()` est plus générale puisqu'elle permet de gérer à la fois le type et la structuration des données. Elle sera présentée plus loin. Pour des raisons pédagogiques, nous utilisons ici la commande `typeof()`.

Énumérons maintenant les divers types de données (aussi appelés modes).

1.2.1.1 Type numérique (`numeric`)

Il y a deux types numériques : les entiers (`integer`) et les réels (`double`). Lorsque vous saisissez :

```
> a <- 1
> b <- 3.4
> c <- as.integer(a)
> typeof(c)
[1] "integer"
```

les variables `a` et `b` sont du type `"double"` et la variable `c` a la même valeur que `a` excepté qu'elle a été forcée à être du type `"integer"`. L'intérêt du type `"integer"` est que le stockage d'entiers nécessite moins de place mémoire que le stockage de `"double"`s. Les instructions commençant par `as.` sont très courantes en **R** pour convertir une donnée en un type différent. Nous verrons dans la sous-section 1.2.2.1 comment vérifier que le type d'un objet est numérique.

1.2.1.2 † Type complexe (`complex`)

On fabrique un nombre complexe à l'aide de la lettre `i`. On utilise les fonctions `Re()` pour la partie réelle, `Im()` pour la partie imaginaire, `Mod()` pour le module et `Arg()` pour l'argument.

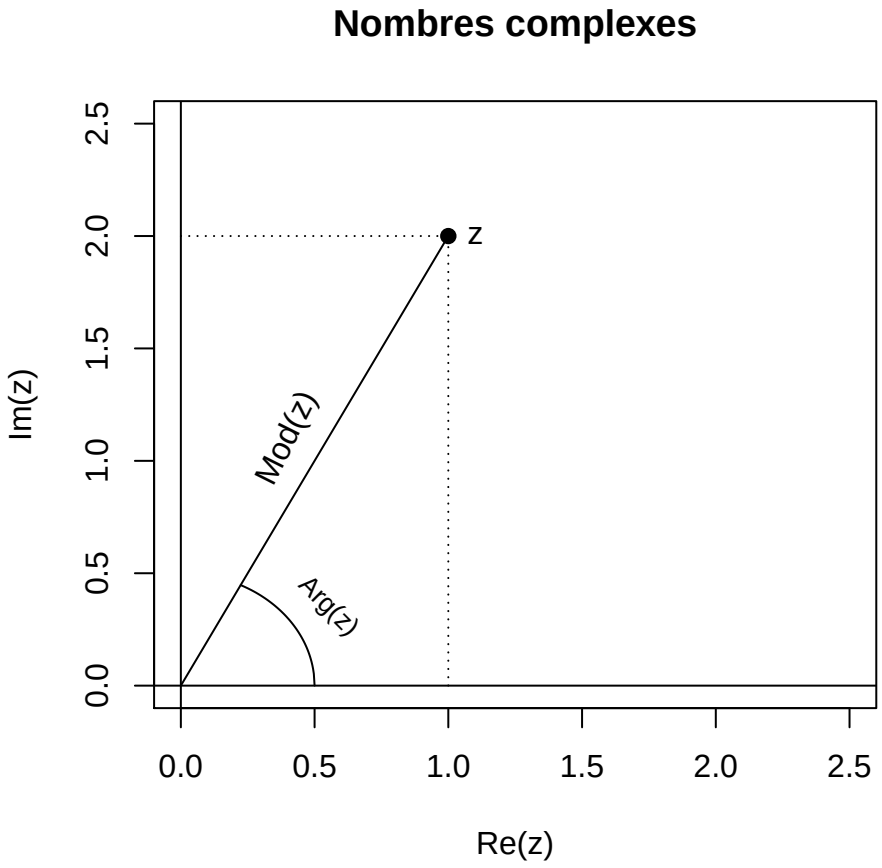


FIGURE 1.2 – Caractéristiques d'un nombre complexe.

Voici quelques exemples :

```
> 1i
[1] 0+1i
> z <- 1+2i
> typeof(z)
[1] "complex"
> is.complex(z) # Permet de savoir si un objet est du type
                # complex.
[1] TRUE
> Re(z)
[1] 1
> Im(z)
[1] 2
> Mod(z)
[1] 2.236068
> Arg(z)
[1] 1.107149
```

1.2.1.3 Type booléen ou logique (`logical`)

Le type `logical`, résultat d'une condition logique, peut prendre les valeurs `TRUE` ou `FALSE`. Voici quelques instructions pour créer des valeurs logiques :

```
> b>a
[1] TRUE
> a==b
[1] FALSE
> is.numeric(a)
[1] TRUE
> is.integer(a)
[1] FALSE
> x <- TRUE
> is.logical(x)
[1] TRUE
```

Attention

`TRUE` et `FALSE` peuvent être saisis de manière plus succincte en tapant respectivement `T` et `F`. Mais cette approche n'est pas recommandée.

Lorsque cela se révèle nécessaire, ce type de données est naturellement converti en type numérique sans qu'il y ait à le spécifier : `TRUE` vaut `1` et `FALSE` vaut `0`. Cela est illustré par l'exemple suivant :

```
> TRUE + T + FALSE*F + T*FALSE + F
[1] 2
```

1.2.1.4 Données manquantes (`NA`)

Une donnée manquante ou non définie est indiquée par l'instruction `NA` (pour *non available* = non disponible), et plusieurs fonctions existent pour

gérer ce type de données. En fait, R considère ce type de données comme étant une valeur logique constante. Il ne s'agit donc pas d'un type de données à proprement parler. Voici quelques exemples faisant intervenir l'instruction `NA` :

```
> x <- c(3,NA,6)
> is.na(x)
[1] FALSE  TRUE FALSE
> mean(x)               # Tentative de calcul de la moyenne de x.
[1] NA
> mean(x,na.rm=TRUE)    # Le paramètre na.rm signifie enlever les
                        # NA (NA.remove).
[1] 4.5
```

Cette notion est très importante lors de la lecture de fichiers de données statistiques et sera développée dans le chapitre 3.

Attention

Ne pas confondre `NA` avec le mot réservé `NaN` signifiant *not a number* :

```
> 0/0
[1] NaN
```

Notez également que l'instruction suivante ne renvoie pas `NaN` mais l'infini, représenté en R par le mot réservé `Inf`.

```
> 3/0
[1] Inf
```

1.2.1.5 Type chaînes de caractères (`character`)

Toute information mise entre guillemets (simple `'` ou double `"`) correspond à une chaîne de caractères :

```
> a <- "R est mon ami"
> mode(a)
[1] "character"
> is.character(a)
[1] TRUE
```

Les conversions en chaîne de caractères depuis un autre type sont possibles. Les conversions réciproques s'appliquent dès lors que le contenu entre les guillemets peut être correctement interprété par R. Notez enfin que certaines conversions se font de manière automatique. Voici quelques exemples :

```
> as.character(2.3)          # Conversion vers une chaîne de
                             # caractères.
[1] "2.3"
> b <- "2.3"
> as.numeric(b)              # Conversion depuis une chaîne de
                             # caractères.
[1] 2.3
> as.integer("3.4")          # Conversion depuis une chaîne de
                             # caractères.
[1] 3
> c(2,"3")                   # Conversion automatique.
[1] "2" "3"
> as.integer("3.quatre")     # Conversion impossible.
[1] NA
```

Remarque

Nous verrons au chapitre 3 les différences existant entre les double et simple guillemets.

1.2.1.6 † Données brutes (raw)

R offre la possibilité de travailler directement avec des octets (affichés sous forme hexadécimale). Cela peut parfois être utile lors de la lecture de certains fichiers au format binaire. Nous en verrons des exemples dans le chapitre 5.

```
> x <- as.raw(15)
> x
[1] 0f
> mode(x)
[1] "raw"
```

Récapitulatif

Table 1.1 – Les différents types de données en R.

Type des données	Type sous R	Présentation
réel (entier ou non)	`numeric`	`3.27`
complexe	`complex`	`3+2i`
logique (vrai/faux)	`logical`	`TRUE` ou `FALSE`
manquant	`logical`	`NA`
texte (chaîne)	`character`	`"texte"`
binaires	`raw`	`1c`

Astuce

La fonction `storage.mode()` permet d'obtenir ou de définir le type ou le mode de stockage d'un objet.

1.2.2 Structures de données

R offre la possibilité d'organiser (de structurer) les différents types de données définies précédemment. La fonction `class()` permettra de manipuler des structures. Nous présentons les plus utiles dans les pages qui viennent.

1.2.2.1 Les vecteurs (`vector`)

Cette structure de données est la plus simple. Elle représente **une suite de données de même type**. La fonction permettant de créer ce type de structure (c'est-à-dire les vecteurs) est la fonction `c()` (pour collection ou concaténation). D'autres fonctions comme `seq()` ou bien les deux points `:` permettent aussi de créer des vecteurs. Notez que lors de la création d'un vecteur, il est possible de mélanger des données de plusieurs types différents. **R** se charge alors d'opérer une conversion implicite vers le type de données le plus fréquent comme vous pouvez le constater dans les exemples ci-dessous.

```
> c(3,1,7)
[1] 3 1 7
> c(3,TRUE,7)
[1] 3 1 7
> c(3,T,"7")
[1] "3"    "TRUE" "7"
> seq(from=0,to=1,by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(from=0,to=20,length=5)
[1]  0  5 10 15 20
> vec <- 2:36
> vec
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[20] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

Notons qu'il est possible de « nommer » les éléments d'un vecteur à l'aide de la fonction `names()`.

```
> vec <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
> names(vec) <- letters[1:9] # 9 premières lettres de l'alphabet.
> vec
a b c d e f g h i
1 3 6 2 7 4 8 1 0
```

Attention

Les indications *[1]* et *[26]* fournissent le rang de l'élément qui les suit dans le vecteur `vec`.

```
> is.vector(vec)
[1] TRUE
> x <- 1:3
> x
[1] 1 2 3
> y <- c(1,2,3)
> y
[1] 1 2 3
> class(x)
[1] "integer"
> class(y)
[1] "numeric"
```

On se serait en fait attendu à voir s'afficher `"vector of doubles"` ou `"vector of integers"` plutôt que `"numeric"` ou `"integer"`, mais aucun logiciel n'est parfait !

Expert

Notez que les instructions `c()` et `:` fournissent le même affichage, mais `x` et `y` sont ici stockées en interne de façon différente. Le type `integer` utilise moins de mémoire que le type `numeric`.

1.2.2.2 Les matrices (`matrix`), les tableaux (`arrays`)

Ces deux notions généralisent la notion de vecteur puisqu'elles représentent des suites à double indice pour les matrices et à multiples indices pour les tableaux (*array*). Ici aussi **les éléments doivent avoir le même type**, sinon des conversions implicites seront effectuées.

L'instruction suivante :

```
> X <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> X
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

permet de créer (et stocker dans la variable `X`) une matrice comportant quatre lignes (*row* signifie ligne) et trois colonnes remplies par lignes successives (`byrow`

=TRUE) avec les éléments du vecteur `1:12` (c'est-à-dire les douze premiers entiers).

De la même manière, il est possible de créer une matrice remplie par colonnes successives (`byrow=FALSE`).

```
> Y <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)
> Y
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> class(Y)
[1] "matrix"
```

La fonction `array()` permet de créer des matrices multidimensionnelles à plus de deux dimensions comme cela est illustré sur la figure suivante (pour un *array* ayant trois dimensions).

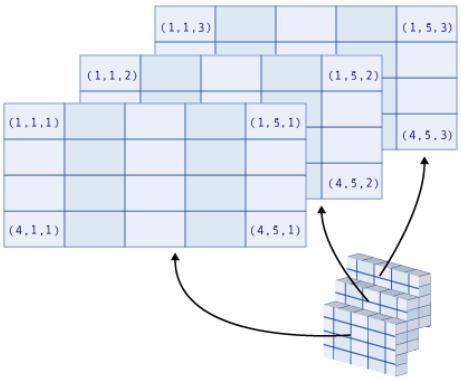


FIGURE 1.3 – Illustration d'une *array*.

```
> X <- array(1:12,dim=c(2,2,3))
> X
, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
     [,1] [,2]
```

```
[1,]    9   11
[2,]   10   12
> class(X)
[1] "array"
```

Attention

Il est possible de créer des tableaux à plus de trois dimensions au moyen du paramètre `dim` qui peut en effet avoir une longueur supérieure à 3.

1.2.2.3 Les listes (`list`)

La structure du langage R la plus souple et à la fois la plus riche est celle de la liste. Contrairement aux structures précédentes, les listes permettent de **regrouper dans une même structure des données de types différents** sans pour autant les altérer. De façon générale, chaque élément d'une liste peut ainsi être un vecteur, une matrice, un *array* ou même une liste. Voici un premier exemple :

```
> A <- list(TRUE,-1:3,matrix(1:4,nrow=2),c(1+2i,3),
+           "Une chaîne de caractères")
> A
[[1]]
[1] TRUE
[[2]]
[1] -1  0  1  2  3
[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[[4]]
[1] 1+2i 3+0i
[[5]]
[1] "Une chaîne de caractères"
> class(A)
[1] "list"
```

Dans une telle structure hétérogène en types de données, la manière d'ordonner les éléments est très souvent complètement arbitraire. C'est pourquoi il est possible de les nommer de façon explicite, ce qui rend la sortie plus conviviale. En voici un exemple :

```
> B <- list(une.matrice=matrix(1:4,nrow=2),
+           des.complexes=c(1+2i,3))
> B
$une.matrice
     [,1] [,2]
```

```
[1,]    1    3
[2,]    2    4
$des.complexes
[1] 1+2i 3+0i
> liste1 <- list(complexe=1+1i,logique=FALSE)
> liste2 <- list(chaine="J'apprends R",vecteur=1:2)
> C <- list("Ma première liste"=liste1,Ma.seconde.liste=liste2)
> C
$`Ma première liste`
$`Ma première liste`$complexe
[1] 1+1i
$`Ma première liste`$logique
[1] FALSE
$Ma.seconde.liste
$Ma.seconde.liste$chaine
[1] "J'apprends R"
$Ma.seconde.liste$vecteur
[1] 1 2
```

Renvoi

Cette façon de procéder (en nommant les éléments) permet de faciliter l'extraction d'éléments d'une liste (voir chapitre 3, page 113).

1.2.2.4 Le tableau individus × variables (`data.frame`)

Le tableau individus × variables est la structure par excellence en statistique. Cette notion est exprimée dans R par le *data.frame*. Conceptuellement, c'est une matrice dont les lignes correspondent aux individus et les colonnes aux variables (ou caractères) mesurées sur ces derniers. **Chaque colonne représente une variable particulière dont tous les éléments sont du même type**. Les colonnes de la matrice-données peuvent être nommées. Voici un exemple de création d'un *data.frame* :

```
> IMC <- data.frame(Sexe=c("H","F","H","F","H","F"),
+     Taille=c(1.83,1.76,1.82,1.60,1.90,1.66),
+     Poids=c(67,58,66,48,75,55),
+     row.names=c("Rémy","Lol","Pierre","Domi","Ben","Cécile"))
> IMC
       Sexe Taille Poids
Rémy      H   1.83    67
Lol       F   1.76    58
Pierre    H   1.82    66
Domi      F   1.60    48
Ben       H   1.90    75
Cécile    F   1.66    55
> is.data.frame(IMC)
```

```
[1] TRUE
> class(IMC)
[1] "data.frame"
> str(IMC)
'data.frame':        6 obs. of  3 variables:
 $ Sexe  : Factor w/ 2 levels "F","H": 2 1 2 1 2 1
 $ Taille: num  1.83 1.76 1.82 1.6 1.9 1.66
 $ Poids : num  67 58 66 48 75 55
```

Remarque

La fonction `str()` permet d'afficher la structure de chacune des colonnes d'un *data.frame*.

Expert

Les *data.frame* peuvent être vus comme des listes de vecteurs de même longueur. Cela est d'autant plus vrai que c'est de cette façon que R structure dans son fonctionnement interne un *data.frame*.

```
> is.list(IMC)
[1] TRUE
```

1.2.2.5 Les facteurs (`factor`) et les variables ordinales (`ordered`)

R permet d'organiser les chaînes de caractères de façon plus astucieuse au moyen de la fonction `factor()` :

```
> x <- factor(c("bleu","vert","bleu","rouge",
+                      "bleu","vert","vert"))
> x
[1] bleu  vert  bleu  rouge bleu  vert  vert
Levels: bleu rouge vert
> levels(x)
[1] "bleu"  "rouge" "vert"
> class(x)
[1] "factor"
```

Astuce

La fonction `cut()` permet de recoder une variable continue en facteurs.

```
> Poids <- c(55,63,83,57,75,90,73,67,58,84,87,79,48,52)
> cut(Poids,3)
```

```
 [1] (48,62] (62,76] (76,90] (48,62] (62,76] (76,90] (62,76]
 [8] (62,76] (48,62] (76,90] (76,90] (76,90] (48,62] (48,62]
Levels: (48,62] (62,76] (76,90]
```

Il est bien évidemment possible de mettre des facteurs dans un *data.frame*. R indique les différents niveaux (*levels*) du facteur. La fonction `factor()` est donc celle à utiliser pour stocker des variables qualitatives. Pour les variables ordinales, il est plutôt conseillé d'utiliser la fonction `ordered()` :

```
> z <- ordered(c("Petit","Grand","Moyen","Grand","Moyen",
+       "Petit","Petit"),levels=c("Petit","Moyen","Grand"))
> class(z)
[1] "ordered" "factor"
```

Le paramètre `levels` de la fonction `ordered()` permet de spécifier l'ordre des modalités de la variable.

Renvoi

Des exemples d'utilisation de ces deux fonctions sont présentés au chapitre 9, pages 369 et 371.

Astuce

La fonction `gl()` produit des facteurs en spécifiant leurs niveaux.

```
> gl(n = 2,k = 8,labels = c("Control", "Treat"))
 [1] Control Control Control Control Control Control Control
 [8] Control Treat   Treat   Treat   Treat   Treat   Treat
[15] Treat   Treat
Levels: Control Treat
```

Dans l'instruction ci-dessus, `n` et `k` sont deux entiers, le premier donnant le nombre de niveaux et le second le nombre de répétitions.

Expert

R permet d'organiser un vecteur de chaînes de caractères de façon plus efficace en prenant en compte les éléments qui se répètent. Cette approche permet d'obtenir une meilleure gestion de la mémoire. En effet, chacun des éléments du facteur ou de la variable ordinale est en fait codé sous la forme d'un entier.

1.2.2.6 Les dates

R permet de structurer les données constituées par des dates, au moyen de la fonction `as.Date()` par exemple.

```
> dates <- c("27/02/92","27/02/92","14/01/92","28/02/92","01/02/92")
> dates <- as.Date(dates, "%d/%m/%y")
> dates
[1] "1992-02-27" "1992-02-27" "1992-01-14" "1992-02-28"
[5] "1992-02-01"
> class(dates)
[1] "Date"
```

Nous reviendrons en détails sur les fonctions permettant de manipuler des dates au chapitre 3.

1.2.2.7 Les séries temporelles

Lorsque les données constituent des valeurs indicées par le temps, il peut être utile, au moyen de la fonction `ts()`, de les organiser dans une structure R qui reflètera l'aspect temporel de ces données.

```
> ts(1:10, frequency = 4, start = c(1959, 2)) # 2ème trimestre de
                                              # 1959.
     Qtr1 Qtr2 Qtr3 Qtr4
1959         1    2    3
1960    4    5    6    7
1961    8    9   10
```

Renvoi

Le lecteur pourra consulter avec profit le livre [4] dans lequel sont exposées les techniques de base pour la modélisation de séries temporelles, les fonctions R permettant ces modélisations, et l'application de ces fonctions sur plusieurs jeux de données réelles.

Récapitulatif

TABLE 1.2 – Les différentes structures de données en **R**.

Structure des données	**Instruction R**	**Description**
vecteur	`c()`	Suite d'éléments de même nature.
matrice	`matrix()`	Tableau à deux dimensions dont les éléments sont de même nature.
tableau multidimensionnel	`array()`	Plus général que la matrice ; tableau à plusieurs dimensions.
liste	`list()`	Suite de structures **R** de nature différente et quelconque.
tableau individus × variables	`data.frame()`	Tableau à deux dimensions dont les lignes sont des individus et les colonnes des variables (numériques ou facteurs). Les colonnes peuvent être de nature différente, mais doivent avoir la même longueur. Les éléments à l'intérieur d'une même colonne sont tous de la même nature.
facteur	`factor()`, `ordered()`	Vecteur de chaînes de caractères associé à une table des modalités.
dates	`as.Date()`	Vecteur de dates.
série temporelle	`ts()`	Série temporelle, contenant les valeurs d'une variable à certains temps.

Termes à retenir

`<-, ->` : flèches d'affectation dans une variable
`mode(), typeof()` : récupérer la nature d'un objet
`is.numeric()` : déterminer si un objet est de nature numérique
`TRUE, FALSE, is.logical()` : Vrai, Faux, déterminer si un objet est de nature booléenne
`is.character()` : déterminer si un objet est une chaîne de caractères
`NA, is.na()` : valeur manquante, déterminer s'il existe des valeurs manquantes
`class()` : déterminer la structure d'un objet
`c()` : créer une suite d'éléments de même nature
`matrix(), array()` : créer une matrice, un tableau multidimensionnel
`list()` : créer une liste, collection de structures différentes
`data.frame()` : créer un tableau individus × caractères
`factor()` : créer un facteur

Exercices

1.1- Que renvoie cette instruction : `1:3^2` ?

1.2- Que renvoie cette instruction : `(1:5)*2` ?

1.3- Que renvoient ces instructions : `var<-3` ? `Var*2` ?

1.4- Que renvoient ces instructions : `x<-2` ? `2x<-2*x` ?

1.5- Que renvoient ces instructions : `racine.de.quatre <- sqrt(4)` ? `racine.de.quatre` ?

1.6- Que renvoient ces instructions : `x<-1` ? `x< -1` ?

1.7- Que renvoie cette instruction : `Un chiffre pair <- 16` ?

1.8- Que renvoie cette instruction : `"Un chiffre pair" <- 16` ?

1.9- Que renvoie cette instruction : `"2x" <- 14` ?

1.10- Que renvoie cette instruction : `Un chiffre pair` ?

1.11- Complétez cette sortie de R, où deux symboles ont été enlevés :

```
> 2
+
[1] 6
```

1.12- Que renvoie cette instruction : `TRUE + T +FALSE*F + T*FALSE +F` ?

1.13- Quels sont les cinq types de données sous R ?

1.14- Donnez l'instruction R permettant d'obtenir l'affichage suivant :

```
> X
     [,1] [,2]  [,3]
```

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

1.15- Quelles sont les structures (classes) de données que **R** met à votre disposition ?

Fiche de TP

Étude sur l'indice de masse corporelle

Un échantillon de dossiers d'enfants a été saisi. Ce sont des enfants vus lors d'une visite en 1re section de maternelle en 1996-1997 dans des écoles de Bordeaux (Gironde, France). L'échantillon présenté ici est constitué de dix enfants âgés de 3 ou 4 ans.

Les données disponibles pour chaque enfant sont :
- le sexe : F pour fille et G pour garçons ;
- le fait que leur école soit située en ZEP (zone d'éducation prioritaire, c'est-à-dire réunissant plusieurs indices de précarité sociale) : O pour oui et N pour non ;
- l'âge en années et en mois à la date de la visite (deux variables, une pour le nombre d'années et une pour le nombre de mois) ;
- le poids en kg arrondi à 100 g près ;
- la taille en cm arrondie à 0,5 cm près.

Prénom	Érika	Célia	Éric	Ève	Paul	Jean	Adam	Louis	Jules	Léo
`Sexe`	F	F	G	F	G	G	G	G	G	G
`ZEP`	O	O	O	O	N	O	N	O	O	O
`Poids`	16	14	13.5	15.4	16.5	16	17	14.8	17	16.7
`An`	3	3	3	4	3	4	3	3	4	3
`Mois`	5	10	5	0	8	0	11	9	1	3
`Taille`	100.0	97.0	95.5	101.0	100.0	98.5	103.0	98.0	101.5	100.0

En statistique, il est très important de connaître le type des variables étudiées : qualitatives, ordinales ou quantitatives. **R** permet de spécifier explicitement ce type au moyen des fonctions de structure que nous avons vues dans ce chapitre.

Voilà quelques manipulations à effectuer avec R. Pensez à bien utiliser la stratégie de travail introduite en début de chapitre.

1.1- Choisissez la fonction R appropriée pour enregistrer les données de chacune des variables précédentes dans des vecteurs que vous nommerez `Individus`, `Poids`, `Taille` et `Sexe`.

1.2- Calculez la moyenne des variables pour lesquelles cela est possible.

1.3- Calculez l'IMC des individus et regroupez les valeurs obtenues dans un vecteur nommé `IMC` (faites attention aux unités).

1.4- Regroupez ces variables dans la structure R qui vous paraît la plus adaptée.

1.5- Utilisez l'aide en ligne de R afin d'obtenir des informations sur la fonction `plot()`.

1.6- Tracez le nuage de points du `Poids` en fonction de la `Taille`. Pensez à fournir un titre à votre graphique et à annoter vos axes.

Chapitre 2

Importation-exportation et production de données

Pré-requis et objectif

- Lecture du chapitre 1.
- Ce chapitre décrit les instructions à utiliser pour entrer des données sous R. Il présente les différentes possibilités offertes par R pour importer ou exporter des données, depuis et vers des logiciels aussi différents qu'Excel, que SPSS, Minitab, SAS ou Matlab, ainsi que l'interaction avec les bases de données (requêtes SQL). La lecture du très complet document `http//cran.r-project.org/doc/manuals/R-data.pdf` peut se révéler enrichissante (en anglais).

SECTION 2.1

Importer des données

2.1.1 Importer des données depuis un fichier texte ASCII

Soit vous disposez déjà d'un fichier texte au format ASCII qui contient vos données, soit vous pouvez les entrer vous-même dans un fichier à l'aide d'un éditeur de texte comme Wordpad sous Microsoft Windows ou bien Emacs sous Linux.

Remarque

Entrer à la main des données dans un fichier à l'aide d'un éditeur de texte est valable pour un nombre peu important de valeurs. Si vous avez un grand nombre de données, il vaut mieux utiliser un tableur (voir la section suivante).

Il existe trois fonctions R principales à utiliser pour importer des données depuis un fichier texte, présentées dans le tableau suivant.

TABLE 2.1 – Fonctions d'importation de données.

Nom de la fonction	Description
`read.table()`	À privilégier pour des jeux de données organisés sous la forme de tableaux, comme cela est souvent le cas en statistique.
`read.ftable()`	Permet de lire des tableaux de contingence.
`scan()`	Beaucoup plus flexible et puissante. Elle devra être utilisée dans tous les autres cas.

2.1.1.1 Lecture de données avec `read.table()`

L'instruction R suivante va lire les données présentes dans un fichier (à sélectionner dans une fenêtre de dialogue) et les rapatrier dans R sous la forme d'un *data.frame* que nous avons choisi de nommer `donnees`.

```
donnees <- read.table(file=file.choose(),header=TRUE,sep="\t",
                      dec=".",row.names=1)
```

La fonction `read.table()` comprend de nombreux paramètres dont les plus utilisés sont décrits dans le tableau suivant.

TABLE 2.2 – Paramètres principaux de `read.table()`.

Nom du paramètre	Description
`file=chemin/vers/fichier`	Emplacement et nom du fichier à lire.
`header=TRUE`	(*header* = entête). Valeur logique indiquant si le fichier contient le nom des variables sur la première ligne.
`sep="\t"`	Les valeurs sur chaque ligne sont séparées par ce caractère (`="\t"`=TABULATION ; `""` = un ou plusieurs espaces ; `","` = , ; etc.).
`dec="."`	Séparateur décimal pour les nombres (`"."` ou `","`).
`row.names=1`	La première colonne du fichier contient le nom des individus. Si ce n'est pas le cas, il suffit d'omettre ce paramètre.

Lors de l'utilisation de la fonction `read.table()`, il est nécessaire de spécifier la valeur du paramètre `file` qui doit contenir, dans une chaîne de caractères, le nom du fichier à lire ainsi qu'optionnellement son chemin d'accès complet. Vous avez pu constater que nous avons utilisé la fonction `file.choose()`

qui ouvre une fenêtre de dialogue permettant de sélectionner un fichier et de renvoyer la chaîne de caractères requise. Il s'agit en fait d'un moyen commode pour récupérer le chemin vers le fichier, mais il est également possible de le spécifier de façon explicite :

```
donnees <- read.table(file="C:/MonDossier/mesdonnees.txt")
```

Attention

Notez la spécification des chemins de fichiers au moyen de *slashes* (/). Cette façon de procéder est issue de l'environnement UNIX. L'utilisation d'anti-*slashes* (\) comme sous Microsoft ne fonctionne pas sous R, sauf à doubler tous les anti-*slashes* (\\).

Une dernière option consistera à utiliser la fonction `setwd()` pour changer le répertoire courant (équivalant à utiliser le menu « Fichier/Changer le répertoire courant ») puis à donner au paramètre `file` uniquement le nom du fichier sans le chemin d'accès.

```
setwd("C:/MonDossier")
mon.fichier <- "mesdonnees.txt"
donnees <- read.table(file=mon.fichier)
```

Vos données sont maintenant accessibles dans la console de R puisqu'elles sont stockées dans l'objet que vous avez choisi de nommer `donnees`. Vous pouvez les « visualiser » en tapant `donnees` ou encore en tapant `head(donnees)` ou `tail(donnees)` pour n'afficher que le début ou la fin du jeu de données.

Astuce

- La fonction `attach()` (voir chapitre 7) permet d'avoir accès aux variables (colonnes) du *data.frame* directement en tapant leur nom tel qu'il est écrit sur la première ligne du fichier au format ASCII (si c'est le cas bien entendu).

  ```
  attach(donnees)
  ```

- Si votre fichier contient des lignes complètement vides ou bien des lignes non terminées, utilisez les deux paramètres (et leur valeur d'appel) `fill=TRUE` et `blank.lines.skip=FALSE`.

Prise en main

Commencez par créer un dossier nommé `DossierData`. Maintenant, téléchargez le fichier `http://www.biostatisticien.eu/springeR/Intima_Media.txt` et sauvez-le dans le dossier `DossierData`.

Utilisez la fonction `readLines()` pour visualiser le début du fichier de données, vous faire une idée de la façon dont il est structuré, et ainsi déterminer quels paramètres de la fonction `read.table()` il vous faudra utiliser.

```
setwd("chemin/vers/DossierData/")  # Remplacez chemin/vers
                                   # par votre chemin.
readLines("Intima_Media.txt",n=5)
```

Vous voyez apparaître la sortie suivante :

```
[1] "SEXE AGE taille poids tabac paqan SPORT mesure alcool"
[2] "1 33 170 70 1 1 0 0,52 1"
[3] "2 33 177 67 2 20 0 0,42 1"
[4] "2 53 164 63 1 30 0 0,65 0"
[5] "2 42 169 76 1 26 1 0,48 1"
```

Vous remarquez que la première ligne contient le nom des variables, que les champs sont séparés par des simples espaces et que le séparateur décimal est une virgule. Par conséquent, vous devez utiliser les paramètres `header=TRUE`, `sep=" "` et `dec=","`.

```
mesdonnees <- read.table("Intima_Media.txt",sep=" ",
                         header=TRUE,dec=",")
mesdonnees        # Afin d'afficher le contenu
                  # de mesdonnees.
head(mesdonnees)  # Affiche uniquement les premières
                  # lignes du data.frame.
```

Vous noterez la présence de valeurs manquantes symbolisées par `NA`.

On peut maintenant vérifier le type de structure de l'objet `mesdonnees` ainsi que les types de chacune de ses colonnes :

```
class(mesdonnees)
str(mesdonnees)
```

Utilisons la fonction `attach()` afin de pouvoir accéder aux variables du tableau.

```
attach(mesdonnees)
```

Il vous est désormais possible de faire des calculs sur les variables dont les noms sont donnés par `names(mesdonnees)`. Par exemple

```
mean(AGE)    # Moyenne des âges.
var(taille)  # Variance des tailles.
```

Vous noterez l'importance de respecter la casse (minuscules/majuscules).

La fonction `read.table()` contient un très grand nombre de paramètres. Mais puisque de nombreux jeux de données ont un format standard, quelques fonctions sont disponibles pour les lire. Ces fonctions consistent en fait à appeler `read.table()` avec les valeurs de certains paramètres fixés par défaut.

Ainsi, si vous avez un fichier d'extension `.csv` (pour *comma separated values*) créé par exemple depuis le tableur d'OpenOffice (ou de LibreOffice), vous pouvez aussi utiliser la fonction suivante :

```
read.csv(file.choose())  # Pour lire des données séparées par
                         # des virgules
                         # (avec le . pour séparateur décimal).
read.csv2(file.choose()) # Pour lire des données séparées par
                         # des points-virgules
                         # (avec la , pour séparateur décimal).
```

Pour lire des données séparées par des tabulations, on utilisera plutôt :

```
read.delim(file.choose()) # (avec le . pour séparateur décimal).
read.delim2(file.choose())# (avec le , pour séparateur décimal).
```

2.1.1.2 Lecture de données avec `read.ftable()`

Il arrive parfois que l'on ne dispose pas de toutes les données individuelles, mais uniquement d'un résumé présenté sous la forme d'un tableau de contingence. Dans ce cas, il faut utiliser la fonction d'importation `read.ftable()`.

Supposons par exemple que le contenu du fichier Intima_ftable.txt se présente sous la forme suivante :

```
            "alcool" "ne boit pas" "boit occasionnellement" "boit régulièrement"
"SEXE" "tabac"
"H"    "ne fume pas"            6                     19                     7
       "a arrêté de fumer"      0                      9                     0
       "fume"                   1                      6                     5
"F"    "ne fume pas"           12                     26                     2
       "a arrêté de fumer"      3                      5                     1
       "fume"                   1                      6                     1
```

On utilisera les instructions ci-dessous pour lire et afficher ces données dans R :

```
Intima.table <- read.ftable("Intima_ftable.txt",row.var.names
                =c("SEXE","tabac"),col.vars=list("alcool"=
                 c("ne boit pas","boit occasionnellement",
                 "boit régulièrement")))
ftable(Intima.table)
```

Le résultat sera alors le suivant :

```
                  alcool ne boit pas boit occasionnellement boit régulièrement
SEXE tabac
H    ne fume pas                   6                     19                   7
     a arrêté de fumer             0                      9                   0
     fume                          1                      6                   5
F    ne fume pas                  12                     26                   2
     a arrêté de fumer             3                      5                   1
     fume                          1                      6                   1
```

Renvoi

Nous présenterons l'analyse descriptive de ce type de données dans le chapitre 9, pages 374, 381 et 401. Notons qu'il est aussi possible d'effectuer des tests statistiques classiques sur des tableaux de contingence tels que le khi-deux d'indépendance présenté au chapitre 11, page 468. Vous pouvez également consulter l'article suivant `http://www.jstatsoft.org/v17/i03/paper` qui présente plusieurs outils pour analyser des données de ce type.

2.1.1.3 Lecture de données avec la fonction `scan()`

La fonction `scan()` possède de très nombreux paramètres et elle est à privilégier lorsque les données ne sont pas organisées sous la forme d'un tableau rectangulaire. Il sera utile de consulter la documentation à son sujet (`help(scan)`).

Par exemple, supposons que votre fichier de données, nommé Intima_Media2.txt, contienne les lignes suivantes :

```
Description du fichier:
-----------------------

Les données individuelles sont enregistrées les unes à la
suite des autres pour les neuf variables suivantes:
SEXE AGE taille poids tabac paqan SPORT mesure alcool

Données:
--------
1 33 170 70 1 1 0 0,52 1 2 33 177 67 2 20 0 0,42 1
2 53 164 63 1 30 0 0,65 0 2 42 169
76 1 26 1 0,48 1
```

Les commandes que nous conseillons d'utiliser pour lire ce fichier sont indiquées ci-dessous. Le paramètre `skip=`*n* permet d'omettre la lecture des *n* premières lignes du fichier.

```
# Lecture du nom des variables:
nom.variable<-scan("Intima_Media2.txt",skip=5,nlines=1,what="")
# Lecture des données:
donnee <- scan("Intima_Media2.txt",skip=9,dec=",")
matable <- as.data.frame(matrix(donnee,ncol=9,byrow=TRUE))
colnames(matable) <- nom.variable
```

Voici le résultat obtenu :

```
  SEXE AGE taille poids tabac paqan SPORT mesure alcool
1    1  33    170    70     1     1     0   0.52      1
2    2  33    177    67     2    20     0   0.42      1
3    2  53    164    63     1    30     0   0.65      0
4    2  42    169    76     1    26     1   0.48      1
```

Astuce

Notez qu'il est possible d'utiliser les fonctions `read.table()` et `scan()` pour aller lire directement des fichiers ASCII sur l'Internet.

```
read.table("http://www.biostatisticien.eu/springeR/
                    temperature.dat")
```

2.1.2 Importer des données depuis Excel ou le tableur d'OpenOffice

2.1.2.1 Utiliser le copier-coller

On sélectionne à l'aide de la souris (dans le tableur) la plage de données que l'on souhaite incorporer dans R. Une fois que les données ont été sélectionnées, on les copie (à partir du menu Edit ou en utilisant la combinaison de touches `CTRL+C` sous Windows ou `COMMAND+C` sous Mac) dans le presse-papiers.

Il suffit ensuite de taper l'instruction suivante dans la console de R pour que les données y soient transférées depuis le presse-papiers.

```
x <- read.table(file("clipboard"),sep="\t",header=TRUE,dec=",")
```

Astuce

L'instruction `fix(x)` ouvre un mini-tableur dans R permettant de visualiser et de modifier les données présentes dans `x`. Elle est plus utile que

l'instruction `edit()` qui n'est pas prévue pour opérer des modifications. Dans le même ordre d'idées, la fonction `View()` permet uniquement d'afficher (dans un mini tableur) mais sans pouvoir les modifier les données présentes dans `x`.

Attention

Prenez garde au fait qu'Excel pourrait contenir des formules ou autres caractères cachés derrière la plage de valeurs que vous souhaitez copier. Une façon de s'affranchir d'éventuels problèmes reliés à ce fait consiste à d'abord copier, puis à effectuer un collage spécial de la plage de ces valeurs sur une feuille vierge du tableur. Vous pourrez ensuite utiliser la fonction `read.table()` comme cela est indiqué ci-dessus.

2.1.2.2 Passer par un fichier ASCII intermédiaire

Il vous faut enregistrer votre fichier dans un format ASCII puis vous référer à la section précédente.

- Dans Excel, allez dans `Fichier / Enregistrer sous ...` et choisissez `Type de fichier: Texte (séparateur: tabulation) (*.txt) (*.txt)` puis sauvez.

- Dans OpenOffice, allez dans `Fichier / Enregistrer sous ...` et choisissez `Type de Fichier: Texte CSV (.csv;.txt)` puis sauvez.
  Dans la fenêtre suivante, choisissez successivement :
  – séparateur de champs : `Tab`
  – séparateur de texte : `"`
  puis cliquez sur OK.

2.1.2.3 Utiliser des *packages* spécialisés

Il existe quelques *packages* permettant de lire directement des fichiers `*.xls` depuis R. On peut notamment citer la fonction `read.xls()` du *package* `gdata` qui fonctionne très bien dès lors qu'une version fonctionnelle de PERL (logiciel gratuit, disponible par exemple via l'installation du fichier `http://www.biostatisticien.eu/springeR/Rtools29.exe`) est présente sur votre ordinateur. Il est aussi possible d'utiliser le *package* `xlsReadWrite`.

2.1.3 Importer des données depuis SPSS, Minitab, SAS ou Matlab

Le tableau suivant donne les *packages* et les fonctions R à utiliser pour importer des données depuis quelques logiciels commerciaux usuels.

TABLE 2.3 – *Packages* et fonctions R d'importation de données depuis quelques logiciels usuels.

Logiciel	*Package*	Fonction R	Extension du fichier	Format du résultat
SPSS	`foreign`	`read.spss()`	`*.sav`	`list`
Minitab	`foreign`	`read.mtp()`	`*.mtp`	`list`
SAS	`foreign`	`read.xport()`	`*.xpt`	`data.frame`
Matlab	`R.matlab`	`readMat()`	`*.mat`	`list`

La fonction `lookup.xport()` renvoie de l'information (sous la forme d'une liste) sur la librairie SAS d'un fichier SAS XPORT d'extension `*.xpt`.

Attention

Notez tout d'abord que sous l'environnement Windows le *package* `foreign` est déjà pré-installé (mais pas chargé) dans R et qu'il n'est pas possible d'en installer une autre version depuis le CRAN (qui ne dispose que des versions Linux et Mac).
Notez maintenant les points suivants. La fonction `read.spss()` peut nécessiter l'utilisation du paramètre `reencode="utf8"` sous Linux. La fonction `read.mtp()` fonctionne sur des fichiers contenant uniquement des données numériques. La fonction `read.xport()` ne permet actuellement pas de lire des fichiers directement depuis l'Internet.

2.1.4 Les gros fichiers de données

Le logiciel R est correctement outillé pour la lecture de gros jeux de données. Pour cela, il faut indiquer de façon explicite le type de chacune des colonnes. Sinon, R devra lire tout le fichier pour vérifier que les colonnes numériques sont bien numériques (le début du fichier pourrait par exemple contenir des nombres, puis plus loin des chaînes de caractères). L'exemple suivant, issu de données génomiques réputées pour être volumineuses, permet d'illustrer ce point. Il vous faut d'abord télécharger le fichier `http://www.biostatisticien.eu/springeR/dbsnp123.dat` (50 Mo) sur votre disque dur, puis essayer les instructions ci-après.

Attention

Notez que si vous essayez de taper les commandes ci-dessous, votre session R risque d'être « gelée » pour quelques minutes.

```
tmps <- Sys.time() # Récupère l'heure courante.
dbsnp <- read.table("dbsnp123.dat")
Sys.time()-tmps
Time difference of 5.063645 mins

tmps <- Sys.time()
dbsnp<-read.table("dbsnp123.dat",colClasses=rep("character",3))
Sys.time()-tmps
Time difference of 13.75810 secs
```

Il convient donc de noter qu'une utilisation judicieuse de R aide à gérer de très gros jeux de données de façon relativement rapide. La limite est essentiellement la quantité de RAM disponible. Par ailleurs, la fonction `scan()` donne des temps d'exécution du même ordre de grandeur.

Les gros jeux de données sont aussi parfois stockés dans un format binaire. Dans ce cas, la fonction `readBin()` permet de les lire. Nous en verrons un exemple dans la partie Travaux pratiques du chapitre 5.

Renvoi

Si R affiche un message indiquant un défaut de mémoire, vous pourriez consulter avec profit la section 7.8.

Expert

Si la fonction `scan()` est correctement utilisée, la lecture d'un fichier texte est très rapide (autant qu'avec le logiciel SAS par exemple).

Si votre fichier est vraiment gros, vous devriez considérer la possibilité de stocker vos données dans une base de données (par exemple MySQL) et d'y accéder par segments. Voir la section 2.4 pour plus de détails.

Notez également l'existence des *packages* `R.huge` et `filehash`, ce dernier étant plus général que `R.huge`, permettant de gérer des gros fichiers de données. Dans le *package* `filehash`, la limitation est la taille de l'espace disponible sur le disque dur.

SECTION 2.2

Exporter des données

2.2.1 Exporter des données vers un fichier texte ASCII

Il faut utiliser la fonction `write.table()`.

Si vous avez dans R un *data.frame* nommé `donnees` qui contient les données que vous voulez sauvegarder dans un fichier texte, utilisez l'instruction :

```
write.table(donnees, file = "mon-fichier.txt", sep = "\t")
```

Remarque

Notez qu'il existe aussi une fonction `write()` qui s'utilise sur des objets du type vecteur ou matrice et qui possède un paramètre intéressant (`ncolumns`) permettant de spécifier le nombre de colonnes dans le fichier résultant. Prenez garde toutefois au fait que votre fichier contiendra une version transposée de la matrice ou du vecteur à écrire.

2.2.2 Exporter des données vers Excel ou OpenOffice Calc

Tapez par exemple dans la console de R :

```
X <- data.frame(Poids=c(80,90,75),Taille=c(182,190,160))
write.table(X,file("clipboard"),sep="\t",dec=",",row.names=FALSE)
```

Les données ont ainsi été copiées dans le presse-papiers (*clipboard*). Il vous suffira alors de les coller dans votre tableur, par exemple au moyen de la combinaison de touches `CTRL+V`.

Vous pouvez aussi utiliser le *package* `xlsReadWrite` (seulement pour Microsoft Windows).

SECTION 2.3

Création de données

2.3.1 Entrer des données jouets

Nous allons voir dans cette section comment vous pouvez créer rapidement quelques données. Cela peut se révéler utile pour expérimenter les diverses fonctions de R sur de petites séries de données.

Nous décrivons maintenant les principales fonctions que sont `c()`, `seq()`, `:()` et `rep()`.

• La fonction `c()` permet de créer un vecteur par concaténation de ses paramètres d'entrée.

```
> c(1,5,8,2.3)
[1] 1.0 5.0 8.0 2.3
```

• La fonction `seq()` permet de générer une suite de valeurs, sous la forme d'un vecteur.

```
> seq(from=4,to=5)
[1] 4 5
> seq(from=4,to=5,by=0.1)
 [1] 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0
> seq(from=4,to=5,length=8)
[1] 4.000000 4.142857 4.285714 4.428571 4.571429 4.714286
[7] 4.857143 5.000000
```

Remarque

Les fonctions `c()` et `rep()` peuvent aussi être utilisées pour créer des vecteurs de chaînes de caractères.

• La fonction `":"()` permet de générer une suite d'entiers. Notez qu'il s'agit bien d'une fonction R puisque la syntaxe classique fonctionne :

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

Cependant, la syntaxe suivante est plus commode :

```
> 1:12
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

• La fonction `rep()` duplique les valeurs de son premier paramètre d'entrée, de plusieurs façons astucieuses. Nous laissons le soin au lecteur de s'assurer qu'il comprend bien toutes ces instructions.

```
> rep(1,4)
[1] 1 1 1 1
> rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4
> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
> rep(1:4, c(2,1,2,3))
[1] 1 1 2 3 3 4 4 4
> rep(1:4, each = 2, len = 4)
[1] 1 1 2 2
```

```
> rep(1:4, each = 2, len = 10)
 [1] 1 1 2 2 3 3 4 4 1 1
> rep(1:4, each = 2, times = 3)
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

2.3.2 Générer des données pseudo-aléatoires

La fonction `runif()` génère une suite de nombres répartis (uniformément) au hasard.

```
> runif(5)
[1] 0.42880942 0.05190332 0.26417767 0.39879073 0.83613414
> runif(5,min=2,max=7)
[1] 6.323606 5.076762 5.875549 3.777843 4.029250
```

La fonction `rnorm()` génère une suite de nombres provenant d'une variable aléatoire suivant une loi normale.

```
> rnorm(5)
[1]  0.5436154 -0.7076248 -0.3694458 -1.3219757  1.2805975
> rnorm(5,mean=2,sd=3)
[1]  4.0022452  7.0752649  2.0037842 -0.2273839  3.8290533
```

La fonction `sample()` permet d'effectuer un tirage aléatoire avec remise dans une urne.

```
> urne <- 0:9}
+ sample (urne, 20, replace = TRUE)
```

Renvoi

Nous verrons de nombreuses autres fonctions de ce type dans le chapitre 10, page 436.

2.3.3 Entrer des données issues d'un support papier

- **Créer un vecteur en utilisant la fonction `scan()`**

Cette fonction, plus conviviale que `c()` dans ce contexte, permet de rentrer rapidement des données à la volée.

```
> z <- scan() # R attend que vous entriez des données.
1: 4.2
2: 5.6
3: 8.9
4: 1
5: 2.3
6:      # Un appui sur la touche ENTRÉE à vide
        # provoque l'arrêt de la procédure.
```

```
Read 5 items
> z
[1] 4.2 5.6 8.9 1.0 2.3
```

- **Créer plusieurs vecteurs de longueurs différentes**

On peut utiliser la fonction `data.entry()`. Cette dernière ne renvoie rien mais stocke dans l'environnement de travail les variables entrées à la main dans le mini-tableur qui s'affiche.

```
# L'instruction suivante (expliquée plus loin) permet d'effacer
# tous les objets de la session.
rm(list=ls())
data.entry("")
```

Vous pouvez alors modifier les noms des variables (colonnes) et entrer des données. Les colonnes peuvent ici contenir un nombre différent d'observations. Puis, si vous quittez le mini-tableur et tapez l'instruction `ls()`, vous verrez alors s'afficher les variables que vous venez de créer.

Mac

Le fonctionnement de cette fonction peut différer suivant le système d'exploitation utilisé.

- **Créer un tableau individus × variables**

Si l'on veut entrer des données directement dans le mini-tableur de R (un peu comme sous Excel), il suffira d'utiliser la fonction `de()` (*data entry*) comme cela est indiqué dans l'instruction ci-dessous.

```
X <- as.data.frame(de(""))
```

Attention

Pensez à modifier le nom des variables et aussi le type des colonnes (`numeric` ou `character`) en cliquant sur les cases de la première ligne du tableau (celle contenant le nom des variables). Lorsque vous avez terminé d'entrer vos données, il vous faut fermer cette fenêtre pour avoir de nouveau accès à la console de R.

Notez que si vous souhaitez apporter de petites modifications à votre tableau de données `X`, il vous suffit d'utiliser la fonction `fix()` sur celui-ci : `fix(X)`.

Astuce

L'instruction (optionnelle) suivante permet de nommer les lignes de `X` :

```
rownames(X) <- paste("ind",1:nrow(X),sep="")
```

Des noms d'individus apparaîtront ainsi dans la première colonne du mini-tableur (celle nommée `row.names`).

SECTION 2.4

† Lecture/écriture dans les bases de données

R peut communiquer avec la plupart des systèmes de gestion de bases de données (SGBD). Dans cette section, nous effectuons un survol rapide des opérations les plus courantes lors de l'utilisation du SGBD MySQL.

EasyPHP est un environnement comprenant deux serveurs (un serveur web Apache et un serveur de bases de données MySQL), un interpréteur de script (PHP), ainsi qu'une administration SQL nommée phpMyAdmin. Téléchargez, installez puis exécutez la dernière version d'EasyPHP (`http://www.easyphp.org`).

Remarque

Il vous faudra peut-être configurer votre pare-feu afin qu'il permette l'utilisation des services lancés par EasyPHP (mysqld et Apache).

2.4.1 Créer une base de données et une table

Après avoir lancé EasyPHP, faites un clic droit sur son icone  présent dans la partie droite de la barre des tâches (cliquez éventuellement sur le petit triangle blanc `Afficher les icônes cachées`, situé à droite de la barre des tâches), et sélectionnez l'entrée **Administration**. Votre navigateur devrait alors s'ouvrir (si cela ne fonctionne pas, essayez d'utiliser le navigateur `firefox` et de configurer Apache par un clic droit sur l'icône d'EasyPHP, puis `Configuration` : `Listen 127.0.0.1:80`). Sur la page qui s'affiche, cliquez sur le bouton **ouvrir** de la section **MODULES** afin d'accéder à la page d'administration de phpMyAdmin. Cliquez ensuite, dans la nouvelle page, sur l'onglet **Bases de données**, et créez une nouvelle base de données nommée `IMC`. Cliquez ensuite sur l'icone **Nouvelle table** qui apparaît dans le panneau de gauche.

Entrez comme nom de la table : `matable`.

Définissez ensuite quatre champs pour votre table (un par ligne), que vous remplirez de la façon suivante :

- **Nom**=`Prenom`, **Type**=`VARCHAR` et **Taille/Valeurs**=`20` ;
- **Nom**=`Poids`, **Type**=`FLOAT` et **Taille/Valeurs**=`3` ;
- **Nom**=`Taille`, **Type**=`FLOAT` et **Taille/Valeurs**=`3` ;
- **Nom**=`IMC`, **Type**=`FLOAT` et **Taille/Valeurs**=`5`.

Cliquez sur **Sauver**.

2.4.2 Créer une source de données compatible avec MySQL

Les fonctions `odbcConnect()`, `sqlQuery()` et `odbcClose()` du *package* `RODBC` permettent de gérer des bases de données sous différents systèmes (PostgreSql, MySQL, etc.) par l'intermédiaire de R grâce à un lien ODBC (*Open DataBase Connectivity*) préalablement créé avec une base de données déjà existante. Voyons comment créer une source de données ODBC sous Microsoft Windows, compatible avec MySQL.

Commencez par installer MyODBC (MySQL Connector/ODBC) (`http://dev.mysql.com/downloads/connector/odbc`), puis lancez le fichier `C:\Windows\System32\odbcad32.exe` permettant d'afficher la fenêtre Sources de données (ODBC).

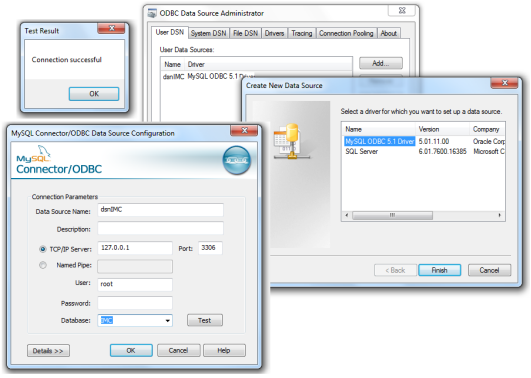
Astuce

Pour un ordinateur 64 bits, vous pourriez avoir à utiliser le fichier `C:\Windows\Syswow64\odbcad32.exe`.

Cliquez sur **Ajouter** et sélectionnez l'entrée **MySQL ODBC 5.1 Driver**. Dans la fenêtre qui s'ouvre alors, renseignez les champs suivants :

Data Source Name : `dsnIMC`
TCP/IP Server : `127.0.0.1` **Port :** `3306`
User : `root`
Database : `IMC`

Cliquez ensuite sur le bouton **Test**. Le message **Connection successful** devrait apparaître si tout s'est bien déroulé. Cliquez alors sur le bouton **OK** (deux fois) pour fermer les boîtes de dialogue ouvertes.



Nous venons ainsi de configurer un lien ODBC qui permettra à **R** de communiquer avec MySQL.

Linux

Vérifiez que le fichier `/etc/odbcinst.ini` contient les références aux pilotes de la base de données MySQL. Il faut aussi modifier (en mode super-utilisateur) le fichier `/etc/odbc.ini` pour qu'il contienne les lignes ci-dessous.

```
[dsnIMC]        # nom de la source de données.
Description =
Driver = MySQL
Server = localhost
Database = IMC
Port = 3306
```

2.4.3 Écrire dans une table

Nous allons écrire des informations dans la table `matable` de la base `IMC`. Les instructions suivantes permettent l'ajout du poids, de la taille, et de la valeur de l'IMC (fixée à 0 temporairement) pour un sujet nommé Pierre.

```
> require("RODBC")
> Connex <- odbcConnect(dsn="dsnIMC",uid="root",pwd="")
> requete <- "INSERT INTO matable VALUES ('Pierre',72,182,0)"
> resultat <- sqlQuery(Connex,requete)
> odbcClose(Connex)
```

Voyons maintenant un exemple d'insertion multiple.

```
> Prenoms <- c("Benoit","Rémy")
> Poids <- c(70,75)
```

```
> Taille <- c(190,184)
> IMC <- round(Poids/(Taille/100)^2,3)
> mat <- cbind(Prenoms,Poids,Taille,IMC)
> insertmult <- function(vect)
+     paste("(",toString(c(encodeString(vect[1],quote="'"),
+       vect[-1])),")",sep="")
> ainserer <- toString(apply(mat,1,insertmult))
> ainserer
[1] "('Benoit', 70, 190, 19.391), ('Rémy', 75, 184, 22.153)"
> require("RODBC")
> Connex <- odbcConnect(dsn="dsnIMC",uid="root",pwd="")
> requete <- paste("INSERT INTO matable (Prenom,Poids,Taille,IMC)
+                  VALUES ",ainserer,sep="")
> resultat <- sqlQuery(Connex,requete)
> odbcClose(Connex)
```

Astuce

Vous pouvez retourner dans phpMyAdmin pour vérifier que la table `matable` a bien été modifiée (onglet **Afficher**).

2.4.4 Lire dans une table

Afin de lire, depuis R, les informations présentes dans la table `matable`, vous pouvez utiliser les instructions suivantes.

```
> require("RODBC")
> Connex <- odbcConnect(dsn="dsnIMC",uid="root",pwd="")
> requete <- "SELECT * FROM matable"
> data <- sqlQuery(Connex,requete)
> odbcClose(Connex)
> data
  Prenom Poids Taille    IMC
1 Pierre    72    182  0.000
2 Benoit    70    190 19.391
3   Rémy    75    184 22.153
```

Termes à retenir

`read.table()` : lire un fichier de données rectangulaires
`scan()` : lire des données ligne par ligne
`read.ftable()` : lire des tables de contingence
`ftable()` : affichage formaté de tables de contingence
`readLines()` : lire et afficher quelques lignes d'un fichier
`file.choose()` : ouverture d'une boîte de dialogue de sélection de fichier
`file`, `header`, `sep`, `dec`, `row.names`, `skip` : paramètres principaux de `read.table()`
`read.spss()`, `read.mtp()`, `read.xport()`, `readMat()` : importation depuis des logiciels externes
`write.table()` : écrire un fichier de données
`file("clipboard")` : copier ou coller dans le presse-papiers
`c()` : créer une suite d'éléments de même nature
`seq()` : créer une séquence de nombres ou de chaînes de caractères
`rep()` : répète les valeurs de son premier paramètre d'entrée
`de()`, `data.entry()` : entrer des données au moyen d'un mini-tableur
`fix()` : permet de modifier un *data.frame* ou une matrice au moyen d'un mini-tableur

Exercices

2.1- Quelles sont les trois fonctions R principales à utiliser pour importer des données depuis un fichier texte au format ASCII ?

2.2- L'une des fonctions usuelles de lecture de données possède les paramètres suivants : `header`, `sep`, `dec`, `row.names`, `skip`, `nrows`. Expliquez à quoi ils servent. Donner un exemple de la valeur que peut prendre chacun de ces paramètres.

2.3- À quoi sert la fonction `readLines()` ?

2.4- À quoi sert la fonction `fix()` ?

2.5- Décrivez les particularités des fonctions `read.csv()`, `read.csv2()`, `read.delim()` et `read.delim2()`.

2.6- À quoi sert la fonction `read.ftable()` ?

2.7- En quoi les fonctions `scan()` et `read.table()` se différencient-elles ?

2.8- Expliquez en détail comment vous feriez pour importer des données se trouvant dans une feuille de classeur d'Excel.

2.9- Quel *package* contient plusieurs fonctions permettant d'importer des données depuis des logiciels commerciaux de statistique ?

2.10- Lors de la lecture de gros fichiers de données, quel paramètre de la fonction `read.table()` permet d'augmenter considérablement la vitesse de lecture ?

2.11- Quelle est la fonction R à utiliser pour écrire dans un fichier le jeu de données contenu dans un *data.frame* ? Quelle autre fonction connaissez-vous ?

2.12- Citez les quatre fonctions de base permettant de fabriquer des vecteurs.

2.13- Indiquez comment utiliser la fonction `seq()` pour obtenir le vecteur suivant :
`[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0`

2.14- Donnez l'instruction R la plus concise permettant d'obtenir le vecteur suivant :

```
1 1 2 2 3 3
```

2.15- Donnez l'instruction R la plus concise permettant d'obtenir le vecteur suivant :

```
1 2 3 1 2 3
```

2.16- Donnez deux fonctions R permettant d'entrer des données à la main dans le mini-tableur de R.

Fiche de TP

Lecture de différents jeux de données

A- Rentrer des données issues d'un format papier

• Boutons de fièvre

Cinq traitements contre les boutons de fièvre, dont un placebo, ont été administrés par tirage au sort à trente patients (six patients par groupe de traitement). Le délai (en jours) entre l'apparition des boutons et la cicatrisation complète a été recueilli chez chaque patient.

Traitements				
trt1 (placebo)	trt2	trt3	trt4	trt5
5	4	6	7	9
8	6	4	4	3
7	6	4	6	5
7	3	5	6	7
10	5	4	3	7
8	6	3	5	6

La question que l'on se pose est de savoir s'il existe une différence entre les traitements. Il s'agit donc ici de comparer les moyennes des délais de cicatrisation observés dans cinq échantillons indépendants (groupes de traitement). L'analyse statistique adéquate s'appelle l'ANOVA ; elle sera présentée au chapitre 13. Nous allons voir ici comment entrer ces données dans R uniquement pour calculer la moyenne de l'échantillon de chaque traitement.

2.1- Entrez directement les données dans R à l'aide de la fonction `de()`.

2.2- Utilisez la fonction `attach()` puis la fonction `mean()` pour calculer la moyenne de chacun des traitements.

2.3- Calculez simultanément les moyennes des traitements au moyen de la fonction `colMeans()`.

2.4- Enregistrez votre *data.frame* dans un fichier nommé boutons.txt en utilisant la fonction `write.table()`.

2.5- Vérifiez que tout s'est bien passé en ouvrant le fichier au moyen d'un éditeur de texte.

2.6- Utilisez la fonction `rm()` pour effacer tous les objets R que vous venez de créer dans votre environnement de travail.

2.7- Importez le fichier boutons.txt en utilisant `read.table()` et affichez-le.

• Facteurs de risque de l'athérosclérose

Lors d'une étude sur les facteurs de risque de l'athérosclérose, des données ont été recueillies et résumées dans le tableau de contingence suivant :

		alcool ne boit pas	boit occasionnellement	boit régulièrement
SEXE	tabac			
H	ne fume pas	6	19	7
	a arrêté de fumer	0	9	0
	fume	1	6	5
F	ne fume pas	12	26	2
	a arrêté de fumer	3	5	1
	fume	1	6	1

Il peut être intéressant de savoir s'il y a une dépendance entre les habitudes tabagiques et alcooliques suivant le sexe. Pour entrer ce type de données dans R, il faut suivre plusieurs étapes.

2.1- Utilisez la fonction `scan()` afin d'obtenir une matrice `X` de taille 6×3 qui contiendra uniquement les données.

2.2- Utilisez l'instruction `class(X) <- "ftable"` pour spécifier qu'il va s'agir d'une table de contingence.

2.3- Tapez les deux instructions :

```
attributes(X)$col.vars <- list(alcool=c("ne boit pas",
            "boit occasionnellement","boit régulièrement"))
attributes(X)$row.vars <- list(SEXE=c("H","F"),tabac=
              c("ne fume pas","a arrêté de fumer","fume"))
```

2.4- Affichez votre tableau de contingence ainsi créé.

2.5- Enregistrez votre tableau de contingence dans un fichier nommé athero.txt en utilisant la fonction `write.ftable()`.

2.6- Vérifiez que tout s'est bien passé en ouvrant le fichier au moyen d'un éditeur de texte.

2.7- Utilisez la fonction `rm()` pour effacer tous les objets R que vous venez de créer dans votre environnement de travail.

2.8- Importez le fichier athero.txt en utilisant `read.ftable()` et affichez-le.

B- Importer depuis un logiciel externe

Lors de l'étude de l'IMC (indice de masse corporelle) chez des enfants, un fichier de données a été recueilli sous plusieurs formats différents par une équipe de statisticiens. Nous allons nous entraîner à lire ces différents formats. Il existe donc plusieurs fichiers portant le nom de base imcenfant, mais avec des extensions différentes.

2.1- Importez le fichier imcenfant.xls dans un *data.frame* nommé `imc.XLS`.

2.2- Importez le fichier imcenfant.xpt dans un *data.frame* nommé `imc.SAS`.

2.3- Importez le fichier imcenfant.sav dans un *data.frame* nommé `imc.SPSS`.

2.4- Importez le fichier imcenfant.mat dans un *data.frame* nommé `imc.MAT`. Pour ce fichier, la procédure étant plus complexe que pour les autres, nous la détaillons :

```
x <- readMat("imcenfant.mat")
class(x) # x est une liste
x        # on voit que les données sont dans $imc[,,1]
x <- x$imc[,,1]
# Notez que les éléments de SEXE et zep
# sont enregistrés dans une liste.
x$SEXE
class(x$SEXE) <- "character"
x$SEXE
class(x$zep) <- "character"
imc.MAT <- as.data.frame(x)
```

2.5- Afin de vérifier que l'importation s'est bien déroulée, utilisez la fonction `summary()` sur tous ces *data.frames*. Celle-ci va afficher quelques résumés numériques.

2.6- Sauvez l'un de ces *data.frames*, qui sont tous identiques, dans un fichier nommé imcenfant.txt.

C- Importer des fichiers de données plus compliqués

Dans la pratique d'un statisticien, il arrive régulièrement que l'on rencontre des fichiers de données ayant un format d'enregistrement non standard. Nous allons donc nous entraîner à lire plusieurs fichiers de ce type sur lesquels nous serons amenés à réaliser des analyses statistiques.

2.1- Importez le fichier raf98.gra dans la structure la plus adaptée. Pour cela, lire le fichier associé formatgeoide.txt qui contient la description du format de ce fichier.

2.2- Importez le fichier Infarct.xls dans un *data.frame* en vous assurant de bien traiter les valeurs manquantes.

2.3- Importez le fichier nutriage.txt contenant treize variables mesurées sur 226 individus, dans un *data.frame* (indice : utilisez entre autres les fonctions `as.data.frame()` et `t()`).

2.4- Importez le fichier Poids_naissance.txt contenant dix variables mesurées sur 189 individus dans un *data.frame*. Celui-ci devra contenir le nom des variables et le nom des individus (correspondant à la colonne `Id`). Pensez à utiliser l'aide en ligne.

Chapitre 3

Manipulation de données, fonctions

Pré-requis et objectif

- Lecture des chapitres 1 et 2.
- Nous évoquons les fonctions élémentaires de manipulation de données. Nous décrivons aussi les principales structures de contrôle. Ce chapitre décrit l'utilisation de l'outil d'extraction de composantes d'un objet. Il s'agit d'une façon de procéder très puissante et il convient de bien dominer cet outil afin d'utiliser R de la façon la plus efficace possible. L'extraction directe et l'extraction par masque logique seront présentées. Nous présentons également les possibilités offertes par R en ce qui concerne la manipulation de chaînes de caractères et de dates.

SECTION 3.1

Opérations sur les vecteurs, matrices et listes

3.1.1 Arithmétique vectorielle

Le logiciel R présente l'avantage de pouvoir opérer sur des vecteurs ou des matrices. Ainsi, la troisième instruction ci-dessous

```
> x <- c(1,2,4,6,3)
> y <- c(4,7,8,1,1)
> x+y
[1]  5  9 12  7  4
```

va renvoyer en une seule opération le vecteur des sommes $(x_1 + y_1, \ldots, x_n + y_n)$.

Attention

Cela est d'ailleurs l'**une des grandes forces** de ce logiciel, appelée vectorisation, et il faut prendre l'habitude de travailler de cette façon. Il est ainsi **fortement déconseillé d'utiliser des boucles de programmation** comme cela est couramment pratiqué dans d'autres langages sous peine de voir l'exécution de son code considérablement ralentie.

R opère de la même façon pour de très nombreuses fonctions telles que : `+`, `*`, `-`, `/`, `exp`, `log`, `sin`, `cos`, `tan`, `sqrt`, etc.

Ainsi, l'instruction suivante calcule l'exponentielle de tous les éléments de la matrice `M` :

```
> M <- matrix(1:9,nrow=3)
> exp(M)
          [,1]      [,2]     [,3]
[1,]  2.718282  54.59815 1096.633
[2,]  7.389056 148.41316 2980.958
[3,] 20.085537 403.42879 8103.084
```

3.1.2 Le recyclage

Un point important à noter à ce stade est la façon dont R se comporte lorsque les deux vecteurs fournis à l'une des fonctions ci-dessus ne sont pas de la même longueur. R va alors compléter le vecteur le plus court en réutilisant les valeurs de ce vecteur. L'exemple suivant devrait permettre de bien comprendre ce fonctionnement :

```
> x <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15) # Vecteur de
                                               # longueur 15.
> y <- c(1,2,3,4,5,6,7,8,9,10)                # Vecteur de
                                               # longueur 10.
> x+y                                         # Vecteur de
                                               # longueur 15.
 [1]  2  4  6  8 10 12 14 16 18 20 12 14 16 18 20
```

R a donc complété le vecteur y ainsi : `c(1,2,3,4,5,6,7,8,9,10,1,2,3,4,5)` en se resservant de ses propres valeurs de façon circulaire.

Remarque

Cette propriété s'appelle le **recyclage**. Il est important d'avoir conscience de ce mode de fonctionnement, car il peut être la source d'erreurs

difficiles à détecter. D'ailleurs, R affiche en général un message d'avertissement :

```
Warning message:
In x + y :
  la taille d'un objet plus long n'est pas multiple
  de la taille d'un objet plus court
```

Voilà un autre exemple de recyclage utilisé lors de la création d'une matrice. Le vecteur `1:4` est ainsi réutilisé en boucle pour remplir la matrice qui est déclarée de taille 3×3.

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

3.1.3 Fonctions basiques

Voyons maintenant quelques fonctions basiques de manipulation de données qui sont très souvent utilisées et qu'il est donc indispensable de connaître.

- `length()` : renvoie la longueur d'un vecteur.

  ```
  > length(c(1,3,6,2,7,4,8,1,0))
  [1] 9
  ```

- `sort()` : permet d'ordonner les éléments d'un vecteur, par valeurs croissantes ou décroissantes.

  ```
  > sort(c(1,3,6,2,7,4,8,1,0))
  [1] 0 1 1 2 3 4 6 7 8
  > sort(c(1,3,6,2,7,4,8,1,0),decreasing=TRUE)
  [1] 8 7 6 4 3 2 1 1 0
  ```

- `rev()` : réarrange les éléments d'un vecteur en sens inverse.

  ```
  > rev(c(1,3,6,2,7,4,8,1,0))
  [1] 0 1 8 4 7 2 6 3 1
  ```

- `order()`, `rank()` : la première commande renvoie le vecteur des indices de classement (croissant ou décroissant) des éléments du paramètre d'appel utilisé. En cas d'*ex æquo*, le calcul est toujours effectué de gauche à droite. La deuxième commande, comme son nom l'indique, en renvoie les rangs. Les *ex æquo* peuvent être gérés de plusieurs manières.

  ```
  > vec <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
  > names(vec) <- 1:9
  > vec
  ```

```
1 2 3 4 5 6 7 8 9
1 3 6 2 7 4 8 1 0
> sort(vec)
9 1 8 4 2 6 3 5 7
0 1 1 2 3 4 6 7 8
> order(vec)
[1] 9 1 8 4 2 6 3 5 7
> rank(vec)
  1   2   3   4   5   6   7   8   9
2.5 5.0 7.0 4.0 8.0 6.0 9.0 2.5 1.0
```

- `unique()` : comme son nom l'indique, enlève les doublons d'un vecteur.

```
> unique(c(1,3,6,2,7,4,8,1,0))
[1] 1 3 6 2 7 4 8 0
```

- `duplicated()` : indique les valeurs qui commencent à être répétées (parcourues de gauche à droite).

```
> duplicated(c(1,3,6,2,7,4,8,1,0))
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

3.1.4 Opérations sur les matrices ou les *data.frames*

Nous allons décrire ici plusieurs fonctions R spécialisées permettant d'obtenir de l'information sur une matrice (ou un *data.frame*) ou encore de manipuler ses lignes et ses colonnes.

Renvoi

Les opérations classiques de calcul matriciel (multiplication, décomposition, déterminant, etc.) sont détaillées dans le chapitre 8, page 343.

3.1.4.1 Informations sur l'architecture

Voilà déjà quelques fonctions permettant d'obtenir de l'information sur une matrice ou un *data.frame* :

- `dim()` : taille de la matrice ou du *data.frame*.
- `nrow()` : nombre de lignes.
- `ncol()` : nombre de colonnes.
- `dimnames()` : noms des lignes et des colonnes (sous la forme d'une liste).
- `names()`, `colnames()` : noms des colonnes.
- `rownames()` : noms des lignes.

Prise en main

Importez, dans un objet R que vous nommerez `X`, les données contenues dans le fichier `http://www.biostatisticien.eu/springeR/Poids_naissance.xls` et utilisez les fonctions mentionnées ci-dessus sur `X`. Veuillez noter que la première colonne du fichier contient l'identifiant des patients.

3.1.4.2 Fusion de tables

Il est souvent très utile de pouvoir combiner (fusionner) plusieurs matrices ou *data.frames*. Les fonctions de base permettant d'obtenir ce résultat sont `cbind()` pour la fusion des colonnes et `rbind()` pour la fusion des lignes.

• Fusion de colonnes

La fonction générique est `cbind()`.

```
> cbind(1:4,5:8)
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Toutefois, cette fonction n'est pas optimale, comme nous allons le voir sur l'exemple suivant. Essayons de réunir (combiner) en colonnes les deux tables suivantes.

X1=

Id	SEXE	Poids
1	H	75
2	F	68
3	F	48
4	H	72

∪

X2=

Id	SEXE	Taille
1	H	182
2	F	165
3	F	160
4	H	178

```
> X1 <- data.frame(Id=1:4,SEXE=c("H","F","F","H"),
+                          Poids=c(75,68,48,72))
> X2 <- data.frame(Id=1:4,SEXE=c("H","F","F","H"),
+                          Taille=c(182,165,160,178))
> cbind(X1,X2)
  Id SEXE Poids Id SEXE Taille
```

```
1  1    H    75  1    H    182
2  2    F    68  2    F    165
3  3    F    48  3    F    160
4  4    H    72  4    H    178
```

Cela fonctionne, mais il est gênant que les colonnes `Id` et `SEXE` soient dupliquées. Une fonction très utile dans ce contexte est la fonction `merge()` :

```
> merge(X1,X2)
  Id SEXE Poids Taille
1  1    H    75    182
2  2    F    68    165
3  3    F    48    160
4  4    H    72    178
```

Maintenant, supposons que les individus ne soient pas classés de la même manière dans les deux tables.

	Id	SEXE	Poids
	1	H	75
X1=	2	F	68
	3	F	48
	4	H	72

∪

	Id	SEXE	Taille
	2	F	165
X3=	1	H	182
	4	H	178
	3	F	160

Dans ce cas, la fonction `cbind()` ne pourra pas être utilisée alors que l'instruction `merge()` fonctionne encore :

```
> X3 <- data.frame(Id=c(2,1,4,3),SEXE=c("F","H","H","F"),
+                                  Taille=c(165,182,178,160))
> merge(X1,X3)
  Id SEXE Poids Taille
1  1    H    75    182
2  2    F    68    165
3  3    F    48    160
4  4    H    72    178
```

Vous aurez donc noté que, par défaut, la fonction `merge()` permet de fusionner deux *data.frames*. Dénotons par `X` et par `Y` les deux *data.frames* que nous désirons fusionner, et par `Z` le *data.frame* résultant de la fusion de `X` et de `Y`. La fusion est effectuée sur la base des colonnes de ces deux *data.frames* ayant les mêmes noms. On parlera de « colonnes communes » pour désigner ces colonnes. Il est possible de spécifier (forcer) les colonnes que l'on désire considérer comme communes au moyen du paramètre `by`. La valeur à fournir à ce paramètre consiste ou bien en un vecteur de noms, ou bien en un vecteur d'indices, ou encore en un vecteur de logiques. Les autres colonnes seront alors considérées par `merge()` comme étant différentes, et ce même si elles portent le même nom. Le fonctionnement de la fonction `merge()` est alors le suivant :

- pour chaque ligne (individu) du *data.frame* `X`, la fonction `merge()` commence par comparer les éléments de cette ligne à ceux de chacune des lignes de `Y`, mais uniquement sur le sous-ensemble des colonnes communes ;
- si elle trouve un appariement parfait, elle considère alors qu'il s'agit du même individu : cet individu est alors ajouté à `Z` puis complété avec les valeurs des colonnes non communes de `X`, puis avec les valeurs des colonnes non communes de `Y` ;
- si l'appariement n'est pas parfait, il s'agit alors de deux individus différents qui sont soit ajoutés chacun sur une ligne différente de `Z` et complétés par des `NA` (si le paramètre `all` prend la valeur `TRUE`), soit retirés (si le paramètre `all` prend la valeur `FALSE`, ce qui est le comportement par défaut) ;
- l'opération se poursuit alors pour la ligne suivante, et ce jusqu'à la dernière ligne.

Voici un exemple qui permettra de clarifier les choses.

```
> X <- data.frame(SEXE=c("F","H","H","F"),Taille=c(165,182,
+      178,160),Poids=c(50,65,67,55),Revenu=c(80,90,60,50))
> Y <- data.frame(SEXE=c("F","H","H","F"),Taille=c(165,182,
+      178,160),Poids=c(55,65,67,85),Salaire=c(70,90,40,40),
+           row.names=4:7)
> X
  SEXE Taille Poids Revenu
1    F    165    50     80
2    H    182    65     90
3    H    178    67     60
4    F    160    55     50
> Y
  SEXE Taille Poids Salaire
4    F    165    55      70
5    H    182    65      90
6    H    178    67      40
7    F    160    85      40
> merge(X,Y,by=c("SEXE","Poids"))
  SEXE Poids Taille.x Revenu Taille.y Salaire
1    F    55      160     50      165      70
2    H    65      182     90      182      90
3    H    67      178     60      178      40
> merge(X,Y,by=c("SEXE","Poids"),all=TRUE)
  SEXE Poids Taille.x Revenu Taille.y Salaire
1    F    50      165     80       NA      NA
2    F    55      160     50      165      70
3    F    85       NA     NA      160      40
4    H    65      182     90      182      90
5    H    67      178     60      178      40
```

Attention

Vous aurez noté que, par défaut, la fonction `merge()` ne prend pas en compte les noms des individus dans les *data.frames* `X` et `Y` pour déterminer les individus communs. Il est toutefois possible de contourner ce fonctionnement soit en ajoutant une colonne `Id` à `X` et à `Y` pour identifier les individus, soit en utilisant le nom `"row.names"` comme valeur du paramètre `by`.

```
> merge(X,Y,by=c("row.names","Poids"))
  Row.names Poids SEXE.x Taille.x Revenu SEXE.y Taille.y
1         4    55      F      160     50      F      165
  Salaire
1      70
> merge(X,Y,by=c("row.names","Poids"),all=TRUE)
  Row.names Poids SEXE.x Taille.x Revenu SEXE.y Taille.y
1         1    50      F      165     80   <NA>       NA
2         2    65      H      182     90   <NA>       NA
3         3    67      H      178     60   <NA>       NA
4         4    55      F      160     50      F      165
5         5    65   <NA>       NA     NA      H      182
6         6    67   <NA>       NA     NA      H      178
7         7    85   <NA>       NA     NA      F      160
  Salaire
1      NA
2      NA
3      NA
4      70
5      90
6      40
7      40
```

• **Fusion de lignes**

La fonction générique est `rbind()`.

```
> rbind(1:4,5:8)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

La fonction `smartbind()` du *package* `gtools` est plus évoluée comme on peut le voir sur l'exemple suivant :

```
> require("gtools")
> df1 <- data.frame(A=1:5, B=LETTERS[1:5])    # Les crochets []
                                              # permettant
                                              # l'extraction
> df2 <- data.frame(A=6:10, E=letters[1:5])   # d'éléments seront
```

```
                                          # vus à la section 3.4.
> smartbind(df1, df2)
     A    B    E
1:1  1    A <NA>
1:2  2    B <NA>
1:3  3    C <NA>
1:4  4    D <NA>
1:5  5    E <NA>
2:1  6 <NA>    a
2:2  7 <NA>    b
2:3  8 <NA>    c
2:4  9 <NA>    d
2:5 10 <NA>    e
```

Astuce

Notons l'existence du *package* `gdata` qui contient plusieurs fonctions très intéressantes pour manipuler des données.

3.1.4.3 La fonction `apply()`

Une fonction très utilisée est la fonction `apply()` qui applique une fonction donnée (fournie comme valeur du paramètre `FUN`) aux lignes (`MARGIN=1`) ou bien aux colonnes (`MARGIN=2`) d'une matrice ou d'un *data.frame*.

```
> X <- matrix(c(1:4, 1, 6:8), nr = 2)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    7
[2,]    2    4    6    8
> apply(X, MARGIN=1, FUN=mean)
[1] 3 5
> apply(X, MARGIN=2, FUN=sum)
[1]  3  7  7 15
```

Astuce

Lorsque les opérations à effectuer consistent à sommer ou à moyenner les lignes ou les colonnes, on peut aussi utiliser respectivement : `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()`.

Prise en main

Voyons comment il est possible de calculer la somme des carrés de chacune des lignes d'une matrice. Pour cela, commencez par créer une matrice `M` de taille 5×2 contenant les chiffres de votre choix. Ensuite, utilisez la fonction `apply()` sur les lignes de cette matrice `M`. Vous prendrez le paramètre et sa valeur associée `FUN=function(x) {sum(x^2)}`.

Attention

Dans la pratique précédente, nous avons vu rapidement comment utiliser sa propre fonction (`sum(x^2)`) dans l'appel de `apply()`, à l'aide du mot réservé `function`. Nous verrons plus en détails dans le chapitre 6 comment créer des fonctions plus élaborées.

3.1.4.4 La fonction `sweep()`

La fonction `sweep()` est très utilisée. Elle permet de « retirer » d'une table (dans un sens spécifié par la valeur d'appel du paramètre `FUN`), à chacune des lignes (`MARGIN=1`) ou à chacune des colonnes (`MARGIN=2`), une statistique donnée (par la valeur d'appel du paramètre `STATS`). Les deux exemples suivants permettront de bien comprendre cette fonction.

```
> X
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    7
[2,]    2    4    6    8
> # Retranche 3 à la ligne 1, et 5 à la ligne 2.
> sweep(X,MARGIN=1,STATS=c(3,5),FUN="-")
     [,1] [,2] [,3] [,4]
[1,]   -2    0   -2    4
[2,]   -3   -1    1    3
> # Divise les deux premières colonnes par 2, et les deux
  # dernières par 3.
> sweep(X,MARGIN=2,STATS=c(2,2,3,3),FUN="/")
     [,1] [,2]      [,3]     [,4]
[1,]  0.5  1.5 0.3333333 2.333333
[2,]  1.0  2.0 2.0000000 2.666667
```

3.1.4.5 La fonction `stack()`

La fonction `stack()` permet d'empiler dans un seul vecteur les valeurs de certaines colonnes d'un *data.frame*. Cette fonction renvoie un *data.frame* dont

la première colonne contient le vecteur ainsi empilé, et dont la deuxième colonne contient un facteur indiquant l'origine de chaque observation. La fonction `unstack()` effectue l'opération inverse. Cette fonction apparaît particulièrement utile en analyse de la variance (ANOVA).

```
> X <- data.frame(trt1=c(1,6,3,5),trt2=c(8,8,3,1))
> X
  trt1 trt2
1    1    8
2    6    8
3    3    3
4    5    1
> stack(X)
  values  ind
1      1 trt1
2      6 trt1
3      3 trt1
4      5 trt1
5      8 trt2
6      8 trt2
7      3 trt2
8      1 trt2
```

3.1.4.6 La fonction `aggregate()`

La fonction `aggregate()` permet le découpage d'un *data.frame* en sous-populations suivant un facteur (spécifié par le paramètre `by`) et d'appliquer une fonction donnée sur chacune de ces sous-populations.

```
> X<-data.frame(Poids=c(80,75,60,52),Taille=c(180,170,165,150),
+                          Cholesterol=c(44,12,23,34),
+                      sexe=c("Homme","Homme","Femme","Femme"))
> X
  Poids Taille Cholesterol  sexe
1    80    180          44 Homme
2    75    170          12 Homme
3    60    165          23 Femme
4    52    150          34 Femme
> aggregate(X[,-4],by=list(Sexe=X[,4]),FUN=mean)
   Sexe Poids Taille Cholesterol
1 Femme  56.0  157.5        28.5
2 Homme  77.5  175.0        28.0
```

Remarque

L'instruction `X[,-4]` permet d'extraire toutes les colonnes de `X` sauf la quatrième. Les instructions d'extraction seront vues plus en détail à la section 3.4.

3.1.4.7 La fonction `transform()`

Elle permet d'opérer des transformations sur les colonnes d'un *data.frame*. L'instruction suivante permet par exemple d'obtenir la taille en mètres à partir de la taille en centimètres, et de rajouter dans le *data.frame* une colonne contenant l'IMC.

```
> X <- transform(X,Taille=Taille/100,IMC=Poids/(Taille/100)^2)
> X
  Poids Taille Cholesterol  sexe      IMC
1    80   1.80          44 Homme 24.69136
2    75   1.70          12 Homme 25.95156
3    60   1.65          23 Femme 22.03857
4    52   1.50          34 Femme 23.11111
```

Renvoi

Notez l'existence du package `plyr` qui permet de manipuler de façon simplifiée et efficace des tableaux de données.

3.1.5 Opérations sur les listes

Les fonctions `lapply()` et `sapply()` sont similaires à la fonction `apply()`, mais appliquent une fonction à chacun des éléments d'une liste. La première renvoie le résultat sous la forme d'une liste, la seconde sous la forme d'un vecteur dans la mesure du possible.

```
> x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,
+            FALSE,TRUE))
> lapply(x,mean) # Moyenne de chaque élément de la liste.
$a
[1] 5.5
$beta
[1] 4.535125
$logic
[1] 0.5
> lapply(x,quantile,probs=(1:3)/4) # Médiane et quartiles des
                                   # éléments de la liste.
$a
 25%  50%  75%
3.25 5.50 7.75
$beta
      25%       50%       75%
0.2516074 1.0000000 5.0536690
$logic
25% 50% 75%
0.0 0.5 1.0
> sapply(x, quantile) # Quantiles des éléments de la liste.
```

```
          a        beta logic
0%    1.00  0.04978707   0.0
25%   3.25  0.25160736   0.0
50%   5.50  1.00000000   0.5
75%   7.75  5.05366896   1.0
100% 10.00 20.08553692   1.0
> i36 <- sapply(3:6, seq) # Crée une liste de vecteurs.
> i36
[[1]]
[1] 1 2 3
[[2]]
[1] 1 2 3 4
[[3]]
[1] 1 2 3 4 5
[[4]]
[1] 1 2 3 4 5 6
> sapply(i36, sum) # Somme de chacun des vecteurs de la liste.
[1]  6 10 15 21
```

Astuce

La fonction `do.call()` prend deux paramètres : le premier est le nom d'une fonction et le second celui d'une liste. Elle exécute cette dernière fonction avec comme valeurs d'entrée les différents éléments de la liste. Nous verrons un exemple d'utilisation de cette fonction dans la fiche de TP. Mentionnons aussi l'existence de la fonction `mapply()`.

SECTION 3.2

Opérations logiques et relationnelles

Les deux valeurs logiques sont VRAI (`TRUE` ou `T`) et FAUX (`FALSE` ou `F`). Notez également que `NA` est considérée en **R** comme une valeur logique (dite constante). Les vecteurs de logiques sont très utiles en **R**, notamment dans l'**extraction d'éléments par masque logique** que nous verrons plus loin.

Voici le tableau des opérateurs et fonctions agissant sur ou créant des logiques.

TABLE 3.1 – Opérateurs et fonctions agissant sur ou créant des logiques.

Opérateur en R	Description	Exemple	Résultat
`logical()`	Créer un vecteur de logiques	`logical(2)`	F F
`as.logical()`	Transforme en logiques	`as.logical(c(0,1))`	F T
`is.logical()`	Le paramètre effectif est-il logique ?	`is.logical(F)`	T
`x < y`	Est-ce que $x_i < y_i$?	`c(1,4)<c(2,3)`	T F
`x > y`	Est-ce que $x_i > y_i$?	`c(1,4)>c(2,3)`	F T
`x <= y`	Est-ce que $x_i <= y_i$?	`c(1,4)<=c(1,3)`	T F
`x >= y`	Est-ce que $x_i >= y_i$?	`c(1,4)>=c(1,3)`	T T
`x == y`	Est-ce que $x_i = y_i$?	`c(1,4)==c(1,3)`	T F
`x != y`	Est-ce que $x_i \neq y_i$?	`c(1,4)!=c(1,3)`	F T
`!x`	Négation de x	`!c(T,F)`	F T
`x & y`	Conjonctions terme à terme	`c(T,T) & c(T,F)`	T F
`x && y`	Conjonctions séquentielles	`F && T && T`	F
`x | y`	OU logique terme à terme	`c(T,T) | c(T,F)`	T T
`x || y`	OU séquentiel	`F || T || F`	T
`xor(x, y)`	OU exclusif	`xor(c(T,T),c(T,F))`	F T
`any(x)`	`TRUE` si l'un des x_i est `TRUE`	`any(c(T,F))`	T
`all(x)`	`TRUE` si tous les x_i sont `TRUE`	`all(c(T,F))`	F
`all.equal(x,y)`	Est-ce que $x_i \approx y_i$? (voir le paramètre `tolerance`)	`all.equal(0.2-0.1,0.3-0.2)`	T
`identical(x,y)`	`TRUE` si $\forall i, x_i = y_i$	`identical(1,as.integer(1))`	F

Renvoi

Une explication de la différence entre conjonctions terme à terme et conjonctions séquentielles sera fournie à la page 125.

Attention

Prenez garde à la différence de résultat entre les deux instructions suivantes :

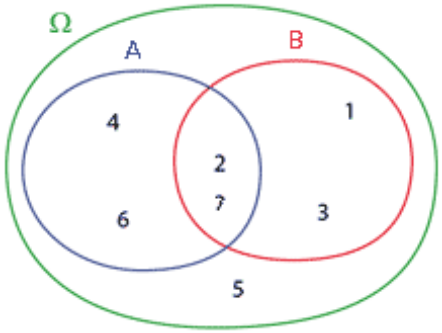
```
> all.equal(0.2-0.1,0.3-0.2)
[1] TRUE
> (0.2-0.1) == (0.3-0.2)
[1] FALSE
```

Cela est dû au fait que l'ordinateur effectue les calculs avec une précision limitée, et que la fonction `all.equal()` admet un paramètre optionnel de tolérance aux erreurs d'arrondis. Nous en reparlerons dans la section dédiée aux instructions de condition. Voir aussi la section 3.9 qui explique pourquoi la seconde instruction ci-dessus retourne `FALSE`.

SECTION 3.3

Opérations ensemblistes

Il est possible en **R** d'effectuer les opérations ensemblistes usuelles.




```
> A <- c(4,6,2,7)  # Un premier ensemble.
> B <- c(2,1,7,3)  # Un deuxième ensemble.
> vec <- c(2,3,7)  # Quelques éléments.
```

Table 3.2 – Opérations ensemblistes.

Opération ensembliste	Instruction **R**	Résultat
Appartenance : $a \in A$	`is.element(vec,A)` ou bien `vec %in% A`	`T F T`
Inclusion : $A \subset B$	`all(A %in% B)`	`F`
Contenance : $A \supset B$	`all(B %in% A)`	`F`
Intersection : $A \cap B$	`intersect(A,B)`	`2 7`
Réunion : $A \cup B$	`union(A,B)`	`4 6 2 7 1 3`
Complémentaire : $A \backslash B$	`setdiff(A,B)`	`4 6`
Différence symétrique : $(A \cup B) \backslash (A \cap B)$	`setdiff(union(A,B),intersect(A,B))`	`4 6 1 3`

Astuce

Il est très facile en **R** de définir ses propres fonctions ensemblistes, comme par exemple les fonctions d'inclusion, de contenance et de différence symétrique.

```
> "%inclus%" <- function(A,B) all(A %in% B)
> "%contient%" <- function(A,B) B %inclus% A
> "%diffsym%" <- function(A,B) setdiff(union(A,B),
+                                      intersect(A,B))
```

SECTION 3.4

Extraction et insertion d'éléments

L'objectif dans cette section est de voir comment il est possible d'extraire une partie d'un vecteur, d'une matrice ou encore d'une liste. R possède en effet pour ce faire des mécanismes très particuliers qui peuvent apparaître un peu déroutants de prime abord, mais qui se révèlent être ensuite de très puissants outils.

3.4.1 Extraction/Insertion dans les vecteurs

• Extraction

La fonction à utiliser pour extraire des composantes d'un vecteur est `"["()`. Elle peut prendre les paramètres d'appel suivants :

- un vecteur des indices des éléments à extraire ;
- un vecteur des indices des éléments à ne pas extraire ;
- un vecteur de logiques `TRUE/FALSE` indiquant quels éléments extraire.

Voici quelques exemples qui sont plus parlants qu'un long discours.

```
> vec <- c(2,4,6,8,3)
> vec[2]
[1] 4
> "["(vec,2)          # Note : "[" est bien une fonction.
[1] 4
> vec[-2]             # Tous les éléments sauf le deuxième.
[1] 2 6 8 3
> vec[2:5]
[1] 4 6 8 3
> vec[-c(1,5)]
[1] 4 6 8
> vec[c(T,F,F,T,T)] # Extraction par masque logique.
[1] 2 8 3
> vec>4
[1] FALSE FALSE  TRUE  TRUE FALSE
> vec[vec>4]          # Extraction par masque logique.
[1] 6 8
```

Attention

Il est important de noter ici la simplicité de la syntaxe d'une instruction comme `x[y>0]` qui extrait de `x` tous les éléments dont les indices sont les i tels que $y_i > 0$.

```
> x <- 1:5
> y <- c(-1,2,-3,4,-2)
> x[y>0]
[1] 2 4
```

Il vous faut apprendre à utiliser le plus souvent possible ce type de construction, dit par **masque logique**, qui possède le double avantage de produire un code facile à lire et qui est exécuté très rapidement par R.

Notons également l'existence des fonctions `which()`, `which.min()` et `which.max()` qui rendent souvent de précieux services.

```
> masque <- c(TRUE,FALSE,TRUE,NA,FALSE,FALSE,TRUE)
> which(masque) # Renvoie les indices des valeurs TRUE.
[1] 1 3 7
> x <- c(0:4,0:5,11)
> which.min(x)  # Renvoie l'indice de la plus petite valeur.
[1] 1
> which.max(x)  # Renvoie l'indice de la plus grande valeur.
[1] 12
```

Attention

Il est important de bien signaler le fait que R ne gère pas l'indice 0, contrairement à certains autres langages de programmation.

• **Remplacement**

Le remplacement d'éléments dans un vecteur procède de manière similaire à celle de l'extraction. Pour cela, il suffit de sélectionner les éléments voulus comme si nous souhaitions les extraire, puis d'utiliser le symbole d'affectation `<-` suivi des éléments de remplacement. Bien entendu, vous devez spécifier le même nombre d'éléments de remplacement que ceux sélectionnés.

Voyons tout de suite quelques exemples de ce principe.

```
> z
[1] 0 0 0 2 0
> z[c(1,5)] <- 1
> z
[1] 1 0 0 2 1
> z[which.max(z)] <- 0
> z
[1] 1 0 0 0 1
> z[z==0] <- 8  # Les z_i tels que z_i
```

```
                # vaut 0 sont remplacés par 8.
> z
[1] 1 8 8 8 1
```

• **Insertion**

L'insertion ou le rajout d'éléments dans un vecteur pré-existant utilise la fonction `c()`.

```
> vecA <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
> vecA
[1] 1 3 6 2 7 4 8 1 0
> (vecB <- c(vecA, 4, 1))
 [1] 1 3 6 2 7 4 8 1 0 4 1
> (vecC <- c(vecA[1:4], 8, 5, vecA[5:9]))
 [1] 1 3 6 2 8 5 7 4 8 1 0
```

Ce mécanisme offre la possibilité de compléter un vecteur que l'on veut éviter de dimensionner au départ.

```
> a <- c()
> a <- c(a,2)
> a <- c(a,7)
> a
[1] 2 7
```

Prise en main

• Créez le vecteur `taille <- c(182,150,160,140.5,191)` et le vecteur `sexe <- c(0,1,1,1,0)` contenant les tailles (cm) et le sexe (codé en 0 = H / 1 = F) de cinq personnes. Extrayez du vecteur `taille` les tailles des hommes. Utilisez l'approche d'extraction par indice puis par masque logique.

• Extrayez du vecteur suivant tous les nombres compris entre 2 et 3 :

```
> x <- c(0.1,0.5,2.1,3.5,2.8,2.7,1.9,2.2,5.6)
```

3.4.2 Extraction/Insertion dans les matrices

• **Extraction**

Pour extraire des éléments d'une matrice `X`, on peut utiliser deux approches, chacune ayant une syntaxe propre.

3.1- *L'extraction par indice :* `X[indl,inc]`, où `indl` désigne le vecteur des indices des lignes et `indc` désigne le vecteur des indices des colonnes à extraire. L'omission de `indl` (respectivement de `indc`) signifie que l'on sélectionne toutes les lignes (respectivement les colonnes). Notons également que l'on peut faire précéder `indl` et/ou `indc` du signe moins (`-`) pour indiquer plutôt les éléments à ne pas extraire.

3.2- *L'extraction par masque logique :* `X[masque]`, où `masque` est une matrice de logiques `TRUE/FALSE` de la même taille que `X` indiquant quels éléments extraire.

Voici quelques exemples utilisant la première approche :

```
> Mat <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> Mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> Mat[2,3]          # Extraction de l'élément à l'intersection
                    # ligne 2/colonne 3.
[1] 6
> Mat[,1]           # On prend toutes les lignes, et seulement la
                    # colonne 1.
[1]  1  4  7 10
> Mat[c(1,4),]      # On prend toutes les colonnes, et les lignes 1
                    # et 4.
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   10   11   12
> Mat[3,-c(1,3)]    # On prend la ligne 3 et la colonne 2.
[1] 8
```

Voilà maintenant un exemple utilisant un masque logique :

```
> MatLogique <- matrix(c(TRUE,FALSE),nrow=4,ncol=3)
> MatLogique        # Possède la même taille que Mat.
      [,1]  [,2]  [,3]
[1,]  TRUE  TRUE  TRUE
[2,] FALSE FALSE FALSE
[3,]  TRUE  TRUE  TRUE
[4,] FALSE FALSE FALSE
> Mat[MatLogique] # Assurez-vous de comprendre cette
                  # instruction.
[1] 1 7 2 8 3 9
```

Astuce

Il se trouve qu'une matrice est stockée dans R sous la forme d'un long vecteur, empilement de toutes les colonnes de cette matrice les unes sur les autres. Essayez par exemple la commande `as.vector(Mat)`. On peut donc extraire des éléments d'une matrice sans utiliser la forme `[lignes,colonnes]`, mais en utilisant l'extraction vectorielle `[ind]` où `ind` est le vecteur des indices (ou bien un vecteur de logiques) des éléments à extraire de la grande colonne ainsi constituée.

```
> ind
[1] 2 4 6 8 3
> Mat[ind]
[1]  4 10  5 11  7
```

Attention

Le fonctionnement de la fonction `"["()` mène parfois à un changement de la structure de l'objet manipulé. Voyons ceci sur l'exemple ci-dessous.

```
> m <- matrix(1:6,nrow=2) ; m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m[,1]
[1] 1 2
> class(m[,1])
[1] "integer"
```

On constate que l'extraction a transformé le résultat en simple vecteur (ligne), ce qui peut être gênant puisqu'ici on pouvait s'attendre à obtenir une matrice colonne. Mais il existe une solution à ce problème comme le démontre le code suivant.

```
> m[,1,drop=FALSE]
     [,1]
[1,]    1
[2,]    2
```

Prise en main

Essayez d'intercaler de façon automatique le vecteur `c(0,2,3,4)` dans le vecteur `c(1,0,0,0)` afin d'obtenir le vecteur `c(1,0,0,2,0,3,0,4)` (indice : utilisez les fonctions `cbind()`, `t()` et `as.vector()`).

Comme dans le cas vectoriel, la fonction `which()` permet de récupérer les indices des éléments de la matrice qui vérifient une certaine condition. Voyons cela sur un exemple.

```
> m <- matrix(c(1,2,3,1,2,3,2,1,3),3,3)
> m
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    1
[3,]    3    3    3
> which(m == 1)              # m est considérée comme un empilement
                             # de ses colonnes.
[1] 1 4 8
> which(m == 1,arr.ind=TRUE)   # Récupération des indices sous la
                               # forme de couples.
     row col
[1,]   1   1
[2,]   1   2
[3,]   2   3
```

• Insertion

L'insertion d'éléments procède de la même manière que dans le cas vectoriel. Il faut utiliser le symbole d'affectation `<-` pour remplacer des éléments, sélectionnés soit au moyen de leurs indices, soit par masque logique, par d'autres éléments.

```
> m
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    1
[3,]    3    3    3
> m[m!=2] <- 0
> m
     [,1] [,2] [,3]
[1,]    0    0    2
[2,]    2    2    0
[3,]    0    0    0
> Mat <- Mat[-4,] ; Mat
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> m[Mat>7] <- Mat[Mat>7]
> m
     [,1] [,2] [,3]
[1,]    0    0    2
[2,]    2    2    0
[3,]    0    8    9
```

Astuce

Il existe une autre fonction permettant d'extraire (et donc d'insérer) des éléments : `subset()`. Essayez par exemple `subset(airquality, Temp > 80, select = c(Ozone, Temp))`.

Prise en main

```
> m1 <- matrix(c(0,22,0,23,34,0,0,0,28),ncol=3)
> m2 <- matrix(c(10,1,4,10,9,9,2,6,4),ncol=3)
> m1
     [,1] [,2] [,3]
[1,]    0   23    0
[2,]   22   34    0
[3,]    0    0   28
> m2
     [,1] [,2] [,3]
[1,]   10   10    2
[2,]    1    9    6
[3,]    4    9    4
```

Remplacez toutes les valeurs non nulles de `m1` par les valeurs de `m2` correspondantes.
Enlevez la deuxième colonne de `m1`.

3.4.3 Extraction/Insertion dans les *arrays*

L'extraction et l'insertion dans les *arrays* fonctionne de la même manière que dans les matrices, mis à part qu'il peut y avoir plus de deux dimensions. Nous nous contentons donc de donner quelques exemples et laissons le lecteur s'assurer qu'il les comprend bien.

```
> A <- array(1:12,dim=c(2,2,3))
> A
, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
     [,1] [,2]
[1,]    9   11
[2,]   10   12
> A[2,2,1]
[1] 4
> A[1,2,3] <- 4 # Remplace 11 par 4.
> which(A==4,arr.ind=TRUE)
     dim1 dim2 dim3
[1,]    2    2    1
[2,]    1    2    3
> A[which(A==4,arr.ind=TRUE)]
[1] 4 4
> length(A[A>4])
[1] 7
```

3.4.4 Extraction/Insertion dans les listes

• **Extraction**

L'extraction dans les listes est légèrement plus compliquée que dans les matrices. Chaque élément d'une liste est en effet lui-même une liste. L'utilisation de la fonction `"["()` sur une liste renvoie donc une autre liste.

```
> L <- list(12,c(34,67,8),Mat,1:15,list(10,11))
> class(L)
[1] "list"
> L
[[1]]
[1] 12
[[2]]
[1] 34 67  8
[[3]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[[4]]
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[[5]]
[[5]][[1]]
[1] 10
[[5]][[2]]
[1] 11
> L[2]
[[1]]
[1] 34 67  8
> class(L[2])
[1] "list"
> L[c(3,4)]
[[1]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[[2]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

Puisqu'une liste est faite pour stocker des éléments de nature différente, il faudra utiliser la fonction `"[["()` pour accéder au contenu des éléments d'une liste.

```
> L[[2]]
[1] 34 67  8
> "[["(L,2)
[1] 34 67  8
> class(L[[2]])
[1] "numeric"
> L[[5]][[2]]
[1] 11
```

Attention

Les instructions suivantes génèrent une erreur :

```
> L[2,3]
Error in L[2, 3] : nombre de dimensions incorrect
> L[[2,3]]
Error in L[[2, 3]] : nombre d'indices incorrect
```

Expert

R définit ce que l'on appelle l'**indexation récursive**. Ainsi, l'instruction suivante commence par récupérer le contenu du deuxième élément de la liste

`L` (c'est-à-dire le vecteur `c(34,67,8)`), puis extrait le troisième élément de ce vecteur.

```
> L[[c(2,3)]] # Indexation récursive.
[1] 8
```

Par ailleurs, **R** offre un mécanisme permettant de nommer explicitement les différents éléments d'une liste, et il est ensuite possible d'utiliser le symbole `$` pour extraire nommément certains éléments de la liste.

```
> L <- list(voitures=c("FORD","PEUGEOT"),climat=
+                       c("Tropical","Tempéré"))
> L[["voitures"]]
[1] "FORD"    "PEUGEOT"
> L$voitures
[1] "FORD"    "PEUGEOT"
> L$climat
[1] "Tropical" "Tempéré"
```

• **Insertion**

Le mécanisme d'insertion fonctionne comme précédemment en utilisant la flèche `<-`.

```
> L$climat[2] <- "Continental"
> L
$voitures
[1] "FORD"    "PEUGEOT"
$climat
[1] "Tropical"    "Continental"
```

Astuce

Le nom d'une colonne peut contenir des espaces. Pour y accéder, il faut alors utiliser des guillemets.

```
> L <- list("belles voitures"=c("FORD","PEUGEOT"))
> L
$`belles voitures`
[1] "FORD"    "PEUGEOT"
> L$"belles voitures"
[1] "FORD"    "PEUGEOT"
```

SECTION 3.5

Manipulation de chaînes de caractères

La manipulation de chaînes de caractères se révèle très utile dans le traitement de nombreux fichiers statistiques, ou dans l'annotation de certains graphiques. Nous allons donc présenter les fonctions R les plus importantes dans ce contexte.

Nous avons déjà vu que la création d'une chaîne de caractères s'obtient au moyen des guillemets `""`, ou bien de l'utilisation de la fonction `as.character()`.

```
> chaine <- c("un","deux","trois")
> chaine
[1] "un"    "deux"  "trois"
> as.character(1:3)
[1] "1" "2" "3"
```

Astuce

La fonction `noquote()` permet de supprimer l'affichage des guillemets dans les sorties de R.

```
> noquote(chaine)
[1] un    deux  trois
```

Les fonctions `sQuote()` et `dQuote()` permettent de gérer plusieurs styles de guillemets.
La fonction `format()` permet de produire un affichage personnalisé, notamment des *data.frames*.

```
> zz <- data.frame("Les prénoms"=c("Pierre","Benoit","Rémy"),
+                                              check.names=FALSE)
> zz
  Les prénoms
1      Pierre
2      Benoit
3        Rémy
> format(zz, justify = "left")
  Les prénoms
1      Pierre
2      Benoit
3      Rémy
```

D'autres fonctions intéressantes permettant de gérer l'affichage sont `cat()`, `sprintf()` et `print()`.

La fonction `nchar()` compte le nombre de symboles d'une chaîne. Elle peut être appliquée à un vecteur de chaînes.

```
> chaine1 <- c("a","ab","B","bba","un","!@","brut")
> nchar(chaine1)    # Compte le nombre de symboles dans chaque
                    # chaine.
[1] 1 2 1 3 2 2 4
> chaine1[nchar(chaine1)>2]
[1] "bba"  "brut"
```

Astuce

Les commandes `letters` et `LETTERS` renvoient les vingt-six lettres de l'alphabet en minuscules et en majuscules.

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> chaine1[chaine1 %in% c(letters,LETTERS)]
[1] "a" "B"
```

La fonction `paste()` permet de concaténer plusieurs chaînes.

```
> chaine2 <- c("e","D")
> paste(chaine1,chaine2) # Concaténation des chaînes.
[1] "a e"    "ab D"   "B e"    "bba D"  "un e"   "!@ D"
[7] "brut e"
> paste(chaine1,chaine2,sep="-")  # On peut indiquer le
                                  # séparateur entre les
                                  # chaînes.
[1] "a-e"    "ab-D"   "B-e"    "bba-D"  "un-e"   "!@-D"
[7] "brut-e"
> paste(chaine1,chaine2,collapse="",sep="")  # collapse permet de
                                             # coller les
                                             # chaînes.
[1] "aeabDBebbaDune!@Dbrute"
```

La fonction `substring()` permet d'extraire des sous-chaînes d'une chaîne.

```
> substring("abcdef",first=1:3,last=2:4)
[1] "ab" "bc" "cd"
```

La fonction `strsplit()` permet de découper une chaîne.

```
> strsplit(c("05 Jan","06 Fev"), split=" ")
[[1]]
[1] "05"  "Jan"
[[2]]
[1] "06"  "Fev"
```

La fonction `grep()` permet de rechercher un motif dans un vecteur de chaînes. Elle renvoie les indices des éléments du vecteur qui contiennent cette chaîne.

```
> grep("i",c("Pierre","Benoit","Rems"))
[1] 1 2
```

La fonction `gsub()` remplace toutes les occurrences d'un motif trouvé dans une chaîne par une autre chaîne.

```
> gsub("i","L",c("Pierre","Benoit","Rems"))
[1] "PLerre" "BenoLt" "Rems"
```

La fonction `sub()` remplace uniquement la première occurrence trouvée.

```
> sub("r","L",c("Pierre","Benoit","Rems"))
[1] "PieLre" "Benoit" "Rems"
```

Astuce

Les fonctions `tolower()` et `toupper()` permettent respectivement de passer une chaîne en lettres minuscules ou en majuscules.

```
> tolower("mINISCULE")
[1] "miniscule"
> toupper("Majuscule")
[1] "MAJUSCULE"
```

Prise en main

Entrez le *data.frame* suivant :

```
> X <- data.frame(date=c("03 JANVI","02 JUILL","15 NOVEM"),
+                 ensol=c(10,15,12))
> X
      date ensol
1 03 JANVI    10
2 02 JUILL    15
3 15 NOVEM    12
```

Retirez la première colonne de `X` et rajouter deux nouvelles colonnes, l'une nommée `jour` qui contiendra le numéro du jour codé en numérique (1 ou 2 chiffres), et l'autre nommée `mois` qui contiendra le mois codé sur 4 lettres minuscules (indice : utiliser la fonction `transform()`).

SECTION 3.6

Manipulation de dates et d'unités de temps

3.6.1 Affichage de la date courante

Dans R, il existe deux fonctions permettant d'afficher la date courante : `Sys.time()` et `date()`.

```
> Sys.time()
[1] "2014-06-25 21:40:43 EST"
> date()
[1] "Wed Jun 25 21:40:43 2014"
```

Il est possible d'extraire l'année, le mois, le jour, l'heure, la minute et la seconde de la façon suivante :

```
> as.numeric(substring(Sys.time(),c(1,6,9,12,15,18),
+            c(4,7,10,13,16,19)))
[1] 2014    6   25   21   40   43
```

Astuce

La fonction `system.time()` permet de connaître le temps d'exécution d'une instruction. La fonction `Sys.sleep()` permet de stopper l'exécution d'une liste d'instructions pendant un nombre donné de secondes.

3.6.2 Extraction de dates

Il arrive fréquemment en statistique que l'on soit amené à traiter des fichiers de données contenant des dates. R dispose de plusieurs fonctions permettant de bien gérer ce type de données qui seraient sans cela très difficiles à manipuler.

La première instruction à connaître est `strptime()` qui permet de récupérer une date depuis une chaîne de caractères pour la mettre dans un objet R de classe POSIXlt (liste nommée de vecteurs contenant des informations sur la date et l'heure).

```
[1] "C"
> strptime("27/mar/73",format="%d/%b/%y")
[1] "1973-03-27 EST"

[1] "fr_FR.UTF-8"
```

Vous aurez noté dans l'instruction précédente l'utilisation du paramètre `format` qui permet de décrire comment la date et/ou l'heure sont codées dans

la chaîne de caractères. De nombreux codes sont disponibles, et nous les détaillons dans le tableau qui suit.

Remarque

Si jamais l'instruction précédente renvoie `NA`, il pourrait apparaître nécessaire d'utiliser au préalable l'instruction suivante pour changer les paramètres régionaux (*locale*) par défaut utilisés par R :

```
> Sys.setlocale("LC_TIME","C")
```

TABLE 3.3 – Codes pour la fonction `strptime()`. Les exemples ont été fabriqués avec `format(Sys.time(),"%`*lettre*`")`.

Code	Description	Exemple
%a	Jour de la semaine abrégé	sam
%A	Jour de la semaine complet	samedi
%b	Mois abrégé	avr
%B	Mois complet	avril
%c	Date et heure, dépend du lieu	sam 18 avr 2009 09:06:24 CEST
%d	Jour du mois (01-31)	18
%H	Heure (0-24)	13
%I	Heure (0-12)	09
%j	Jour de l'année (001-366)	108
%m	Mois (01-12)	04
%M	Minute (00-59)	07
%S	Seconde (00-61, avec 2 « secondes intercalaires »)	50
%U	Semaine de l'année (00-53), le 1er dimanche étant compté comme le jour 1 de la semaine 1	15
%w	Jour de la semaine (0-6), dimanche étant le 0	6
%W	Semaine de l'année (00-53), le 1er lundi étant compté comme le jour 1 de la semaine 1	15
%x	Date, dépend du lieu	2009-04-18
%X	Temps, dépend du lieu	09:08:40
%y	Année sans le siècle	09
%Y	Année avec le siècle	2009
%z	Décalage de Greenwich, '-0800' est 8 heures à l'ouest de Greenwich	+0000
%Z	Fuseau horaire en chaîne de caractères (sortie seulement)	CEST

Astuce

Sous Linux, l'instruction `man strptime` tapée dans une fenêtre de terminal peut vous donner des codes supplémentaires.

Prise en main

Essayez de lire les dates suivantes avec la fonction `strptime()`.

```
> dates1
[1] "3jan1948"   "4jan1950"   "30avr1961"  "18juil1990"

> dates2
[1] "01/21/99 21:04:22" "03/28/99 22:19:55"
[3] "07/15/99 03:01:32"
```

Notez l'existence des fonctions `weekdays()` et `months()` qui permettent de récupérer le jour et le mois d'une date au format POSIXlt.

3.6.3 Opérations sur des dates

Lorsque l'on veut faire des manipulations de dates, il faut toujours commencer par convertir les dates et les heures en des objets ayant une classe `POSIXlt` ou `POSIXct`. R propose deux fonctions pour cela : `as.POSIXct()` qui représente le nombre de secondes depuis le 1er janvier 1970 comme un vecteur numérique et `as.POSIXlt()` qui est une liste nommée de vecteurs représentant :

- `sec` 0-61 : secondes.
- `min` 0-59 : minutes.
- `hour` 0-23 : heures.
- `mday` 1-31 : jour du mois.
- `mon` 0-11 : nombre de mois après le 1er mois de l'année.
- `year` : nombre d'années depuis 1900.
- `wday` 0-6 : jour de la semaine, débutant le dimanche.
- `yday` 0-365 : jour de l'année.
- `isdst` : indicateur de l'heure d'été. Positif si en vigueur, zéro sinon (négatif si inconnu).

Voici quelques instructions utilisant ces fonctions :

```
> z <- Sys.time() # Au format POSIXct.
> class(z) ; is.double(z)
[1] "POSIXct" "POSIXt"
[1] TRUE
```

```
> z
[1] "2014-06-25 21:40:43 EST"
> as.numeric(z) # Nombre de secondes depuis le 1er janvier 1970.
[1] 1403696443
> # On peut changer l'origine:
> as.POSIXct(as.numeric(z), origin="1960-01-01")
[1] "2004-06-24 21:40:43 EST"
> # Une quarantaine d'années se sont écoulées:
> as.numeric(z)/60/60/24/7/c(53,52)
[1] 43.79107 44.63320
> z <- as.POSIXlt(z)  # Au format POSIXlt.
> class(z) ; is.list(z)
[1] "POSIXlt" "POSIXt"
[1] TRUE
> z
[1] "2014-06-25 21:40:43 EST"

> names(z)
NULL
> z$year # Nombre d'années depuis 1900.
[1] 114
```

Notez que les fonctions `as.POSIXct()` et `as.POSIXlt()` peuvent être utilisées soit sur des vecteurs de valeurs numériques, soit sur des vecteurs de chaînes de caractères. Dans le premier cas, il faudra fournir une valeur au paramètre `origin` sous la forme d'une chaîne de caractères représentant une date. Dans l'autre cas, chaque chaîne de caractères devra être dans un format tel que `"2001-02-03"` ou `"2001/02/03"`, suivie de façon optionnelle par un espace et une heure au format `"14:52"` ou `"14:52:03"`. Il pourra être intéressant d'utiliser la fonction `strptime()` pour obtenir un format compatible avec ces fonctions (voir la table 3.3 pour une description).

```
> as.POSIXct("2001/02/03")
[1] "2001-02-03 EST"
> as.numeric(as.POSIXct("2001/02/03"))
[1] 981118800
> as.POSIXlt("2001/02/03")$wday
[1] 6
> lct <- Sys.getlocale("LC_TIME") # Récupérer les paramètres
                                  # régionaux.
> Sys.setlocale("LC_TIME","C")    # Voir la remarque en page
                                  # 120.
[1] "C"
> as.POSIXlt(strptime("27/mar/73",format="%d/%b/%y"))
[1] "1973-03-27 EST"
> Sys.setlocale("LC_TIME",lct)    # Redéfinir les paramètres
                                  # régionaux précédents.
[1] "fr_FR.UTF-8"
```

Remarque

La classe `Date` permet aussi de représenter des dates.

```
> z <- as.Date(c("2006-06-01", "2007-01-01"))
> class(z)
[1] "Date"
> z[1] + 100 # On ajoute 100 jours.
[1] "2006-09-09"
> z[2]-z[1]
Time difference of 214 days
> z[2] < z[1]
[1] FALSE
```

L'avantage de stocker des dates dans des objets ayant l'une des classes décrites ci-dessus est, hormis le joli affichage qui en découle, de pouvoir ensuite effectuer des opérations sur ces dates (différence entre deux dates, test d'antériorité, etc.), comme cela est illustré dans les exemples suivants.

```
> date2 <- as.POSIXlt("2009-04-15")
> date1 <- as.POSIXlt("2000-11-24")
> date2-date1
Time difference of 3064.042 days
> difftime(date2,date1,units="hours")
Time difference of 73537 hours
> date1 <= date2
[1] TRUE
```

Expert

Le *package* `chron` contient de nombreuses fonctionnalités pour gérer des données de dates.

SECTION 3.7

Structures de contrôle

Comme tout langage de programmation, R dispose des indispensables structures de contrôle qui permettent de diriger le flot d'exécution d'un programme.

3.7.1 Instructions de condition

• Instruction `switch()` (aiguiller)

Elle s'utilise de la façon suivante :

```
switch(<expr:test>,<expr:cas1>=<code1>,<expr:cas2>=<code2>, etc.)
```

Dans l'instruction ci-dessus, `<expr:test>` est soit un nombre, soit une chaîne de caractères. Cette instruction renvoie `<code1>` si `<expr:test>` vaut `<expr:cas1>`, `<code2>` si `<expr:test>` vaut `<expr:cas2>`, etc. Si `<expr:test>` n'est égal à aucun des `<expr:cas`i`>`, la fonction `switch()` renvoie alors `NULL`.

En voici un exemple d'utilisation :

```
> x <- rcauchy(10) # Génération de dix nombres aléatoires issus
                   # d'une loi de Cauchy.
> entree <- "moyenne"
> switch(entree,moyenne = mean(x),mediane = median(x))
[1] 3.842243
> entree <- "mediane"
> switch(entree,moyenne = mean(x),mediane = median(x))
[1] 0.6206986
> entree <- "variance"
> switch(entree,moyenne = mean(x),mediane = median(x))
```

Remarque

Il est possible de fournir une seule valeur non nommée, c'est-à-dire un `<code`i`>` sans le `<expr:cas`i`>=` associé. Cela aura pour effet de renvoyer cette valeur à la place de `NULL` lorsque la valeur du paramètre `EXPR` ne sera égale à aucun des `cas`.

```
> switch(EXPR = "b", a=4, b=2:3, "Sinon: rien")
[1] 2 3
> switch(EXPR = "QQ", a=4, b=2:3, "Sinon: rien")
[1] "Sinon: rien"
```

• Instructions `if` et `else` (si et sinon)

L'instruction conditionnelle `if` est utilisée sous les deux formes suivantes :

```
if (<cond>) <expr:vrai>
ou
if (<cond>) <expr:vrai> else <expr:faux>
```

Le paramètre `<cond>` doit prendre l'une des valeurs `TRUE` ou `FALSE`. Notez que `<cond>` est d'abord transformé en `as.logical(<cond>)`, ce qui autorise l'uti-

lisation de nombres réels (`0` étant le seul nombre qui sera converti en `FALSE`) ou encore des chaînes de caractères `"T"` ou `"TRUE"` et `"F"` ou `"FALSE"`. Notez également que `<cond>` doit être de longueur 1. Si ce n'est pas le cas, seule la première composante de `<cond>` sera prise en compte et R affichera un message d'avertissement.
Bien évidemment, dans la pratique, `<cond>` sera souvent le résultat d'opérations logiques élaborées faisant intervenir les opérateurs logiques que nous avons décrits un peu plus haut. Voici un exemple d'utilisation de ces instructions. Assurez-vous de bien le comprendre.

```
> if (TRUE) 1+1
[1] 2
> x <- 2
> y <- -3
> if (x <= y) {
+   z <- y-x
+   print("x plus petit que y")
+ } else {
+   z <- x-y
+   print("x plus grand que y")
+   z
+ }
[1] "x plus grand que y"
[1] 5
```

Attention

Notez que les blocs d'instructions multiples doivent être délimités par des accolades.

Astuce

La fonction `ifelse()` permet d'exécuter l'une ou l'autre de deux instructions appliquées à un vecteur suivant que les valeurs d'une condition logique sont vraies ou fausses. Par exemple :

```
> x <- c(3:-2)
> sqrt(ifelse(x >= 0, x, NA))
[1] 1.732051 1.414214 1.000000 0.000000       NA       NA
```

• Opérateurs logiques à privilégier

Il est important de prendre garde à la bonne utilisation des opérateurs logiques. Dans le contexte des instructions de condition, nous conseillons :

– d'utiliser `x && y` plutôt que `x & y` ;

– d'utiliser `x || y` plutôt que `x | y`.

Nous l'illustrons sur l'exemple ci-dessous. Si on utilise la forme longue `&&`, les évaluations des conditions logiques après le `if` sont effectuées de gauche à droite tant que les conditions successives sont réalisées (à `TRUE`).

```
> as.logical(x <- 2) # as.logical(x, nombre réel non nul) renvoie
                     # TRUE.
[1] TRUE
> x
[1] 2
> rm(x) # On efface x.
> if (FALSE & as.logical(x <- 2)) 4*7  # <cond> est évaluée à
                                       # FALSE. Les deux
                                       # parties de <cond> sont
                                       # évaluées.
> x
[1] 2
> if (FALSE && (x <- 3)) 4*7 # En utilisant &&, seule la première
                             # partie de <cond> est évaluée.
> x
[1] 2
```

Bien entendu, pour la forme longue `||` les évaluations des conditions logiques après le `if` sont effectuées de gauche à droite jusqu'à ce qu'il soit rencontré un `TRUE`. Ces formes longues seront donc à privilégier lors de l'écriture de vos programmes puisqu'elles sont plus rapidement traitées.

• La fonction `all.equal()` pour l'instruction `if()`

Lors de l'utilisation de l'instruction `if()` (surtout lorsque des valeurs réelles sont en jeu), il convient :
– d'utiliser `isTRUE(all.equal(x,y))` plutôt que `x == y` ;
– d'utiliser `!isTRUE(all.equal(x,y))` plutôt que `x != y`.

Nous l'illustrons sur l'exemple suivant, dans lequel `x` et `y` ne contiennent pas exactement la même valeur du fait de la limite de précision de la machine :

```
> x <- 0.1
> y <- 0.1
> x==y
[1] TRUE
> x <- 0.2-0.1   # Il semblerait que
> y <- 0.3-0.2   # x soit égal à y.
> x == y         # Ce n'est pas le cas, car l'ordinateur a une
                 # précision limitée. Voir la section
                 # 3.9.
[1] FALSE
```

```
> all.equal(x,y,tolerance=10^-6) # La fonction all.equal() permet
                                 # de régler ce problème.
[1] TRUE
```

Astuce

La fonction `all.equal()` contient un paramètre `tolerance` permettant de se fixer une limite de tolérance pour accepter la différence entre deux valeurs comme étant nulle.

3.7.2 Instructions de boucles

Une boucle est une structure de contrôle destinée à exécuter une portion de code plusieurs fois de suite, tant qu'une condition de sortie n'est pas satisfaite ou bien qu'un nombre de boucles spécifié à l'avance n'a pas été atteint.

R possède trois instructions de boucle : `for()`, `while()` et `repeat`. Les mots réservés `next` et `break` fournissent par ailleurs un contrôle supplémentaire de l'exécution d'un code. L'instruction `break` provoque une sortie immédiate de la boucle en cours d'exécution. L'instruction `next` amène le curseur d'exécution du programme au départ de la boucle. La prochaine itération de la boucle (s'il y en a une) est ensuite exécutée. Aucune instruction après `next` dans la boucle courante n'est exécutée.

• **Instruction `for()` (pour)**

La syntaxe de cette instruction est la suivante :

```
for (i in vect) <Instructions>
```

Voici deux exemples :

```
> for (i in 1:3) print(i)
[1] 1
[1] 2
[1] 3
> x <- c(1,3,7,2)
> for (var in x) print(2*var)
[1] 2
[1] 6
[1] 14
[1] 4
```

Astuce

La liste d'instructions ci-dessous fait défiler un compteur décroissant :
`n <- 100 ; for (i in 1:n) {flush.console();cat(n-i,"\r"); Sys.sleep(0.1)}`

• **Instruction `while()` (tant que)**

La syntaxe de cette instruction est la suivante :

```
while(<condition>) <expression>
```

Par exemple :

```
> x <- 2
> y <- 1
> while(x+y<7) x <- x+y
> x
[1] 6
```

• **Instructions `next` (suivant), `break` (interrompre)**

```
> for (i in 1:4) {
+   if (i == 3) break
+   for (j in 6:8) {
+    if (j==7) next
+    j <- i+j
+   }
+ }
> i
[1] 3
> j
[1] 10
```

• **Instructions `repeat` (répéter), `break` (interrompre)**

```
> i <- 0
> repeat {
+   i<-i+1
+   if (i==4) break
+ }
```

Attention

Lorsque cela est possible, il est préférable d'éviter l'utilisation de boucles en R, car cela entraîne souvent un accroissement du temps de calcul (qui peut être mesuré grâce à la fonction `system.time()`). La plupart des opérations en R sont en effet vectorisées, c'est-à-dire qu'elles peuvent opérer sur des vecteurs, et ce calcul est effectué dans un langage compilé, ce qui est beaucoup plus rapide.

```
> system.time(for (i in 1:1000000) sqrt(i))
   user  system elapsed
  0.156   0.028   0.182
> system.time(sqrt(1:1000000))
   user  system elapsed
  0.012   0.000   0.010
```

Par ailleurs, des fonctions comme `apply()`, `tapply()` et `sapply()` fournissent une façon de réaliser des boucles de façon implicite et souvent très commode.

SECTION 3.8

Création de fonctions

Nous avons déjà vu en 1.1.4, page 47, quelques notions succinctes sur l'exécution de fonctions R. Le langage R offre également la possibilité de créer ses propres fonctions. Nous en proposons un aperçu dans cette section. Le lecteur devra s'attarder sur chacun des codes présentés afin de s'assurer qu'il en comprenne bien le fonctionnement.

Renvoi

Une présentation plus formelle de la programmation de fonctions est présentée au chapitre 6.

Afin d'illustrer simplement le processus de création de fonctions, nous allons nous appuyer sur le calcul de l'indice de masse corporelle (IMC) à partir du poids (en kg) et de la taille (en m) suivant la formule bien connue

$$IMC = \frac{Poids}{Taille^2}.$$

Celle-ci se programme facilement en R de la façon suivante :

```
> IMC <- function(poids,taille) {
+  imc <- poids/taille^2
+  names(imc) <- "IMC"
```

```
+  return(imc)
+ }
```

Attention

L'usage de la fonction `return()` n'est pas obligatoire dans le code ci-dessus, mais vous devriez prendre l'habitude de l'utiliser. Il existe en effet des contextes où cela s'avère absolument nécessaire :

```
> f <- function(x) {
+  res <- vector("numeric",length(x))
+  for (i in 1:10) {
+   res[i] <- rnorm(1) + x[i]
+  }
+ # Omission de return(res)
+ }
> f(1:10) # Ne renvoie rien!
```

Exécutons maintenant la fonction `IMC()` que nous avons codée ci-dessus.

```
> IMC(70,1.82)
     IMC
21.13271
> IMC(1.82,70) # Nous pouvons noter qu'il n'est pas possible de
               # permuter les arguments d'une fonction,
         IMC
0.0003714286
> IMC(taille=1.82,poids=70) # sauf s'ils sont précédés de leurs
                            # "intitulés".
     IMC
21.13271
```

La fonction que nous venons de proposer ne renvoie qu'une seule valeur. Le code ci-dessous permettra de retourner une liste de plusieurs variables.

```
> IMC <- function(poids,taille) {
+  imc <- poids/taille^2
+  res <- list(poids,taille,imc)
+ return(res)
+ }
```

L'instruction ci-dessous permet de se rendre compte que la nouvelle fonction `IMC()` renvoie bien une liste, dont les éléments ne sont pas nommés.

```
> IMC(70,1.82)
[[1]]
[1] 70
[[2]]
[1] 1.82
[[3]]
[1] 21.13271
```

Prise en main

Programmez une fonction nommée `biroot()` permettant le calcul des racines d'un polynôme de degré 2, c'est-à-dire les valeurs x solutions de l'équation $ax^2 + bx + c = 0$. Nous rappelons que celles-ci peuvent être des nombres complexes et sont définies par

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

avec

$$\Delta = b^2 - 4ac.$$

Si $\Delta = 0$, il y aura une solution unique et vous ne renverrez que celle-ci. Si $\Delta < 0$, alors la racine carrée de Δ est le nombre complexe z tel que $z^2 = \Delta$. (Indice : utilisez l'instruction `as.complex(Delta)`)
Vous comparerez les résultats de votre fonction avec ceux obtenus au moyen de la fonction `polyroot()` (pensez à utiliser l'aide en ligne sur cette fonction : `?polyroot`).

Pour nommer les éléments de la liste, on peut utiliser le code suivant.

```
> IMC <- function(poids,taille) {
+  imc <- poids/taille^2
+  res <- list(Poids=poids,Taille=taille,IMC=imc)
+ return(res)
+ }
```

Voici le résultat de cette opération.

```
> IMC(70,1.82)
$Poids
[1] 70
$Taille
[1] 1.82
$IMC
[1] 21.13271
```

Supposons maintenant que nous cherchions à calculer l'IMC de plusieurs individus, par exemple de Benoit et de Pierre :

```
> Benoit <- c(74,1.90)
> Pierre <- c(70,1.82)
> donnees <- rbind(Benoit,Pierre)
```

Nous pourrions penser à utiliser le code ci-dessous.

```
> for (i in 1:2) {
+  print(IMC(donnees[i,1],donnees[i,2]))
+ }
$Poids
Benoit
    74
$Taille
Benoit
   1.9
$IMC
  Benoit
20.49861
$Poids
Pierre
    70
$Taille
Pierre
  1.82
$IMC
  Pierre
21.13271
```

Mais le lecteur avisé aura remarqué que de nombreuses fonctions R (comme les opérateurs de division, de multiplication ou d'exponentiation) fonctionnent très bien avec des vecteurs. Il ne sera donc pas étonné par la sortie suivante :

```
> IMC(c(70,74),c(1.82,1.90))
$Poids
[1] 70 74
$Taille
[1] 1.82 1.90
$IMC
[1] 21.13271 20.49861
```

Le code ci-dessous illustre l'utilisation d'un paramètre possédant une valeur par défaut, ainsi que celui de la fonction `stop()` qui permet de gérer certaines erreurs d'entrée.

```
> IMC <- function(poids,taille,unite.taille="m") {
+  if (length(poids) != length(taille))
+      stop("Les vecteurs poids et taille doivent
+                          être de même longueur.")
+  if (unite.taille == "cm") taille <- taille/100
+  imc <- poids/taille^2
+  res <- list(Poids=poids,Taille=taille,IMC=imc)
+ return(res)
+ }
```

Expert

L'utilisation de la fonction `stop()` peut parfois entraîner des désagréments. Par exemple, dans une étude de simulations, il arrive souvent que l'on appelle une même fonction à plusieurs reprises. Si lors d'un appel, la fonction en question renvoie une erreur, la simulation est interrompue. Il est alors conseillé d'utiliser la fonction `try()`. En cas d'erreur, celle-ci stocke le message d'erreur dans un objet sans interrompre la simulation.

```
> set.seed(123)
> x <- rnorm(50)
> execute <- function(x) {
+  x <- sample(x,replace=TRUE)
+  if(length(unique(x)) > 30) mean(x)
+  else {stop("trop de points identiques")}
+ }
> res <- lapply(1:100, function(i) try(execute(x), TRUE))

> res[8:10]
[[1]]
[1] 0.2030573
[[2]]
[1] "Erreur dans execute(x) : trop de points identiques"
attr(,"class")
[1] "try-error"
[[3]]
[1] 0.235046
```

Il est possible d'attribuer à un individu un type de corpulence en fonction de sa valeur d'IMC. La correspondance entre ces deux mesures est fournie dans le tableau 3.4.

Table 3.4 – Correspondance entre IMC et types de corpulence.

famine	maigreur	corpulence normale	surpoids	obésité modérée	obésité sévère	obésité morbide
[15; 16.5[	[16.5; 18.5[	[18.5; 25[	[25; 30[	[30; 35[	[35; 40[	[40; 41]

La fonction ci-dessous permet de renvoyer à un utilisateur son type de corpulence à partir de la valeur de son IMC.

```
> affiche.corpulence <- function(imc) {
+  if (imc<16.5) corpulence <- "famine" else {
+   if (imc<18.5) corpulence <- "maigreur" else {
+    if (imc<25) corpulence <- "corpulence normale" else {
+     if (imc<30) corpulence <- "surpoids" else {
```

```
+      if (imc<35) corpulence <- "obésité modérée" else {
+       if (imc<40) corpulence <- "obésité sévère" else {
+        corpulence <- "obésité morbide"}}}}}}
+  cat(paste("Votre IMC vaut: ",round(imc,3),".\nCela correspond
+              au type de corpulence: ",corpulence,".\n",sep=""))
+ }
```

Utilisons là sur un exemple.

```
> affiche.corpulence(IMC(70,1.82)$IMC)
Votre IMC vaut: 21.133.
Cela correspond
             au type de corpulence: corpulence normale.
```

Il est en fait possible de simplifier le code de la fonction `affiche.corpulence()` au moyen de la fonction `switch()`, comme nous pouvons le voir ci-dessous :

```
> affiche.corpulence <- function(imc) {
+  intervalles.IMC <- c(15,16.5,18.5,25,30,35,40,41)
+  code <- as.character(rank(c(imc,intervalles.IMC),
+                            ties.method="max")[1])
+  corpulence <- switch(code,"2"="famine","3"="maigreur",
+   "4"="corpulence normale","5"="surpoids","6"="obésité modérée",
+   "7"="obésité sévère","8"="obésité morbide")
+  cat(paste("Votre IMC vaut: ",round(imc,3),".\nCela correspond
+              au type de corpulence: ",corpulence,".\n",sep=""))
+ }

> affiche.corpulence(IMC(70,1.82)$IMC)
Votre IMC vaut: 21.133.
Cela correspond
             au type de corpulence: corpulence normale.
```

Toutefois, cette fonction renvoie des résultats erronés lorsqu'elle est utilisée sur un vecteur :

```
> affiche.corpulence(IMC(c(70,74),c(1.82,1.90))$IMC)
Votre IMC vaut: 21.133.
Cela correspond
             au type de corpulence: surpoids.
 Votre IMC vaut: 20.499.
Cela correspond
             au type de corpulence: surpoids.
```

Il est possible d'améliorer cette fonction afin qu'elle fonctionne sur plusieurs individus simultanément. Vous noterez également l'utilisation du mot réservé `NULL` et de la fonction `is.null()` qui permettent de gérer de façon astucieuse l'argument `noms`.

```
> affiche.corpulence <- function(imc,noms=NULL) {
+  intervalles.IMC <- c(15,16.5,18.5,25,30,35,40,41)
```

```
+  n <- length(imc)
+  if (is.null(noms)) noms <- paste("Sujet numéro",1:n)
+  if (length(noms) != n) stop(paste("Vous devez entrer un vecteur
+                                     de 'noms' de longueur",n))
+  code <- vector("integer",length=n)
+  corpulence <- vector("character",length=n)
+  for (i in 1:n) {
+   code[i] <- as.character(rank(c(imc[i],intervalles.IMC),
+                  ties.method="max")[1])
+   corpulence[i] <- switch(code[i],"2"="famine","3"="maigreur",
+  "4"="corpulence normale","5"="surpoids","6"="obésité modérée",
+  "7"="obésité sévère","8"="obésité morbide")
+  cat(paste(noms[i],":\nVotre IMC vaut: ",round(imc[i],3),".
+    \nCela correspond au type de corpulence: ",
+      corpulence[i],".\n",sep=""))
+  }
+ }
```

Voyons son utilisation sur un exemple.

```
> affiche.corpulence(IMC(c(70,74),c(1.82,1.90))$IMC)
Sujet numéro 1:
Votre IMC vaut: 21.133.

Cela correspond au type de corpulence: corpulence normale.
Sujet numéro 2:
Votre IMC vaut: 20.499.

Cela correspond au type de corpulence: corpulence normale.
```

Astuce

Dans la fonction précédente, nous avons pu déclarer, directement avec la bonne longueur, les vecteurs `code` et `corpulence` à l'aide de la fonction `vector()`. Il est néanmoins possible de gérer un vecteur que l'on veut éviter de dimensionner au départ (soit car l'on ne connaît pas sa longueur à l'avance, soit pour une autre raison). Voyons ceci via l'exemple de la construction des premiers termes de la suite de Fibonnacci, qui est définie par :

$$a_1 = 0, a_2 = 1, a_i = a_{i-1} + a_{i-2}, \forall i \geq 3.$$

```
> a <- c(0, 1)
> for (i in 3:10) a <- c(a, a[i-1]+a[i-2])
> a
 [1]  0  1  1  2  3  5  8 13 21 34
```

Il est également possible d'utiliser la fonction `while()` afin d'afficher tous les termes de la suite de Fibonnacci jusqu'au premier terme supérieur à 1 000.

```
> a <- c(0,1)
> i <- 3
> while(a[i-1]<1000) {
+  a <- c(a, a[i-1]+a[i-2])
+  i <- i+1
+ }
> a
 [1]    0    1    1    2    3    5    8   13   21   34   55
[12]   89  144  233  377  610  987 1597
```

On peut aussi utiliser l'instruction `break` pour obtenir un programme légèrement différent.

```
> a <- c(0,1)
> i <- 3
> while(TRUE) { # Crée une boucle infinie.
+  a <- c(a, a[i-1]+a[i-2])
+  if (a[i]>1000) break; # Permet d'interrompre la boucle.
+  i <- i+1
+ }
> a
 [1]    0    1    1    2    3    5    8   13   21   34   55
[12]   89  144  233  377  610  987 1597
```

Attention

La façon de procéder présentée dans l'astuce précédente n'est pas forcément optimale en termes d'allocation de mémoire, comme nous le verrons au chapitre 6.

SECTION 3.9

† Représentation des nombres à virgule fixe, flottante

Cette note, un peu technique, vise à aider l'utilisateur de R à identifier et éviter certaines erreurs de calcul dues à l'utilisation des nombres dits à *virgule flottante*.

3.9.1 Représentation d'un nombre à l'aide d'une base

Ayant fixé une *base* b (valeur entière supérieure ou égale à 2), tout nombre réel $x \in \mathbb{R}$ peut s'écrire sous une représentation dite à *virgule fixe* par :

$$x = \sum_{i=+\infty}^{i=-\infty} m_i b^i,$$

où les coefficients de la représentation m_i (aussi appelés chiffres) appartiennent à $\{0, 1, \ldots, b-1\}$, $\forall i \in \mathbb{Z}$.

Par exemple, le nombre $x = 10.625$ peut s'écrire en *représentation décimale* ($b = 10$) sous la forme :

$$x = \mathbf{1} \times 10^1 + \mathbf{0} \times 10^0 + \mathbf{6} \times 10^{-1} + \mathbf{2} \times 10^{-2} + \mathbf{5} \times 10^{-3},$$

d'où l'écriture **10.625** qui donne les coefficients de la représentation (aussi appelés *chiffres*). En notant que $10.625 = 8 + 2 + 0.5 + 0.125$, ce même nombre x peut s'écrire en *représentation binaire* ($b = 2$; les seuls chiffres utilisés sont donc 0 et 1) sous la forme :

$$x = \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{1} \times 2^1 + \mathbf{0} \times 2^0 + \mathbf{1} \times 2^{-1} + \mathbf{0} \times 2^{-2} + \mathbf{1} \times 2^{-3}.$$

Ainsi, on écrit aussi la représentation du nombre 10.625 sous la forme (dite binaire) **1010.101**, qui explicite les coefficients de la représentation binaire.

Astuce

Nous avons intégré dans le *package* associé à ce livre les fonctions `dec2bin()` et `bin2dec()` permettant de passer d'une représentation décimale à une représentation binaire, ou l'inverse.

```
> bin2dec(1010.101)
[1] 10.625
> dec2bin(10.625,3)
[1] "1010.101"
```

Remarque

Un nombre dont l'écriture en base 10 est limitée (nombre fini de chiffres après la virgule) est appelé un *nombre décimal*. Un nombre dont l'écriture en base 2 est limitée (nombre fini de chiffres après la virgule) est appelé un *nombre dyadique*. Certains nombres ne sont bien évidemment ni décimaux, ni dyadiques. On peut noter qu'un nombre dyadique est toujours décimal

mais que la réciproque est fausse. Par conséquent, une perte d'information est toujours possible lors du passage d'une représentation à une autre. Par ailleurs, les fractions dyadiques décimales ont le même nombre de chiffres que leurs équivalents binaires, alors que les valeurs décimales qui ne sont pas dyadiques ont des équivalents binaires infinis.

L'ordinateur est une machine conçue pour travailler avec des nombres binaires car les deux chiffres 0 et 1 s'y traduisent facilement par l'absence ou par le passage d'un courant électrique. Puisque les représentations à virgule fixe sont souvent très dispendieuses en nombres de *bits* (le *bit*, *binary digit* en anglais, est un chiffre binaire, c'est-à-dire 0 ou 1), l'ordinateur fonctionne en général avec des représentations dites à virgule flottante. Celles-ci sont présentées dans la sous-section suivante.

3.9.2 Représentation à virgule flottante

3.9.2.1 Définitions

Ayant fixé une *base* b (ex. : $b = 2$ pour le système binaire ou $b = 10$ pour le système décimal pour ne citer que les plus classiques), tout nombre réel $x \in \mathbb{R}$, peut être écrit sous la forme :

$$x = (-1)^s m b^e$$

où

- s est appelé le *bit* de signe (de x), et vaut 0 ou 1 ;
- m est appelé la *mantisse* et peut s'écrire $m = m_1.m_2m_3\cdots m_\infty$ où chaque $m_i \in \{0, 1, \ldots, b-1\}$ est appelé un *digit* (et où le premier « . » après m_1 symbolise le séparateur décimal, communément appelé la « virgule » en français) ;
- $e \in \mathbb{Z}$ est appelé l'*exposant*.

Cette écriture est dite *représentation à virgule flottante* de x dans la base b. L'entier m_1 est la *partie entière* de la mantisse, le reste des chiffres (*digits*), à savoir $m_2 \cdots m_\infty$ est la *partie fractionnaire* de la mantisse. Notons qu'il faut imposer une contrainte de non-nullité du premier chiffre ($m_1 \neq 0$) afin d'assurer l'unicité de la représentation. Bien sûr, cette contrainte ne peut pas s'appliquer pour le cas particulier du zéro.

Donnons un premier exemple. Dans un système décimal, on peut écrire le nombre -0.6345 sous une représentation à virgule flottante en prenant $s = 1$, $m_1 = 6$, $m_2 = 3$, $m_3 = 4$, $m_4 = 5$, $m_i = 0, \forall i > 4$, $b = 10$ et $e = -1$:

$$-0.6345 = (-1)^1 6.345 \times 10^{-1}.$$

Remarque

On comprend ici la raison de l'appelation « virgule flottante », puisque la « virgule » (le point en fait dans la notation anglo-saxonne utilisée dans ce livre) a été déplacée pour exprimer le même nombre de façon différente.

En fait, dans un ordinateur, tout nombre réel x est représenté en utilisant une base binaire, et à l'aide de *bits* (0 ou 1), en utilisant la formule légèrement différente suivante :

$$x = (-1)^s(1+f)2^{(e-1023)} \tag{3.1}$$

où s est un entier codé sur un seul *bit* (appelé *bit* de signe), e est un nombre entier positif ou nul codé sur 11 *bits* (pouvant donc prendre les valeurs de 0 à $2^{11}-1 = 2\,047$), et $f \in [0;1)$ est un nombre codé sur 52 *bits* qui s'écrit donc

$$f = \sum_{i=1}^{i=52} k_i 2^{-i}, \tag{3.2}$$

où les k_i peuvent prendre les valeurs 0 ou 1. Une valeur de 0 pour e indiquera par convention que le nombre x vaut 0, alors qu'une valeur de 2 047 pour e indiquera par convention que x vaut `+Inf` ou `-Inf`, suivant la valeur de s. Illustrons ceci sur deux exemples.

```
> s <- 1; e <- 2047; f <- 0; x <- (-1)^s * 2^(e-1023) * (1+f); x
[1] -Inf
> s <- 0; e <- 2047; f <- 0; x <- (-1)^s * 2^(e-1023) * (1+f); x
[1] Inf
```

3.9.2.2 Limite de cette représentation due à la mantisse

La grande majorité des ordinateurs actuels utilisent une représentation en virgule flottante. Toutefois, il est primordial de noter que, du fait de leurs limites physiques, cette représentation est imparfaite. En effet, la mantisse m ne peut avoir qu'un nombre limité (fini) de *digits*, et l'entier relatif e est borné, $e \in [e_{min}, e_{max}]$, puisqu'on ne peut pas coder une infinité d'entiers sur une machine physique (un ordinateur est évidemment constitué d'un nombre fini de composantes).

Un exemple simple permettant de comprendre cette limitation est le suivant. La représentation en virgule flottante dans une base décimale du nombre réel 1/3 s'écrit :

$$1/3 = (-1)^0 \times 3.33333\cdots333\cdots \times 10^{-1}.$$

Il est évidemment impossible de stocker une infinité de 3 sur un ordinateur. Ceci est encore plus frappant avec le nombre $\pi = 3.141592653589793\cdots$ pour lequel il n'est pas possible de trouver une suite finie de chiffres qui se répète

dans les chiffres après la virgule. Mais c'est aussi vrai pour des nombres qui pourraient paraître plus simples, comme 0.1 ou 0.9 par exemple.

Ceci est illustré par la sortie ci-dessous, pour le nombre 0.9, qui peut s'écrire en virgule flottante sous la forme $0.9 = (-1)^0 \times 2^{-1} \times (1 + f)$ avec $f = 0.8$. Mais, puisque f n'est codé « que » sur 52 *bits* (voir (3.2)), l'ordinateur n'en conservera qu'une approximation. La fonction d'affichage `formatC()` permet de se faire une idée de la valeur utilisée par R.

```
> dec2bin(0.8,52)
[1] "0.11001100110011001100110011001100110011001100110011001101"
> f <-  2^(-1)+2^(-2)+2^(-5)+2^(-6)+2^(-9)+2^(-10)+2^(-13)+
+  2^(-14)+2^(-17)+2^(-18)+2^(-21)+2^(-22)+2^(-25)+2^(-26)+
+  2^(-29)+2^(-30)+2^(-33)+2^(-34)+2^(-37)+2^(-38)+2^(-41)+
+  2^(-42)+2^(-45)+2^(-46)+2^(-49)+2^(-50)+2^(-52)
> f # Attention, l'affichage est en fait ici tronqué.
[1] 0.8
> formatC(f,50) # Partie fractionnaire de la mantisse la plus
                # proche de 0.8 que l'ordinateur est capable
                # d'obtenir sur 52 bits.
[1] "0.80000000000000004440892098500626161694526672363281"
> formatC(0.8,50)
[1] "0.80000000000000004440892098500626161694526672363281"
```

Nous pouvons maintenant afficher la valeur conservée et utilisée par l'ordinateur en lieu et place de la valeur 0.9 :

```
> formatC(0.9,50)
[1] "0.90000000000000022204460492503130808472633361816 41"
> formatC((-1)^0 * 2^(-1) * (1+f),50)
[1] "0.90000000000000022204460492503130808472633361816 41"
> formatC((-1)^0 * 2^(-1) * (1+0.8),50)
[1] "0.90000000000000022204460492503130808472633361816 41"
```

3.9.2.3 Éviter certaines chausse-trappes numériques

Commençons par apporter un éclairage sur une bizarrerie numérique.

```
> identical(1.0-0.9,0.1)
[1] FALSE
> (1.0-0.9) == 0.1
[1] FALSE
```

Il nous est maintenant possible de comprendre cette sortie. En effet, les nombres $1.0 - 0.9$ et 0.1 son représentés de manière différente, car bien que 1.0 soit parfaitement représenté, 0.1 et 0.9 ne le sont pas

```
> formatC(1.0,50)
[1] "                                                1"
```

```
> formatC(0.9,50)
[1] "0.90000000000000022204460492503130808472633361816 41"
> formatC(1.0-0.9,50)
[1] "0.09999999999999997779553950749686919152736663818 3594"
> formatC(0.1,50)
[1] "0.10000000000000000555111512312578270211815834045 41"
```

et par conséquent, on a :

```
> formatC(1.0 -
+  0.90000000000000022204460492503130808472633361816 41,50)
[1] "0.09999999999999997779553950749686919152736663818 3594"
> # qui est bien différent de:
> formatC(0.1,50)
[1] "0.10000000000000000555111512312578270211815834045 41"
```

Astuce

Il est plus judicieux d'utiliser la fonction `all.equal()` pour comparer deux nombres, car celle-ci comporte un argument de tolérance numérique.

```
> all.equal(1.0-0.9,0.1,tol=10^(-6))
[1] TRUE
```

Attention

Il ne faut pas utiliser des constructions comme `while(x != 0)` ou `if (x != 0)`. En effet, si `x` prend par hasard une valeur comme 1.0-0.9-0.1 lors de l'exécution de votre code, les deux instructions précédentes ne se comporteront pas comme prévu. Il faut les remplacer par `while(!isTRUE(all.equal(x,0)))` et `if(!isTRUE(all.equal(x,0)))`.

Nous laissons au lecteur le soin de comprendre les sorties ci-dessous.

```
> as.integer(100*(1-.34))
[1] 65
> floor(100*(1-.34))
[1] 65
> round(100*(1-.34))
[1] 66
> 100*(1-.34)-66
[1] -1.421085e-14
> floor(100*(1-.38))
[1] 62
> round(100*(1-.38))
[1] 62
> 100*(1-.38)-62
[1] 0
```

```
> 0.900000000000000022204460492503130808472633361816 41 - 0.9
[1] 0
```

Renvoi

La lecture de [10] pourra s'avérer utile dans ce contexte.

3.9.2.4 Limite de cette représentation due à l'exposant

- Plus petit et plus grand nombre pouvant être représenté.

Un autre problème entraîné par la représentation en virgule flottante est le suivant. Puisque l'exposant e de la représentation

$$x = (-1)^s(1+f)2^{(e-1023)} \tag{3.3}$$

est forcément borné (car codé sur 11 *bits*), il existe un plus petit et un plus grand nombre réel qu'il est possible de représenter à l'aide d'un ordinateur donné. La tentative de représenter un nombre en dehors de cette plage devrait produire, respectivement, un débordement par valeur inférieure (*underflow*) ou par valeur supérieure (*overflow*). R est toutefois bien conçu et renverra plutôt `-Inf` ou `+Inf`. Illustrons ceci via quelques instructions R.

```
> .Machine$double.xmin # Plus petit nombre réel pouvant être
                       # codé:
[1] 2.225074e-308
> # que l'on retrouve par:
> s <- 0; e <- 0; f <- sum(2^(-(1:52)))
> x <- (-1)^s * 2^(e-1023) * (1+f); x
[1] 2.225074e-308
> .Machine$double.xmax # Plus grand nombre réel pouvant être
                       # codé:
[1] 1.797693e+308
> # que l'on retrouve par:
> s <- 0; e <- 2046; f <- sum(2^(-(1:52)))
> x=(-1)^s * 2^(e-1023) * (1+f); x
[1] 1.797693e+308
```

Astuce

Le *package* `Brobdingnag` permet de gérer des nombres beaucoup plus grands.

- Trous entre deux nombres.

Une dernière mise en garde importante vaut la peine d'être faite. Il est en effet important de noter qu'il n'est pas possible d'obtenir certains nombres

entre `.Machine$double.xmin` et `.Machine$double.xmax` (même les nombres entiers).

Prenons par exemple le nombre 2^{150}, qui sera représenté par l'ordinateur sous la forme :

$$(-1)^0(1+0)2^{(1173-1023)}.$$

Alors, le nombre immédiatement supérieur que l'ordinateur pourra coder sera donné pour les valeurs $s = 0$, $e = 1173$ et $f = 2^{-52}$ (plus petite valeur fractionnaire non nulle de la mantisse), soit :

$$(-1)^0(1+2^{-52})2^{150} = 2^{150} + 2^{150-52} = 2^{150} + 2^{98}.$$

Il peut donc y avoir un énorme « trou » (d'une longueur de $2^{98} \approx 3.2e+29$ ici) entre deux nombres « consécutifs » !

Ceci permet donc d'expliquer l'étrangeté de la sortie ci-dessous :

```
> a <- 2^150; b <- a + 2^97; b == a
[1] TRUE
> b-a
[1] 0
> a <- 2^150; b <- a + 2^98; b == a
[1] FALSE
> b-a
[1] 3.169127e+29
```

Astuce

D'autres informations sur les limites de l'ordinateur dans la représentation des nombres sont accessibles au travers de l'instruction `.Machine` :

```
> noquote(unlist(format(.Machine)))
           double.eps        double.neg.eps
         2.220446e-16          1.110223e-16
          double.xmin           double.xmax
        2.225074e-308         1.797693e+308
          double.base          double.digits
                     2                    53
      double.rounding           double.guard
                     5                     0
    double.ulp.digits double.neg.ulp.digits
                   -52                   -53
      double.exponent         double.min.exp
                    11                 -1022
       double.max.exp           integer.max
                  1024            2147483647
         sizeof.long       sizeof.longlong
```

```
                8                          8
sizeof.longdouble          sizeof.pointer
               16                          8
```

Renvoi

Le lecteur désireux d'approfondir ses connaissances dans ce domaine pourra lire le document *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, disponible à l'adresse : `http://biostatisticien.eu/springeR/FloatingPoint.pdf`.

Termes à retenir

`length()` : longueur d'un vecteur
`sort()` : ordonne les éléments d'un vecteur
`rev()` : réarrange les éléments d'un vecteur en sens inverse
`order()` : renvoie le vecteur des rangs de classement des éléments de son paramètre effectif
`unique()` : enlève les doublons d'un vecteur
`dim()` : taille de la matrice ou du *data.frame*
`nrow(), ncol()` : nombre de lignes, de colonnes
`dimnames()` : noms des lignes et des colonnes
`rownames(), colnames()` : noms des lignes, des colonnes
`rbind(), cbind()` : fusion de lignes, de colonnes d'une matrice
`merge()` : fusion intelligente de colonnes
`apply()` : appliquer une fonction aux lignes ou aux colonnes d'une matrice
`lapply(), sapply()` : applique une fonction aux éléments d'une liste
`<, <=, >, >=, ==, !=` : opérateurs logiques de comparaison
`!, &, &&, |, ||` : opérateurs logiques terme à terme
`any(x)` : renvoie `TRUE` si l'un des x_i est vrai
`all(x)` : renvoie `TRUE` si tous les x_i sont vrais
`if`, `else`, `switch` : instructions de condition
`for()`, `while()`, `repeat` : instructions de boucle
`"["()` : opérateur d'extraction pour les vecteurs et les matrices
`"[["()` : opérateur d'extraction pour les listes
`which()` : donne les indices des valeurs `TRUE` d'un objet de logiques
`nchar()` : nombre de symboles d'une chaîne
`paste()` : concaténation de deux chaînes
`substring()` : extraction de sous-chaînes
`strsplit()` : découpage de chaînes
`grep()` : recherche de motif dans une chaîne
`sub(), gsub()` : remplacer les occurrences d'un motif dans une chaîne
`Sys.time()` : afficher la date
`strptime()` : extraction de dates dans une chaîne
`as.POSIXlt()` : convertir au format POSIXlt
`difftime()` : calcule la différence entre deux dates

Exercices

3.1- Que renvoie cette instruction : `c(1,4)*c(2,3)` ?

3.2- Que renvoie cette instruction : `matrix(1:2,ncol=2,nrow=2)` ?

3.3- Comment pouvez-vous récupérer les noms des lignes et des colonnes d'un *data.frame* ?

3.4- Donnez l'instruction permettant de fusionner ces deux tables :

```
> X
     Sexe Poids
Jean    H    80
Paul    F    60
> Y
      Yeux Taille
Jean Bleus    180
Paul Verts    160
```

3.5- Donnez l'instruction permettant de calculer le produit de tous les éléments (respectivement des éléments de chacune des colonnes) d'une matrice numérique `X`.

3.6- Que renvoie cette instruction : `vec<-c(2,4,6,8,3);vec[2];vec[-2]` ?

3.7- On a mesuré le poids et la taille de plusieurs hommes et on a stocké ces valeurs dans les vecteurs `poids` et `taille`. Donnez l'instruction R permettant d'obtenir le poids des hommes dont la taille est supérieure à 180 cm.

3.8- Que renvoie l'instruction suivante :
`Mat<-matrix(1:12,nrow=4,byrow=TRUE);Mat[3,];Mat[2,2:3]` ?

3.9- Comment faire pour remplacer la quatrième composante de la liste suivante par `1:10` ?
`L<-list(12,c(34,67),Mat,1:15,list(10,11))`

3.10- Que renvoie cette instruction : `L[[2]][2]` ?

3.11- Donnez l'instruction R permettant d'obtenir les poids et taille de toutes les femmes du tableau suivant (vous pouvez utiliser la fonction `attach()`) :

```
> X
   poids taille sexe
1     79    163    H
2     90    163    F
3     87    198    H
4     63    164    F
5     90    168    F
6     71    178    F
7     58    191    H
8     80    194    F
9     91    185    F
10    89    176    H
```

3.12- Que renvoie cette instruction : `(1:3)[any(c(T,F,T))]` ? Et celle-ci : `(1:3)[all(c(T,F,T))]` ?

3.13- Que renvoie cette instruction : `c(T,T,F) | c(F,T,F)` ?
Et celle-ci : `c(T,T,F) || c(F,T,F)` ?

3.14- Que renvoie cette instruction : `nchar(c("abcd","efgh"))` ?

3.15- Que renvoie l'instruction suivante :
`paste(c("a","b"),c("c","d"),collapse="",sep="")` ?

3.16- Que renvoie cette instruction : `strsplit(c("ab;cd"),";")` ?

3.17- Que renvoie cette instruction : `substring("abcdef",3,c(2,4))` ?

3.18- Comment remplacer les lettres majuscules par des minuscules dans `c("Pierre","Paul","Pascal")` ?

3.19- Quelle est la fonction permettant de récupérer une date dans une chaîne de caractères ?

3.20- Pouvez-vous expliquer pourquoi le deuxième nombre de la dernière sortie n'est pas égal à 36.21313 ?

```
> logp <- function(x) log(max(x,exp(1)))
> x <- -2.4
> delta <- 0.1
> (abs(x))^(4+delta)/(logp(abs(x)))^2
[1] 36.21313
> x <- seq(from=-2.8,to=-2,length=3)
> x
[1] -2.8 -2.4 -2.0
> (abs(x))^(4+delta)/(logp(abs(x)))^2
[1] 64.26795 34.15959 16.17594
```

Proposez une solution à ce problème.

Fiche de TP

Manipulation de différents jeux de données

A- Quelques manipulations sur les jeux de données présentés en début d'ouvrage

Ces fichiers peuvent être récupérés à l'adresse `http://www.biostatisticien.eu/springeR/`. Notez que vous pouvez aussi ajouter le nom du fichier à la fin de cette adresse pour récupérer directement le fichier depuis R (par exemple `http://www.biostatisticien.eu/springeR/nutri1.xls`).

• Fichier de données nutriage.xls

Le fichier de données nutriage.xls présenté en début d'ouvrage est en fait issu de la fusion de deux fichiers de données initiaux, chacun saisi par un opérateur différent. Nous vous proposons de le reconstruire pour les cas de figure suivants. Vous utiliserez uniquement R et vous n'éditerez pas le fichier à la main.

3.1- Les individus sont initialement séparés dans deux fichiers (nutri1.xls et nutri2.xls). Notez que le nom des variables est indiqué en lettres majuscules dans le premier fichier et en lettres minuscules dans le second.

3.2- Certains individus sont présents dans les deux fichiers (nutri3.xls et nutri4.xls). Les noms des variables sont identiques.

3.3- Même question que 3.2, mais des erreurs se sont glissées et il vous faudra détecter les individus correspondants, par exemple ceux ayant un poids à 200 kg (nutri5.xls et nutri6.xls).

3.4- Les variables sont séparées dans deux fichiers (nutri7.xls et nutri8.xls) qui contiennent les mêmes individus.

3.5- Même question que 3.4, mais une variable possède trop de valeurs manquantes et nous choisissons de la supprimer (nutri9.xls et nutri10.xls).

3.6- Même question que 3.4 mais un individu possède trop de valeurs manquantes. Supprimez-le (nutri11.xls et nutri12.xls).

3.7- Dans le fichier nutriage.xls, combien de personnes pratiquent le végétarisme (pas de viande, pas de poisson).

• Fichier Intima_media.xls

3.1- Rajoutez une colonne IMC contenant l'IMC de chaque individu au *data.frame* issu de ce fichier de données.

3.2- Récupérez la mesure de l'intima-média pour les personnes ayant un `IMC>30`.

3.3- Extrayez les femmes « sportives ».

3.4- Extrayez les « non-obèses » chez les personnes âgées de 50 ans ou plus (obèse=`IMC>30`).

• Fichier imcenfant.xls

3.1- Rajoutez une colonne `IMC`.

3.2- Extrayez les enfants ayant un `IMC < 15` et un âge `<= 3.5` ans.

3.3- Donnez le nombre d'enfants vérifiant les conditions ci-dessus.

• Fichier Poids_naissance.xls

3.1- Rajoutez une variable `PTL1` (nombre d'antécédents de prématurés) à 3 modalités (où la troisième modalité sera codée 2 et correspondra à « 2 ou plus » antécédents de prématurés).

3.2- Même question avec `FVT` (nombre de visites à un médecin) pour rajouter `FVT1`.

3.3- Ordonnez le fichier suivant les poids de naissance (`BWT`) croissants.

3.4- Extrayez les individus ayant des mères noires ou blanches et qui fument.

B- Gestion des valeurs manquantes

Importez dans un *data.frame* le fichier de données suivant :
http://www.biostatisticien.eu/springeR/Infarct.xls

3.1- Quelles sont les lignes qui ont des valeurs manquantes ?

3.2- Quels sont les individus qui ont plus d'une valeur manquante ?

3.3- Quelles sont les variables ayant des valeurs manquantes ?

3.4- Vous allez proposer plusieurs solutions pour supprimer toutes les lignes de ce *data.frame* contenant au moins une valeur manquante. En plus des opérateurs logiques et de la fonction d'extraction, vous ne devez utiliser que :

a) les fonctions `is.na()`, `prod()`, `apply()` et `as.logical()` ;

b) les fonctions `is.na()`, `apply()` et `any()` ;

c) les fonctions `is.na()`, `apply()` et `all()` ;

d) la fonction `complete.cases()` ;

e) la fonction `na.omit()`.

C- Gestion des chaînes de caractères

3.1- Importez le fichier `www.biostatisticien.eu/springeR/dept-pop.csv` dans un *data.frame* nommé `dept`.

3.2- Remplacez la première colonne par deux nouvelles colonnes : l'une nommée `numdep` contiendra le numéro du département, l'autre nommée `Dept` qui n'en contiendra que le nom.

D- Épidémies de grippe en France depuis 1984 [1]

3.1- Importez le fichier `http://www.biostatisticien.eu/springeR/grippe.csv` dans un *data.frame* nommé `grippe`. Assurez-vous de bien gérer les valeurs manquantes.

3.2- Tapez `names(grippe)`. Comme vous le constatez, `grippe$Date` contient des dates sous la forme de l'année (avec le siècle ; exemple : 2003) suivie du numéro de la semaine (à deux chiffres).

3.3- Déterminez quels sont les différents numéros de semaine possibles (indice : utilisez les fonctions `substring()`, `unique()` et `sort()`).

3.4- Dans un premier temps, il s'agit de récupérer ces dates sous **R** dans un objet de classe `POSIXlt`, par exemple au moyen de la fonction `strptime()`. En vous fondant sur le tableau 3.3, et en utilisant cette fonction, transformez la première date (la plus ancienne) au format `POSIX`.

1. Source : `http://www.sentiweb.fr`

3.5- Les données sont en fait actualisées tous les lundis de chaque semaine, depuis la première semaine. Déterminez quelle est la plus ancienne date (jour, mois, année) pour laquelle vous avez des observations (indice : consultez le calendrier suivant :
`http://sentiweb.fr/calendrier.php`).

3.6- Vous devriez donc constater qu'il y a une différence avec le résultat de la question 3.4. Pour régler ce problème, essayez de rajouter un « 1 » à la fin de la première date et de la transformer de nouveau à l'aide de la fonction `strptime()`.

3.7- Affichez les dix premières dates du fichier de données. Utilisez l'astuce de la question précédente pour les transformer à l'aide de la fonction `strptime()`. Est-ce que la dernière date est correcte ? Si non, avez-vous une idée pour régler ce problème ?

3.8- À ce stade, vous devriez constater que le format des dates de ce fichier n'est pas compatible avec le format POSIX (qui nécessite des numéros de semaine de 00 à 53). Il n'est donc pas possible d'utiliser directement la fonction `strptime()` ou `as.POSIXlt()` pour transformer ces dates en un type d'objet facile à gérer par R. Tapez l'instruction :
`date1 <- as.POSIXlt("`*Jour*`,`*Mois*`,`*Année*`",format="`*à spécifier*`")`
où vous remplacerez *Jour*, *Mois* et *Année* par la date la plus ancienne, et *à spécifier* par le bon format de date comme cela est décrit dans le cours.

3.9- Tapez l'instruction `date1` puis `date1+7`. Que constatez-vous ?

3.10- Arrangez-vous pour ajouter sept jours à `date1` (indice : combien y a-t-il de secondes dans une journée ?).

3.11- Maintenant, créez le vecteur `dates` contenant toutes les dates au format POSIX depuis la plus ancienne jusqu'à la plus récente (indice : utilisez la fonction `nrow()`).

3.12- Utilisez la fonction `substring()` sur le vecteur `dates` pour remplacer la première colonne du *data.frame* `grippe` par des dates au format `"année-mois-jour"` (par exemple : `"1992-12-07"`).

3.13- Servez-vous du vecteur `dates` et de ce que vous avez appris sur l'extraction pour sélectionner uniquement la portion du *data.frame* `grippe` pour les dates comprises entre le 15 septembre 1992 et le 3 novembre 1993. Vous stockerez cette sous-table dans un objet nommé `portion`.

3.14- Calculez le nombre de cas de grippes pour cette période dans chacune des régions (indice : attention aux valeurs manquantes, utilisez le paramètre `na.rm`). Vous stockerez les résultats dans un vecteur nommé `casgrippe`.

E- Combinaison de tables ou de listes, autres manipulations

3.1- Entrez sous R les deux tables suivantes (attention aux noms des lignes).

```
> a
  [,1] [,2]
1    1    4
2    2    5
6    3    6
> b
  [,1] [,2]
3    1    5
4    2    6
5    3    7
7    4    8
```

3.2- Combinez `a` et `b` dans une nouvelle table nommée `ab` qui devra contenir :

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

(indice : utilisez `rbind()` et `order()`).

3.3- Concaténez en colonnes, dans une même matrice, les éléments de `list1` :

```
> list1 <- list()
> list1[[1]] <-  matrix(runif(25),nr=5)
> list1[[2]] <-  matrix(runif(30),nr=5)
> list1[[3]] <-  matrix(runif(15),nr=5)
```

(indice : utilisez la fonction `unlist()` ou bien la fonction `do.call()`).

3.4- Concaténez en lignes, dans une même matrice, les éléments de `list2` :

```
> list2 <- list()
> list2[[1]] <-  matrix(runif(25),nc=5)
> list2[[2]] <-  matrix(runif(35),nc=5)
> list2[[3]] <-  matrix(runif(15),nc=5)
```

(indice : utilisez la fonction `unlist()` ou bien la fonction `do.call()`).

3.5- Sélectionnez, de façon automatique, les maladies qui ont `tabac` pour facteur de risque.

```
> tmp
          Maladie FacteursRisques
1        Infarctus    tabac,alcool
2         Hépatite          alcool
3 Cancer du poumon           tabac
```

(indice : utilisez la fonction `grep()`).

Création de fonctions

F- Le chevalier de Méré

Le chevalier de Méré était un grand joueur. Il affectionnait particulièrement deux jeux. Le premier consistait à lancer un dé quatre fois et à parier sur l'apparition d'un 6. Le second consistait à lancer deux dés 24 fois et à parier sur l'apparition d'un double-six. Le chevalier avait remarqué que le premier jeu était « avantageux », c'est-à-dire qu'il y avait plus de 50 % de chances d'apparition d'au moins un 6 sur tous les lancers. Il pensait que le deuxième jeu était également avantageux.

- Proposez le code d'une fonction nommée `quatrejets()` qui renvoie 1 s'il y a eu au moins un six dans les 4 jets d'un dé, et 0 sinon. Vous n'utiliserez pas de boucles.
- Proposez le code d'une fonction nommée `vingtquatrejets()` qui renvoie 1 s'il y a eu au moins un double six dans les 24 jets de deux dés, et 0 sinon. Vous n'utiliserez pas de boucles.
- Proposez le code d'une fonction nommée `meresix()` permettant de confirmer l'intuition du chevalier de Méré. Elle utilisera les deux premières fonctions et aura un paramètre formel `nsim` permettant de fournir le nombre de répétitions du jeu.

(indice : utilisez la fonction `sample()`).

Chapitre 4

R et sa documentation

Pré-requis et objectif

- Lecture du chapitre 1.
- Ce chapitre présente les différents moyens d'accéder à de l'aide sur le logiciel R.

SECTION 4.1

Aide intégrée au logiciel R

4.1.1 La commande `help()`

R contient une aide en ligne (en anglais) très complète et très bien structurée sur toutes les fonctions que vous pouvez utiliser ainsi que sur les différents symboles du langage. Cette aide est accessible au moyen de plusieurs commandes dont la principale est `help()`. Elle s'utilise en mode ligne de commande.

Tapez par exemple :

```
help(help)
```

La commande `help()` possède un alias qui est le point d'interrogation : `?`.

```
?sum
?sd
?"+"
?"[["
```

Attention

Il peut arriver que cet alias ne fonctionne pas, et il est alors nécessaire d'utiliser la fonction `help()` avec des guillemets.

```
?function         # Ne fonctionne pas.
help(function)    # Renvoie une erreur.
help("function") # Appel correct.
```

Décortiquons ensemble le fichier d'aide de la fonction `mean()`, qui permet de calculer une moyenne.

```
?mean
① mean              package:base              R Documentation
② Arithmetic Mean
③ Description:
     Generic function for the (trimmed) arithmetic mean.
④ Usage:
     mean(x, ...)
     ## Default S3 method:
     mean(x, trim = 0, na.rm = FALSE, ...)
⑤ Arguments:
       x: An R object. Currently there are methods for
          numeric dataframes, numeric vectors and dates.
          A complex vector is allowed for 'trim = 0', only.
    trim: the fraction (0 to 0.5) of observations to be trim-
          med from each end of 'x' before the mean is computed.
  na.rm: a logical value indicating whether 'NA' values
          should be stripped before the computation proceeds.
     ...: further arguments passed to or from other methods.
⑥ Value:
     For a data frame, a named vector with the appropriate
     method being applied column by column.
     If 'trim' is zero (the default), the arithmetic mean of
     the values in 'x' is computed.
     If 'trim' is non-zero, a symmetrically trimmed mean is
     computed with a fraction of 'trim' observations deleted
     from each end before the mean is computed.
⑦ References:
     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988)
     _The New S Language_. Wadsworth & Brooks/Cole.
⑧ See Also:
     'weighted.mean', 'mean.POSIXct'
⑨ Examples:
     x <- c(0:10, 50)
     xm <- mean(x)
     c(xm, mean(x, trim = 0.10))
     mean(USArrests, trim = 0.2)
```

Voici les différentes sections de ce fichier d'aide :

① L'en-tête du fichier contenant :
- le nom de la fonction pour laquelle on recherche de l'aide : `mean` ;
- le nom du *package* qui contient cette fonction : `base` ;
- l'origine du fichier d'aide : `R Documentation`.

② Un titre explicite pour la fonction : `Arithmetic Mean`.

③ Une brève description de ce que fait la fonction : `Description`.

④ La façon d'utiliser la fonction avec notamment ses paramètres obligatoires et optionnels : `Usage`.

⑤ Une description des différents paramètres d'entrée de la fonction : `Arguments`.

⑥ Des explications sur ce que renvoie la fonction lorsqu'elle est exécutée : `Value`.

⑦ Des références d'articles ou d'ouvrages ayant rapport avec le domaine statistique d'application de la fonction : `References`.

⑧ La rubrique `See Also` qui donne le nom de fonctions similaires ou en rapport avec la fonction en cours d'investigation.

⑨ Des exemples montrant comment utiliser cette fonction : `Examples`.

Attention

La plupart des fichiers d'aide ont le même format. Vous devez bien comprendre et retenir la logique organisationnelle de la structure des différents fichiers d'aide. Vous devez aussi prendre l'habitude d'utiliser l'aide en ligne dès que vous rencontrez une nouvelle fonction inconnue afin de bien comprendre quels sont ses paramètres et comment elle s'utilise.

Astuce

Vous noterez que tous ces fichiers d'aide ne contiennent pas de graphiques à visualiser, notamment ceux qui pourraient être produits par le code de la section `Examples` ⑨. Cela pourrait toutefois être intéressant, en particulier pour toutes les fonctions graphiques. Une façon de les obtenir consiste à utiliser la fonction `example()`. Vous pouvez aussi consulter le site web *R Graphical Manual* : `http://bm2.genes.nig.ac.jp/RGM2/index.php` qui présente tous les fichiers d'aide de R au format HTML. Dans ces fichiers, les graphiques, lorsqu'il y en a, sont directement incorporés dans la section `Examples`.

4.1.2 Quelques commandes complémentaires

En plus de la commande principale `help()`, il existe quelques autres fonctions complémentaires qui peuvent se révéler utiles lorsque l'on cherche de l'aide

sur une commande donnée. Nous les présentons ci-dessous :

- `help.start()` : cette fonction ouvre un navigateur avec plusieurs liens vers des manuels au format HTML, de l'aide sur toutes les fonctions présentes dans tous les *packages* de R (aussi au format HTML), une FAQ (questions fréquemment posées ou *frequently asked questions*) qui contient les réponses aux questions les plus fréquemment posées sur R. Vous y trouverez également un moteur de recherche dans l'aide par mots clefs. On y trouve aussi quelques autres documents plus techniques.

Linux

Sous Linux, une fois la commande `help.start()` tapée, l'utilisation de la commande `help()` affichera toujours ses résultats dans un navigateur et non plus dans la console. Pour annuler cela, utilisez l'instruction `options(htmlhelp = FALSE)`. Le choix du navigateur, par exemple `firefox`, s'opère au moyen de l'instruction `options(browser="firefox")`.

- `help.search()` ou `??()` : cette fonction est à utiliser lorsque l'on ne connaît pas le nom d'une commande. Elle renvoie une liste des différentes fonctions (et des *packages* dans lesquels elles se trouvent) ayant un rapport avec votre requête. Essayez par exemple : `help.search("mean")`.

- `apropos()` : cette instruction renvoie une liste de toutes les fonctions dont le nom contient le paramètre d'appel. Ainsi, `apropos("mean")` renvoie toutes les fonctions dont le nom contient le mot `mean`.

Expert

Notez l'existence de la fonction `methods()` qui renvoie toutes les méthodes (fonctions) associées à un objet. Essayez par exemple `methods(summary)`.

- `library(help=`*package*`)` : cette commande permet de lister toutes les fonctions présentes dans la librairie *package*. Elle donne les mêmes résultats que la commande `help(package="`*package*`")`. Nous vous conseillons d'essayer les instructions suivantes afin d'obtenir la liste des fonctions principales de R :

```
library(help=base)
library(help=utils)
library(help=datasets)
library(help=stats)
library(help=graphics)
library(help=grDevices)
```

Astuce

La fonction `library(lib.loc = .Library)` renvoie la liste de toutes les librairies (*packages*) installées sur le système.

Inversement, l'instruction `find("`*fonction*`")` permet de savoir à quelle librairie appartient la fonction *fonction*`()`.

```
> find("t.test")
[1] "package:stats"
```

- `vignette()` : les vignettes sont des petits fichiers au format PDF à visée pédagogique qui expliquent certaines notions plus en détail. Tapez `vignette()` pour obtenir la liste de ces vignettes, et par exemple `vignette("xtableGallery")` pour ouvrir le fichier PDF de la vignette du *package* `xtable`.

Mac

L'ensemble des vignettes peut également être consulté dans un navigateur spécial pour vignettes dans le menu « Aide/Vignettes ». Dans ce navigateur, il est possible d'ouvrir le fichier PDF ainsi que le code source R (sous forme d'un fichier d'extension `.R`) et d'accéder directement aux codes des exemples inclus dans la vignette.

Voici trois autres fonctions qui pourront également se révéler utiles :

- `data()` : cette commande renvoie la liste de tous les jeux de données (*datasets*) présents dans R.

- `example()` : cette instruction donne des exemples sur l'utilisation d'une fonction. Ainsi, `example(mean)` exécute les instructions présentes dans la section *Examples* du fichier d'aide obtenu quand on tape `help(mean)`.

- `demo()` : cette instruction est similaire à la précédente, mais n'est accessible que pour un nombre limité de fonctions. Lorsqu'elle est disponible, elle présente un éventail des possibilités offertes par une fonction. Essayez par exemple `demo(graphics)`.

SECTION 4.2

† Aide accessible sur l'Internet

Le site web officiel de R (`http://www.r-project.org`) contient une quantité considérable d'informations sur ce logiciel. Nous vous conseillons de passer un peu de temps à l'explorer. Il existe également d'autres sources d'information que nous détaillons dans les sections suivantes.

4.2.1 Moteurs de recherche

Il existe deux moteurs de recherche principaux sur R :
- http://search.r-project.org/nmz.html

Astuce

La commande `RSiteSearch()` permet de faire une requête sur ce site directement depuis R. Les informations extraites sont alors affichées dans votre navigateur.

- http://www.rseek.org

Notez également l'existence d'une plateforme collaborative de questions-réponses (en anglais) consultable à l'adresse http://stackoverflow.com/questions/tagged/r, ainsi que le site collaboratif de partage de scripts, de codes et d'astuces suivant : http://abcdr.guyader.pro

4.2.2 Forums de discussion

Voici deux forums de discussion en français sur lesquels vous pouvez poser vos questions :
- http://forums.cirad.fr/logiciel-R/index.php
- http://www.developpez.net/forums/f1179/autres-langages/autres-langages/r

Voici un forum en langue anglaise qui semble beaucoup plus fréquenté : http://www.nabble.com/R-f13819.html.

4.2.3 Listes de diffusion (*mailing lists*)

Une liste de distribution de courrier électronique est une utilisation spécifique du courrier électronique qui permet le publipostage (en masse) d'informations à un grand nombre d'utilisateurs possédant une adresse courriel et inscrits à cette liste.

Il existe plusieurs listes de diffusion donnant des informations sur R dont les plus importantes sont citées ci-dessous :
- https://stat.ethz.ch/mailman/listinfo/r-help
- http://blog.gmane.org/gmane.comp.lang.r.general
- http://www.r-project.org/mail.html
- R-announce : https://stat.ethz.ch/mailman/listinfo/r-announce

Notez l'existence du site http://r-project.markmail.org qui permet de faire des recherches dans les archives de ces listes.

Les règles à suivre pour poster des messages sur ces listes sont disponibles ici : http://www.r-project.org/posting-guide.html.

Mac

Une liste dédiée aux utilisateurs Mac : https://stat.ethz.ch/mailman/listinfo/r-sig-mac.

4.2.4 Discussion relayée par l'Internet (IRC)

L'IRC (*Internet relay chat*) est un outil de communication instantanée qui vous permet de dialoguer avec des internautes regroupés sur des thèmes bien définis. Il existe deux canaux IRC portant sur le logiciel R, l'un en langue anglaise (`#R`) et l'autre en français (`#Rfr`).

Pour y accéder, vous pouvez soit utiliser un logiciel client spécialisé tel `xchat` (www.xchat.org) ou encore utiliser votre navigateur en passant par des sites web comme http://webchat.freenode.net.

Voilà les instructions à utiliser sur ces sites ou sur `xchat` pour vous connecter à ces canaux :

```
/server irc.freenode.net
/join #R
/join #Rfr
```

4.2.5 *Wiki*

Un *wiki* est un système de gestion de contenu de site web rendant ses pages librement modifiables par tous les visiteurs y étant autorisés. Les *wikis* sont utilisés pour faciliter l'écriture collaborative de documents avec un minimum de contraintes.

Il existe un *wiki* sur R accessible ici : http://rwiki.sciviews.org.

SECTION 4.3

† Littérature sur R

4.3.1 Sur le web

Il est possible de trouver de la littérature sur R sur l'Internet sous diverses formes :

- **Les *Task Views* :** il s'agit d'une liste de *packages* utiles dans un certain domaine et regroupés par thèmes privilégiés. Une page web décrivant ces *Task Views* est accessible à `http://cran.r-project.org/web/views`.

- **Les foires aux questions (FAQ) :** les questions les plus fréquemment posées sont accessibles ici : `http://cran.r-project.org/faqs.html`.

- **Les revues spécialisées :** il existe deux revues en ligne mettant l'accent sur le logiciel R. Le R Journal antérieurement connu sous le nom de R *News* (`http://journal.r-project.org`) et le *Journal of Statistical Software* (`http://www.jstatsoft.org`).

- **Les manuels de cours :** il y a de nombreux ouvrages disponibles au format PDF sur le site de R. La plupart sont en anglais, mais on peut aussi trouver quelques documents en français, dont prochainement l'ouvrage que vous êtes en train de lire : `http://cran.r-project.org/other-docs.html`
- **Les aide-mémoire :** quelques cartes de référenes spécifiques à R sont disponibles ici :

  – `http://biostatisticien.eu/springeR/Kauffmann.pdf`
  – `http://revue.sesamath.net/IMG/pdf/RCarte_Commandes-R.pdf`

4.3.2 En format papier

De nombreux livres sur R ont été publiés ces dernières années. Nous citons ci-dessous ceux qui nous paraissent les plus intéressants.

- **Statistiques avec R** [15]
- **Data Analysis and Graphics Using R: An Example-based Approach** [33]
- **The R Book** [17]
- **Statistics and Data with R** [13]
- **Software for Data Analysis: Programming with R** [11]
- **Lattice: Multivariate Data Visualization with R** [42]
- **R for SAS and SPSS users** [39]
- **Introductory Statistics with R** [18]
- **A First Course in Statistical Programming with R** [9]
- **A Handbook of Statistical Analyses Using R** [22]
- **A Beginner's Guide to R** [46]
- **R Cookbook** [45]
- **R in a Nutshell** [1]
- **The Art of R Programming** [35]
- **The R Inferno** [10]

Termes à retenir

`help()`, `?()` : fournit de l'aide sur une fonction ou un symbole
`help.search()` : renvoie une liste des fonctions et *packages* associés, ayant un lien avec votre requête
`apropos()` : renvoie une liste de toutes les fonctions dont le nom contient la requête
`library(help=`*package*`)` : donne la liste de toutes les fonctions de la librairie *package*
`data()` : donne la liste de tous les jeux de données présents dans R
`example()` : exécute la section *Examples* du fichier d'aide correspondant
`demo()` : lance une petite démonstration des possibilités offertes par une fonction
`vignette()` : ouvre un fichier PDF à visée pédagogique donnant des détails sur une fonction
`help.start()` : ouvre la version HTML des fichiers d'aide de R
`RSiteSearch()` : opère une requête sur le moteur de recherche du site officiel de R

Exercices

4.1- Quelle est l'instruction R que vous devez taper pour obtenir de l'aide sur la fonction `mean()` ?

4.2- Expliquez à quoi sert la commande `apropos()`.

4.3- Expliquez à quoi sert la commande `example()`.

4.4- Expliquez à quoi sert la commande `RSiteSearch()`.

4.5- Comment est organisé un fichier d'aide ?

4.6- Quelle commande permet d'obtenir la liste des fonctions contenues dans le *package* `stats` ?

4.7- Comment afficher l'un des jeux de données présents sous R ?

Fiche de TP

Où trouver de l'information

4.1- Trouvez la fonction R permettant de lister toutes les combinaisons de k éléments pris parmi n éléments.

4.2- Utilisez cette fonction pour lister toutes les combinaisons de trois éléments pris parmi `c(5,8,2,9)`.

4.3- Trouvez le jeu de données présent dans R qui donne le taux de crimes violents commis aux États-Unis.

4.4- Décrivez le contenu de ce jeu de données.

4.5- Abonnez-vous à la liste de diffusion `https://stat.ethz.ch/mailman/listinfo/r-help`.

4.6- Lisez les règles à suivre avant de poser une question (`http://www.r-project.org/posting-guide.html`).

4.7- Trouvez comment vous désabonner de la liste.

4.8- Connectez-vous sur IRC, par le moyen de votre choix, au canal `Rfr` et prenez contact poliment avec les membres de ce canal.

4.9- Abonnez-vous au forum `http://forums.cirad.fr/logiciel-R/index.php`.

4.10- Parcourez la FAQ de **R** pour Microsoft Windows et tentez de comprendre ce que signifie la *TAB completion*.

4.11- Mettez-la en pratique pour lister les fichiers présents dans le répertoire courant.

Chapitre 5

Techniques pour tracer des courbes et des graphiques

Pré-requis et objectif

- Lecture des chapitres précédents.
- Ce chapitre décrit les possibilités graphiques du logiciel R, mais sans parler des fonctions graphiques spécialisées de haut niveau comme `hist()`, `barplot()`, etc., dont la présentation est reportée au chapitre 9. Elle montre les possibilités génériques de faire des ajustements sur la plupart des graphiques que vous pouvez tracer.

SECTION 5.1

Les fenêtres graphiques

5.1.1 Fenêtre graphique de base, manipulation, sauvegarde

Tous les graphiques créés dans R sont affichés dans des fenêtres spéciales, distinctes de la console, appelées « R graphics : Device *numero-device* », où *numero-device* est un entier donnant le numéro de la fenêtre (ou *device*).

Pour ouvrir une fenêtre graphique, il faut utiliser la commande `dev.new()`, `windows()` ou `win.graph()`.

Ces commandes admettent plusieurs paramètres, dont quelques-uns sont décrits de façon succincte dans le tableau suivant :

`width`	Largeur de la fenêtre graphique en pouces (*inches* : *in*).
`height`	Hauteur de la fenêtre graphique en pouces.
`pointsize`	Taille par défaut des caractères.
`xpinch, ypinch`	Double. Pixels par pouce, horizontalement et verticalement.
`xpos, ypos`	Entier. Position du coin gauche supérieur de la fenêtre, en pixels.

Linux

Sous Linux, il faut utiliser la commande `X11()` à la place de `windows()`.

Mac

Sous Macintosh, il faut utiliser la commande `quartz()`.

Remarque

Certains éditeurs de code, comme RStudio par exemple, gèrent eux-mêmes les fenêtres graphiques intégrées à l'éditeur, ce qui est très pratique. Ils ne supportent pas les commandes d'ouverture de nouvelles fenêtres graphiques.

Lorsque l'on ouvre plusieurs fenêtres graphiques, une seule est la fenêtre dite « active ». C'est la fenêtre dans laquelle toutes les opérations graphiques se passent. À chaque fenêtre est associé un numéro de *device* ; la console possédant le numéro 1.

Voici une liste de quelques fonctions permettant de manipuler les différentes fenêtres graphiques grâce à leur numéro de *device*.

`dev.off(`*numero-device*`)`	Ferme la fenêtre spécifiée par *numero-device* (si aucun numéro de *device* n'est fourni, c'est la fenêtre active qui est fermée).
`graphics.off()`	Ferme toutes les fenêtres graphiques ouvertes.
`dev.list()`	Renvoie les numéros des fenêtres graphiques ouvertes.
`dev.set(`*numero-device*`)`	Active la fenêtre spécifiée par *numero-device*.
`dev.cur()`	Renvoie le numéro de la fenêtre active (1 pour la console).

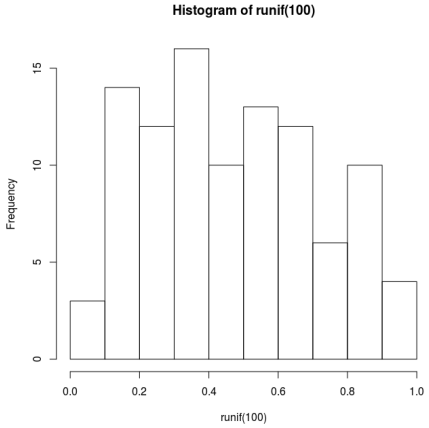
Notez enfin que l'on peut sauvegarder dans un fichier un graphique ayant déjà été tracé, en utilisant la commande `savePlot()` de la façon suivante :

```
savePlot(filename="Rplot",
         type=c("wmf", "png", "jpeg", "jpg", "bmp",
          "ps", "pdf"),device=dev.cur())
```

Le paramètre `filename` est le nom du fichier sous lequel enregistrer le graphique, `type` est le type de fichier (Windows metafile, PNG, JPEG, BMP, Postscript ou PDF) et `device` est le numéro de *device* où se trouve le graphique que

l'on veut enregistrer (par défaut, la fenêtre active). Notez que la disponibilité de certains types de fichiers peut dépendre du système d'exploitation que vous utilisez.

```
> X11(type="cairo")
> hist(runif(100))
> savePlot(filename="mongraph.png",type="png")
> dev.off()
pdf
  2
```



Il est possible d'utiliser deux autres instructions dans ce contexte :
- `dev.copy2eps(file="mongraph.eps")`
- `dev.copy2pdf(file="mongraph.pdf")`

qui créent respectivement des fichiers au format Postscript et PDF.

Astuce

Il existe d'autres fonctions permettant d'enregistrer vos graphiques dans divers formats intéressants : `postscript()`, `pdf()`, `pictex()`, `xfig()`, `bitmap()`, `bmp()`, `jpeg()`, `png()`, `tiff()`. En revanche, leur utilisation se fait de façon un peu différente puisqu'il faut d'abord commencer par taper le nom de l'une des instructions précédentes, puis tracer votre graphique et enfin finir par l'appel de `dev.off()`. Vous noterez que l'utilisation de ces commandes ne provoque pas d'affichage à l'écran.

```
> pdf(file="mongraph.pdf")
> hist(runif(100))
> g <- dev.off()
```

5.1.2 Découpage de la fenêtre graphique : `layout()`

Si vous voulez tracer plusieurs graphiques sur la même fenêtre, **R** vous offre la possibilité de découper cette fenêtre en autant de cases que nécessaire.

Une première possibilité consiste à utiliser la fonction `par()` avec le paramètre `mfrow` (veuillez consulter l'encadré d'alerte de la section 5.7 sur cette fonction `par()`). Ainsi l'exemple ci-dessous découpe la fenêtre graphique en trois lignes et deux colonnes. À chaque appel d'une fonction de tracé, les petites cases graphiques ainsi créées seront remplies successivement ligne par ligne (l'utilisation de `mfcol` aurait conduit à un remplissage colonne par colonne) comme on peut le visualiser sur la figure suivante :

```
> par(mfrow=c(3,2))
```

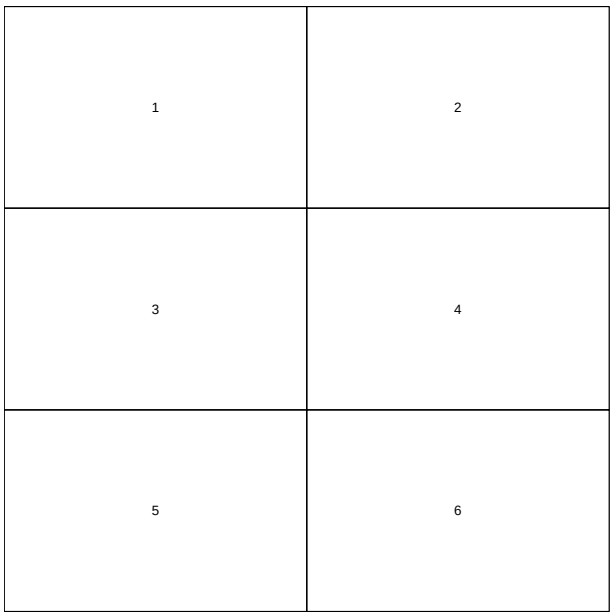


FIGURE 5.1 – Visualisation de l'effet du paramètre `mfrow` de la fonction `par()`. Des numéros ont été ajoutés pour une meilleure compréhension des emplacements successifs où seront tracées les figures.

La fonction `layout()` permet d'obtenir un découpage plus évolué que l'utilisation de la fonction `par()`. Montrons sur un exemple comment ce découpage est spécifié de façon naturelle au moyen du paramètre `mat`, lorsque l'on souhaite par exemple tracer cinq graphiques isolés les uns des autres.

```
> mat <- matrix(c(2,3,0,1,0,0,0,4,0,0,0,5),4,3,byrow=TRUE)
> mat
     [,1] [,2] [,3]
[1,]    2    3    0
[2,]    1    0    0
[3,]    0    4    0
[4,]    0    0    5
```

```
> layout(mat)
```

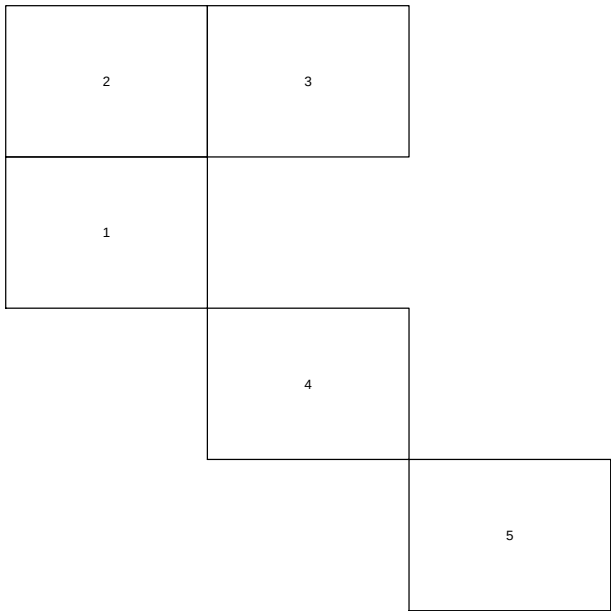


FIGURE 5.2 – Visualisation du potentiel de la fonction `layout()`.

Astuce

Notez qu'il est nécessaire d'utiliser l'instruction `layout.show(5)` afin de faire afficher la figure précédente sous **R**.

Notons qu'à chaque appel d'une fonction **R** de tracé, les différents graphiques obtenus seront successivement affichés dans les cases numérotées, en suivant l'ordre croissant de ces cases.

Il est aussi intéressant de noter que l'on peut spécifier, au moyen du paramètre `widths`, les largeurs respectives de toutes les colonnes de `mat` les unes par rapport aux autres. On peut faire de même pour les hauteurs respectives des lignes, au moyen du paramètre `heights`.

```
> layout(mat,widths=c(1,5,14),heights=c(1,2,4,1))
> layout.show(5)
```

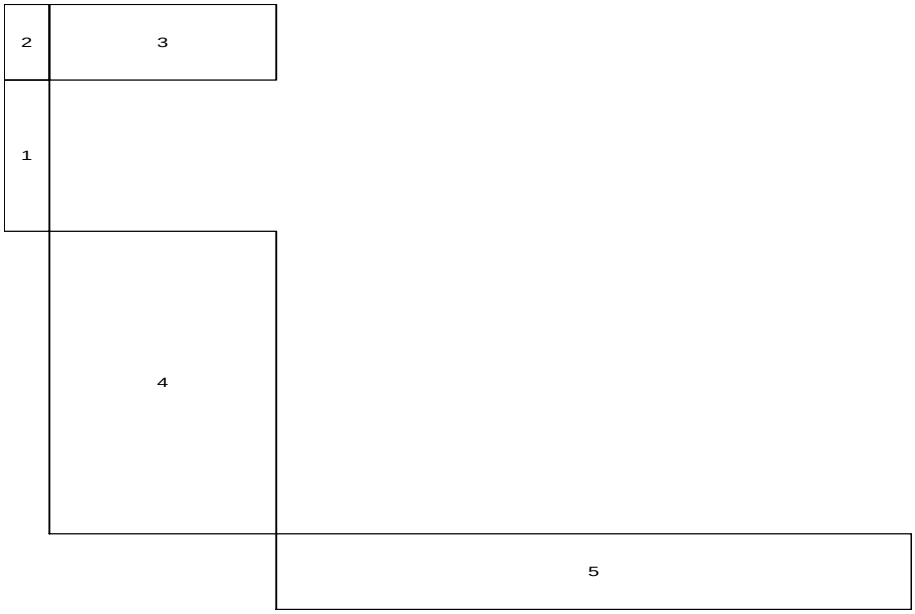


FIGURE 5.3 – La fonction `layout()` et ses paramètres `widths` et `heights`.

SECTION 5.2

Les fonctions de tracé de bas niveau

5.2.1 Les fonctions `plot()` et `points()`

La fonction `plot()` est la fonction générique pour tracer des graphiques. Elle prend comme paramètre d'entrée les coordonnées des points à tracer.

Attention

Il est aussi possible d'utiliser la fonction `plot()` sur un objet R pour lequel une méthode graphique est définie. Nous en verrons des exemples dans les chapitres 12 et 13.

Voilà une partie des paramètres les plus utiles de cette fonction.

Argument	Description
`x`	Vecteur des coordonnées en x des points à tracer.
`y`	Vecteur des coordonnées en y des points à tracer.
`type`	Spécifie le type de graphique à tracer : `"p"` pour des points, `"l"` pour des lignes, `"b"` pour les deux, `"c"` pour des lignes coupées au niveau des points, `"o"` pour des lignes et des points superposés, `"h"` pour des lignes verticales, `"s"` pour des marches d'escalier (*step*) et `"n"` pour ne rien tracer (mais affiche la fenêtre de tracé avec les axes).
`main`	Spécifie le titre principal.
`sub`	Spécifie le sous-titre.
`xlab`	Spécifie le titre de l'axe des x.
`ylab`	Spécifie le titre de l'axe des y.
`xlim`	Vecteur de longueur 2. Spécifie l'intervalle de valeurs à utiliser pour l'axe des x.
`ylim`	Vecteur de longueur 2. Spécifie l'intervalle de valeurs à utiliser pour l'axe des y.
`log`	Chaîne de caractères qui contient `"x"` (respectivement `"y"`, `"xy"` ou `"yx"`) si l'axe des abscisses (respectivement des ordonnées, les deux) doit être affiché en coordonnées logarithmiques.

```
> plot(1:4,c(2,3,4,1),type="b",main="Titre principal",
+   sub="Sous-titre",xlab="Titre pour les x",ylab="Titre pour les y")
```

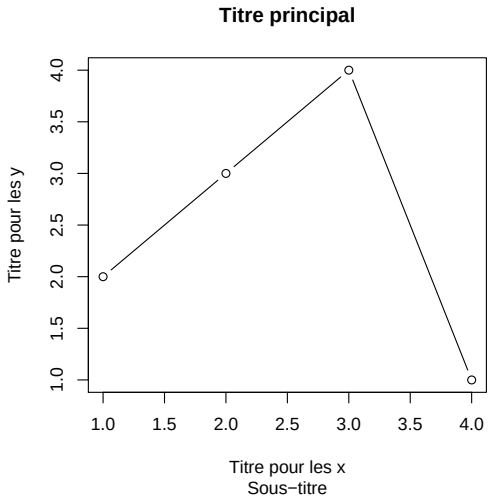


FIGURE 5.4 – La fonction `plot()`.

Notez que des appels successifs de la fonction `plot()` créent à chaque fois un nouveau graphique, en remplacement de l'ancien (sauf si la fenêtre graphique a été découpée comme cela est expliqué à la section précédente).

La fonction `points()` permet de remédier à ce problème en superposant le nouveau graphique sur l'ancien. Elle possède les mêmes paramètres que `plot()`.

```
> points(1:4,c(4,2,1,3),type="l")
```

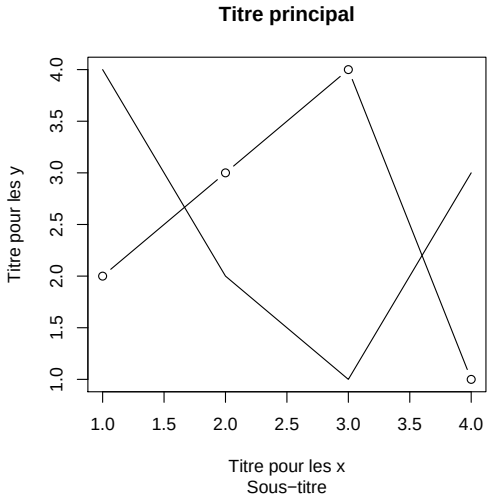


FIGURE 5.5 – La fonction `points()`.

Astuce

La fonction `approx()` permet de produire une interpolation linéaire ou constante par morceaux entre plusieurs points.

5.2.2 Les fonctions `segments()`, `lines()` et `abline()`

Les fonctions `segments()` et `lines()` permettent de joindre des points par des segments de ligne qui sont ajoutés à un graphique préexistant.

```
> plot(0,0,"n")
> segments(x0=0,y0=0,x1=1,y1=1)
> lines(x=c(1,0),y=c(0,1))
```

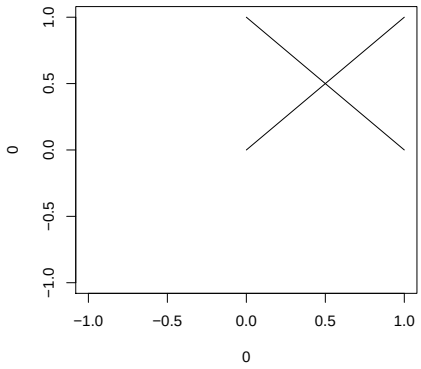


FIGURE 5.6 – Les fonctions `segments()` et `lines()`.

La fonction `abline()` permet soit de tracer une droite d'équation $y = a + bx$ (spécifiée par les paramètres `a` et `b`), soit de tracer une ligne horizontale (paramètre `h`) ou verticale (paramètre `v`).

```
> plot(0,0,"n");abline(h=0,v=0);abline(a=1,b=1)
```

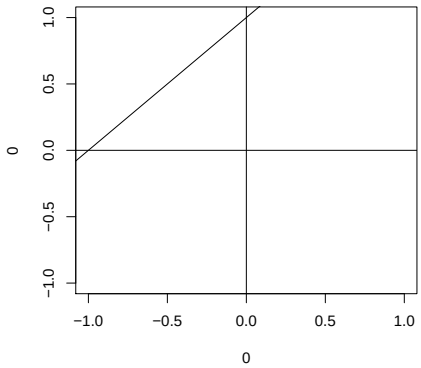
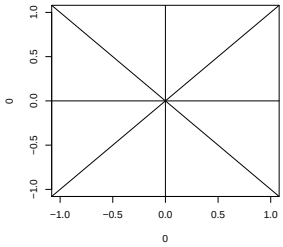


FIGURE 5.7 – La fonction `abline()`.

Prise en main

Reproduire le graphique ci-dessous.



5.2.3 La fonction `arrows()`

Cette fonction permet de tracer des flèches reliant des paires de points. Elle comprend un paramètre `length` permettant d'indiquer la taille de la pointe de la flèche.

```
> x <- runif(12); y <- runif(12)
> i <- order(x,y); x <- x[i]; y <- y[i]
> plot(x,y)
> s <- seq(length(x)-1)
> arrows(x[s], y[s], x[s+1], y[s+1], length=0.1)
```

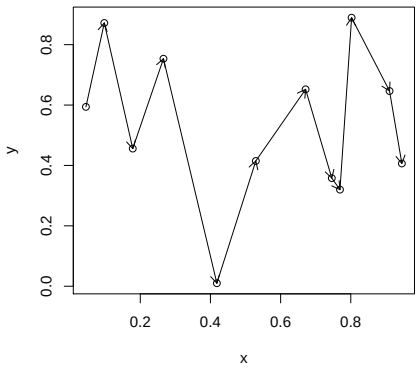


FIGURE 5.8 – La fonction `arrows()`.

5.2.4 La fonction `polygon()`

Comme son nom l'indique, cette fonction permet de tracer des polygones et permet aussi de remplir l'intérieur d'un polygone avec une couleur préspécifiée.

Prise en main

Tapez la commande suivante dans la console de **R** :

```
example(polygon)
```

Astuce

La commande `polygon(locator(10,"l"))` permet de tracer un polygone à dix côtés en cliquant dans la fenêtre graphique aux endroits qui seront les sommets du polygone.

5.2.5 La fonction `curve()`

Cette fonction permet de tracer une courbe dans un repère cartésien, sur un intervalle dont les bornes sont spécifiées par les paramètres `from` et `to`.

```
> curve(x^3-3*x,from=-2,to=2)
```

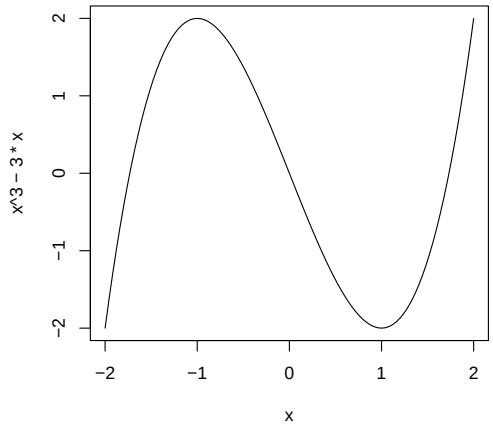


FIGURE 5.9 – La fonction `curve()`.

Notez que l'on peut aussi utiliser le paramètre `add=TRUE` pour indiquer à R de superposer la courbe désirée à un graphique préexistant.

Prise en main

Utilisez l'instruction suivante pour tracer l'histogramme en densité de 10 000 valeurs provenant d'une loi gaussienne standard :

```
hist(rnorm(10000),prob=TRUE,breaks=100)
```

Utilisez la fonction `curve()` pour superposer sur cet histogramme la densité d'une loi $\mathcal{N}(0,1)$, obtenue au moyen de la fonction `dnorm()`.

5.2.6 La fonction `box()`

Cette fonction permet d'ajouter une boîte autour du graphique courant. Le paramètre `bty` permet de gérer le type de boîte ajouté, le paramètre `lty` spécifie le type de ligne utilisé pour tracer la boîte. Notez que la fonction `plot()` ajoute par défaut une boîte au graphique qu'elle trace, sauf si on lui fournit le paramètre `axes=FALSE`.

```
> plot(runif(7), type = "h", axes = FALSE)
> box(lty = "1373")
```

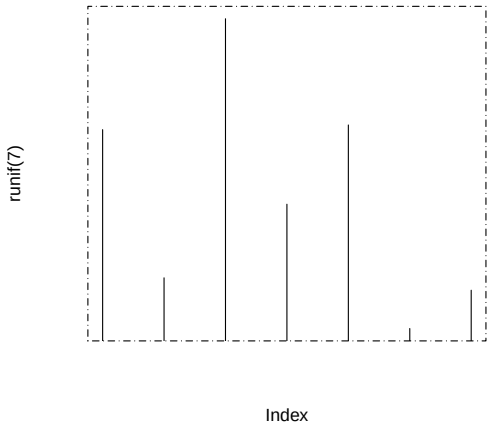


FIGURE 5.10 – La fonction `box()`.

SECTION 5.3

La gestion des couleurs

5.3.1 La fonction `colors()`

Cette fonction renvoie le nom des 657 couleurs que R connaît.

Astuce

Si vous voulez connaître les variantes de la couleur orange, vous pouvez utiliser l'instruction

```
> colors()[grep("orange",colors())]
 [1] "darkorange"  "darkorange1" "darkorange2" "darkorange3"
 [5] "darkorange4" "orange"      "orange1"     "orange2"
 [9] "orange3"     "orange4"     "orangered"   "orangered1"
[13] "orangered2"  "orangered3"  "orangered4"
```

Ces couleurs sont utilisables pour tracer vos graphiques via le paramètre `col`, de la fonction `plot()` par exemple.

```
> plot(1:10,runif(10),type="l",col="orangered")
```

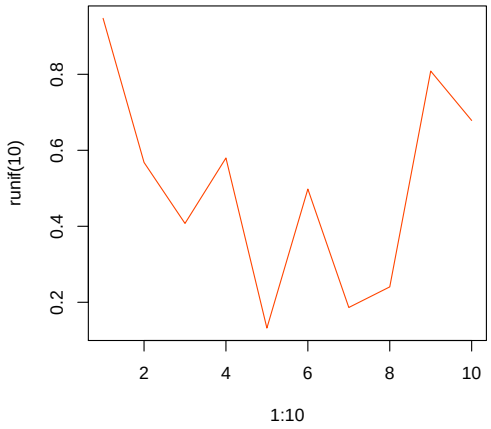


FIGURE 5.11 – Le paramètre `col` de la fonction `plot()`.

Renvoi

Notez que vous pouvez aussi changer la couleur des autres éléments constitutifs du graphique, comme les axes ou le titre par exemple. Pour cela, référez-vous à la section 5.7, page 188, qui traite de la fonction `par()`.

5.3.2 Le codage hexadécimal des couleurs

R offre la possibilité d'utiliser le codage hexadécimal des couleurs, via le paramètre `col` de la fonction `plot()` par exemple. Ainsi, chaque couleur est en fait codée par sa décomposition en couleurs de base : rouge, vert et bleu. Chaque composante peut prendre une valeur entre 0 et 255 (0 : absence totale de la couleur ; 255 : saturation de la couleur). Le codage hexadécimal de ces 256 valeurs donne donc lieu à des codes compris entre `00` et `FF`.

Voici quelques exemples de couleurs :

```
Noir:  #000000
Blanc: #FFFFFF
Vert amande:  #82C46B
Jaune citron: #F7FF3C
Bleu canard:  #048B9A
Bleu nuit:    #10076B
```

Notez que vous pouvez utiliser la fonction `rgb()` pour obtenir ce codage à partir de la décomposition en rouge, vert et bleu d'une couleur.

```
> rgb(red=26,green=204,blue=76,maxColorValue = 255)
[1] "#1ACC4C"
> rgb(red=0.1,green=0.8,blue=0.3)
[1] "#1ACC4D"
```

La fonction `col2rgb()` effectue l'opération inverse :

```
> col2rgb("#1ACC4C")
      [,1]
red     26
green  204
blue    76
```

Il est même possible d'obtenir de la transparence à l'aide du paramètre `alpha` de la fonction `rgb()` :

```
> curve(sin(x),lwd=30,col=rgb(0.8,0.5,0.2),xlim=c(-10,10))
> curve(cos(x),lwd=30,col=rgb(0.1,0.8,0.3,alpha=0.2),add=TRUE)
```

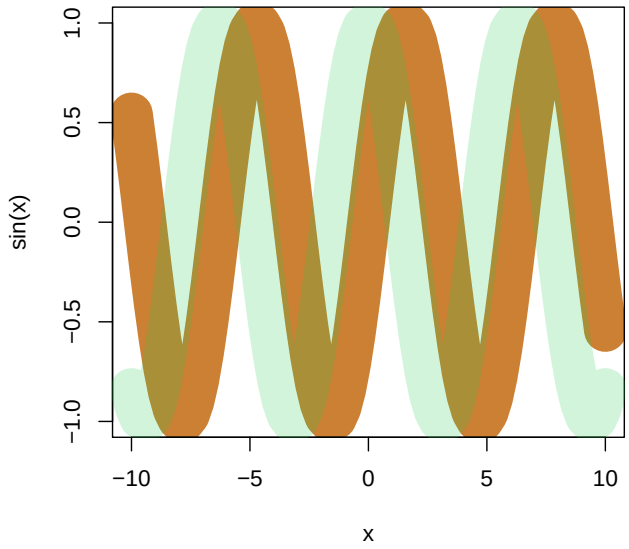
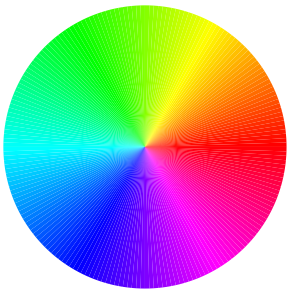


FIGURE 5.12 – Le paramètre `alpha` de la fonction `rgb()`.

Si votre carte graphique le permet, **R** peut donc gérer jusqu'à 256^3 couleurs, soit un peu plus de 16 millions de couleurs. L'exemple ci-après, utilisant la fonction `rainbow()`, permet de s'en faire une petite idée.

```
> pie(rep(1, 200), labels = "", col = rainbow(200), border = NA)
```



```
> n <- 40; plot(1:n, col = rainbow(n), pch = 19, cex = 2)
> grid(col = "grey50")
```

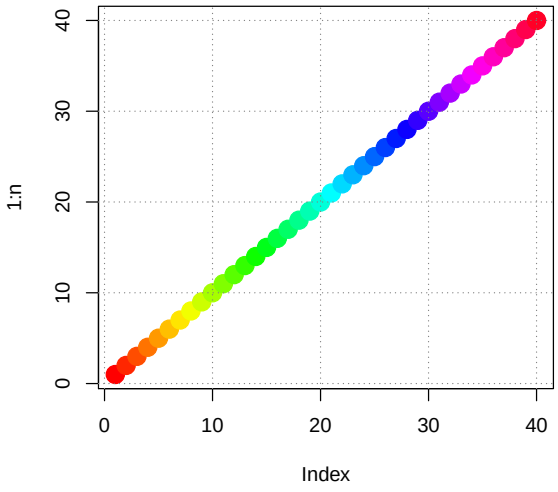


FIGURE 5.13 – Un exemple utilisant la fonction `rainbow()`.

Renvoi

Si vous souhaitez plus de détails sur ce codage ou sur les couleurs en général, allez voir le très beau site dédié aux couleurs `http://www.pourpre.com` et notamment son dictionnaire chromatique.

Par ailleurs, on peut ajouter à **R** le *package* `RColorBrewer`. Cette librairie propose la création automatique de palettes de couleurs idéales pour de belles présentations : dégradés de couleurs dans une teinte, couleurs complémentaires ou divergentes.

```
> require("RColorBrewer")
> display.brewer.all()
```

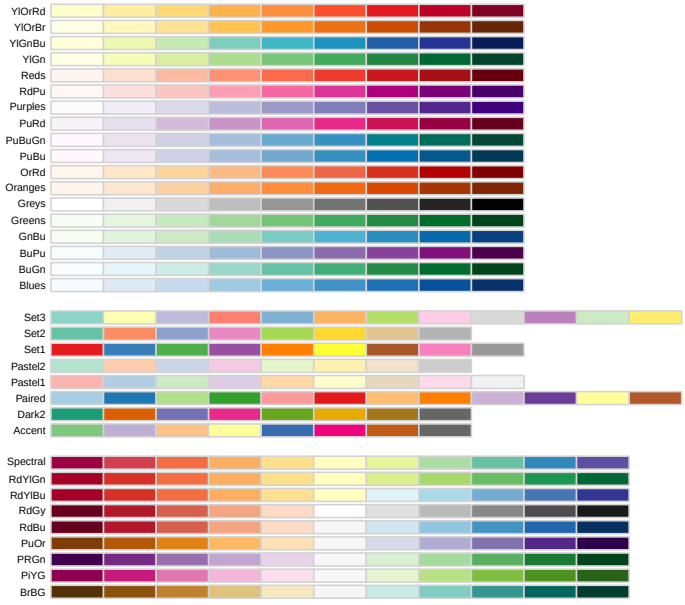


FIGURE 5.14 – La fonction `display.brewer.all()` du *package* `RColorBrewer`.

5.3.3 La fonction `image()`

Cette fonction crée et affiche une grille de rectangles en niveaux de gris ou de couleurs. Ces rectangles sont aussi appelés des pixels (*picture elements*). Elle

peut donc être utilisée pour afficher des données 3D ou des données spatiales, c'est-à-dire des images.

```
> X <- matrix(1:12,nrow=3)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> couleurs <- c("orange","orangered","red","lightblue",
+                    "blue", "white","lightgrey","grey",
+                    "darkgrey","yellow","green","purple")
> image(X,col=couleurs)
> text(rep(c(0,0.5,1),4),rep(c(0,0.3,0.7,1),each=3),1:12,cex=2)
```

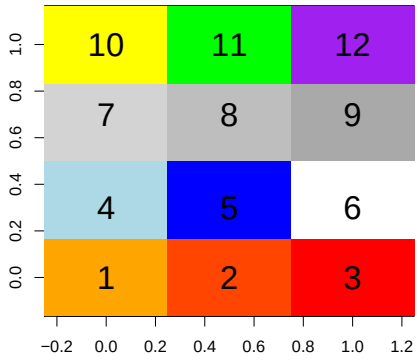


FIGURE 5.15 – La fonction `image()`.

Les numéros dans les cases ont été rajoutés en utilisant la fonction `text()` présentée plus loin.

Attention

Prenez garde à la façon dont sont organisés les rectangles de couleur, de gauche à droite et de bas en haut. Il s'agit donc d'une rotation de 90 degrés dans le sens contraire des aiguilles d'une montre par rapport à l'affichage du contenu de la matrice `X`.

Il est possible d'obtenir un affichage cohérent avec l'organisation des données dans la matrice `X`, en procédant de la façon suivante :

```
> image(as.matrix(rev(as.data.frame(t(X)))),col=couleurs)
> text(rep(c(0,0.33,0.67,1),each=3),rep(c(1,0.5,0),4),1:12,cex=2)
```

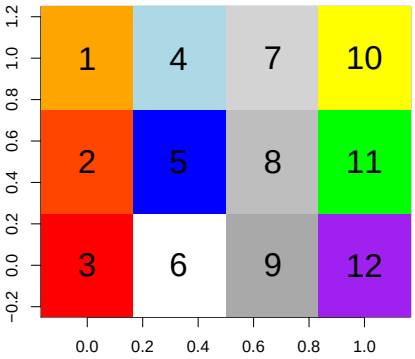


FIGURE 5.16 – La fonction `image()` avec un affichage cohérent avec les données.

Prise en main

Installez puis chargez le *package* `caTools`. Utilisez la fonction `read.gif()` de ce *package* pour lire le fichier `http://www.biostatisticien.eu/springeR/R.gif`. Utilisez la fonction `image()` pour l'afficher dans **R**. Utilisez les couleurs fournies par `read.gif()` et affichez l'image dans le bon sens.

SECTION 5.4

L'ajout de texte

5.4.1 La fonction `text()`

Cette fonction permet d'ajouter du texte sur un graphique. Elle permet aussi, ce qui est très intéressant, d'y ajouter des formules mathématiques. Il faut donner les coordonnées en x et en y du centre de la chaîne de caractères à afficher, et bien entendu la chaîne elle-même. Si l'on veut écrire une expression mathématique, il faut utiliser la fonction `expression()`. La fonction `bquote()` peut aussi être utile.

```
> plot(1:10,1:10,xlab=bquote(x[i]),ylab=bquote(y[i]))
> text(3,6,"du texte")
> text(4,9,expression(widehat(beta) == (X^T * X)^{-1} * X^T * y))
> p <- 4; text(8,4,bquote(beta[.(p)])) # Combiner « math » et
                                       # variables numériques.
```

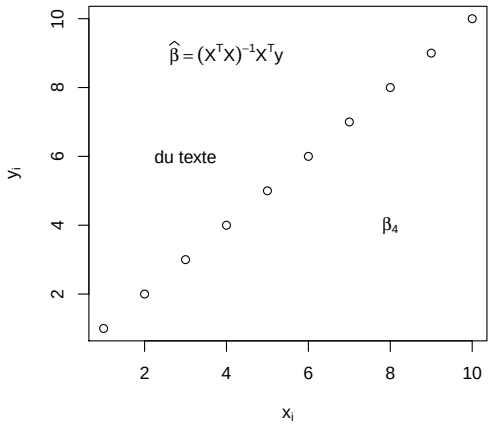


FIGURE 5.17 – La fonction `text()`.

Astuce

Utilisez la commande `demo(plotmath)` pour voir les différentes possibilités offertes pour ajouter des expressions mathématiques sur un graphique. Les commandes à utiliser sont aussi fournies.

Prise en main

Tracez un point aux coordonnées (1,1). Puis ajoutez, aussi aux coordonnées (1,1), le texte `"ABC"` au moyen de la fonction `text()`. Vous observerez également l'effet du paramètre `pos` qui peut prendre les valeurs 1 (en dessous), 2 (à gauche), 3 (au-dessus) et 4 (à droite).

5.4.2 La fonction `mtext()`

Cette fonction permet d'ajouter du texte dans les marges de la fenêtre graphique. Elle permet également d'y ajouter des formules mathématiques.

Elle comprend un paramètre `side` (valant 1=bas, 2=gauche, 3=haut ou 4=droite) qui spécifie dans quelle marge ajouter le texte.

```
> plot(1:10,1:10)
> mtext("bas",side=1)
> mtext("gauche",side=2)
> mtext("haut",side=3)
> mtext(expression(x^2+3*y+hat(beta)),side=4)
```

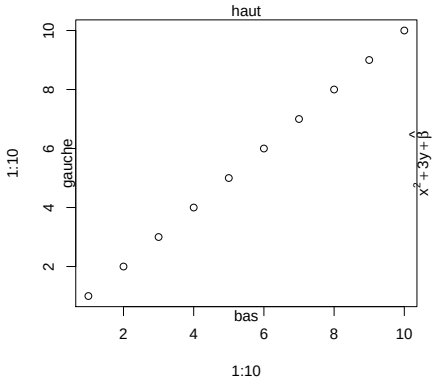


FIGURE 5.18 – La fonction `mtext()`.

SECTION 5.5

Titres, axes et légendes

5.5.1 La fonction `title()`

Cette fonction permet d'ajouter des titres à votre graphique : un titre principal en haut de la figure avec le paramètre `main`, un sous-titre en bas de la figure avec le paramètre `sub`, un titre pour l'axe des x avec le paramètre `xlab`, et un titre pour l'axe des y avec le paramètre `ylab`. Notez que ces paramètres peuvent aussi être spécifiés directement lors de l'appel de fonctions graphiques comme `plot()`.

```
> plot.new()
> box()
> title(main = "Titre principal", sub = "Sous-titre",
+           xlab = "Titre des x", ylab = "Titre des y")
```

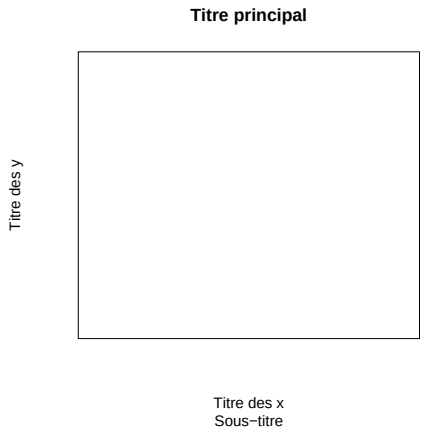


FIGURE 5.19 – La fonction `title()`.

Astuce

Notez qu'il est possible d'écrire un titre sur plusieurs lignes en utilisant le caractère de retour chariot `"\n"`.

```
> plot(1:10,main="Titre sur\n trois\n lignes",
+      xlab="",ylab="")
```

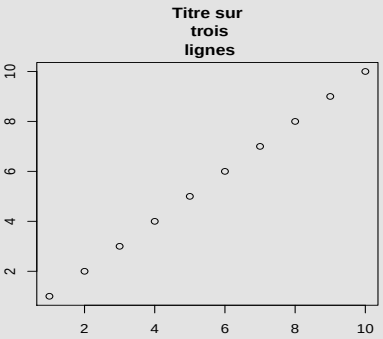


FIGURE 5.20 – Titre sur plusieurs lignes dans un graphique.

5.5.2 La fonction `axis()`

Cette fonction ajoute un axe à un graphique préexistant. Il est possible de spécifier sur quel côté tracer l'axe, les positions des graduations et plusieurs autres paramètres.

Remarque

L'utilisation de la fonction `axis()` se fait en général lorsque l'on veut gérer soi-même plus finement l'aspect des axes. Pour cela, on peut d'abord tracer un premier graphique (par exemple avec la fonction `plot()`) sans les axes, au moyen du paramètre `axes=FALSE`.

Voici quelques-uns des paramètres principaux de la fonction `axis()`.

Argument	Description
`side`	Spécifie sur quel côté tracer l'axe : `side=1` (au-dessous), `side=2` (à gauche), `side=3` (au-dessus), `side=4` (à droite).
`at`	Spécifie où tracer les graduations.
`labels`	Soit un booléen qui spécifie si l'on doit mettre des annotations à chaque graduation, soit une chaîne de caractères spécifiant quoi mettre à chaque graduation.
`tick`	Booléen qui spécifie si les graduations doivent être tracées ou pas.
`col`	Donne la couleur de l'axe.

D'autres paramètres sont disponibles. Vous pouvez les obtenir en consultant l'aide en ligne.

```
> plot.new()
> lines(x=c(0,1),y=c(0,1),col="red")
> axis(side=1,at=c(0,0.5,1),labels=c("a","b","c"),col="blue")
```

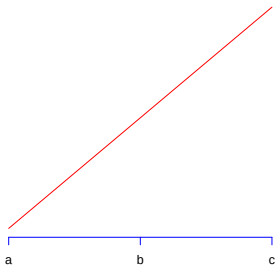


FIGURE 5.21 – La fonction `axis()`.

5.5.3 La fonction `legend()`

Cette fonction est utilisée pour ajouter une légende à un graphique existant.

Voici quelques-uns de ses paramètres :

Argument	Description
`x, y`	Spécifient les coordonnées de l'emplacement de la légende sur un graphique.
`legend`	Vecteur de chaînes de caractères ou d'expressions devant apparaître dans la légende.
`fill`	Vecteur de couleurs devant apparaître à côté du texte dans la boîte de légende.
`lty,lwd`	Entier. Le type des lignes ou l'épaisseur des traits qui apparaissent dans la légende. Il faut indiquer l'un de ces deux paramètres si l'on souhaite obtenir des lignes dans la légende.
`col`	Vecteur de couleurs des points ou des lignes apparaissant dans la légende.

```
> plot(1:4,1:4,col=1:4)
> legend(x=3,y=2.5,legend=c("a","b","c","d"),fill=1:4)
```

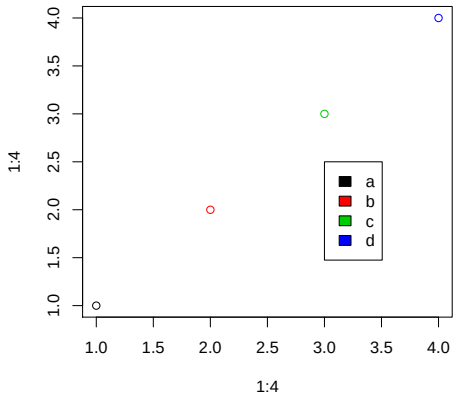


FIGURE 5.22 – La fonction `legend()` avec des carrés.

```
> plot(1:4,1:4,col=1:4,type="b")
> legend(x=3,y=2.5,legend=c("a","b","c","d"),col=1:4,lty=1)
```

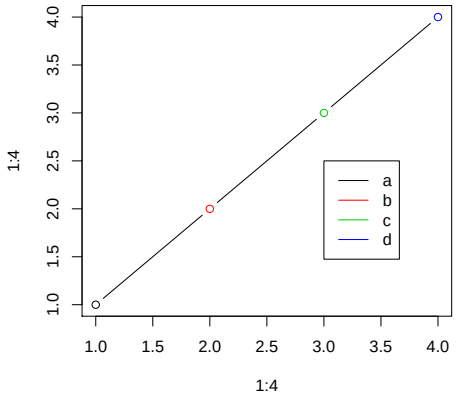


FIGURE 5.23 – La fonction `legend()` avec des segments.

De nombreux autres paramètres sont disponibles, consultables dans l'aide en ligne.

SECTION 5.6

L'interaction avec le graphique

5.6.1 La fonction `locator()`

Elle permet de placer un point sur votre graphique ou d'en repérer les coordonnées au moyen d'un clic de souris. Elle peut par exemple aussi être utile pour placer du texte (ou une légende) à un endroit précis à l'aide de votre souris.

Prise en main

Entrez les commandes suivantes puis cliquez n'importe où sur le graphique obtenu.

```
plot(1,1)
text(locator(1),labels="Ici") # Cliquez sur la
                              # fenêtre graphique.
```

5.6.2 La fonction `identify()`

Elle permet d'identifier et de marquer des points déjà présents sur un graphique. La prise en main suivante permettra de mieux illustrer cette fonction.

Prise en main

Entrez les commandes suivantes puis cliquez près des points sur le graphique obtenu. Notez qu'un clic droit de la souris permet de sortir du mode interactif.

```
> plot(swiss[,1:2])
> x <- identify(swiss[,1:2],labels=rownames(swiss))
> x
```

SECTION 5.7

† La gestion fine des paramètres graphiques : `par()`

La fonction `par()` est une fonction qui possède de très nombreux paramètres vous permettant de peaufiner vos graphiques. Il faut donc utiliser cette fonction pour fixer (ou récupérer) les paramètres graphiques généraux.

L'utilisation de cette commande se fait de la façon suivante :
- `par("`*nom-paramètre*`")` renvoie la valeur par défaut du paramètre *nom-paramètre* de la fonction `par()` ;
- `par("`*nom-paramètre*`"=val)` permet d'attribuer au paramètre *nom-paramètre* la valeur `val` ;
- `par()` renvoie la liste de tous les paramètres graphiques en cours ainsi que leur valeur.

Attention

Avant de faire des modifications des valeurs associées aux paramètres de la fonction `par()`, il est bon de sauvegarder ces valeurs. On pourra ainsi les restaurer par la suite en cas de besoin.

```
# Sauvegarde des valeurs par défaut de par().
sauve.par <- par(no.readonly = TRUE)
# Ensuite, on peut changer certains paramètres graphiques.
par(bg="red")
```

```
# Puis restaurer les anciennes valeurs.
par(sauve.par)
```

Avant de présenter en détail l'utilisation de cette fonction, il est utile de noter que la fenêtre graphique (aussi appelée dispositif graphique, ou *device region*) comprend la zone de figure (*figure region*) qui elle-même contient la région du tracé (*plot region*). Cela est illustré sur la figure 5.24.

Voilà la liste (presque complète) des différents paramètres de la fonction `par()`, accompagnée d'une description succincte. Nous les avons organisés par groupes de fonctionnalités afin qu'il soit plus facile de retrouver le paramètre souhaité.

• Gestion de la fenêtre graphique

TABLE 5.1 – Paramètres de **gestion de la fenêtre graphique**.

Nom	Description
`ask`	Booléen. Si `TRUE`, un message demande à l'utilisateur d'appuyer sur la touche `ENTRÉE` avant que chaque nouveau graphique soit tracé. Utilisez plutôt `devAskNewPage()`.
`din`*	Dimensions `c(largeur,hauteur)` du dispositif graphique, en pouces (*device region inches*).
`fig`	Un vecteur numérique de la forme `c(x1, x2, y1, y2)` qui donne les coordonnées NDC (= *normalized device coordinates*) de la zone de figure dans laquelle sera tracé le graphique (sur la fenêtre graphique).
`fin`	Un vecteur numérique de la forme `c(largeur, hauteur)` qui donne la taille de la zone de figure, en pouces (*figure region inches*).
`mai`	Un vecteur numérique de la forme `c(bas, gauche, haut, droite)` qui donne les tailles des marges, en pouces.
`mar`	Un vecteur numérique de la forme `c(bas, gauche, haut, droit)` qui donne le nombre de lignes de marge sur les 4 côtés du graphique. La valeur par défaut est `c(5, 4, 4, 2) + 0.1`.
`mex`	`mex` est un facteur d'expansion de taille de caractère qui est utilisé pour décrire les coordonnées dans les marges des graphiques. Notez que cela ne change pas la taille de police, mais plutôt spécifie la taille de police (comme un multiple de `csi`) utilisée pour convertir entre `mar` et `mai`, et entre `oma` et `omi`. Sa valeur est 1 lorsque le device est ouvert, puis est réinitialisée quand le *layout* est changé (`cex` est aussi réinitialisé).
`mfcol`, `mfrow`	Un vecteur de la forme `c(nl, nc)`. Les figures successives seront tracées dans une matrice de taille `nl`-par-`nc` dans la fenêtre graphique, par -colonnes- (`mfcol`) ou par -lignes- (`mfrow`) respectivement. Considérez les alternatives, `layout()` et `split.screen()` (*multi-figures filled by columns, resp. rows*).
`mfg`	Un vecteur numérique de la forme `c(i, j)` où `i` et `j` indiquent dans quelle cellule de la matrice de figures doit être tracé le prochain graphique. La matrice de figures doit déjà avoir été définie par le paramètre `mfcol` ou `mfrow`.
`mgp`	La ligne de marge (en unités `mex`) pour le titre des axes, les étiquettes des axes et la ligne des axes. La valeur par défaut est `c(3, 1, 0)`.
`new`	Booléen, par défaut `FALSE`. Si fixé à `TRUE`, la prochaine commande graphique de haut niveau (en fait `plot.new()`) n'effacera pas l'ancien graphique et ajoutera le nouveau par dessus.
`oma`	Un vecteur de la forme `c(bas, gauche, haut, droite)` donnant la taille des marges extérieures en lignes de textes.
`omd`	Un vecteur de la forme `c(x1, x2, y1, y2)` donnant la région à l'intérieur des marges extérieures en NDC (= *normalized device coordinates*), c'est-à-dire en pourcentage (dans [0,1]) de la fenêtre graphique.
`omi`	Un vecteur de la forme `c(bas, gauche, haut, droite)` donnant la taille des marges extérieures, en pouces (*outer margin inches*).
`pin`	Les dimensions de la région du tracé courant, `c(largeur,hauteur)`, en pouces (*plot region inches*).
`plt`	Un vecteur de la forme `c(x1, x2, y1, y2)` donnant les coordonnées de la région de tracé en fractions de la zone de figure courante.
`pty`	Un caractère spécifiant le type de région de tracé à utiliser : `"s"` génère une région de tracé carrée et `"m"` génère la région de tracé maximale (*plot region type*).
`usr`	Un vecteur de la forme `c(x1, x2, y1, y2)` donnant les valeurs extrêmes des coordonnées utilisateur de la région de tracé. Quand une échelle logarithmique est utilisée (c'est-à-dire `par("xlog")` est `TRUE`), alors les limites en x seront $10^{\texttt{par("usr")[1:2]}}$. De même pour les limites en y.
`xpd`	Un booléen ou `NA`. Si `FALSE`, tous les graphiques sont attachés à la région de tracé. Si `TRUE`, tous les graphiques sont attachés à la zone de figure. Si `NA`, tous les graphiques sont attachés à la fenêtre graphique. Voir aussi `clip()`.

* Un astérisque a été accolé aux paramètres qui ne sont pas modifiables par l'utilisateur (en lecture seule).

Le graphique suivant permet de mieux comprendre la portée de certains de ces paramètres.

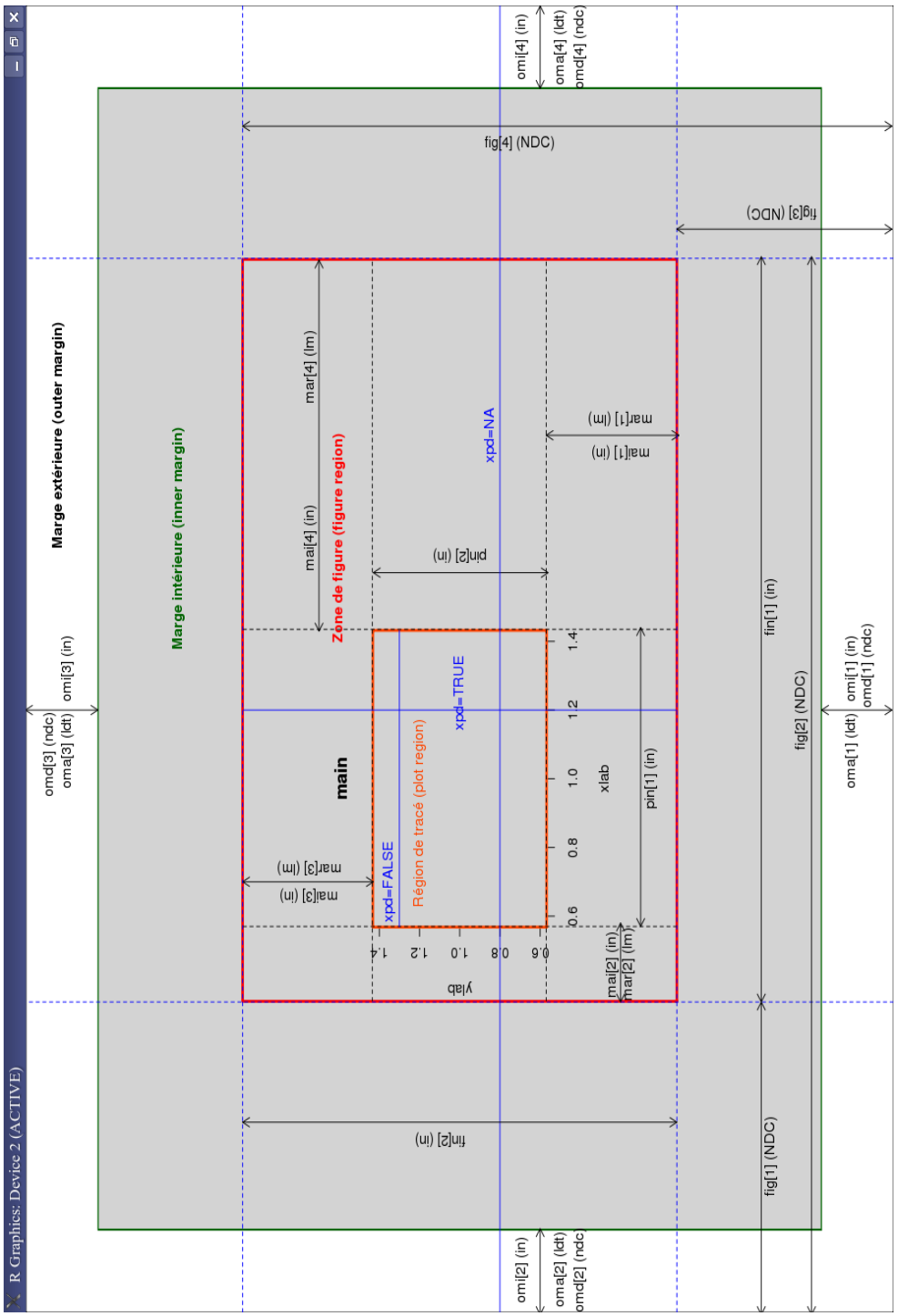


FIGURE 5.24 – Figure illustrant la gestion fine des paramètres graphiques.

• **Gestion de la couleur**

TABLE 5.2 – Paramètres de **gestion de la couleur**.

Nom	Description
`bg`	La couleur d'arrière-plan du dispositif graphique (*Background*).
`col`	La couleur du graphique.
`col.axis`	La couleur pour l'annotation des axes.
`col.lab`	La couleur pour les étiquettes en x et en y.
`col.main`	La couleur pour le titre général.
`col.sub`	La couleur des sous-titres.
`fg`	La couleur d'avant-plan (axes et boîte autour du graphique). Fixe `col` à la même valeur (*foreground*).

Voici une mise en situation de ces paramètres :

```
> par(bg="lightgray",col.axis="darkgreen",col.lab="darkred",
+             col.main="purple",col.sub="black",fg="blue")
> curve(cos(x),xlab="xlab en darkred",main="Titre en purple",
+              xlim=c(-10,10),sub="sub en black")
> curve(sin(x),col="blue",add=T)
```

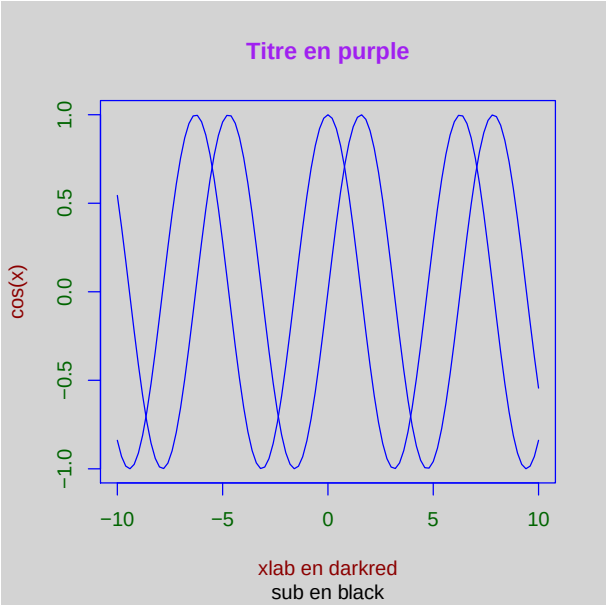


FIGURE 5.25 – Gestion des couleurs sur un graphique.

• **Gestion du texte**

TABLE 5.3 – Paramètres de **gestion du texte** affiché sur le graphique.

Nom	Description
`adj`	La valeur de `adj` détermine la façon dont les chaînes de caractères sont justifiées dans `text()`, `mtext()` et `title()`. Une valeur de `0` produit du texte justifié à gauche, `0.5` du texte centré et `1` du texte justifié à droite. Toute valeur dans $[0, 1]$ est permise et parfois même des valeurs hors de cet intervalle fonctionneront aussi. Notez que le paramètre `adj` de la fonction `text()` autorise aussi `adj = c(x, y)` pour des ajustements différents en x et en y. Notez que pour `text()`, le texte est justifié par rapport à un point alors que pour `mtext()` et `title()`, il est justifié par rapport à la région du tracé ou à la fenêtre graphique.
`ann`	Si fixées à `FALSE`, les fonctions graphiques de haut niveau n'ajoutent pas d'annotations sur les graphiques qu'elles produisent (axes et titres généraux). Par défaut, l'annotation est faite.
`cex`	Une valeur numérique donnant le coefficient de grossissement du texte et des symboles sur le graphique (par rapport à une valeur de référence) (*character expansion*).
`cex.axis`	Le grossissement pour l'annotation des axes (par rapport à une valeur de référence).
`cex.lab`	Le grossissement pour les étiquettes des axes x et y.
`cex.main`	Le grossissement du titre général.
`cex.sub`	Le grossissement des titres secondaires.
`cin`*	Taille des caractères en pouces `c(largeur,hauteur)` (*character inches*).
`cra`*	Taille des caractères en pixels `c(largeur,hauteur)`.
`crt`	Une valeur numérique spécifiant (en degrés) comment les différents caractères doivent être inclinés. Cela doit être des multiples de 90. À comparer avec `srt` qui fait des rotations de chaînes de caractères (*character rotation*).
`csi`*	Hauteur des caractères en pouces. Identique à `par("cin")[2]`.
`cxy`*	Taille des caractères `c(largeur,hauteur)` en unités exprimées par rapport aux coordonnées utilisateur. `par("cxy")` est égal à `par("cin")/par("pin")` fois un facteur d'échelle en coordonnées utilisateur. Notez que `c(strwidth(ch), strheight(ch))` pour une chaîne donnée `ch` est en général beaucoup plus précis.
`family`	Le nom d'une famille de polices pour le texte. La taille maximale autorisée est 200 octets. Ce nom est mis en relation par chaque dispositif graphique avec une description de police spécifique au dispositif. La valeur par défaut est `""` qui signifie que les polices par défaut seront utilisées (voir la fiche d'aide du dispositif). Des valeurs courantes sont `serif`, `sans` et `mono`, les familles de police d'Hershey sont aussi disponibles. Cela peut être spécifié dans la fonction `text()`.
`font`	Un entier qui spécifie quelle police utiliser pour le texte. En général, 1 correspond à du texte ordinaire, 2 à du gras, 3 à du texte en italique et 4 à du gras en italique. 5 devrait être la police de symboles (encodage d'Adobe).
`font.axis`	La police pour l'annotation des axes.
`font.lab`	La police pour les étiquettes des axes x et y (*font for labels*).
`font.main`	La police pour le titre général.
`font.sub`	La police pour les sous-titres (*font for subtitle*).
`ps`	Entier. La taille du texte, en points (mais pas des symboles) (*point size*).
`srt`	Rotation des chaînes de caractères en degrés. Voir le commentaire sur `crt`. Seulement supporté par `text()` (*string rotation*).

* Un astérisque a été accolé aux paramètres qui ne sont pas modifiables par l'utilisateur (en lecture seule).

Voici une mise en situation des paramètres `adj` et `srt` :

```
> par(mfrow = c(1, 3))
> vals <- c(0, 0.5, 1)
> for (adj in vals) {
+ par(adj = adj)
+ plot(0, main = paste("adj =", adj), col.lab = "red",
```

```
+   col.main = "red",type = "n")
+ text(1, 0, "abc", col = "red", cex = 2)
+ abline(h = 0, lty = 2)
+ abline(v = 1, lty = 2)
+ }
> abline(v=0.8,h=-0.5,lty=2)
> text(0.8, -0.5, "abc", col = "red", cex = 2,adj=c(0,1))
> abline(h=0.5,v=0.5,lty=2)


> text(0.8,0.5,"ABC",col="red",cex=2,adj=c(0.5,0.5),srt=120)
```

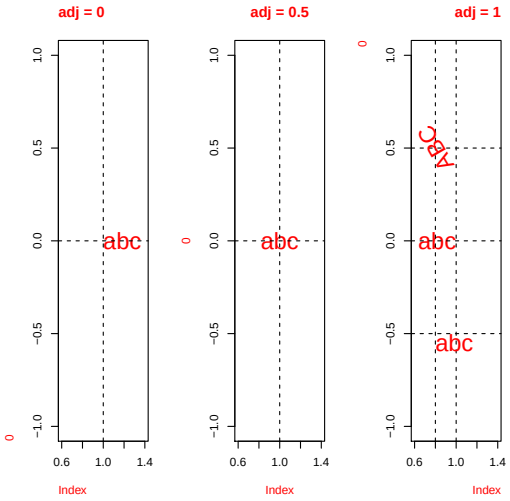


FIGURE 5.26 – Mise en situation des paramètres `adj` et `srt`.

Voici un deuxième exemple illustrant l'utilisation de quelques polices :

```
> par(cex.axis=1.5)
> plot(1:5,y=rep(1,5),type="n",font.axis=2,font.lab=3,xlab=
+  "xlab en italique",ylab="",font.main=4,main="Titre en
+  gras/italique",font.sub=5,sub="Sous-titre en police de
+  symboles")
> text(2,1.2,"Texte ordinaire")
> par(ps=30)
> text(3,1,"Une police Hershey",family="HersheyScript")
> par(ps=14)
```

```
> text(3.5,0.8,"Une autre police Hershey",
+                      family = "HersheyGothicEnglish")
```

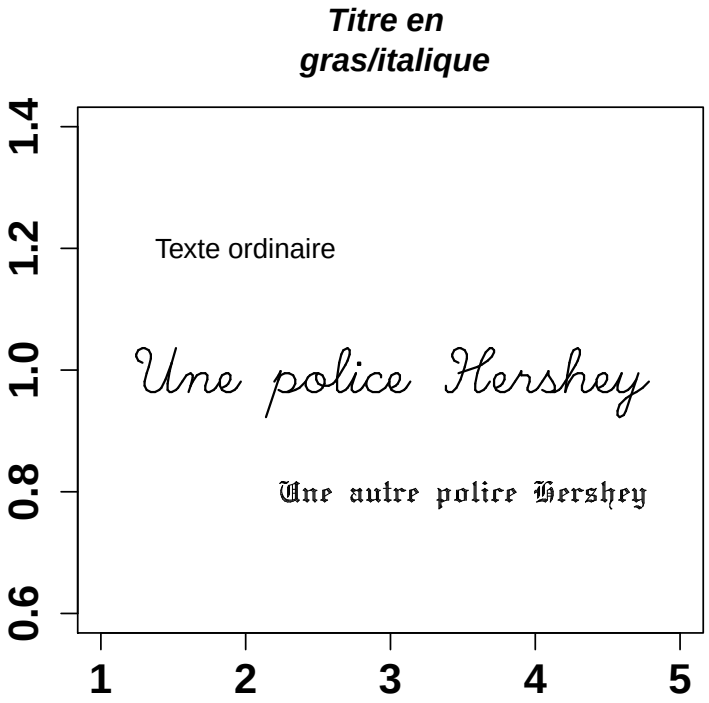


FIGURE 5.27 – Utiliser diverses polices sur un graphique.

Astuce

Pour visionner tous les symboles et polices disponibles sous **R**, il sera avantageux d'utiliser la commande suivante :

```
demo(Hershey)
```

• Gestion des axes

TABLE 5.4 – Paramètres pour la **gestion des axes**.

Nom	Description
`bty`	Une chaîne de caractères qui détermine le type de boîte qui entoure les graphiques (les axes). Si `bty` est l'un des caractères `"o"` (le défaut), `"l"`, `"7"`, `"c"`, `"u"`, ou `"]"` la boîte résultante ressemble au caractère correspondant. Une valeur de `"n"` supprime la boîte (*box type*).
`lab`	Un vecteur numérique de la forme `c(x, y, len)` qui modifie la façon dont les axes sont annotés. Les valeurs de `x` et `y` donnent (approximativement) le nombre de graduations sur les axes x et y. `len` spécifie la taille de l'étiquette. La valeur par défaut est `c(5, 5, 7)`. Notez que cela n'affecte que la façon dont les paramètres `xaxp` et `yaxp` sont fixés quand le système de coordonnées est mis en place, et que ce n'est pas consulté lorsque les axes sont tracés. `len` n'est pas encore implémenté.
`las`	Nombre appartenant à {0,1,2,3}. Style des étiquettes des axes. 0=toujours parallèle aux axes (le défaut), 1=toujours horizontal, 2=toujours perpendiculaire aux axes, 3=toujours vertical. Notez que le paramètre `srt` de `par()` qui gère la rotation des chaînes de caractères n'affecte pas les étiquettes des axes.
`tck`	La longueur des graduations (*tick marks*) sur les axes en fraction du minimum entre la hauteur et la largeur de la région de tracé. Si `tck >= 0.5` il est interprété comme une fraction du côté pertinent, ainsi si `tck=1` une grille est dessinée. Le défaut (`tck = NA`) est d'utiliser `tcl = -0.5`.
`tcl`	La longueur des graduations en fraction de la hauteur d'une ligne de texte. La valeur par défaut est `-0.5`. En mettant `tcl = NA` cela fixe `tck = -0.01`.
`xaxp`	Un vecteur de la forme `c(x1, x2, n)` donnant les coordonnées des graduations extrêmes et le nombre d'intervalles entre les graduations quand `par("xlog")` est `FALSE`. Autrement, quand l'échelle est logarithmique, les trois valeurs ont une signification différente. Voir l'aide en ligne pour plus de détails. Voir aussi `axTicks()`.
`xaxs`	Le style de calcul des intervalles des axes à utiliser pour l'axe des x. Les valeurs possibles sont `"r"`, `"i"`, `"e"`, `"s"`, `"d"`. Les styles sont généralement contrôlés par l'étendue des données, ou `xlim` s'il est fourni. Le style `"r"` (*regular*) étend d'abord l'étendue des données par 4 pour cent à chaque extrémité et ensuite trouve un axe avec de jolies étiquettes qui entrent dans l'étendue. Le style `"i"` (*internal*) trouve uniquement un axe avec des jolies étiquettes qui rentrent dans l'étendue d'origine des données. Le style `"s"` (*standard*) trouve un axe avec de jolies étiquettes dans lequel l'étendue des données d'origine rentre. Le style `"e"` (*extended*) est comme le style `"s"`, mis à part qu'il laisse aussi de la place pour tracer des symboles à l'intérieur de la *bounding box*. Le style `"d"` (*direct*) spécifie que les axes en cours doivent être utilisés sur les graphiques subséquents. Seuls les styles `"r"` et `"i"` sont pour l'instant implémentés.
`xaxt`	Un caractère qui spécifie le type d'axe pour les x. Mettre `"n"` implique qu'un axe est créé mais pas tracé. La valeur standard est `"s"`.
`xlog`	Booléen (voir `log` dans `plot.default()`). Si `TRUE`, une échelle logarithmique est utilisée (par exemple, après `plot(, log = "x")`). Pour une nouvelle fenêtre graphique, le défaut est `FALSE`, c'est-à-dire une échelle linéaire.
`yaxp`	Un vecteur de la forme `c(y1, y2, n)` donnant les coordonnées des graduations extrêmes et le nombre d'intervalles entre ces graduations sauf pour l'échelle logarithmique, voir `xaxp` au-dessus.
`yaxs`	Le style du calcul des intervalles des axes à utiliser pour l'axe des y. Voir `xaxs` au-dessus.
`yaxt`	Un caractère qui spécifie le type d'axes. Si on met `"n"` l'axe est défini mais pas tracé.
`ylog`	Booléen ; voir `xlog` au-dessus.

Voici une mise en situation de quelques-uns de ces paramètres :

```
> # Agrandit la marge du bas pour faire de la place pour les
  # étiquettes inclinées des x.
> par(mar = c(7, 4, 4, 2) + 0.1)
> # Définit un style de boîte, dix graduations en x et
  # y, des étiquettes horizontales,
```

```
> # et des graduations de longueur 1 (ce qui donne un
  # quadrillage).
> par(bty="7",col="blue",lab=c(10,10,1),las=1,tck=1)
> # Crée un graphique sans axe des x et sans étiquette
  # des x.
> plot(1 : 8, xaxt = "n",  xlab = "")
> # Ajoute l'axe des x uniquement avec les graduations.
> axis(1, labels = FALSE)
> # Création du vecteur des étiquettes.
> labels <- paste("Label", 1:8, sep = " ")
> # Rajoute les étiquettes des x aux graduations par
  # défaut.
> text(1:8, par("usr")[3] - 0.25, srt = 45, adj = 1,
+        labels = labels, xpd = TRUE)
> # Rajoute un sous-titre en bas, à la sixième ligne de marge
  # (sur 7).
> mtext(1, text = "Étiquettes de l'axe des X", line = 6)
```

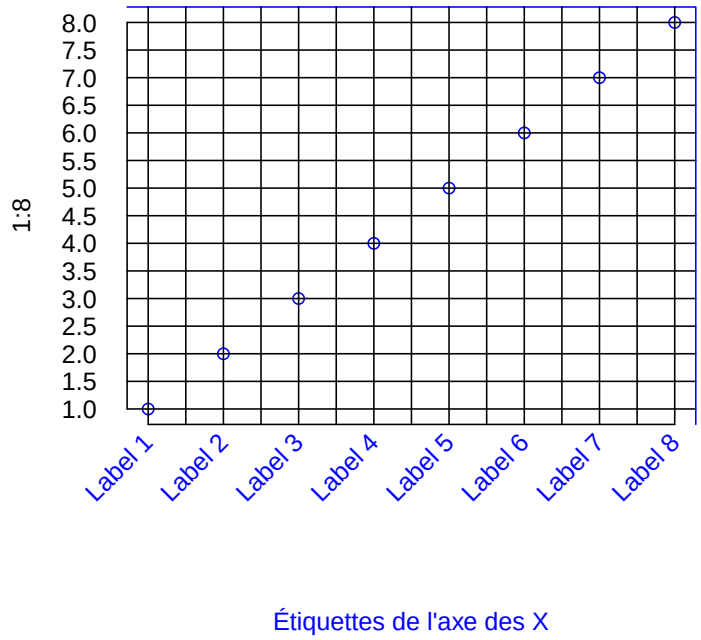


FIGURE 5.28 – Gestion des étiquettes sur un graphique.

• **Gestion des lignes et symboles**

TABLE 5.5 – Paramètres pour la **gestion des lignes et symboles**.

Nom	Description
`lend`	Le style de fin de ligne. Il peut être spécifié au moyen d'un entier ou d'une chaîne de caractères : `0` ou `"round"` signifie qu'un demi-disque est ajouté en fin de ligne ; `1` ou `"butt"` signifie que la ligne se termine droite ; `2` ou `"square"` signifie qu'un petit carré est ajouté à la fin de la ligne (*line end*).
`lheight`	Le multiplicateur de hauteur de ligne. La hauteur d'une ligne de texte utilisée pour espacer verticalement du texte (s'étendant sur plusieurs lignes) est trouvée en multipliant la hauteur des caractères par à la fois le facteur courant d'expansion de caractère et par le multiplicateur de hauteur de ligne. La valeur par défaut est 1. Utilisée dans `text()` et `strheight()`.
`ljoin`	Le style de jointure des lignes. Il peut être spécifié au moyen d'un entier ou d'une chaîne de caractères : `0` ou `"round"` signifie une jointure arrondie, défaut ; `1` ou `"mitre"` signifie une jointure droite ; `2` ou `"bevel"` signifie une jointure biseautée.
`lmitre`	Cela contrôle à partir de quand des jointures droites sont automatiquement converties en des jointures biseautées. La valeur doit être supérieure à 1 et la valeur par défaut est 10. Ne fonctionne pas avec tous les périphériques.
`lty`	Le type de ligne. Peut être spécifié soit par un entier (0=vide, 1=pleine, 2=tirets, 3=points, 4=points et tirets, 5=tirets longs, 6=deux tirets), soit par l'une des chaînes de caractères `"blank"`, `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"` ou `"twodash"`. Notez que `"blank"` utilise des lignes invisibles (c'est-à-dire ne les trace pas). On peut aussi donner une chaîne de caractères (de longueur inférieure ou égale à 8) donnant la longueur des segments de ligne pleins et vides. Voir la section `Spécification du type de lignes` dans l'aide en ligne.
`lwd`	La largeur du trait des courbes (un nombre positif), par défaut `1`.
`pch`	Ou bien un entier spécifiant un symbole ou bien un caractère unique remplaçant les petits cercles dans les graphiques de points.

Le graphique suivant permet de mieux comprendre les paramètres `lend` et `ljoin`.

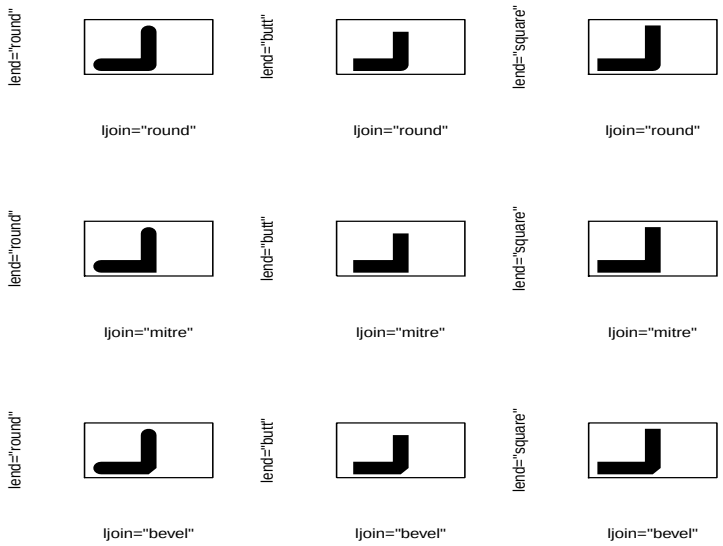


FIGURE 5.29 – Les paramètres `lend` et `ljoin`.

La figure ci-dessous présente les différents symboles obtenus via le paramètre `pch`. Le type des points pour les graphiques est contrôlé avec le paramètre `pch`. Les points de 0 à 20 sont d'une seule couleur contrôlée avec le paramètre `col`. Les points de 21 à 25 ont en plus une couleur de remplissage contrôlée avec le paramètre `bg` de la fonction `points()`.

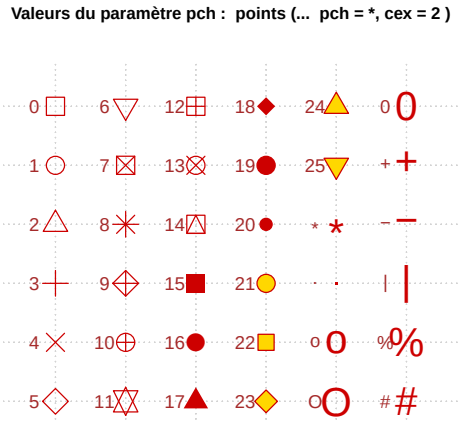


FIGURE 5.30 – Le paramètre `pch`.

Le graphique suivant illustre l'utilisation des paramètres `lty` et `lwd` :

```
> plot(1,1,type="n")
> for (i in 0:6) abline(v=0.6+i*0.1,lty=i,lwd=i)
> abline(v=1.3,lty="92",lwd=10)
```

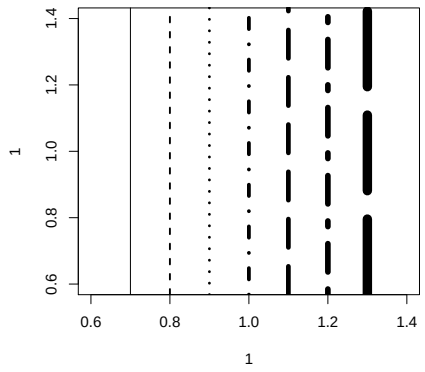


FIGURE 5.31 – Les paramètres `lty` et `lwd`.

SECTION 5.8

† Graphiques avancés : `rgl`, `lattice` et `ggplot2`

Il existe d'autres *packages* dans R permettant de gérer les graphiques de façon plus avancée. Nous ne pourrons pas décrire en détail ces *packages* par manque de place. Nous nous contentons donc de donner quelques exemples frappants qui inciteront peut-être le lecteur plus avancé à s'y intéresser.

- *Package* `rgl`

Ce *package* permet d'obtenir de jolis graphiques en 3D qu'il est possible de faire bouger à l'aide de la souris. Tapez par exemple les commandes suivantes pour en avoir un bref aperçu :

```
require("rgl")
demo(rgl)
example(rgl)
```

- *Package* `lattice`

Notons tout d'abord qu'un livre complet est dédié à ce *package* [42]. Nous donnons uniquement ici un exemple qui montre que les graphiques dans le

package `lattice` peuvent être considérés comme des objets (au sens de la programmation objet), ce qui plaira aux lecteurs avertis. Imaginons par exemple que vous ayez tracé le graphique résultant des instructions suivantes, et que vous constatiez votre erreur dans le titre.

```
x <- 1:100
y <- sin(x)
plot(x,y,type="l",main="Courbe de cosinus")
```

Votre option pour résoudre le problème sera alors de retracer entièrement la figure, avec le bon titre cette fois-ci.

Une utilisation du *package* `lattice` aurait permis d'éviter cet écueil.

```
require("lattice")
xyplot(y~x,type="l",main="Courbe de cosinus")
```

L'instruction suivante permet alors de modifier le titre sans retracer la courbe !

```
update(trellis.last.object(),main="Courbe de sinus")
```

- *Package* `ggplot2`

Nous nous contentons ici de mentionner l'existence du *package* `ggplot2` qui explicite les liens conceptuels entre graphiques et analyses statistiques. Vous pouvez consulter le site de ce *package* à l'adresse `http://ggplot2.org` et le site du livre qui y est associé à l'adresse `http://ggplot2.org/book`.

Termes à retenir

`dev.off()` : fermer la fenêtre graphique active
`savePlot()` : sauvegarder dans un fichier le contenu de la fenêtre graphique active
`layout()` : découpage de la fenêtre graphique en sous-cases
`plot()` : tracer un nuage de points reliés éventuellement entre eux
`points()` : ajouter à un graphique existant un nuage de points éventuellement reliés entre eux
`segments(), lines(), abline()` : ajouter des lignes à un graphique
`arrows()` : ajouter une flèche à un graphique
`polygon()` : tracer un polygone
`curve()` : tracer une courbe donnée par son équation
`box()` : ajouter une boîte autour du graphique courant
`colors()` : renvoie la liste des noms de couleurs connues de R
`text()` : ajouter du texte ou des symboles mathématiques à un graphique
`mtext()` : ajouter du texte dans les marges d'un graphique
`title()` : gestion des titres d'un graphique
`axis()` : ajouter un axe à un graphique
`legend()` : ajouter une légende à un graphique
`locator()` : détecter les coordonnées d'un point sur un graphique au moyen d'un clic de souris
`identify()` : identifier un point déjà présent sur un graphique
`par()` : gestion fine de tous les paramètres graphiques

Exercices

5.1- À quoi sert la commande `windows()` ? Et la commande `dev.off()` ?

5.2- Imaginons qu'un graphique ait été créé à l'aide de la commande `curve(cos(x))`. Quelle instruction R utilisez-vous pour sauvegarder ce graphique dans un fichier au format PDF nommé monplot.pdf ?

5.3- Expliquez en détail à quoi sert l'instruction suivante : `par(mfrow=c(3,2))`.

5.4- À quoi sert la fonction `layout()` ?

5.5- Quelle est la commande à utiliser pour ajouter un nuage de points sur un graphique déjà existant ?

5.6- Quel paramètre de la fonction `plot()` permet d'obtenir des points reliés par des segments de lignes ?

5.7- Citez une fonction permettant de tracer une droite.

5.8- À quoi sert la fonction `curve()` ?

5.9- Quel paramètre utilisez-vous pour gérer la couleur dans les graphiques ?

5.10- Quelle fonction permet d'afficher une image ? Donnez l'instruction permettant d'afficher l'image dont les valeurs sont données dans la matrice `X`, de façon cohérente avec l'affichage de `X` dans la console.

5.11- Quelle fonction permet d'ajouter du texte sur un graphique ?

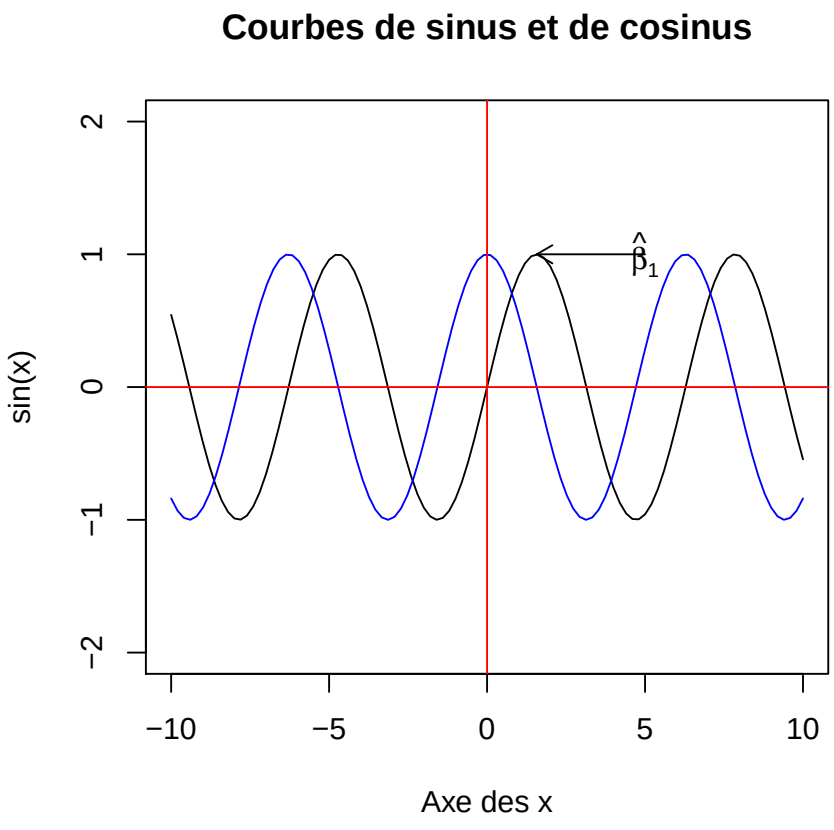
5.12- Quelle fonction vous permet de repérer les coordonnées d'un point sur une fenêtre graphique au moyen d'un clic de souris ?

5.13- Expliquez en détail à quoi sert l'instruction suivante : `par(ask=TRUE)`.

5.14- Quel paramètre de la fonction `par()` vous permet de gérer le type de lignes qui sont tracées par la fonction `curve()` ?

5.15- Quel paramètre de la fonction `par()` vous permet de remplacer les petits cercles dans les nuages de points par d'autres symboles ?

5.16- Donnez la liste d'instructions à utiliser pour obtenir le graphique suivant. Les axes du repère central seront affichés en rouge. La courbe de cosinus sera affichée en bleu.



Fiche de TP

Création de graphiques divers et variés

A- Nombres complexes

5.1- Reproduisez le graphique sur les nombres complexes du chapitre 1 en page 51.

B- Dessiner le drapeau du Canada

5.1- Installez puis chargez le *package* `caTools`.

5.2- Utilisez la fonction `read.gif()` pour lire l'image `http://www.biostatisticien.eu/springeR/canada.gif`

5.3- Affichez cette image en utilisant la fonction `image()`.

5.4- Retracez ce drapeau dans une autre fenêtre en vous servant des fonctions `plot()`, `rect()` et `polygon()` (indice : utilisez la fonction `locator()`).

C- Graphiques de tables de fréquences

Le tableau suivant représente des scores de sensation de brûlure pour seize sujets soumis à une étude visant à tester un nouveau pansement hydrogel. La première colonne contient le numéro du sujet. Les colonnes suivantes contiennent l'indice de sensation de brûlure (sur une échelle de 1 à 4) pour les semaines de 1 (`W1`) à 7 (`W7`).

Nr	W1	W2	W3	W4	W5	W6	W7
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	2
3	1	1	1	1	1	2	3
4	1	1	1	1	1	3	4
5	1	1	1	1	2	3	3
6	1	1	1	1	1	1	1
7	1	1	1	3	4	2	2
8	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1
10	1	1	1	1	1	1	4
11	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1
13	1	2	1	3	2	3	4

```
14      1       1       1       2       2       4       4
15      1       1       1       1       1       1       1
16      1       1       1       1       1       1       1
```

Nous allons proposer une représentation graphique intéressante pour ce type de données.

5.1- Pour la semaine `W7`, calculez le vecteur $(f_1, 1-f_1, f_2, 1-f_2, f_3, 1-f_3, f_4, 1-f_4)$ où f_i est la fréquence de la modalité i ($1 \leq i \leq 4$) observée pendant la semaine `W7` sur les seize sujets. (indice : utilisez la fonction `tabulate()`, `cbind()`, `t()` et `as.vector()`).

5.2- Maintenant, utilisez la fonction `apply()` pour faire le même calcul sur toutes les semaines. Le résultat sera stocké dans une matrice.

5.3- Utilisez la fonction `barplot()` et le paramètre `col=c("black","white")` sur cette matrice. Le graphique que vous obtenez permet de rapidement se faire une idée de l'évolution de la distribution de la variable `Sensation de brulure` au cours du temps.

5.4- Modifiez le graphique précédent afin de changer la couleur des barres représentant les fréquences en rouge. Les numéros des semaines devront être écrits en bleu, en haut du graphique et non plus en bas. Les numéros des modalités devront être écrits à gauche, en bleu. Un titre sera ajouté au graphique.

D- Affichage d'images anatomiques du cerveau

Les données acquises au cours d'un examen d'imagerie par résonance magnétique (IRM) du cerveau humain sont en général stockées dans un fichier binaire d'extension `*.img`. Nous allons voir comment lire et afficher ce type de données.

5.1- Importez le fichier `http://www.biostatisticien.eu/springeR/anat.img`, qui contient l'image d'une seule coupe cérébrale de 256×256 pixels, au moyen de la fonction `readBin()`. Ces données peuvent être considérées comme une suite de 256×256 paires d'octets (`raw`). Vous stockerez ces données dans un objet nommé `octets`.

5.2- Lors de l'écriture de ces données, chaque paire d'octets a en fait été écrite en sens inverse (c'est-à-dire par exemple la paire d'octets `02 56` a été écrite `56 02`). Vous devez donc permuter toutes les paires d'octets deux à deux. Vous stockerez le résultat de cette opération dans `x`.

5.3- Il faut maintenant transformer cette suite de paires d'octets en valeurs numériques que l'on pourra représenter graphiquement. Lorsque l'on a deux octets (par exemple `02 56`), il faut utiliser l'instruction `as.numeric("0x0256")` pour obtenir la valeur décimale correspondant à cette paire d'octets (dans ce cas on obtient `598`). Transformez `x` en valeurs décimales et stockez le résultat dans un objet nommé `valeurs` (indice : utilisez les fonctions `matrix()`, `apply()` et `paste()`).

5.4- Recréez la matrice de taille 256×256 contenant les observations présentes dans `valeurs`.

5.5- Utilisez la fonction `image()` sur la matrice `X` ainsi créée. Vous utiliserez un dégradé de couleurs en niveaux de gris avec une centaine de teintes, obtenu grâce à la fonction `gray()`.

5.6- Notez que c'est exactement ce qui est fait dans le *package* R `AnalyzeFMRI`. Après avoir téléchargé les deux fichiers `http://www.biostatisticien.eu/springeR/anat.img` et `http://www.biostatisticien.eu/springeR/anat.hdr`, vous auriez donc pu obtenir la même chose que précédemment (après installation du *package* `AnalyzeFMRI`) en tapant :

```
require("AnalyzeFMRI")
Y <- f.read.volume("/chemin/vers/anat.img") # Chemin à
                                            # remplacer.
image(X,col=gray(0:1000 / 1000))
```

E- Dessiner la carte d'une région française

Le *package* `maps` contient de nombreuses cartes de divers pays. Nous allons l'utiliser pour représenter les contours d'un département français.

5.1- Installez, puis chargez les *packages* `maps` et `mapdata`.

5.2- Tracez la carte de la France : `map("france")`.

5.3- Récupérez les données des contours des régions françaises : `france <- map("france",plot=FALSE)`

5.4- Affichez le contenu de l'objet `france` et assurez-vous de bien comprendre son organisation. Vous noterez par exemple que les données de latitude/longitude sont organisées dans `france$x`/`france$y` pour chaque département dans `france$names` (jusqu'au prochain `NA`).

5.5- Créez un vecteur `indNA` contenant le vecteur des indices des valeurs manquantes.

5.6- Créez un objet contenant le nom de votre département (par exemple `nomdept <- "Gard"`).

5.7- Créez un objet nommé `inddept` qui contient l'indice du département `nomdept` dans le vecteur `france$names`.

5.8- Tracez la carte de votre département.

5.9- Rajoutez le point d'une localité (ou d'une adresse) sur la carte. Vous pouvez récupérer les coordonnées (latitude/longitude) de cette localité en vous rendant par exemple sur le site `http://www.gpsvisualizer.com/geocode`.

F- Représentation du géoïde en France

Le géoïde peut être considéré comme une surface équipotentielle de pesanteur, passant par l'origine du nivellement, c'est-à-dire ajustée au niveau moyen des mers.

5.1- Importez le fichier `http://www.biostatisticien.eu/springeR/raf98.gra` dans une matrice au moyen de la fonction `scan()`. Avant cela, lire le fichier associé `http://www.biostatisticien.eu/springeR/formatgeoide.txt` qui contient la description du format de ce fichier.

5.2- Essayez de reproduire le graphique disponible ici : `http://www.biostatisticien.eu/springeR/geoide.png`. Ne pas essayer de superposer la carte de France pour l'instant (indice : utilisez les fonctions `scan()`, `layout()`, `par()`, `image()`, `axis()`, `contour()`, `legend()` et `rainbow()`).

Chapitre 6

Programmation en R

Pré-requis et objectif

- Lecture de tous les chapitres précédents. Cependant, un utilisateur non expérimenté pourra dans une première lecture de ce livre survoler le contenu de ce chapitre. Il est en effet bien connu que la programmation dans un langage est d'un niveau plus avancé que celui de l'utilisation d'un langage.
- L'objectif est ici de proposer à l'utilisateur la possibilité de développer de nouvelles fonctions, ce qui correspond en R à étendre le langage. L'utilisateur pourra alors compléter sa compréhension sur le fonctionnement de R.

SECTION 6.1

Préambule

Le point fort du système R est qu'il intègre un vrai langage de programmation. Nous verrons qu'il propose des concepts de programmation très originaux. Le concept d'objet est très présent dans le langage R. La programmation orientée objet utilisée dans R est transparente pour l'utilisateur dans le sens où il n'a pas besoin d'en comprendre la théorie pour pouvoir l'utiliser. Il n'en est pas de même lorsque l'on se place du point de vue du développeur souhaitant respecter l'esprit du langage R.

Problématique
Imaginons qu'un utilisateur débutant en R veuille s'initier à la programmation R en développant quelques fonctions relatives à la méthode bien connue des moindres carrés [1] dans le cadre de la régression linéaire simple. Il imagine

1. Voir par exemple, `http://fr.wikipedia.org/wiki/Méthode_des_moindres_carrés`

très vite l'intérêt de deux tâches particulières : la première consiste à fournir un résumé proposant les estimations et le coefficient de corrélation linéaire ; la seconde vise à proposer un graphique représentant le nuage de points et la droite des moindres carrés. Avec l'expérience acquise à la lecture des chapitres précédents, cet utilisateur ne voit aucune difficulté particulière à produire ces résultats en ligne de commande. Cependant, afin d'éviter la saisie de plusieurs lignes de commandes à chaque fois qu'il désire voir les résultats de ces deux tâches, il aimerait développer deux fonctions bien plus faciles à utiliser dans une pratique courante du logiciel R. Pour ce faire, il aura le soutien d'un utilisateur plus avancé qui pourra le conseiller dans sa démarche dès lors qu'il rencontrera quelques difficultés.

Cette problématique permettra de motiver le lecteur à acquérir les notions présentées dans ce chapitre.

SECTION 6.2

Développer des fonctions

Commençons par présenter quelques éléments théoriques de base permettant de comprendre comment créer une fonction R.

6.2.1 Mise en route rapide : déclaration, création et appel de fonctions

La déclaration d'une fonction se fait selon la forme générale suivante :

```
function(<liste de paramètres>) <corps de fonction>
```

où

- `<liste de paramètres>` est une suite de paramètres (formels) nommés ;
- `<corps de fonction>` représente, comme son nom l'indique, le contenu du code qui sera exécuté à chaque appel de la fonction.

Voici un exemple de déclaration d'une fonction :

```
> function(nom) cat("Bonjour",nom,"!")
function (nom)
cat("Bonjour", nom, "!")
```

Une fonction est considérée par R comme un objet particulier. La création d'une fonction correspond donc à affecter l'objet « fonction R » dans une variable dont le nom correspondra à la fonction elle-même. Par exemple, pour créer la fonction `bonjour()`, on pourra procéder ainsi :

```
> bonjour <- function(nom) cat("Bonjour",nom,"!")
> bonjour
function (nom)
cat("Bonjour", nom, "!")
```

Pour que cette fonction soit exécutée, il faut que son utilisateur appelle la fonction suivie de la liste des paramètres effectifs contenue entre parenthèses. On rappelle qu'**un paramètre effectif est la valeur affectée à un paramètre formel**. On parlera parfois aussi de paramètre d'appel ou de paramètre d'entrée à la place de paramètre effectif.

```
> bonjour("Pierre")
Bonjour Pierre !
```

6.2.2 Concepts de base sur les fonctions

6.2.2.1 Corps de fonction

Le corps de fonction peut être soit une simple instruction R, soit une suite d'instructions R. Dans ce dernier cas, ces instructions doivent être mises entre les caractères `{` et `}` pour délimiter le début et la fin du corps de la fonction ainsi définie. Plusieurs instructions R peuvent être écrites sur une même ligne dès lors qu'elles sont séparées par le caractère `;`. Lorsque le corps de fonction contenant plusieurs instructions R est écrit sur une même ligne, il ne faut alors pas oublier de les placer entre les caractères `{` et `}`. Rappelons que sur une ligne tout le code écrit après le caractère `#` n'est pas interprété par R et donc considéré comme un commentaire.

```
> bonjour <- function(nom) {
+   # Mettre le nom en majuscules.
+   nom <- toupper(nom)
+   cat("Bonjour",nom,"!")
+ }
> bonjour("Pierre")
Bonjour PIERRE !
```

6.2.2.2 Liste de paramètres formels et effectifs

Nous décrivons ici comment fonctionnent conjointement la déclaration de la liste de paramètres formels lors de la définition de la fonction et la saisie de la liste de paramètres effectifs (on dit aussi d'entrée, ou d'appel) lors de l'appel de la fonction.

Déclaration de fonction

À la déclaration de la fonction, **tous les paramètres sont identifiés par des noms uniques**. Ils peuvent être associés à des valeurs par défaut. Une valeur par défaut est spécifiée en utilisant, comme pour la déclaration d'objet liste (`list`), le caractère `=` suivi de la valeur par défaut. Dès lors qu'à l'appel de la fonction aucun paramètre effectif n'est spécifié pour un paramètre, cette valeur par défaut sera affectée à ce paramètre. Cette fonctionnalité a déjà été largement utilisée dans les chapitres précédents, mais nous savons à présent

comment l'introduire dans le développement de nouvelles fonctions. En voici un exemple :

```
> bonjour <- function(nom="Pierre") cat("Bonjour",nom,"!")
> bonjour()
Bonjour Pierre !
```

Insistons tout de suite sur la différence existant entre l'appel simple du nom `bonjour` de la fonction et celui de la fonction suivie de la paire de parenthèses : `bonjour()`. La première forme correspond à l'affichage du contenu de la fonction, comme tout autre objet R, tandis que la seconde forme correspond à l'appel de la fonction (où ici aucun paramètre n'est spécifié). En résumé, toute exécution de fonction doit se faire par l'ajout de la paire de parenthèses avec possiblement la liste des paramètres effectifs.

Nommage et paramètre effectif

Un paramètre effectif en R peut être saisi en y adjoignant également le nom du paramètre formel. Cela n'a évidemment pas de réel intérêt quand la fonction ne dépend que d'un seul paramètre formel. Ajoutons donc à notre fonction `bonjour()` la possibilité de parler plusieurs langues, et voyons quelques appels de cette fonction.

```
> bonjour <- function(nom="Pierre",langue="fr") {
+   cat(switch(langue,fr="Bonjour",esp="Hola",ang="Hi"),nom,"!")
+ }
> bonjour()
Bonjour Pierre !
> bonjour(nom="Ben")
Bonjour Ben !
> bonjour(langue="ang")
Hi Pierre !
```

Lorsque combinée avec la possibilité de fixer des valeurs par défaut [2], nous voyons que cette fonctionnalité propose au développeur une manière de définir les fonctions avec une liste importante de paramètres formels correspondant à des options d'appels. Les utilisateurs pourront alors appeler cette fonction sans être contraints à saisir tous les paramètres effectifs. Ils pourront par exemple affecter une valeur au dernier paramètre formel sans avoir à saisir tous les autres paramètres effectifs de la fonction. Cela permet donc de rassembler en une unique fonction ce que l'on devrait faire en plusieurs fonctions. C'est une véritable spécificité [3] du langage R qui permet un mode de programmation totalement innovant. À titre d'exemple, allez consulter dans l'aide les fonctionnalités de la fonction `seq()` avec les différentes options `by`, `length.out` et `along.with`.

2. La fonction `missing()` est aussi fort utile dans ce type de programmation.
3. Il est à noter que beaucoup de langages de programmation ne disposent pas de cette fonctionnalité.

Nommage partiel et paramètre effectif

Une deuxième fonctionnalité offerte par R dans ce contexte est qu'il est possible, à l'appel de la fonction, de ne pas saisir le nom complet d'un paramètre formel. Considérons les appels suivants de la fonction `bonjour()` :

```
> bonjour(lang="ang")
Hi Pierre !
> bonjour(l="ang")
Hi Pierre !
> bonjour(l="a")
 Pierre !
```

La règle pour déterminer le paramètre formel correspondant au nom partiel fourni à l'appel est la suivante : parmi la liste ordonnée des noms des paramètres formels de la fonction, le paramètre formel sélectionné est le **premier** paramètre formel dont les premières lettres correspondent au nom partiel saisi par l'utilisateur.

Liste complémentaire de paramètres « ... »

Il est possible de fournir une liste complémentaire de paramètres au moyen de la syntaxe `....` Lors de l'appel de la fonction, tous les paramètres « nommés » ne figurant pas dans la liste des paramètres formels sont regroupés dans la structure `....` L'utilisateur pourra alors utiliser dans le corps de la fonction la syntaxe `...` comme s'il faisait un copier-coller de la liste complémentaire de ces paramètres nommés. Mais il vaut mieux illustrer cela à travers un exemple.

```
> test.3points <- function(a="toto",...) print(list(a=a,...))
> test.3points("titi",b="toto")
$a
[1] "titi"
$b
[1] "toto"
```

Plus généralement, on peut dire qu'une règle d'utilisation de la liste complémentaire de paramètres `...` dans un corps de fonction est qu'elle est placée en paramètre d'un, voire de plusieurs appels internes de fonctions.

Expert

Lorsque `...` est contenu dans la liste de paramètres et ne figure pas en dernière position, le « nommage partiel de paramètres » est inopérant pour tous les paramètres à la suite de `....` En effet, un nom partiel de paramètre formel sera alors considéré comme un paramètre formel de la liste complémentaire.

```
> test.3points <- function(aa="toto",...,bb="titi") {
+                    print(list(aa=aa,...,bb=bb))}
> test.3points(a="titi",b="toto")
$aa
[1] "titi"
$b
[1] "toto"
$bb
[1] "titi"
```

Notez que la valeur du paramètre formel `aa` a bien été modifiée, mais que `bb` a conservé sa valeur. Le paramètre formel `b` a ainsi été créé. Si l'on désire modifier la valeur du deuxième paramètre formel `bb`, nous sommes contraints de saisir son nom au complet.

```
> test.3points(a="titi",bb="toto")
$aa
[1] "titi"
$bb
[1] "toto"
```

Un utilisateur adepte de la fonctionnalité de nommage partiel aura été surpris par la sortie suivante lors de l'utilisation de la fonction `paste(..., sep = " ", collapse = NULL)` en s'autorisant la saisie partielle (ici `col`) du paramètre formel `collapse` :

```
> paste(c("toto","titi"),col=", ")
[1] "toto , " "titi , "
```

Puisque le nommage partiel est inopérant, `col` est donc considéré comme un second vecteur à coller et les options par défaut de la fonction `paste()` sont donc activées, à savoir
`sep=" "` et `collapse=NULL`. Pour obtenir le résultat recherché, il faut donc saisir le nom complet du paramètre formel `collapse`.

```
> paste(c("toto","titi"),collapse=", ")
[1] "toto, titi"
```

Astuce

Généralement, lors de l'appel d'une fonction, il faut spécifier la valeur de tout paramètre formel n'ayant pas de valeur par défaut. Dans le cas contraire, une erreur est générée. Il existe cependant deux exceptions. La première correspond au cas où le paramètre n'est pas utilisé dans le corps de la fonction, mais cela est bien entendu fort peu utile et correspond probablement à une erreur de programmation. La seconde est présente lorsque

le développeur a prévu de gérer ce cas dans le corps du programme à l'aide de la fonction `missing()`.

```
> bonjour <- function(nom) {
+   if(missing("nom")) nom <- "Pierre"
+   cat("Bonjour",nom,"!")
+ }
> bonjour()
Bonjour Pierre !
```

6.2.2.3 Objet retourné par une fonction

L'exemple de la fonction `bonjour()` que nous avons introduit précédemment ne retournait aucun objet. Cette fonction se contentait de produire un affichage à l'écran.

```
> res <- bonjour()
Bonjour Pierre !
> res
NULL
```

Lors des chapitres précédents, nous avons utilisé à maintes reprises des fonctions R dont les résultats étaient enregistrés dans des variables (par exemple `x <- c(1,5,3)` où le résultat de la fonction de base `c()` est affecté à la variable `x`). Comme nous nous intéressons maintenant au côté développement de fonctions, voyons comment créer une fonction qui retourne un objet (un résultat non éphémère). Une règle générale pour retourner un objet est d'utiliser la fonction `return()`. Cette instruction stoppe l'exécution du code du corps de la fonction et renvoie l'objet à l'intérieur des parenthèses. En voici un exemple :

```
> bonjour <- function(nom="Pierre") {
+   return(paste("Bonjour",nom,"!",collapse=" "))}
> bonjour()
[1] "Bonjour Pierre !"
> message <- bonjour()
> message
[1] "Bonjour Pierre !"
```

Le premier appel de la fonction retourne l'objet chaîne de caractères sans affectation dans une variable. Le résultat est donc affiché à l'écran comme si l'utilisateur avait saisi en ligne de commande l'objet retourné par la fonction. Le deuxième appel ne produit aucun affichage, car, compte tenu de l'affectation dans la variable `message`, le résultat de la fonction est redirigé dans la variable `message` comme le montre la dernière instruction ci-dessus.

Remarque

Il est possible de retourner un objet sans utiliser la fonction `return()`. La règle est alors que l'objet retourné est le dernier manipulé dans la dernière instruction du corps de la fonction (c'est-à-dire juste avant la sortie de la fonction). Dans l'exemple précédent, nous aurions donc pu éviter l'utilisation de la fonction `return()`

```
> bonjour <- function(nom="Pierre") {
+   paste("Bonjour",nom,"!",collapse=" ")}
> bonjour()
[1] "Bonjour Pierre !"
```

Cependant, nous n'encourageons pas cette pratique car elle ne fonctionne pas toujours, comme on peut le voir ci-dessous où l'on s'attendrait à ce que la fonction renvoie 10 :

```
> fonction.sans.renvoi <- function() {
+   for (i in 1:10) x <- i}
> fonction.sans.renvoi()
```

Pouvez-vous dire si la fonction suivante retourne un objet ? Si la réponse est positive, quel est le contenu de cet objet ?

```
> bonjour <- function(nom="Pierre") {
+    msg <- paste("Bonjour",nom,"!",collapse=" ")}
```

Qu'en pensez-vous, si on vous propose la sortie suivante ?

```
> bonjour()
```

Puisqu'il n'y a aucun affichage, il semble qu'aucun objet n'est retourné. Et pourtant, en êtes-vous certain après cela ?

```
> message <- bonjour()
> message
[1] "Bonjour Pierre !"
```

Donc oui, le dernier objet manipulé est bien la variable `msg`. L'affectation dans la variable `message` de la sortie précédente alloue bien à cette variable le contenu de la variable `msg` dans le corps de la fonction. Le R semble parfois un peu déroutant, mais il faut convenir que ce type d'utilisation n'a rien de rationnel et qu'il est fort à parier qu'aucun développeur n'y verra un quelconque intérêt.

Astuce

Si l'on désire, comme dans le dernier exemple, que la fonction ne produise aucun affichage lors de son appel tout en retournant un objet, il est plus direct d'utiliser la fonction `invisible()` dont le nom est suffisamment explicite.

```
> bonjour <- function(nom="Pierre")
+ invisible(paste("Bonjour",nom,"!",collapse=" "))
> bonjour()
> message <- bonjour()
> message
[1] "Bonjour Pierre !"
```

6.2.2.4 Portée des variables dans le corps de la fonction

La notion de portée d'une variable est très importante pour un langage offrant le développement de fonctions. Le principal intérêt est que les variables définies à l'intérieur du corps de fonction ont une portée locale lors de l'exécution de la fonction. Cela signifie qu'une variable à l'intérieur du corps de la fonction est physiquement différente d'une variable ayant le même nom, mais qui serait définie dans l'espace de travail de votre session R. De manière générale, une portée locale pour une variable signifie qu'elle n'existe qu'à l'intérieur du corps de la fonction. Elle est donc automatiquement effacée de l'espace mémoire de l'ordinateur après l'exécution de la fonction. Modifions notre fonction `bonjour()` en insérant des affichages pour contrôler quelques contenus de variables.

```
> message <- "Bonjour Peter !"
> message # Espace de travail initialisé.
[1] "Bonjour Peter !"
> bonjour <- function(nom="Pierre",message="Bonjour") {
+   print(message)
+   message <- paste(message,nom,"!",collapse=" ")
+   print(message)
+   invisible(message)
+ }
> bonjour()
[1] "Bonjour"
[1] "Bonjour Pierre !"
> message # Espace de travail non modifié!
[1] "Bonjour Peter !"
> message <- bonjour()
[1] "Bonjour"
[1] "Bonjour Pierre !"
> message # Espace de travail modifié!
[1] "Bonjour Pierre !"
```

```
> message <- bonjour(message="Bienvenue")
[1] "Bienvenue"
[1] "Bienvenue Pierre !"
> message # Espace de travail de nouveau modifié!
[1] "Bienvenue Pierre !"
```

Faisons à présent un petit commentaire sur les paramètres de la fonction. Malgré ce que l'on pourrait penser, les variables `nom` et `message` ne sont pas directement évaluées (initialisées aux valeurs d'appel ou aux valeurs par défaut) avant l'exécution du corps de la fonction. Leur initialisation n'est faite que lors de leur première utilisation dans le corps de la fonction. Rappelons que la fonction `missing()` permet de tester si un paramètre formel a été fourni lors de l'appel de la fonction. Cette fonctionnalité ne peut en effet être opérationnelle qu'à la condition de la non-évaluation de la liste des paramètres formels au début du corps de la fonction. Dans le même ordre d'idée, il est possible, en début du corps de fonction, de récupérer l'appel effectif (avec la liste des paramètres complétée) via la fonction `match.call()`.

```
> test.call <- function(aa="titi",...,bb="toto") {
+   print(match.call())}
> test.call(a="toto",b="titi")
test.call(aa = "toto", b = "titi")
```

Expert

Cette dernière création de fonction ne semble pas présenter un grand intérêt, mais, lorsque vous deviendrez un développeur confirmé en R, vous verrez peut-être comment exploiter le résultat de la fonction `match.call()`. Sans fournir d'explications, mais pour donner un avant-goût de ce qu'il est possible de faire en R, modifions la fonction précédente afin qu'elle retourne la liste des paramètres coupée en deux listes : celle (nommée `fonction`) des paramètres effectifs associés à des paramètres formels de la fonction et celle (nommée `divers`) des paramètres effectifs complémentaires. Notez au passage la difficulté de la tâche à prendre en compte le nommage partiel des paramètres.

```
> test.call <- function(aa="titi",...,bb="toto") {
+   args <- as.list(match.call())[-1]
+   dans <- names(args) %in% names(list(...))
+   list(fonction=args[!dans],divers=args[dans])
+ }
> test.call(a="toto",b="titi")
$fonction
$fonction$aa
[1] "toto"
$divers
```

```
$divers$b
[1] "titi"
```

Il suffit de quelques lignes de code pour obtenir le résultat, ce qui fait de R un langage ayant de réelles qualités d'introspection. Et cela n'est rien par rapport à tout ce que le R sait faire dans ce contexte. Attention, il n'est nullement question ici de vous inciter à rentrer tout de suite dans ce type de développement, mais plutôt de vous faire comprendre les possibilités offertes par ce langage.

6.2.3 Application à la problématique

Fort des quelques explications théoriques proposées ci-dessus, notre utilisateur débutant propose alors les codes des fonctions relatives à sa problématique de régression linéaire simple.

```
1  resume.reg1 <- function(y,x) {
2    aEst <- cov(x,y)/var(x)
3    bEst <- mean(y)-aEst*mean(x)
4    return(list(aEst=aEst, bEst=bEst,cor=cor(x,y)))
5  }
6
7  affiche.reg1 <- function(y,x) {
8    aEst <- cov(x,y)/var(x)
9    bEst <- mean(y)-aEst*mean(x)
10   plot(x,y)
11   abline(a=bEst,b=aEst)
12 }
```

Remarque

Notons au passage que dans les anciennes versions de R, il était autorisé d'écrire
`return(aEst=aEst, bEst=bEst,cor=cor(x,y))`
mais que cet usage est en voie d'être abandonné dans les futures versions.

Après avoir chargé ces fonctions soit par un copier-coller, soit à l'aide de la commande `source()`, l'utilisateur fait ses premiers tests sur un exemple sans réel intérêt.

```
> y <- rnorm(10);x <- 1:10
> resume.reg1(y,x)
$aEst
[1] -0.218551
```

```
$bEst
[1] 0.08112825
$cor
[1] -0.5380587

> affiche.reg1(y,x)
```

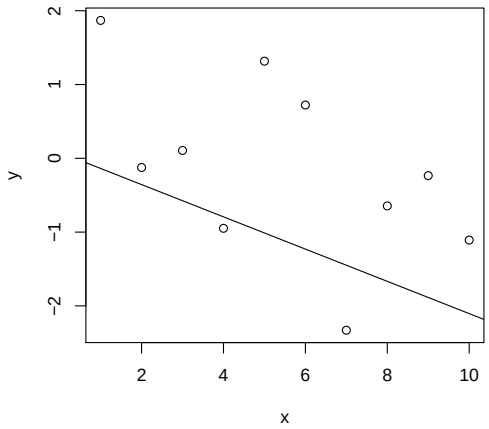


FIGURE 6.1 – Résultat de l'appel de la fonction `affiche.reg1()`.

Nous verrons plus loin comment enrichir ces deux fonctions.

6.2.4 Opérateurs

L'appel d'une fonction sous la forme `<fonction>(<liste de paramètres d'appel>)` n'est pas toujours aisé. Citons par exemple la fonction `seq()`. Laquelle des deux formes équivalentes suivantes préférez-vous ?

```
> seq(1,3)
[1] 1 2 3
> 1:3
[1] 1 2 3
```

Il est fort à parier que ce soit la dernière, car elle est plus synthétique (aucune parenthèse n'est à spécifier) et ainsi plus facile à manipuler notamment dans l'utilisation d'indices (pour les vecteurs, les matrices...). Cette seconde forme est celle correspondant à un opérateur. Le R en utilise en interne.

Il y a deux formes d'opérateurs :
– **opérateur unaire** (un seul paramètre) : `<opérateur> <paramètre1>`

– **opérateur binaire** (deux paramètres) : `<paramètre1> <opérateur> <paramètre2>`

où de manière explicite `<opérateur>` désigne l'opérateur, et `<paramètre1>` et `<paramètre2>` désignent les paramètres effectifs de l'opérateur. Voici une liste non exhaustive des opérateurs utilisés en interne par le R :

```
+, -, *, /, ^, %%, %/%, &, |, !, ==, !=, <, <=, >=, >.
```

Ces opérateurs sont *a priori* non modifiables par l'utilisateur [4]. Le R propose toutefois à l'utilisateur la possibilité de définir ses propres opérateurs. Ils sont de la forme `%<operateur>%` et certains sont déjà proposés dans le système de base, par exemple `%in%` et `%o%` (vus au chapitre 3).

Astuce

Pour afficher le code source de la fonction (l'opérateur) `%in%`, utilisez l'instruction : `get("%in%")`. Vous pouvez alors constater qu'il utilise la fonction `match()` dont il peut être intéressant de connaître l'existence.

Donnons-nous comme objectif de proposer une forme plus synthétique pour concaténer les chaînes de caractères, ce qui se fait normalement à l'aide de la fonction `paste()`.

```
> "%+%" <- function(ch1,ch2) paste(ch1,ch2,sep="")
> nom <- "Pierre"
> "La vie de " %+% nom %+% " est belle!"
[1] "La vie de Pierre est belle!"
> # On obtient donc une simplification de:
> paste("La vie de ", nom ," est belle!",sep="")
[1] "La vie de Pierre est belle!"
```

Notons que le nom non alphanumérique de la fonction nous contraint à le placer entre des guillements. Bien entendu, la préférence entre l'une ou l'autre de ces deux formes est complètement arbitraire. Aussi, il n'est pas question ici d'amoindrir l'intérêt de la fonction `paste()` qui est une fonction bien plus riche que le simple opérateur `%+%` que nous venons d'introduire (au moyen de la fonction `paste()` d'ailleurs). Le but est plutôt de montrer la flexibilité du R en nous permettant, à l'aide d'une simple définition de fonction, de simplifier la syntaxe d'appel.

4. En fait, ce groupe d'opérateurs est utilisable par un utilisateur lors du développement d'une nouvelle classe d'objets. Mais cela nous amènerait trop loin !

Astuce

Vous pouvez créer de nouveaux opérateurs pour définir des fonctions d'ensembles, comme celles présentées en page 105. Par exemple, la réunion entre deux ensembles A et B peut être définie ainsi :

```
> "%union%" <- function(A,B) union(A,B)
> A <- c(4,6,2,7)
> B <- c(2,1,7,3)
> A %union% B
[1] 4 6 2 7 1 3
```

6.2.5 Le R vu comme un langage fonctionnel

Le R est un langage fonctionnel dans le sens où presque toute exécution de code R se fait au moyen d'appels de fonctions, parsemés éventuellement de structures de contrôle. Signalons quelques comportements du R dont on ne soupçonne pas, à l'utilisation, qu'ils sont contrôlés par des fonctions. Nous avons vu que le simple appel d'un objet R se traduit par l'affichage de son contenu. En fait, lors d'une telle instruction, le R appelle (de manière transparente pour l'utilisateur) la fonction `print()` avec pour paramètre effectif le nom de l'objet. Cette fonction a un statut très particulier dans le R (car très couramment utilisée) et de plus amples précisions seront fournies ultérieurement. Toutes les opérations d'affectation (c'est-à-dire instruction contenant `<-`) ont leur exécution gérée par des fonctions dont le nom contient (sans surprise) le signe distinctif `<-` [5]. Le développement et la maintenance du système R peuvent se résumer à la construction d'une panoplie de fonctions. Il y a tout d'abord les fonctions de base qui sont proposées lors de l'installation de base du système R. Elles sont généralement non modifiables par l'utilisateur [6] et lorsque c'est tout de même le cas, il est fortement déconseillé de les modifier pour ne pas rendre votre système R inutilisable. La deuxième famille de fonctions est celle des fonctions directement développées en R [7] par n'importe quel utilisateur. Beaucoup de fonctions sont proposées par la communauté des développeurs R par le biais d'un système de *packages* (dont nous parlerons plus tard).

5. Pour vous en rendre compte, tapez en ligne de commande `apropos("<-")`
6. Le cœur du système R est développé dans le langage C pour des raisons évidentes de rapidité d'exécution, ce qui permet de le rendre plutôt réactif lorsqu'il est utilisé en ligne de commande.
7. Pour favoriser la rapidité d'exécution, il est généralement possible de convertir une fonction R en langage C puis de l'appeler depuis R via son API C.

SECTION 6.3

† Programmation orientée objets

Dans cette section, on ne se contente pas de considérer un objet comme une quantité que l'on peut sauvegarder et réutiliser plus tard. On entre de plus en plus dans l'esprit du langage R en précisant le mécanisme interne orienté objet qui régit la grande partie de son utilisation. Et pourtant, ce qui est assez incroyable, c'est la transparence pour l'utilisateur qui n'a pas à se soucier de connaître le fonctionnement interne de R. C'est à notre avis un point fort de R. Toutefois, à la lecture de cette section, un utilisateur pourra mieux comprendre rétroactivement comment le R fait pour proposer ces résultats. L'une des conséquences attendue sera une utilisation moins « hasardeuse » et ainsi mieux controlée du R.

6.3.1 Comment fonctionne le mécanisme orienté objet du R

6.3.1.1 Classe d'un objet et déclaration d'un objet

En R, ce qui importe est de spécifier la classe d'un objet à l'aide de la fonction `"class<-"()`. Rappelons que la fonction `class()` permet quant à elle de consulter la classe de l'objet.

```
> obj <- 1:10
> class(obj)
[1] "integer"
> class(obj) <- "TheClass"
> class(obj)
[1] "TheClass"
> class(obj) <- "LaClasse"
> obj
 [1]  1  2  3  4  5  6  7  8  9 10
attr(,"class")
[1] "LaClasse"
```

L'objet `obj` de classe `integer` est devenu un objet de classe `LaClasse`. Le dernier affichage de l'objet `obj` nous indique par ailleurs la classe de l'objet où `attr` est un résumé de attribut. Nous reviendrons à la fin de ce chapitre sur la notion d'attribut. Nous pouvons ici nous contenter de comprendre le sens de l'affichage `attr(,"class")` qui littéralement peut se traduire par « l'attribut de classe ».

Expert

Cela étant dit, ce n'est pas tout à fait vrai, car l'objet `obj` a conservé sa caractéristique d'être aussi de la classe `integer`, comme le montre la sortie suivante :

```
> obj*2
 [1]  2  4  6  8 10 12 14 16 18 20
attr(,"class")
[1] "LaClasse"
```

En effet, tous les éléments du vecteur `obj` ont été multipliés par 2. Nous espérons que dans les prochaines versions de R, la sortie de la fonction `class()` appliquée à un objet semblable pourra être du style `[1] "LaClasse" "integer"` permettant ainsi de mieux traduire la vraie nature de l'objet.

Pour savoir si un objet est d'une certaine classe, on peut le faire des deux manières suivantes :

```
> class(obj)=="LaClasse"
[1] TRUE
> inherits(obj,"LaClasse")
[1] TRUE
```

La fonction `inherits()` (en français, hérite) sera toutefois à privilégier, comme nous le verrons plus tard quand nous considérerons des objets dits polymorphes ayant plusieurs classes.

Astuce

Pour voir la classe de la fonction `function()`, vous pouvez utiliser l'instruction suivante :

```
> class(function() {})
[1] "function"
```

Pour la fonction `":"()`, il faut utiliser `class(get(":"))`.

6.3.1.2 Déclaration et utilisation d'une méthode d'un objet

Le mécanisme de programmation orientée objet de R est relativement simple et plutôt original comparativement à beaucoup d'autres langages de programmation. Pour comprendre ce mécanisme, illustrons-le sur l'exemple le plus utilisé en R, à savoir l'affichage d'un quelconque objet R au moyen de la fonction `print()`. Pour ce faire, analysons la série de sorties R suivantes :

```
> vect <- 1:10
> class(vect)
[1] "integer"
> vect
 [1]  1  2  3  4  5  6  7  8  9 10
> print(vect)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Jusqu'ici, rien d'apparemment surprenant, excepté peut-être qu'il faut souligner que la simple saisie d'un objet R en ligne de commande semble provoquer un appel de la fonction `print()` avec pour paramètre effectif l'objet considéré. Cette idée [8] est confirmée par un deuxième exemple illustrant l'affichage d'un objet de classe `formula` caractérisé par la présence du caractère tilde (~). L'exemple suivant enregistre dans la variable `form` la formule (de classe `formula`) exprimant la relation entre `y` et `x`. Notez qu'ici les objets `y` et `x` n'ont pas besoin d'exister puisqu'aucune évaluation n'est faite lors de la déclaration d'une formule [9].

```
> form <- y~x
> class(form)
[1] "formula"
> form
y ~ x
> print(form)
y ~ x
```

On peut remarquer que la fonction `print()` fonctionne différemment pour des objets de classes différentes. En effet, pour la variable `form` (de classe `"formula"`), `print()` a renvoyé `y~x` (c'est-à-dire l'instruction présente à droite de la flèche d'affectation. Pour la variable `vect`, l'appel de `print()` renvoie `[1] 1 2 3 4 5 6 7 8 9 10` alors que l'on aurait pu s'attendre à voir s'afficher `1:10`. Voici le code de la fonction `print()` :

```
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x29bc4c8>
<environment: namespace:base>
```

Le corps de cette fonction indique que la fonction `UseMethod()` doit être exécutée. Cette fonction est appelée en R *fonction générique*. Comme une tour de contrôle, elle sert à rediriger l'objet selon sa classe sur le bon appel de fonction. Pour le dernier exemple ci-dessus, cela a pour conséquence d'appeler la fonction d'affichage associée à la classe de l'objet `formula` de la forme `print.formula()`.

8. En fait, pour l'auto-affichage des objets de base (vecteurs, matrices, listes, etc.) dans la console, R n'utilise pas la fonction `print()`, mais appelle la fonction `PrintValueEnv`, codée en `C`, à laquelle l'utilisateur n'a pas un accès direct

9. Nous ne rentrons pas davantage dans les détails à ce stade, car nous reviendrons plus tard sur cette classe d'objets très originale.

Pour reprendre le jargon communément utilisé en programmation orientée objet, ce dernier type de fonction de forme générale `<methode>.<classe>` est appelé *méthode*. Cela nous éclaire sur la dénomination de la fonction `UseMethod()` dans le corps de la fonction générique `print()`.

Voilà donc ce qui se passe en coulisse en lieu et place d'un simple affichage de l'objet `form` :

```
> form # Provoque un appel de la fonction print(),
       # elle-même appelant la fonction print.formula().
y ~ x
> stats:::print.formula(form)
y ~ x
```

Expert

Pour vérifier comment il est facile de changer le comportement général du R par la simple modification d'une fonction, redéfinissons la fonction d'affichage pour la classe `formula`. On se contentera de reprendre l'affichage standard en y adjoignant la chaîne de caractères `"formule:"`.

```
> print.formula <- function(obj,...) {
+ cat(paste("formule:",paste(sapply(obj[c(2,1,3)],
+             as.character),collapse="")))
+ invisible(obj)
+ }
> y~x
formule: y~x
```

Si vous êtes débutant en R, le but n'est pas ici de comprendre précisément le code R ayant permis d'obtenir le résultat. En effet, malgré la simplicité apparente du code, il faut pour le maîtriser connaître des notions que l'on ne pourra aborder dans cet ouvrage. Encore une fois, le but est plutôt de révéler la puissance introspective du R puisqu'il permet notamment la manipulation de ses entités de base.

Pour rétablir le comportement initial du R quant à l'affichage des formules, vous aurez certainement compris qu'une simple élimination de la nouvelle fonction `print.formula()` suffit via l'instruction en ligne de commande `rm(print.formula)`. Nous ne le supprimons pas encore, car nous avons besoin de ce comportement par la suite.

Donc, si nous avons bien compris le fonctionnement de la fonction `print()`, nous pourrions supposer l'existence d'une fonction `print.integer()`. Tapons quelques instructions pour le vérifier :

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

```
> print.integer(vect)
Error in eval(substitute(expr), envir, enclos) :
  impossible de trouver la fonction "print.integer"
```

Nous constatons donc que la fonction `print.integer()` n'existe pas. En fait, lorsqu'il n'y a pas de méthode associée à une classe, le R exécute la méthode par défaut, de forme générale `<méthode>.default`, à savoir ici `print.default()`. Voyons donc ce que nous fournit la sortie de cette fonction pour nos deux exemples :

```
> print.default(vect)
 [1]  1  2  3  4  5  6  7  8  9 10
> print.default(form)
y ~ x
attr(,"class")
[1] "formula"
attr(,".Environment")
<environment: R_GlobalEnv>
> # À comparer avec:
> form
formule: y~x
```

Cette fois-ci, nous avons l'explication complète de ce qui se passe en coulisse. Nous constatons aussi que l'affichage d'une formule (c'est-à-dire un objet de classe `formula`) n'utilise pas la méthode par défaut comme le suggère la dernière sortie.

Astuce

Soulignons aussi que la fonction `print.default()` sert à afficher tous les objets (ou structures) de base du R quand ces objets sont passés comme paramètre effectif de la fonction `print()`.

En résumé, pour définir une nouvelle famille de méthodes, appelée ici `<méthode>` (nom de la famille de méthodes que vous voulez créer), applicables à tout type d'objet, il faut :

- tout d'abord déclarer la ***fonction générique*** sous la forme suivante :
  `<méthode> <- function(obj,...) UseMethod("<méthode>")`
- puis créer une ***méthode*** `<méthode>` pour une classe `<classe>` comme suit :
  `<méthode>.<classe> <- function(obj,<liste de paramètres>)`
  `<corps de la méthode>`
  où `<liste de paramètres>` et `<corps de la méthode>` sont respectivement une liste optionnelle de paramètres formels et le contenu de cette méthode, qui n'est autre qu'une fonction lorsqu'elle est appelée dans sa version longue.

Remarque

Notez que l'on peut dissocier, dans la déclaration de la famille de méthodes, le nom de la fonction générique et le paramètre de la fonction `UseMethod()` correspondant au nom de la méthode à appeler. Ainsi, on peut très facilement définir un alias, appelé `<alias>`, de la famille précédente de méthodes par la simple définition d'une nouvelle fonction générique :

```
<alias> <- function(obj,...) UseMethod("<méthode>")
```

Il en résulte que les deux appels en ligne de commandes `<méthode>(<objet>)` et `<alias>(<objet>)` pour un objet `<objet>` de classe `<classe>` sont équivalents à `<méthode>.<classe>(<objet>)`. À titre d'application plutôt surprenante, on peut assez facilement traduire les méthodes courantes du R en français comme cela est illustré ci-dessous :

```
> voir <- function(obj,...) UseMethod("print")
> voir(vect)
 [1]  1  2  3  4  5  6  7  8  9 10
> voir(form)
formule: y~x
> rm(print.formula) # Supprimons notre méthode pour
                    # revenir au mode normal.
> voir(form)
y ~ x
> form
y ~ x
```

6.3.2 Retour à la problématique

L'utilisateur se rend compte qu'il a répété l'exécution des estimations de a et b deux fois lors de la création des fonctions `affiche.reg1()` et `resume.reg1()` présentées à la section 6.2.3 (lignes 2 et 3, et lignes 8 et 9). Il demande alors conseil à un utilisateur plus avancé qui lui suggère d'utiliser le concept de programmation orientée objet. Après s'être documenté, il propose de créer une fonction [10] pour retourner l'objet de classe `reg1` afin de pouvoir le réutiliser par la suite comme premier paramètre d'appel de toute méthode pour ladite classe.

```
1 reglin <- function(y,x) {
2   aEst <- cov(x,y)/var(x)
3   bEst <- mean(y)-aEst*mean(x)
4   reg <- list(y=y,x=x,aEst=aEst,bEst=bEst)
```

10. Ce type de fonction est souvent appelé constructeur dans le jargon de la programmation orientée objet.

```
5   class(reg) <- "reg1"
6   return(reg)
7 }
```

Il définit maintenant la méthode `affiche.reg1()` qui pourra être utilisée sur tout objet de classe `reg1`.

```
1 affiche.reg1 <- function(reg) {
2   plot(reg$y,reg$x)
3   abline(a=reg$bEst,b=reg$aEst)
4 }
5
6 resume.reg1 <- function(reg) return(reg)
```

Il se lance alors dans quelques tests.

```
> reg <- reglin(y,x)
> resume(reg)
Error in eval(substitute(expr), envir, enclos) :
  impossible de trouver la fonction "resume"
> affiche(reg)
Error in eval(substitute(expr), envir, enclos) :
  impossible de trouver la fonction "affiche"
```

Surpris des erreurs d'affichage, il vérifie qu'il a bien défini sa fonction

```
> resume.reg1(reg)
$y
 [1]  1.8690093 -0.1242685  0.1070288 -0.9485351  1.3166447
 [6]  0.7226569 -2.3292535 -0.6452325 -0.2341175 -1.1081607
$x
 [1]  1  2  3  4  5  6  7  8  9 10
$aEst
[1] -0.218551
$bEst
[1] 0.08112825
attr(,"class")
[1] "reg1"
```

L'utilisateur avancé lui signale que son erreur provient du fait qu'il a oublié de déclarer les fonctions génériques `resume()` et `affiche()` qui ne sont pas standard contrairement à quelques autres comme `print()` et `summary()`.

```
1 resume <- function(x, ...) UseMethod("resume")
2 affiche <- function(x, ...) UseMethod("affiche")
```

Exécutons de nouveau les instructions précédentes.

```
> resume(reg)
$y
```

```
 [1]  1.8690093 -0.1242685  0.1070288 -0.9485351  1.3166447
 [6]  0.7226569 -2.3292535 -0.6452325 -0.2341175 -1.1081607
$x
 [1]  1  2  3  4  5  6  7  8  9 10
$aEst
[1] -0.218551
$bEst
[1] 0.08112825
attr(,"class")
[1] "reg1"
> affiche(reg)
```

Le dernier affichage qui produit le même graphique que celui de la figure 6.1 n'est pas fourni ici. Lorsque l'on sait que la méthode `print.reg1()` n'a pas été définie, on peut se demander ce que génère la simple instruction réduite au nom de l'objet.

```
> reg
$y
 [1]  1.8690093 -0.1242685  0.1070288 -0.9485351  1.3166447
 [6]  0.7226569 -2.3292535 -0.6452325 -0.2341175 -1.1081607
$x
 [1]  1  2  3  4  5  6  7  8  9 10
$aEst
[1] -0.218551
$bEst
[1] 0.08112825
attr(,"class")
[1] "reg1"
```

Nous savions déjà que c'est la méthode `print.default()` qui est ici sollicitée.

6.3.3 Information sur les méthodes

Pour obtenir des informations relatives aux méthodes, R offre la fonction `methods()` qui est très informative :

```
> methods("formula") # Ou plus directement methods(formula).
[1] formula.character*  formula.data.frame* formula.default*
[4] formula.formula*    formula.glm*        formula.lm*
[7] formula.nls*        formula.terms*
   Non-visible functions are asterisked
> methods(class="formula")
 [1] aggregate.formula*     alias.formula*
 [3] all.equal.formula      ansari.test.formula*
 [5] bartlett.test.formula* boxplot.formula*
 [7] cdplot.formula*        cor.test.formula*
 [9] deriv3.formula*        deriv.formula*
```

```
[11] fligner.test.formula*  [.formula*
[13] formula.formula*       friedman.test.formula*
[15] ftable.formula*        getInitial.formula*
[17] kruskal.test.formula*  lines.formula*
[19] mood.test.formula*     mosaicplot.formula*
[21] pairs.formula*         plot.formula*
[23] points.formula*        ppr.formula*
[25] prcomp.formula*        princomp.formula*
[27] print.formula*         quade.test.formula*
[29] selfStart.formula*     spineplot.formula*
[31] stripchart.formula*    sunflowerplot.formula*
[33] terms.formula          text.formula*
[35] t.test.formula*        update.formula
[37] var.test.formula*      wilcox.test.formula*
   Non-visible functions are asterisked
```

Attention

Veuillez ne pas confondre ces deux utilisations. La première instruction fournit toutes les méthodes (de la forme `<méthode>.<classe>`) associées à la fonction générique `formula`. La seconde nous propose toutes les méthodes de la classe `formula`.

Voici quelques exemples permettant de mieux comprendre la distinction entre les deux utilisations de la fonction `methods()`.

```
> class(y~x)
[1] "formula"
> update(y~x,.~.+z) # On applique la méthode update() à un
                    # objet de classe formula.
y ~ x + z
> update.formula
function (old, new, ...)
{
    tmp <- .Internal(update.formula(as.formula(old),
                                    as.formula(new)))
    out <- formula(terms.formula(tmp, simplify = TRUE))
    return(out)
}
<environment: namespace:stats>
> form <- "y~x"
> class(form)
[1] "character"
> formula(form)
y ~ x
> formula.character
Erreur : objet "formula.character" non trouvé
```

Astuce

Les fonctions suivies d'un astérisque sont exécutables, mais leur corps de fonction n'est pas visualisable. On peut toutefois utiliser la fonction `getAnywhere()`.

```
> getAnywhere(formula.character)
A single object matching 'formula.character' was found
It was found in the following places
  registered S3 method for formula from namespace stats
  namespace:stats
with value
function (x, env = parent.frame(), ...)
{
    ff <- formula(eval(parse(text = x)[[1L]]))
    environment(ff) <- env
    ff
}
<bytecode: 0x620a368>
<environment: namespace:stats>
```

6.3.4 Héritage de classe

Dans le cadre de notre problématique, l'utilisateur avancé informe notre utilisateur néophyte que le R dispose déjà d'un ensemble de fonctions pour le traitement des modèles linéaires. En effet, la fonction `lm()` est dédiée (comme on le verra au chapitre 12) à ce type de traitement. Il lui dit cependant qu'à sa connaissance, les traitements particuliers qu'il propose sur le modèle de régression linéaire simple n'existent pas. Ils s'associent alors pour développer cette extension avec pour principal objectif de ne pas « réinventer la roue » et d'exploiter le mieux possible les capacités existantes du système R.

Dans le concept de programmation orientée objet, la notion d'héritage de classe semble appropriée pour ce type d'extension. Le terme d'héritage exprime qu'un objet d'une certaine classe de base pourra aussi se comporter comme tous les autres objets de classes supplémentaires. Le système R dispose de ce mécanisme, et ce par un biais très simple consistant à associer à un objet une suite de classes. Ainsi, lorsqu'une méthode est appliquée pour un certain objet ayant une hiérarchie de classes, la première classe est d'abord sollicitée. Si la méthode pour cette classe existe, alors elle est exécutée. Dans le cas contraire, le R teste si, dans la hiérarchie des classes, il existe une méthode exécutable. Si tel est le cas, celle-ci est exécutée. Dans le cas contraire, la méthode par défaut est exécutée dès lors que celle-ci est définie. Finalement, si ce n'est toujours pas le cas, une erreur d'exécution est alors générée. Comme un exemple vaut souvent mieux qu'un long discours, illustrons cette notion sur la problématique de nos deux associés. Commençons par déclarer la fonction

constructeur de la nouvelle classe `lm1` héritant directement de la classe existante `lm`. Leur intention est toutefois d'orienter le développement dans la langue d'origine du R, à savoir l'anglais, dans un but de partager éventuellement leur travail avec la communauté des utilisateurs R.

```
1 lm1 <- function(...) {
2   obj <- lm(...)
3   if(ncol(model.frame(obj))>2) stop("plus d'une
4            variable indépendante")
5   class(obj) <- c("lm1",class(obj)) # Ou c("lm1","lm")
6   obj
7 }
```

Appliquons cela aux mêmes variables que précédemment.

```
> reg <- lm1(y~x)
> reg
Call:
lm(formula = ..1)
Coefficients:
(Intercept)            x
     1.0646      -0.2186
```

Déjà, nous pouvons constater la notion d'héritage. En effet, l'objet s'affiche différemment de la sortie de la fonction `print.default()` alors qu'aucune méthode `print.lm1` n'est définie. Cela s'explique par le fait que le système R dispose déjà de la méthode `print.lm()` (définie dans le package système `stats`) et que l'objet `reg` hérite des méthodes de la classe `lm`. Pour vérifier que cet objet hérite bien de cette classe, il y a plusieurs possibilités dont la plus simple est de visualiser le contenu de l'attribut `class` via la fonction `class()`. Un développeur préfèrera peut-être la fonction `inherits()`, plus directe dans son utilisation.

```
> class(reg)
[1] "lm1" "lm"
> inherits(reg,"lm")
[1] TRUE
> stats:::print.lm(reg)
Call:
lm(formula = ..1)
Coefficients:
(Intercept)            x
     1.0646      -0.2186
```

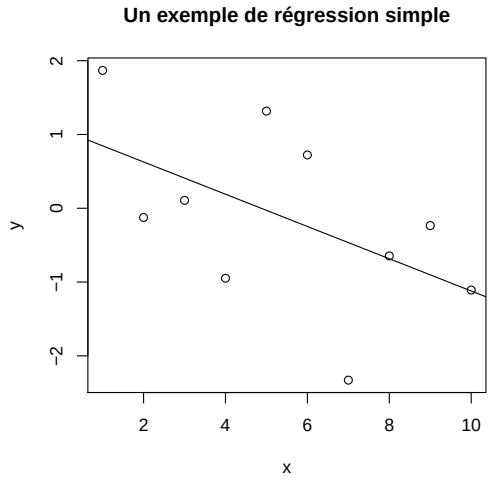
La ligne 4 (que nous ne commenterons pas ici) dans la fonction `lm1()` teste si la formule est bien celle d'un modèle de régression simple. Voyons ce qu'il en est sur un exemple.

```
> lm1(y~x+log(x))
Error in lm1(y ~ x + log(x)) : #(*plus d'une*)
          #(*variable indépendante*)
```

Poursuivons le développement des fonctions dans le même esprit que

```
1 plot.lm1 <- function(obj,...) {
2   plot(formula(obj),...)
3   abline(obj)
4 }
```

```
> summary(reg)
Call:
lm(formula = ..1)
Residuals:
     Min       1Q   Median       3Q      Max
-1.86400 -0.63931  0.02565  0.89407  1.34479
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.0646     0.7511   1.417    0.194
x            -0.2186     0.1210  -1.805    0.109
Residual standard error: 1.099 on 8 degrees of freedom
Multiple R-squared:  0.2895,        Adjusted R-squared:  0.2007
F-statistic:  3.26 on 1 and 8 DF,  p-value: 0.1086
> plot(reg,main="Un exemple de régression simple")
```



Dans l'appel ci-dessus de `summary()`, puisque la méthode `summary.lm1()` n'a pas été développée, c'est la méthode standard `summary.lm()` qui est exécutée. En effet, l'objet `reg` de classe `lm1` hérite ensuite de la classe `lm` pour

bénéficier de toutes les méthodes standards proposées par le système R pour traiter des modèles linéaires. Pour le second appel, à savoir la méthode `plot()`, la méthode finalement invoquée est `plot.lm1` tout fraîchement créée.

Remarque

Notons cependant que le R dispose en standard d'une méthode `plot.lm()` qui permet de proposer un ensemble de graphiques pour une analyse plus approfondie des résultats (voir le chapitre 12). Ayant volontairement changé le comportement par défaut du R pour le cas particulier de la régression linéaire simple, il sera tout de même possible de bénéficier de cette méthode en la désignant explicitement (`plot.lm(reg)`).

Expert

Le mécanisme de programmation orientée objet est extrêmement simple dans sa conception. Il existe bon nombre de langages de programmation orientée objet.Une différence importante est que la grande majorité propose une encapsulation des champs de l'objet et des méthodes dont l'un des intérêts est de pouvoir modifier les champs de l'objet lui-même dans une méthode. Cela n'est pas directement possible en R compte tenu de la stricte portée locale des variables à l'intérieur du code d'une fonction R. L'utilisateur peut tout de même adopter ce type de programmation s'il le désire. Toute méthode `<méthode>.<classe>()` dont l'objectif est de modifier des champs de l'objet `<objet>` (de classe `<classe>`) devra retourner l'objet lui-même. L'utilisateur de la fonction générique `<méthode>()` pourra alors affecter son résultat à l'objet initial comme suit :
`<object> <- <méthode>(<objet>)`. Le risque est tout de même un ralentissement d'autant plus important que le contenu des champs de l'objet est volumineux. La raison est qu'une duplication complète de l'objet est effectuée. Nous espérons qu'un jour les développeurs de R proposeront en standard une fonctionnalité plus élégante (et analogue à ce que propose la majorité des langages de programmation orientée objet) consistant en la modification à l'intérieur du corps de la méthode uniquement des champs considérés (souvent peu nombreux). Lorsque vous deviendrez (et nous le souhaitons) un utilisateur averti, vous remarquerez que la notion de pointeur (pourtant très courante en programmation) n'est pas directement proposée au développeur R (voir cependant la fonction `tracemem()` ainsi que 7.8.2.2, en page 324).

SECTION 6.4

† Aller plus loin en programmation R

Avant de programmer dans un langage, il est bon de connaître l'esprit dans lequel il a été conçu. Dans cette section, nous mettrons plus en avant des structures du langage R qu'il n'est pas nécessaire de connaître dans une première utilisation du R, mais qui se révèlent fort appréciables quand on décide d'approfondir son niveau d'utilisation du langage R. Ce sont des éléments qui font du R un langage original et puissant. Nous conseillons à un utilisateur débutant d'éventuellement survoler cette section sans chercher à en maîtriser les concepts qui y sont abordés. En effet, toutes les informations récoltées ici sont de deuxième niveau, dans le sens où elles peuvent être évitées tout en permettant une utilisation tout à fait sérieuse du R.

6.4.1 Attributs R

Un objet R contient une *information principale* véhiculée au moyen des structures de base du R présentées dans ce livre. Il y a un deuxième niveau d'information que l'on qualifiera d'*information annexe* ou d'*information secondaire*. Ce type d'information est attaché à un objet au moyen d'attributs et accessible via la fonction `attributes()`.

```
> mat <- matrix(1:10,nrow=2)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> class(mat)
[1] "matrix"
> attributes(mat)
$dim
[1] 2 5
```

Nous commenterons plus tard cette sortie. Insistons de nouveau sur le fait que ce mécanisme se veut transparent pour l'utilisateur qui sera généralement plus intéressé par le contenu de l'objet R. Dans une utilisation de base, il est par ailleurs déconseillé de directement modifier les attributs. Ce point de vue est justifié par le fait que de nombreuses fonctions R sont proposées pour les manipuler indirectement. En revanche, un développeur plus curieux de comprendre comment le R fonctionne pourra découvrir quelques caractéristiques complémentaires qui généralement éclairent sur le comportement de l'objet lui-même. Nous avons déjà, sans le faire de manière directe, manipulé l'attribut `class` au moyen des fonctions `class()` et `"class<-"()`. Par la suite, nous manipulerons aussi les trois autres principaux attributs `dim`, `names` et `dimnames` fort utilisés dans la gestion interne du R. Illustrons l'utilisation des attributs sur un exemple sans réel intérêt autre que celui de présenter comment manipuler les attributs.

La fonction complémentaire `attr()` permet de manipuler un seul attribut à la fois alors que la fonction `attributes()` renvoie l'ensemble des attributs sous forme d'une liste R.

```
> vect <- 1:10
> attr(vect,"test") # Renvoie NULL, car vect n'a pas d'attribut
                    # test.
NULL
> attributes(vect)  # NULL car vect n'a pas d'attribut.
NULL
> # Affectation d'un attribut "attrib1" contenant la chaîne de
  # caractères "TEST1".
> attr(vect,"attrib1") <- "TEST1"
> attr(vect,"attrib1")
[1] "TEST1"
> # Affectation d'un attribut "attrib2" contenant le vecteur
  # c(1,3)
> attributes(vect)$attrib2 <- c(1,3)
> attributes(vect)
$attrib1
[1] "TEST1"
$attrib2
[1] 1 3
> attr(vect,"attrib2")
[1] 1 3
> # Modification de l'attribut "attrib1" et suppression de
  # l'attribut "attrib2"
> attributes(vect)$attrib1 <- 3:1
> attr(vect,"attrib2") <- NULL
> attributes(vect)
$attrib1
[1] 3 2 1
> # Suppression simultanée de tous les attributs
> attributes(vect) <- NULL
> attributes(vect)
NULL
```

Le mécanisme d'accès aux attributs est donc simple d'utilisation. L'exemple précédent nous a montré comment changer les attributs à l'aide des fonctions `"attr<-"()` et `"attributes<-"()`. La valeur d'un attribut est n'importe quel objet R. Enfin, l'affectation d'un attribut à `NULL` le supprime.

6.4.1.1 Attribut `class`

Dans la section précédente sur la programmation orientée objet, nous avons déjà manipulé l'attribut `class` au moyen des fonctions `class()` et `"class<-"()`. Cela illustre bien le fait qu'il n'est pas nécessaire de savoir les manipuler directement. Reprenons l'exemple que nous avions choisi pour montrer quand il y a

équivalence entre la manipulation de cet attribut et les deux fonctions d'utilité `class()` et `"class<-"()`.

```
> form <- y~x
> attributes(form)
$class
[1] "formula"
$.Environment
<environment: R_GlobalEnv>
> class(form)
[1] "formula"
> obj <- 1:10
> attr(obj,"class") # Pas d'attribut class.
NULL
> class(obj)        # Et pourtant!
[1] "integer"
> attr(obj,"class") <- "LaClasse" # Équivalent à class(obj) <-
                                  # "LaClasse".
> class(obj)
[1] "LaClasse"
```

Il ne reste plus rien à dire sur cet attribut même s'il joue un rôle central dans le mécanisme de programmation orientée objet du R.

6.4.1.2 Attribut `dim`

L'attribut `dim` joue un rôle primordial dans le comportement des objets matrices (`matrix`) et tableaux (`array`). Prenons tout d'abord l'exemple d'une matrice :

```
> mat <- matrix(1:12,nrow=2)
> mat
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> attr(mat,"dim")
[1] 2 6
> attributes(mat)
$dim
[1] 2 6
> attr(mat,"dim") <- c(3,4)  # Changement de forme : 3 lignes et
                             # 4 colonnes.
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> attributes(mat)$dim <- c(2,6) # Retour à la forme initiale.
> mat
```

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

Dans l'exemple ci-dessus, en changeant l'attribut `dim`, nous avons pu modifier la forme de la matrice. Puisque, comme nous l'avons déjà précisé, la manipulation des attributs se veut la plus transparente possible pour l'utilisateur, il est fort à parier qu'il existe pour l'utilisateur des fonctions analogues avec des noms plus explicites. C'est bien entendu le cas pour notre exemple grâce aux fonctions `dim()` et `"dim<-"()` :

```
> dim(mat)
[1] 2 6
> dim(mat) <- c(1,12) # Changement de forme : 1 ligne et 12
                      # colonnes.
> mat
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
[1,]    1    2    3    4    5    6    7    8    9    10    11
     [,12]
[1,]    12
> dim(mat) <- c(2,6)  # Retour à la forme initiale.
```

Pour vraiment comprendre comment R représente les objets comme les matrices (`matrix`) et les tableaux (`array`), analysons les sorties suivantes :

```
> mat
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> class(mat)
[1] "matrix"
> dim(mat) <- NULL # Ou attributes(mat)$dim<-NULL ou
                   # attributes(mat) <- NULL.
> mat
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> is.vector(mat)
[1] TRUE
> class(mat)
[1] "integer"
> dim(mat) <- c(2,2,3)
> mat
, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

```
, , 3
     [,1] [,2]
[1,]    9   11
[2,]   10   12
> is.vector(mat)
[1] FALSE
> class(mat)
[1] "array"
```

En supprimant l'attribut `dim`, nous constatons que l'objet `mat` est devenu un simple vecteur. En lui affectant un vecteur à trois entiers, nous constatons que l'objet `mat` est maintenant un tableau de dimension 3. Nous comprenons alors que la différence de comportement entre les vecteurs, les matrices et les tableaux réside dans la valeur de l'attribut `dim`.

Attention

Malgré une même sortie d'affichage, un vecteur et un tableau à un indice sont traités différemment en R comme le montrent ces quelques lignes de commandes :

```
> dim(mat) <- 12
> mat
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> is.vector(mat)
[1] FALSE
> class(mat)
[1] "array"
> identical(mat,1:12)
[1] FALSE
> dim(mat) <- NULL
> mat
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> is.vector(mat)
[1] TRUE
> class(mat)
[1] "integer"
> identical(mat,1:12)
[1] TRUE
```

Nous pourrions croire que nous avons tout dit sur l'utilisation de l'attribut `dim` et pourtant il reste une dernière chose assez intéressante à noter. La seule différence existant entre un vecteur et une liste (`list`) est que les éléments d'un vecteur doivent être du même type. Les matrices et tableaux contiennent généralement des éléments de même nature. Cette contrainte est notamment très importante pour les opérations matricielles. Mais en tant que structure de stockage, il est possible d'imaginer que les concepts de matrices et tableaux

soient étendus aux listes (`list`) en leur affectant un attribut `dim` jouant le même rôle que précédemment pour les vecteurs. En lisant la documentation relative aux instructions `matrix()` et `array()`, nous pouvons constater que c'est en effet le cas puiqu'il est possible de fournir en premier paramètre d'appel de ces fonctions une liste (`list`) à la place d'un vecteur (comme précédemment). Appliquons-le alors sur une matrice bien que cela soit aussi réalisable pour un tableau dès lors que le nombre d'éléments de la liste est en accord avec sa dimension.

```
> lmat <- matrix(list(7,1:2,1:3,1:4,1:5,1:6),nrow=2)
> lmat      # Renvoie la structure et non le contenu trop
            # difficile à afficher.
     [,1]      [,2]      [,3]
[1,] 7         Integer,3 Integer,5
[2,] Integer,2 Integer,4 Integer,6
> dim(lmat)
[1] 2 3
> is.list(lmat)
[1] TRUE
> lmat[1,2] # Extraction de l'élément en ligne 1 et colonne 2.
[[1]]
[1] 1 2 3
> lmat[,-2] # Extraction de la sous-matrice sans la deuxième
            # colonne.
     [,1]      [,2]
[1,] 7         Integer,5
[2,] Integer,2 Integer,6
> dim(lmat) <- NULL
> lmat      # Ce n'est plus qu'une liste.
[[1]]
[1] 7
[[2]]
[1] 1 2
[[3]]
[1] 1 2 3
[[4]]
[1] 1 2 3 4
[[5]]
[1] 1 2 3 4 5
[[6]]
[1] 1 2 3 4 5 6
> is.list(lmat)
[1] TRUE
```

6.4.1.3 Attributs `names` et `dimnames`

L'attribut `names` joue un rôle essentiel dans le nommage des éléments d'une liste.

```
> li <- list(1:3,letters[1:3])
> li
[[1]]
[1] 1 2 3
[[2]]
[1] "a" "b" "c"
> attributes(li)
NULL
> attributes(li)$names <- c("chiffres","lettres")
> li
$chiffres
[1] 1 2 3
$lettres
[1] "a" "b" "c"
```

La première et la quatrième instructions sont donc équivalentes à la déclaration suivante plus usuelle :

```
> li <- list(chiffres=1:3,lettres=letters[1:3]))
```

Cela est moins connu, car moins utile, mais cet attribut est aussi utilisable pour tout type de vecteur.

```
> vect <- 1:3
> attr(vect,"names") <- letters[1:3]
> vect
a b c
1 2 3
> # Ou plus directement:
> vect2 <- c(a=1,b=2,c=3)
> vect2
a b c
1 2 3
```

Il n'est pas nécessaire de manipuler directement l'attribut `names`. En effet, son accès et son changement de valeur peuvent se faire de manière plus explicite comme suit :

```
> names(li)
[1] "chiffres" "lettres"
> names(li) <- c("chif","lett")
> li
$chif
[1] 1 2 3
$lett
[1] "a" "b" "c"
> names(vect)
[1] "a" "b" "c"
> names(vect) <- toupper(names(vect))
> vect
```

```
A B C
1 2 3
```

Pour des objets à plusieurs indices, tels les matrices (`matrix`) et plus généralement les tableaux (`array`), la gestion des noms d'indice se fait en interne en modifiant l'attribut `dimnames`. Nous illustrons rapidement cela sur un exemple.

```
> mat <- matrix(1:6,nr=2)
> mat
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> attributes(mat) # Modifiable comme un attribut.
$dim
[1] 2 3
> rownames(mat)   # Noms de lignes.
NULL
> colnames(mat)   # Noms de colonnes.
NULL
> dimnames(mat)   # Noms des lignes et colonnes sous forme de
                  # liste.
NULL
> colnames(mat) <- paste("V",1:3,sep="")
> rownames(mat) <- c("a","b")
> mat
  V1 V2 V3
a  1  3  5
b  2  4  6
```

Pour un objet tableau à plus de deux indices, les fonctions `rownames` et `colnames` n'ont plus de sens et il faut soit directement modifier l'attribut `dimnames`, soit utiliser la fonction `"dimnames<-"()`.

Remarque

Les matrices de données (`data.frame`) ont un statut un peu à part. Elles sont définies comme des listes et plus souvent manipulées comme des matrices. Les attributs gérant les affichages des lignes et des colonnes sont `row.names` et `names` (et non `col.names`) :

```
> df <- data.frame(a=1,b=1:2)
> df
  a b
1 1 1
2 1 2
> attributes(df)
$names
[1] "a" "b"
$row.names
```

```
[1] 1 2
$class
[1] "data.frame"
> names(df)      # Comme liste.
[1] "a" "b"
> dimnames(df)   # Comme tableau : une liste de deux vecteurs.
[[1]]
[1] "1" "2"
[[2]]
[1] "a" "b"
> rownames(df)   # Comme matrice : accès aux noms des lignes.
[1] "1" "2"
> colnames(df)   # Comme matrice : accès aux noms des colonnes.
[1] "a" "b"
```

Les quatre dernières lignes fournissent les appels pour accéder à ces attributs sans avoir à les manipuler directement. Les formes correspondantes existent pour en modifier le contenu. Au passage, notons que l'attribut `class` précise la classe de l'objet.

6.4.2 Autres objets R

On peut dire que l'une des spécificités du langage R est que la très grande majorité des quantités manipulées en R sont allouables dans des variables et ainsi réutilisables plus tard. Les quelques rares exceptions sont pour la plupart les structures de contrôle. Les objets R sont de différents types, appelés classes. Nous avons déjà vu les classes d'objets servant à stocker les structures de données les plus courantes. Par la suite, nous choisirons de décrire trois autres types d'objets. De manière assez surprenante pour un utilisateur non averti, nous verrons qu'une formule et un environnement sont aussi considérés par le R comme des objets à part entière. Nous introduirons aussi la notion d'expression R qui est un objet dans lequel on peut stocker un morceau de code R à exécuter en différé.

6.4.2.1 Expression R

Dans les premières utilisations du système R, on a passé sous silence des structures R qui permettent de décrire les bases syntaxiques du langage R. Respectant sa philosophie de pouvoir enregistrer le plus de composantes possibles, le R est capable de manipuler une expression R et de la découper en une suite d'entités atomiques (par exemple, `call`, `name` ...). Nous ne ferons ici qu'évoquer ses capacités sans rentrer dans les détails. Nous nous limitons aux traitements des expressions R qui ont un réel intérêt en tant que développeur R. Il est difficile de définir rigoureusement ce qu'est une expression R. Nous proposons la définition suivante fondée sur l'utilisation du R en ligne de commande. Une

expression R peut donc être vue comme du code R saisi consécutivement en lignes de commandes jusqu'à ce qu'il soit exécuté par l'interpréteur R (c'est-à-dire nouvelle apparition du caractère > nous invitant à saisir une nouvelle commande). Cette expression peut bien entendu être multiligne. La fonction `expression()` permet de déclarer une expression R lorsqu'elle est utilisée avec un seul paramètre d'appel. On peut cependant fournir une séquence de plusieurs expressions consécutives, chaque expression correspondant à un paramètre effectif dans l'appel de la fonction. Un objet expression n'est pas évalué par l'interpréteur R, mais peut être enregistré pour être évalué plus tard autant de fois qu'on le désire. L'évaluation d'une expression R se fait au moyen de la fonction `eval()`. Tous ces points sont illustrés ci-dessous.

```
> expression(v<-"valeur")              # Expression v<-"valeur"
                                       # non évaluée.
expression(v <- "valeur")
> v
Error in eval(substitute(expr), envir, enclos) : objet 'v' introuvable
> expression(v<-"valeur") -> expr      # Enregistrée dans l'objet
                                       # expr.
> expr
expression(v <- "valeur")
> eval(expr)                           # Évaluation de expr.
> v                                    # Voilà le résultat
                                       # attendu.
[1] "valeur"
> expression(v<-"valeur2",v) -> expr   # Équivalent à 2 lignes de
                                       # commandes non évaluées.
> expr
expression(v <- "valeur2", v)
> eval(expr)                    # La deuxième instruction affiche
                                # le contenu de v.
[1] "valeur2"
```

En tant que développeur, on peut trouver intéressant de convertir une chaîne de caractères décrivant du code R en une expression R à évaluer plus tard. La fonction `parse()` est utilisée pour cela :

```
> parse(text='v<-"valeur"') -> expr
> expr
expression(v<-"valeur")
> eval(expr)
> v
[1] "valeur"
```

Le paramètre formel `text` est utilisé ici pour lire une chaîne de caractères, mais la première utilisation de la fonction est la lecture d'un fichier contenant du code R dont on peut préciser le nom en premier paramètre effectif.

Astuce

Voici un exemple d'utilisation des fonctions `eval()` et `parse()` :

```
> for (i in 1:3) eval(parse(text=paste("a",i," <- i",sep="")))
> a2
[1] 2
```

À présent, nous allons manipuler la fonction `expression()` pour décrire quelques fonctionnements internes du R. Nous allons ainsi mieux comprendre pourquoi il se dit que le langage R est un langage fonctionnel (c'est-à-dire fondé sur l'utilisation intensive des fonctions). Il est étonnant de voir jusqu'à quel point ce principe est vrai. Le premier point abordé souligne que l'affectation est considérée lors de l'exécution comme un opérateur, c'est-à-dire une fonction avec deux paramètres. Le premier correspond à la variable à affecter, le second représente son contenu.

```
> toto <- "TOTO"
> toto
[1] "TOTO"
> "<-"(toto,"TOTO2")                # Équivalent à : toto <-
                                    # "TOTO2"
> toto
[1] "TOTO2"
> expression("<-"(toto,"TOTO2"))    # comme le prouve la sortie de
                                    # cette expression.
expression(toto <- "TOTO2")
```

Poursuivons notre visite avec l'utilisation des parenthèses qui permet notamment d'ordonner les priorités d'exécution dans une expression R. Le R les traite encore comme une fonction.

```
> 30*(10+20)
[1] 900
> 30*"("(10+20)  # En coulisses, voilà ce qui est exécuté.
[1] 900
> expression(30*(10+20))
expression(30 * (10 + 20))
> expression(30*"("(10+20))
expression(30 * (10 + 20))
```

Il en va de même pour la notion de bloc d'expressions qui se définit comme une suite d'expressions R qui sont regroupées entre les deux délimiteurs accolades ouverte `"{"` et fermée `"}"`.

```
> {
+ print("ligne1")
+ print("ligne2")
```

```
+ }
[1] "ligne1"
[1] "ligne2"
> "{"(print("ligne1"),print("ligne2"))
[1] "ligne1"
[1] "ligne2"
> expression({
+   print (      "ligne1"    ) # Commentaire non interprété.
+
+   # Même chose pour ce nouveau commentaire.
+   print("ligne2")
+ })
expression({
    print("ligne1")
    print("ligne2")
})
> expression( "{"(print("ligne1"),print("ligne2")) )
expression({
    print("ligne1")
    print("ligne2")
})
```

Soulignons comment les commentaires et espaces de tout genre sont ignorés par l'interpréteur R.

Par la suite, nous aurons à décrire la syntaxe de quelques instructions R. Notons qu'à l'intérieur d'un bloc, on peut, par souci de présentation, ajouter autant de retours à la ligne que l'on veut sans affecter son exécution.

6.4.2.2 Formule R

L'objet formule (`formula`) est une véritable originalité de R. Son usage est principalement d'établir une relation entre deux parties séparées par un tilde ~. Les deux parties doivent être des expressions R. À la lumière de ce que nous avons vu sur la fonction `expression()`, nous pouvons observer comment le R convertit, lors de son exécution, l'expression d'une formule R en une fonction `"~"()`.

```
> y~x
y ~ x
> "~"(y,x)              # Expression équivalente,
y ~ x
> expression("~"(y,x))  # comme le prouve cette expression.
expression(y ~ x)
```

Pour un développeur, c'est un objet qui permet de proposer une interface utilisateur plutôt facile à utiliser compte tenu de sa forme plus proche du langage littéraire. Par exemple, la formule R `y~x` peut exprimer que les variables `y` et `x` sont liées entre elles ou, plus causalement, que `y` se détermine en fonction de `x`. Dans un contexte général, il est de la responsabilité du développeur

d'interpréter la formule pour réaliser les tâches souhaitées. Cela reste toutefois d'un niveau très avancé, et nous renvoyons le lecteur à la documentation de R. Proposons quelques exemples de formules sans signification particulière, mais qui serviront à nous familiariser avec ce nouvel objet :

```
> y~x
y ~ x
> y~(x+y:z)*t|v
y ~ (x + y:z) * t | v
> y1+y2|w ~ (x+y:z)*t|v
y1 + y2 | w ~ (x + y:z) * t | v
```

Il est bon de souligner que même si les quantités exprimées dans les formules ci-dessous ne sont pas des objets R existants aucune erreur n'est retournée. En revanche, n'oubliez pas qu'une erreur de syntaxe engendre un message d'erreur :

```
> y~x+y)*t|v
Erreur : ')' inattendu(e) dans "y~x+y)"
```

Concentrons-nous maintenant sur son utilisation dans le système R. Comme la formule n'est pas un objet courant, l'utilisateur ne devine pas forcément qu'elle s'enregistre comme tout autre objet R.

```
> form <- y~x
> form
y ~ x
```

Les deux utilisations principales sont dans les contextes des graphiques et des statistiques. L'usage dans les graphiques est une alternative à celle que l'on a déjà vue dans le chapitre 5.

```
> x <- runif(10)
> y <- runif(10)
> plot(x,y)
> plot(y~x)
```

Le graphique produit n'est pas fourni ici puisque le seul intérêt est de montrer que les deux instructions avec ou sans formule sont équivalentes. Notons l'interversion des variables `x` et `y` dans ces deux utilisations, la version avec formule `plot(y~x)` exprimant plus littéralement l'action effectuée : représentation graphique (c'est-à-dire `plot()` en anglais) de `y` *en fonction de* `x`. Cette version élégante (c'est du moins notre avis) est bien entendu proposée pour les fonctions complémentaires `points()` et `lines()`.

Dans un contexte de nature statistique, une fonction relative à un traitement spécifique à un certain modèle statistique prend comme paramètre d'entrée (souvent le premier, ce qui souligne son grand intérêt) une formule établissant la relation entre les variables du modèle. L'exemple le plus simple est certainement le modèle de régression linéaire (*linear model* en anglais) dont voici un exemple [11] :

11. Il n'est nullement question de traiter ici plus en détail comment R gère le modèle de régression linéaire puisque cela sera abordé au chapitre 12.

```
> lm(y~x)   # x et y doivent être définis (c'est le cas ici!).
Call:
lm(formula = y ~ x)
Coefficients:
(Intercept)            x
     0.8095      -0.5574
> lm(form)  # Rappel: form <- y~x
Call:
lm(formula = form)
Coefficients:
(Intercept)            x
     0.8095      -0.5574
```

Outre le côté agréable de la syntaxe, l'objet formule fournit une interface avec l'utilisateur très performante pour décrire le modèle. Cela est confirmé par le fait qu'à la différence de l'utilisation pour les graphiques, il n'y a aucune alternative pour décrire la relation entre les variables dans le modèle. En effet, nous pourrions penser que `lm(y,x)` aurait pu être envisageable. Mais comment dans ce cas écrire de manière équivalente sous forme d'une liste de paramètres d'entrée la formule `y~(x+z)*t` dont on verra (voir le chapitre 13) qu'elle a une interprétation tout à fait valide ?

Concernant les opérations sur les formules, introduisons à présent un exemple d'utilisation de la fonction `update()` qui permet de modifier une formule en fonction d'une autre.

```
> update(y~x,.~.+z)      # Modifier y~x en y~x+z.
y ~ x + z
> form <- y~x            # La même chose en enregistrant les
                         # modèles.
> form2 <- update(form,.~.+z)
> form2
y ~ x + z
> update(form2,.~.-x) # On peut aussi supprimer une variable.
y ~ z
```

Au vu de ces exemples, analysons la syntaxe de la fonction `update()`. Le premier paramètre formel est la formule à modifier alors que le deuxième exprime, par une formule ayant une syntaxe spécifique, les opérations à appliquer pour l'obtention de la nouvelle formule. Il nous reste donc à interpréter la syntaxe de la deuxième formule. Tout point « · » avant le caractère tilde « ~ » est remplacé par l'expression gauche (avant le tilde) de la formule initiale. De la même manière, tout point « · » après le caractère tilde « ~ » est remplacé par l'expression droite (après le tilde) de la formule initiale.

6.4.2.3 Environnement R

Dans la conception de tout langage de programmation, il est nécessaire de proposer la notion d'environnement qui peut être vu comme un espace de stockage d'objets R. Lorque vous ouvrez votre session R, un premier environnement

`.GlobalEnv` est créé par le R. Il est appelé espace de travail (*workspace* en anglais) et tous les objets manipulés dans cette session en ligne de commande y sont stockés. Même si le but ici est uniquement de survoler ce concept, précisons que le concept de fonction dépend intrinsèquement de la notion d'environnement. Sans rentrer dans les détails, donnons-en juste un petit aperçu. Lorsque dans le corps d'une fonction, vous créez un nouvel objet, le R a pris le soin de déclarer en interne un environnement propre à cette fonction pour y stocker le contenu de l'objet. La raison est que, si celui-ci a le même nom qu'un objet de l'environnement `.GlobalEnv`, ce dernier ne verra pas sa valeur écrasée par celle de l'objet défini dans le corps de la fonction. Pour mieux comprendre à quoi correspond un environnement, précisons qu'un objet défini dans l'environnement `.GlobalEnv` a sa valeur accessible dans le corps de la fonction. Pour autant, comme nous l'avons précisé précédemment, sa valeur ne pourra être modifiée par une affectation avec le même nom d'objet. L'accès à un objet défini dans un environnement différent de celui associé à la fonction s'explique par le fait que la déclaration d'un environnement est faite en précisant un parent qui est lui-même un environnement. Un environnement peut cependant ne pas avoir de parent comme c'est le cas pour l'environnement de départ `.GlobalEnv`. Ainsi, lorsqu'un objet n'est pas directement disponible dans l'environnement d'une fonction, la recherche de l'objet est alors faite dans l'environnement parent. Si celui-ci n'est toujours pas disponible, il y a deux cas possibles : soit il existe un environnement parent et la recherche de l'objet est poursuivie dans celui-ci, soit il n'y en a pas et un message d'erreur est retourné, précisant que l'objet est introuvable. Ce processus d'exploration est alors appliqué récursivement jusqu'à l'obtention éventuelle de l'objet. La plupart des déclarations des environnements se font de manière invisible en interne par le système R. Nous reviendrons plus tard sur cette notion quand nous évoquerons plus en détail le développement de fonctions. Un point absolument surprenant est qu'un environnement est lui-même considéré comme un objet R. On peut alors déclarer un environnement afin d'exécuter une partie de code spécifique sans que cela n'affecte l'espace de travail `.GlobalEnv`. La fonction `local()` prenant pour premier paramètre d'appel le code à exécuter et pour deuxième paramètre d'appel l'environnement où l'exécution doit se dérouler est particulièrement utile pour cela :

```
> a <- 12; b <- 13
> espace <- new.env() # Par défaut, le parent est celui où
                      # new.env est appelé.
> local({
+ a <- b+2
+ a
+ },espace)
[1] 15
> a # Valeur de a non modifiée dans .GlobalEnv.
[1] 12
> espace$a # Valeur de a dans l'environnement espace.
```

```
[1] 15
```

La fonction porte bien son nom puisque la valeur de `a` dans l'environnement de travail `.GlobalEnv` a bien été préservée. Comme cela est précisé en commentaire, le parent de `espace` généré via `new.env()` est ici `.GlobalEnv`, mais on aurait pu le préciser en fournissant une valeur pour le paramètre formel `parent`. Prenons deux exemples de déclaration de parent.

```
> espace2 <- new.env(parent=emptyenv())
> local(a<-b+2,espace2) # Erreur !!!
Error in eval(expr, envir, enclos) :
  impossible de trouver la fonction "<-"
> espace2$a   # Sans surprise, l'objet a n'existe pas!
NULL
```

L'environnement `espace2` est très peu utile, car son environnement parent est un environnement vide (c'est-à-dire pas de parent ; déclaré à l'aide de la fonction `emptyenv()`). L'erreur d'exécution dans le code local est due au fait que même l'affectation `<-` est une fonction qui est inaccessible, car l'environnement vide ne connaît vraiment rien du R et en particulier pas les fonctions de base de R. La fonction `globalenv()` retourne l'environnement global `.GlobalEnv` qui est toujours le premier dans la liste d'accès des environnements de R.

```
> espace3 <- new.env(parent=parent.env(globalenv()))
> local(a<-b+2,espace3) # Erreur, car b de .GlobalEnv est
                        # inaccessible!
Error in eval(expr, envir, enclos) : objet 'b' introuvable
> local(a<-15,espace3)
> a
[1] 12
> espace3$a
[1] 15
```

L'environnement est finalement relativement agréable à utiliser puisqu'il s'utilise dans le même style qu'une liste.

```
> espace3$b <- b-1
> b
[1] 13
> espace3$b
[1] 12
```

Pour plus de détails sur le sujet, nous renvoyons le lecteur vers l'aide en ligne assez complète qui s'adresse tout de même à des utilisateurs confirmés.

SECTION 6.5

† Interfacer R et C/C++ ou Fortran

Vous vous demandez peut-être pourquoi vous devriez considérer l'éventualité de coder une partie de vos programmes en langage `C/C++` ou `Fortran` ? Il

peut y avoir plusieurs raisons à cela, comme par exemple :
- utiliser depuis R une routine déjà existante, anciennement programmée en C/C++ ou en Fortran ;
- améliorer la rapidité d'exécution de l'un de vos programmes codés en R ;
- utiliser les capacités graphiques de R ou encore certaines fonctions de R, sur les résultats numériques renvoyés par un code C/C++ ou Fortran.

Astuce

La dernière version de R intègre un *byte compiler* qui accélère sensiblement la vitesse de certains calculs. Vous pouvez aussi utiliser la version de R fournie par la compagnie Revolution Analytics (http://www.revolutionanalytics.com). Elle a été optimisée pour accroître la vitesse d'exécution de certains calculs, par exemple en s'appuyant sur une architecture multi-cœurs lorsqu'elle est disponible.

Attention

Interfacer R et C/C++ ou Fortran est beaucoup plus aisé sous Linux (ou MacOS) que sous un environnement Microsoft Windows car ce dernier ne contient pas par défaut plusieurs outils nécessaires. Notez que les auteurs de ce livre utilisent Linux la plupart du temps !

Renvoi

Nous supposons que le lecteur possède déjà quelques bases dans la programmation en C/C++ et/ou Fortran. Si ce n'est pas le cas, il pourrait consulter avec profit les livres [29] et [44] pour C et C++, et [12] pour Fortran.

Dans cette section, nous ne prétendons nullement être exhaustifs. Nous allons présenter uniquement quelques exemples simples permettant d'illustrer les points précédents. Ce faisant, nous vous fournirons quelques bases qui vous permettront, nous l'espérons, de vous débrouiller seul par la suite.

Astuce

Veuillez noter que nous nous sommes limités à la présentation des interfaces incluses dans le système R mais que nous invitons fortement les lecteurs ayant des projets où la vitesse d'exécution est d'un grand intérêt à regarder le *package* Rcpp car il facilite considérablement le développement de tels outils et leur intégration dans le système R. Il y a de plus un nombre croissant de *packages* qui reposent sur celui-ci. Certains des auteurs de ce livre sont par exemple en train de faire migrer des projets reposant sur

l'API C du système R vers ce nouvel environnement reposant sur le *package* Rcpp. Le choix de ne pas l'introduire dans ce livre est surtout motivé par le fait qu'il existe déjà un livre complet dédié à ce *package* [21] et que la documentation proposée par les auteurs est abondante. Les notions abordées dans cette section sont aussi une bonne mise à l'étrier pour découvrir le développement via le *package* Rcpp.

Attention

Avant de débuter, il vous faut installer des compilateurs C/C++ et Fortran, puisque le système Microsoft Windows n'en est pas pourvu par défaut. Le gratuiciel Rtools, contenant plusieurs outils issus du monde Linux, a été conçu pour cela. Il est disponible à l'adresse http://cran.r-project.org/bin/windows/Rtools. Choisissez Full installation to build 32 or 64 bit R 2.14.2+ si vous avez un processeur 64 bits. Cochez la case appropriée lors de l'installation de Rtools, afin que la variable PATH soit correctement configurée pour ce logiciel. Il faut aussi modifier la variable d'environnement système Path afin qu'elle contienne le chemin d'accès vers le dossier du logiciel R (une façon de connaître ce chemin consiste à effectuer un clic droit sur l'icône de R présent sur le bureau, afin d'en afficher les propriétés). Ceci donnera la possibilité d'appeler le logiciel R depuis une fenêtre de commandes MS-DOS, comme cela sera présenté un peu plus loin. Pour cela, faites un clic droit sur le bureau de Windows, sélectionnez Nouveau/Raccourci, puis entrez l'instruction suivante dans la fenêtre qui s'affiche : control.exe sysdm.cpl,System,3
Après avoir créé ce raccourci sur le bureau, double-cliquez dessus, et dans la fenêtre qui s'ouvre, cliquez sur Variables d'environnement.... Modifiez alors la valeur de la variable système Path pour y ajouter **au début** (en utilisant le ; comme séparateur) le chemin d'accès vers le dossier contenant l'exécutable R (qui devrait ressembler à C:\Program Files\R\R-3.1.0\bin\i386 ou C:\Program Files\R\R-3.1.0\bin\x64) et ceux vers les dossiers de Rtools (qui devraient ressembler à C:\Rtools\bin et C:\Rtools\gcc-4.6.3\bin), si ce n'est pas déjà fait.

6.5.1 Création et exécution d'une fonction C/C++ ou Fortran

Voyons maintenant, sur un premier exemple, comment il est possible d'accélérer l'exécution d'un programme en utilisant le langage C/C++ ou Fortran. La fonction R combn() permet de générer toutes les combinaisons d'un certain nombre d'éléments piochés dans un vecteur donné. Ainsi, l'instruction

ci-dessous génère toutes les combinaisons de taille 3 à partir des éléments du vecteur `1:5`.

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

Lorsque l'on cherche à obtenir les `choose(n,m)` combinaisons (soit 1 313 400 par exemple si $n = 200$ et $m = 3$) à partir d'un vecteur de plus grande taille n, le temps de calcul peut devenir plus important.

```
> system.time(x <- combn(200,3))
   user  system elapsed
  2.013   0.040   2.054
```

On constate, en utilisant la fonction `system.time()`, que le calcul ci-dessus prend plusieurs secondes sur l'ordinateur utilisé lors de l'écriture de ce livre (prenez une valeur supérieure à 200 au besoin, si votre ordinateur est plus rapide).

Astuce

La fonction `permn()` du *package* `combinat` peut être utilisée pour générer toutes les permutations des élements d'un vecteur.

Une version simplifiée de la fonction originale `combn()` de R est donnée ci-dessous :

```
> combnR <- function(n,m) {
+  a <- 1:m ; e <- 0 ; h <- m
+  combmat <- matrix(0,nrow=m,ncol=choose(n,m))
+  combmat[,1] <- 1:m
+  i <- 2
+  nmmp1 <- n - m + 1
+  mp1 <- m + 1
+  while (a[1] != nmmp1) {
+   if (e<n-h) {
+    h <- 1 ; e <- a[m] ; a[m-h+1] <- e + 1
+    combmat[,i] <- a
+    i <- i + 1
+  } else {
+   h <- h + 1 ; e <- a[mp1-h]
+   a[(m-h+1):m] <- e + 1:h
+   combmat[,i] <- a
+   i <- i + 1
+  }}
```

```
+ return(combmat)
+ }
```

Nous proposons ci-après deux codes C/C++ et deux codes Fortran permettant de réaliser la même opération, mais dans un délai beaucoup plus court.

• Création de la fonction C/C++

Code C++ de la fonction combnC, disponible dans le fichier http://biostatisticien.eu/springeR/combn.cpp :

```
#include <math.h>
extern "C" {
void combnC(int *combmat, int *n, int *m) {
 int i, j, e, h, nmmp1, mp1;
 int *a;
 a=new int[m[0]];
 for (i=1;i<=m[0];i=i+1) a[i-1]=i;
 e=0;
 h=m[0];
 for (i=1;i<=*(m+0);i=i+1) combmat[i-1]=i;
 i=2;
 nmmp1=n[0] - m[0] + 1;
 mp1=m[0] + 1;
 while(a[0] != nmmp1) {
  if(e < n[0] - h) {
  h=1;
  e=a[m[0]-1];
  a[m[0] - h]=e + 1;
  for (j=1;j<=m[0];j++) combmat[(i-1)*m[0]+j-1]=a[j-1];
  i=i+1;
  }  else {
  h=h + 1;
  e=a[mp1 - h-1];
  for (j=1;j<=h;j=j+1) a[m[0] - h + j-1]=e + j;
  for (j=1;j<=m[0];j++) combmat[(i-1)*m[0]+j-1]=a[j-1];
  i=i + 1; }          }
 delete[] a;
}}
```

Code de la fonction principale, disponible dans le fichier http://biostatisticien.eu/springeR/main.cpp :

```
#include <iostream>
using namespace std;
```

```
 3 extern "C" {
 4 int main() {
 5   void combnC(int *combmat, int *n, int *m);
 6   int *n, *m, *combmat, j;
 7   double Cnm;
 8   n=new int[1];
 9   m=new int[1];
10   n[0]=5;
11   m[0]=3;
12   Cnm=10;
13   combmat=new int[(int)Cnm*m[0]];
14   combnC(combmat,n,m);
15   for (j=1;j<=Cnm*m[0];j++) cout<< combmat[j-1] << "␣";
16 }}
```

Le symbole ␣ ci-dessus indique que l'on doit entrer un espace. Par ailleurs, on remarque que les indices commencent à zéro en C/C++, contrairement à R où ils commencent à 1.

• Création de la fonction Fortran

Code Fortran de la sous-routine combnF, disponible dans le fichier http://biostatisticien.eu/springeR/combn.f90 :

```
 1 SUBROUTINE combnF(combmat,n,m)
 2
 3 integer, intent(in) :: n,m
 4 integer             :: i,j,e,h,nmmp1,mp1
 5 integer,dimension(m) :: a
 6 integer,dimension(*), intent(out)::combmat
 7
 8 do  i=1,m
 9 a(i)=i
10 end do
11 e=0
12 h=m
13 do  i=1,m
14 combmat(i)=i
15 end do
16 i=2
17 nmmp1=n-m+1
18 mp1=m+1
19 do while (a(1) .ne. nmmp1)
20 if (e < n-h) then
21 h=1
```

```
22 e=a(m)
23 a(m-h+1)=e+1
24 do  j=1,m
25 combmat((i-1)*m+j)=a(j)
26 end do
27 i=i+1
28 else
29 h=h+1
30 e=a(mp1-h)
31 do 40 j=1,h
32 a(m-h+j)=e+j
33 40 continue
34 do j=1,m
35 combmat((i-1)*m+j)=a(j)
36 end do
37 i=i+1
38 endif
39 enddo
40 END SUBROUTINE combnF
```

Code de la fonction principale, disponible dans le fichier http://biostatisticien.eu/springeR/main.f90 :

```
1 PROGRAM main
2 integer :: n,m,Cnm,j,k
3 integer, allocatable,dimension(:) :: combmat
4 n=5
5 m=3
6 Cnm=10
7 k=Cnm*m
8 allocate(combmat(k))
9 CALL combnF(combmat,n,m)
10 write(*,*) (combmat(j) ,j=1,k)
11 deallocate(combmat)
12 end PROGRAM main
```

• Compilation et exécution de la fonction C/C++ ou Fortran

Afin de pouvoir utiliser le code C++ ou Fortran donné plus haut, il vous faut le compiler, c'est-à-dire le transformer en un fichier dit exécutable. Pour cela, il suffit d'ouvrir une fenêtre de commandes MS-DOS, par le menu Démarrer/Exécuter de Windows (ou en utilisant la combinaison de touches [WINDOWS+R]) et en tapant l'instruction cmd suivie de la touche ENTRÉE. Dans cette fenêtre noire, nous tapons les deux instructions qui suivent l'encadré.

Attention

Vous pourriez avoir besoin de vous déplacer, au moyen de la commande `MS-DOS cd` (pour *change directory*), vers le répertoire où vous avez enregistré vos fichiers. Par exemple, si vous avez créé vos fichiers sur le Bureau de Windows, utilisez :

```
cd Bureau
```

ou peut-être

```
cd Desktop
```

Notez que, sous `MS-DOS`, la commande `dir` vous permet d'afficher les fichiers et dossiers contenus dans le répertoire courant.

```
:: Pour compiler le code C/C++:
g++ -o moncombn.exe combn.cpp main.cpp
:: Pour compiler le code Fortran:
gfortran -o moncombn.exe combn.f90 main.f90
:: Exécution de la fonction:
moncombn.exe
```

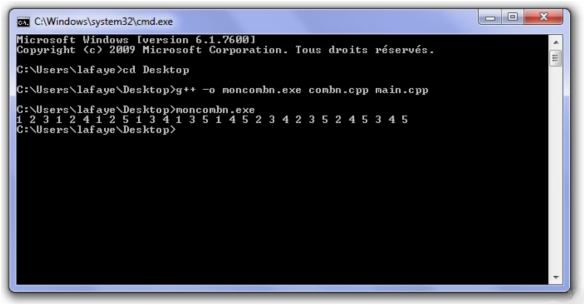
La première instruction a pour effet de compiler notre code `C++` ou `Fortran` afin de produire l'exécutable moncombn.exe. L'autre lance cet exécutable et affiche, sans mise en forme toutefois, le résultat des calculs.

Astuce

La fonction `system()` du logiciel R permet d'exécuter une commande `DOS`. Par exemple, depuis R, on peut taper :

```
> system("moncombn.exe")
1 2 3 1 2 4 1 2 5 1 3 4 1 3 5 1 4 5 2 3 4 2 3 5 2 4 5 3 4 5 >
```

Notez que vous devrez bien sûr préalablement changer le répertoire courant de R, au moyen de la fonction `setwd()` par exemple, afin de vous placer dans le dossier contenant le fichier moncombn.exe.



Astuce

L'option de compilation (*flag* en anglais) `-Wall` permet d'afficher tous les avertissements ou erreurs (*all warnings*) lors de la compilation (lorsqu'il y en a bien entendu !) :

```
g++ -o moncombn.exe combn.cpp main.cpp -Wall
```

Nous allons maintenant produire les $\binom{200}{3} = 1\ 313\ 400$ sous-vecteurs constitués des différentes combinaisons de trois éléments piochés dans le vecteur `1:200`. Pour la version `C/C++`, modifiez les lignes 10, 12 et 15 du code de la fonction `main` présenté en page 255. Ces lignes de code deviennent :

```
n[0]=200;
Cnm=1313400;
// for (j=1;j<=Cnm*m[0];j++) cout << combmat[j-1] << "␣";
```

Pour la version `Fortran`, modifiez les lignes 4, 6 et 10 du code de la fonction `main` présenté en page 257. Ces lignes de code deviennent :

```
n=200
Cnm=1313400
!write(*,*) (combmat(j) ,j=1,k)
```

Nous avons commenté la dernière ligne ci-dessus (au moyen de `//` pour `C/C++` et de `!` pour `Fortran`) pour que l'appel du programme `moncombn.exe` n'affiche plus le résultat (devenu très volumineux) du calcul, ce qui prendrait beaucoup de temps. Mais le calcul est bien effectué. Nous sommes ainsi cohérents avec le calcul précédent effectué sous R, pour lequel le résultat n'était pas affiché mais stocké dans la variable `x`. Après avoir sauvegardé vos modifications, recompilez, puis lancez votre code :

```
:: Pour compiler le code C/C++:
g++ -o moncombn.exe combn.cpp main.cpp
:: Pour compiler le code Fortran:
gfortran -o moncombn.exe combn.f90 main.f90
```

```
:: Exécution de la fonction:
moncombn.exe
```

Vous pouvez constater que ce calcul (sans affichage du résultat) est effectué extrêmement rapidement.

6.5.2 Appel du code C/C++ (ou Fortran) depuis R

Nous allons maintenant voir comment il est possible d'appeler le code C++ du fichier combn.cpp (ou plutôt une version compilée de celui-ci) directement depuis R, sans utiliser de fonction main. Pour cela, on crée une enveloppe (*wrapper* en anglais) R contenant un appel vers la fonction C++.

Remarque

R ne peut appeler que des fonctions C/C++ ou Fortran ne retournant pas de valeur. Les fonctions C/C++ doivent donc être de type void et les routines Fortran, des subroutines. Les résultats devront être mis dans des arguments de la fonction d'appel.

Récupérez le fichier http://biostatisticien.eu/springeR/combn.R dont le code est présenté ci-dessous :

```
1  combnRC <- function(n,m) {
2   combmat <- matrix(0,nrow=m,ncol=choose(n,m))
3   lib <- paste("combn",.Platform$dynlib.ext,sep="")
4   dyn.load(lib)
5   out <- .C("combnC",res=as.integer(combmat),
6               as.integer(n),as.integer(m))
7   combmat <- matrix(out$res,nrow=m,byrow=F)
8   dyn.unload(lib)
9  return(combmat)
10 }
```

Remarque

Pour appeler le code Fortran, remplacez la ligne 5 du code ci-dessus par

```
out <- .Fortran("combnF",res=as.integer(combmat),
```

Les fonctions dyn.load() et dyn.unload() permettent respectivement de charger et de décharger de la mémoire de R les ressources présentes dans un fichier DLL (pour *Dynamic Link Library*, ou en français bibliothèque de liens

dynamiques). Une DLL contient des fonctions qui pourront être appelées pendant l'exécution d'un programme, sans que celles-ci soient incluses dans son exécutable. Ici, il s'agit du fichier combn.dll (qui ne contient qu'une seule fonction) qui sera créé un peu plus loin.

Les fonctions `.C()` et `.Fortran()` (qui renvoient une liste) permettent respectivement de passer des valeurs depuis R vers une fonction `C/C++` ou `Fortran` puis de récupérer le résultat de ces dernières. Il faut utiliser les instructions `as.integer()`, `as.double()` ou `as.character()` pour déclarer respectivement dans R des objets constitués de valeurs entières, de valeurs décimales (numériques) ou de chaînes de caractères, afin qu'elles soient « reçues » correctement par l'un des arguments de la fonction `C/C++` ou `Fortran`.

Pour une fonction `C/C++`, tous les arguments devront être des pointeurs ; par exemple des pointeurs d'entiers (`int *`), de réels (`double *`) ou encore pointeurs de pointeurs de caractères (`char **`). Le tableau 6.1 donne les équivalents entre les types dans R, `C/C++` et `Fortran`.

Table 6.1 – Conventions sur les types des arguments. Tapez `?.Fortran` pour plus de détails.

R	C/C++	Fortran
integer	int *	INTEGER
numeric	double *	DOUBLE PRECISION
numeric	float *	REAL
complex	Rcomplex *	DOUBLE COMPLEX
logical	int *	integer
character	char **	CHARACTER*255
list	SEXP *	non autorisé
autre type	SEXP	non autorisé

Attention

Contrairement à R, où il est très facile d'obtenir la longueur d'un vecteur `x` au moyen de l'instruction `length(x)`, en `C/C++` il n'est pas possible de connaître la longueur de `x`. Il peut alors parfois s'avérer nécessaire de fournir à la fonction `.C()` à la fois le vecteur `x` et sa longueur, par exemple de la façon suivante pour une hypothétique fonction nommée `fonctionC` :

```
x <- c(1.2,0.7,3,2,4,1,0.9)
.C("fonctionC",as.double(x),as.integer(length(x)))
```

Les arguments de la fonction C/C++ `fonctionC` sont : `double *x` et `int *n`. La même remarque s'applique pour une fonction en `Fortran`.

Remarque

La fonction C/C++ `combnC` ne renvoie rien (*void* en anglais) directement. Par contre, la valeur de ses arguments, qui sont des pointeurs, peut être modifiée pendant l'exécution de celle-ci. Et il est ensuite possible d'accéder directement (par leur adresse en fait) aux valeurs de ces pointeurs. C'est de cette façon (mais de manière transparente pour l'utilisateur) que R fonctionne au moyen de la fonction `.C()`.
Par ailleurs, vous aurez noté en ligne 5 du code de la fonction `combnRC()` ci-dessus, l'utilisation de `res=` dans l'appel de la fonction `.C()`. Ceci permet ensuite d'utiliser `out$res` directement, à la place de `out[[1]]`. On peut utiliser un autre nom que `res`, et ceci sur n'importe lequel des arguments de la fonction `.C()`. Nous aurions ainsi pu utiliser `val=as.integer(m)`, ce que nous n'avons pas fait car cette valeur n'a pas été modifiée par `combnC`, et est donc déjà connue (comme étant `m`). Le même genre de remarque s'applique pour une fonction en `Fortran`.

Nous allons maintenant créer le fichier combn.dll, qui sera celui appelé par R. Pour cela, tapez les instructions suivantes dans la fenêtre `MS-DOS` :

```
:: En C/C++:
g++ -c combn.cpp -o combn.o
g++ -shared -o combn.dll combn.o
:: En Fortran:
gfortran -c combn.f90 -o combn.o
g++ -shared -o combn.dll combn.o
```

Astuce

Une façon équivalente (ou presque car des arguments d'optimisation pourraient être utilisés par le compilateur, ce qui peut d'ailleurs gêner une éventuelle activité de débogage) pour créer cette librairie dynamique est (après avoir éventuellement effacé les fichiers combn.o et combn.dll) d'utiliser l'unique instruction : `R CMD SHLIB combn.cpp -o combn.dll`

La première instruction crée le fichier objet `combn.o`, qui contient le code machine de la fonction contenue dans le fichier `combn.cpp`. La deuxième instruction crée la bibliothèque (librairie) dynamique `combn.dll`. À cette étape, le compilateur nous informe des éventuelles erreurs à corriger dans notre programme (avec le numéro de ligne correspondant).

Astuce

Notez qu'il est possible d'inclure plusieurs fichiers objets dans une même librairie, qui contiendra ainsi plusieurs fonctions. Par exemple, si nous disposions du fichier choose.o contenant le code machine d'une fonction calculant les coefficients binomiaux, il serait possible d'incorporer deux fonctions dans une DLL ainsi :

```
g++ -shared -o combn.dll combn.o choose.o
```

Linux

Sous Linux, les « fichiers DLL » ont plutôt une extension `.so` (pour *shared object*). Par conséquent, il vous faudra remplacer toutes les occurences de l'extension `.dll` par l'extension `.so`.

Mac

Sous MacOS, les « fichiers DLL » ont plutôt une extension `.dylib` (pour *dynamic library*). Par conséquent, il vous faudra remplacer toutes les occurences de l'extension `.dll` par l'extension `.dylib`. Notez aussi que sous MacOS, il faudra remplacer `g++ -shared` par `g++ -dynamic`.

Dans R, on peut maintenant (après s'être placé dans le bon répertoire) lancer les instructions suivantes :

```
> combn(5,3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2     3
[2,]    2    2    2    3    3    4    3    3    4     4
[3,]    3    4    5    4    5    5    4    5    5     5
> source("combn.R")
> combnRC(5,3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2     3
[2,]    2    2    2    3    3    4    3    3    4     4
[3,]    3    4    5    4    5    5    4    5    5     5
> system.time(x <- combn(200,3))
   user  system elapsed
  2.056   0.024   2.073
> system.time(x <- combnRC(200,3))
   user  system elapsed
  0.236   0.000   0.235
```

On constate ainsi un gain de temps important avec l'utilisation de notre nouvelle fonction R qui utilise le code écrit en C/C++.

Prise en main

Codez en R pur, et en version hybride R-C/C++ (ou R-Fortran), les fonctions `ar1simR()`, et `ar1simRC()-ar1simC` (ou `ar1simRF()-ar1simF`). Ces fonctions ont trois arguments d'entrée : $n \in \mathbb{N}$, $\phi \in (-1, 1)$ et $M \in \mathbb{N}$. Le calcul effectué par ces fonctions est le suivant.

Pour $m = 1, \ldots, M$:

a) simulation d'un vecteur aléatoire $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^\mathsf{T}$ de loi $\mathcal{N}_n(\mathbf{0}; \mathcal{I}_n)$;

b) fabrication d'un vecteur $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$, avec $x_1 = \epsilon_1$, et tel que pour tout $t = 2, \ldots, n$, on ait $x_t = \phi x_{t-1} + \epsilon_t$;

c) calcul de l'estimateur des moindres carrés conditionnel $\hat{\phi}_m$ de ϕ :

$$\hat{\phi}_m = \frac{\sum_{t=2}^{n} x_{t-1} x_t}{\sum_{t=2}^{n} x_{t-1}^2}.$$

Les fonctions créées devront retourner la valeur $\overline{\hat{\phi}} - \phi = \frac{1}{M} \sum_{m=1}^{M} \hat{\phi}_m - \phi$, permettant ainsi d'évaluer numériquement le biais de l'estimateur $\hat{\phi}$ de ϕ.

Comparez ensuite la vitesse d'exécution de la version en R pur avec celle de la version appelant du code `C/C++` (ou `Fortran`). Pour cela, traçez un graphique des valeurs $(M, time_M)$ pour $M = 1\,000, 2\,000, \ldots, 100\,000$. Vous prendrez $n = 1\,000$ et $\phi = 0,75$.

Note : La fonction `arima.sim()` permet de réaliser les points a) et b) ci-dessus, tandis que la fonction `arima()` permet d'effectuer le point c). Ces deux fonctions, déjà existantes sous R, ne doivent pas être utilisées dans cette prise en main. Elles sont très rapides puisque codées en `C`, mais ne se limitent pas à effectuer les calculs précédents.

Astuce

Afin de faciliter le développement de vos codes, un bon éditeur est toujours utile. Celui-ci devrait minimalement offrir l'indentation et la coloration syntaxique. Vous pourriez envisager d'utiliser l'un des gratuiciels suivants :

- un éditeur de code R comme `RStudio`, `Tinn-R` ou `Emacs` ;
- un éditeur de code source `C/C++` et `Fortran` comme `Emacs` ou `Code::Blocks` (disponible à l'adresse `http://www.codeblocks.org`).

Astuce

Le *package* `rbenchmark` permet de comparer facilement le gain espéré, en termes de temps de calcul, lors de l'utilisation d'une fonction R-C/C++ ou R-Fortran *versus* une fonction R pur. Essayez par exemple de le vérifier sur les résultats obtenus dans la prise en main précédente au moyen du code qui suit :

```
n <- 1000
phi <- 0.75
M <- 2000
dyn.load("ar1sim.dll")
benchmark(Rcode=ar1simR(n,phi,M),
          Ccode=.C("ar1simC",as.integer(n),phi,
                   as.integer(M),res=0.0)$res,
          replications=1000)
```

Astuce

Le `Fortran` et le R stockent les matrices (tableaux) de la même manière : les lignes d'une même colonne sont stockées consécutivement en mémoire. En `C/C++`, ce sont les colonnes d'une même ligne qui sont consécutives. Il faut donc faire attention lorsque l'on envoie une matrice de R vers `C/C++`. Par exemple, l'élément d'indice `[i,j]` d'une matrice R correspond à l'élément d'indice `[(j-1)*nombre-de-lignes + (i-1)]` en `C/C++` (en `C/C++`, les indices commencent à 0).

6.5.3 Appel de librairies `C/C++` ou `Fortran` externes

Il est possible d'utiliser une fonction présente dans une librairie **externe** à R au moyen des fonctions R `.C()` (pour les librairies `C/C++`) ou `.Fortran()` (pour les librairies `Fortran`).

Astuce

Une utilisation amusante de cette approche, permettant de vérouiller la session Windows, est la suivante :

```
dyn.load(file.choose()) # Sélectionner le fichier
                        # C:/windows/system32/user32.dll
.C("LockWorkStation")
```

Il est également possible d'appeler une librairie de fonctions externes directement depuis votre code `C/C++` ou `Fortran`. Nous allons présenter quelques

librairies scientifiques qui nous semblent intéressantes :
- l'API (*Application Programming Interface*) C de R ;
- la librairie C++ newmat ;
- les librairies Fortran BLAS et LAPACK.

Renvoi

D'autres librairies existent, certaines gratuites (voire libres) comme :
- en C/C++ :
  - http://www.gnu.org/software/gsl
  - http://www.math.uiowa.edu/~dstewart/meschach
  - http://www.nrbook.com/a/bookcpdf.php
- en Fortran :
  - http://calgo.acm.org
  - http://www.nrbook.com/a/bookfpdf.php
  - http://www.nrbook.com/a/bookf90pdf.php
  - http://math-atlas.sourceforge.net

d'autres payantes :
- en C/C++ :
  - http://www.nag.co.uk/numeric/CL/CLdescription.asp
  - http://www.vni.com/products/imsl/c/imslc.php
- en Fortran :
  - http://www.nag.co.uk/numeric/RunderWindows.asp
  - http://www.nag.co.uk/numeric/fl/FLdescription.asp
  - http://www.nag.co.uk/numeric/fn/FNdescription.asp
  - http://www.vni.com/products/imsl/fortran/overview.php

6.5.3.1 L'API R

L'API C de R est une librairie mise au point par les développeurs de R. Elle peut être utilisée à partir d'un programme C/C++ sans avoir besoin de faire appel à R (on lui donne alors le nom d'API R *Standalone*). Elle peut également être utilisée depuis un code C/C++ qui sera lui-même appelé par R, comme nous l'avons vu dans la section précédente. L'avantage est de pouvoir utiliser des routines existantes sans avoir besoin de les reprogrammer soi-même. Pour l'utiliser, vous devrez inclure dans votre code source en C/C++ deux fichiers d'entête (R.h et Rmath.h) qui sont nécessaires afin de déclarer ou définir certaines fonctions ou constantes mathématiques.

Renvoi

La documentation (en anglais) pour cette librairie, y compris la liste des fonctions et constantes qu'elle renferme, est disponible à l'adresse http://cran.r-project.org/doc/manuals/R-exts.html#The-R-API.

Vous pourrez aussi consulter avec profit le dossier `nmath/` dans les sources de R, disponibles à l'adresse http://svn.r-project.org/R/trunk/src/nmath.

Nous présentons ci-dessous un code C/C++ disponible à l'adresse http://biostatisticien.eu/springeR/integ.cpp permettant de calculer l'intégrale suivante :

$$\int_{\epsilon_1}^{\pi} \Phi(t+\epsilon_2)dt,$$

où ϵ_1 et ϵ_2 sont les réalisations de deux variables aléatoires indépendantes (respectivement normale et uniforme) et où $\Phi(\cdot)$ désigne la fonction de répartition d'une loi $\mathcal{N}(0,1)$. Le seul intérêt de cet exemple est d'illustrer l'utilisation de fontions de l'API R pour simuler des variables aléatoires, calculer une probabilité ou encore effectuer un calcul d'intégration numérique.

```
#include <R.h>
#include <Rmath.h>
extern "C" {

  typedef void integr_fn(double *x, int n, void *ex);
  void f(double *t, int n, void *ex);

  void testintegral(double *res) {

    // Fonction d'intégration numérique de l'API R
    void Rdqags(integr_fn f,void *ex,double *a,
                double *b,double *epsabs,
                double *epsrel,double *result,
                double *abserr,int *neval,
                int *ier,int *limit,int *lenw,
                int *last,int *iwork, double *work);

    GetRNGstate(); // Lire la graine du générateur de R

    double *a, *b, *epsabs, *epsrel, *result,
      *ex, *abserr, *work;
    int *last, *limit, *lenw, *ier, *neval, *iwork;

    ex = new double[1]; a = new double[1];
    b = new double[1]; epsabs = new double[1];
    epsrel = new double[1]; result = new double[1];
    abserr = new double[1]; neval = new int[1];
    ier = new int[1]; limit = new int[1];
    lenw = new int[1]; last = new int[1];
```

```
30      limit[0] = 100;
31      lenw[0] = 4 * limit[0];
32      iwork =  new int[limit[0]];
33      work = new double[lenw[0]];
34
35      a[0] = rnorm(0.0,1.0);    // ϵ1 de loi N(0,1)
36      b[0] = M_PI; // La constante π ≃ 3.141593···
37      ex[0] = runif(0.0,1.0);   // ϵ2 de loi U[0,1]
38
39      // On calcule l'integrale
40      Rdqags(f, ex, a, b, epsabs, epsrel,
41                result, abserr, neval, ier,
42                limit, lenw, last,
43                iwork, work);
44
45      // Le résultat est stocké dans res[0]
46      res[0] = result[0];
47
48      PutRNGstate(); // Écrire la graine du générateur
49
50      // On libère de la mémoire
51      delete[] ex, a, b, epsabs, epsrel, result, abserr,
52        neval, ier, limit, lenw, last, iwork, work;
53    }
54
55    // Définition de la fonction à intégrer
56    void f(double *t, int n, void *ex) {
57      int i;
58      double eps2;
59      eps2 = ((double*)ex)[0];
60      for (i=0;i<n;i++) {
61        t[i] = pnorm(t[i]+eps2,0.0,1.0,1,0);}
62    }
63  }
```

Les instructions de compilation de cette fonction afin d'obtenir un fichier DLL sont les suivantes :

```
g++ -c integ.cpp -o integ.o -I"C:\Program Files\R\R-3.1.0\include"
g++ -shared -o integ.dll integ.o ^
     -L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Attention

Vous aurez noté qu'il a été nécessaire d'indiquer respectivement les chemins d'accès vers les dossiers contenant les fichiers R.h, Rmath.h et R.dll. Modifiez ceux-ci au besoin suivant la configuration de votre système. Par ailleurs, le symbole ^ indique, en langage MS-DOS, une ligne incomplète.

Linux

```
g++ -c integ.cpp -o integ.o  -I"/usr/lib/R/include" -fPIC
g++ -shared -o integ.so integ.o -I"/usr/lib/R/include" \
        -L"/usr/lib" -lR
```

Maintenant, pour réaliser le calcul sous R, il suffit d'utiliser les instructions suivantes :

```
> dyn.load(paste("integ",.Platform$dynlib.ext,sep=""))
> # c'est-à-dire: dyn.load("integ.dll") sous Microsoft Windows.
> .C("testintegral",val=0.0)$val
[1] 2.407529
```

Bien entendu, le résultat de ce calcul varie suivant les réalisations ϵ_1 et ϵ_2 obtenues.

6.5.3.2 La librairie newmat

La librairie newmat permet de manipuler différents types de matrices et d'effectuer des opérations classiques telles que : multiplication, transposition, inversion, calculs de valeurs propres, décompositions, etc.

Renvoi

La documentation complète de cette librairie est disponible (en anglais) ici : http://www.robertnz.net/nm11.htm

Le code ci-dessous, disponible à l'adresse http://biostatisticien.eu/springeR/inv.cpp, est un code C/C++ utilisant cette librairie pour inverser une matrice et pourra être appelé depuis R.

```
1 #define WANT_STREAM
2 #define WANT_MATH
3 #include "newmatap.h"
4 #include "newmatio.h"
5 #ifdef use_namespace
```

```
 6 using namespace NEWMAT;
 7 #endif
 8 extern "C" {
 9   void invC(double *values, int *nrow) {
10     int i, j;
11     Matrix M(nrow[0],nrow[0]);
12     M << values;
13     M << M.i(); // Calcul de l'inverse de M
14     for (i=1;i<=nrow[0];i++) {
15       for(j=1;j<=nrow[0];j++) {
16         values[nrow[0]*(i-1)+j-1] = M(i,j);
17         }
18     }
19     M.Release();
20     return;
21   }}
```

Astuce

Téléchargez le fichier http://www.robertnz.net/ftp/newmat11.zip et décompressez-le dans `C:/newmat`. Ensuite, tapez les instructions suivantes dans une fenêtre `MSDOS` :

```
cd \
cd newmat
g++ -O2 -c *.cpp
ar cr newmat.a *.o
ranlib newmat.a
cp newmat.a newmat.dll
```

Après quelques minutes, les librairies `newmat.a` et `newmat.dll` sont créées dans le dossier `C:\newmat`.

Vous devez maintenant créer la librairie `inv.dll` (ou `inv.so` sous Linux) au moyen des instructions suivantes :

```
cd dossier contenant le fichier inv.cpp
g++ -IC:\newmat -o inv.o -c inv.cpp
R CMD SHLIB inv.cpp -IC:\newmat C:/newmat/newmat.a
```

Linux

```
g++ -I/usr/include/R  -I/usr/local/include -Inewmat -fpic \
    -c inv.cpp -o inv.o
R CMD SHLIB inv.cpp -Inewmat newmat/newmat.a
```

Vous pouvez ensuite utiliser le code C/C++ précédent depuis R de la façon suivante. Commencez par sauver le code suivant dans un fichier nommé inv.R :

```
> inv <- function(M) {
+  n <- nrow(M)
+  return(matrix(.C("invC",Minv=as.vector(M),n)$Minv,nrow=n,ncol=n))
+ }
```

Puis lancez les instructions :

```
> dyn.load(paste("inv",.Platform$dynlib.ext,sep=""))
> A <- matrix(rnorm(9),nrow=3)
> solve(A) # La fonction R solve() permet le calcul
           # de l'inverse d'une matrice.
           [,1]       [,2]       [,3]
[1,] -5.3289747 -6.6112370 -7.7277894
[2,]  1.5293527  2.6146603  2.0573112
[3,]  0.5656936  0.1808586 -0.4919597
> inv(A)
           [,1]       [,2]       [,3]
[1,] -5.3289747 -6.6112370 -7.7277894
[2,]  1.5293527  2.6146603  2.0573112
[3,]  0.5656936  0.1808586 -0.4919597
```

Les deux fonctions solve() et inv() permettent donc d'obtenir le même résultat pour ce qui est de l'inversion d'une matrice. On peut voir ci-dessous que le gain en temps de calcul pour cette seule opération est appréciable.

```
> require("rbenchmark")
> benchmark(Rcode=solve(A),Ccode=inv(A),replications=100000)
   test replications elapsed relative user.self sys.self
2 Ccode       100000   1.078    1.000     1.052    0.032
1 Rcode       100000   2.765    2.565     2.692    0.092
  user.child sys.child
2          0         0
1          0         0
```

6.5.3.3 Les librairies BLAS et LAPACK

Les librairies BLAS (*Basic Linear Algebra Subprograms*) et LAPACK (*Linear Algebra PACKage*) sont des librairies Fortran permettant d'effectuer de nombreuses opérations matricielles. Nous allons voir sur un exemple simple comment les utiliser.

Commencez par installer le logiciel de (dé)compression 7-zip disponible à l'adresse http://www.7-zip.org/download.html. Utilisez ce logiciel (deux fois) pour décompresser (en deux étapes) le fichier http://www.netlib.org/lapack/lapack.tgz. Tous les fichiers et sous-dossiers obtenus (BLAS, CMAKE, etc.) devront être placés directement dans un dossier nommé C:\lapack. Ce

dossier contiendra par exemple à la racine un fichier nommé `make.inc.example`, qu'il vous faudra renommer en `make.inc` après en avoir modifié la ligne `SHELL = /bin/sh` en `SHELL = sh`. Ensuite, tapez les instructions suivantes dans une fenêtre de commande `MS-DOS` :

```
cd C:\lapack
make lapacklib blaslib
```

Au bout de plusieurs minutes, les librairies statiques `librefblas.a` ainsi que `liblapack.a` seront créées.

Renvoi

La documentation pour ces librairies peut être consultée à l'adresse http://www.netlib.org/lapack/lug. Il est aussi bon de regarder directement les codes sources des routines `BLAS` et `LAPACK` que l'on souhaite utiliser, car ils contiennent la description détaillée des arguments de ces routines.

Voilà le code `Fortran`, disponible à l'adresse http://biostatisticien.eu/springeR/inv.f90, d'une sous-routine permettant de calculer l'inverse d'une matrice. Il utilise les sous-routines `external DGETRF` et `DGETRI` de la librairie `Lapack`.

```
1  ! Renvoie l'inverse d'une matrice calculée en trouvant
2  ! la décomposition LU. Dépend de LAPACK.
3  subroutine invF(A,Ainv,m)
4    double precision, dimension(m,m), intent(in) :: A
5    double precision, dimension(size(A,1),size(A,2)), &
6                          intent(inout) :: Ainv
7
8    ! array de travail pour LAPACK
9    double precision, dimension(size(A,1)) :: work
10   integer, dimension(size(A,1)) :: ipiv !indices pivot
11   integer :: n, info, m
12
13   ! Procedures externes définies dans LAPACK
14   external DGETRF
15   external DGETRI
16
17   ! Stocke A dans Ainv pour l'empêcher
18   ! d'être écrasée par LAPACK
19   Ainv = A
20   n = size(A,1)
21
22   ! DGETRF calcule une factorisation LU
```

```
23   ! d'une matrice générale A de dimension M x N
24   ! en utilisant un pivot partiel avec des échanges de lignes.
25   call DGETRF(n, n, Ainv, n, ipiv, info)
26
27   if (info /= 0) then
28      stop 'La matrice est numériquement singulière!'
29   end if
30
31   ! DGETRI calcule l'inverse d'une matrice en utilisant
32   ! la factorisation LU calculée par DGETRF.
33   call DGETRI(n, Ainv, n, ipiv, work, n, info)
34
35   if (info /= 0) then
36      stop 'Inversion de la matrice: échec!'
37   end if
38 end subroutine invF
```

Pour compiler ce code, lancez les instructions suivantes dans une fenêtre de commandes MS-DOS :

```
cd %HOMEPATH%/Desktop # À adapter à votre cas.
gfortran -c inv.f90 -o invBis.o -I"C:/lapack"
gfortran -shared -o invBis.dll invBis.o -I"C:/lapack" ^
     C:/lapack/liblapack.a C:/lapack/librefblas.a
```

Linux

Sous Linux, on utilisera les instructions suivantes :

```
gfortran -c inv.f90 -o invBis.o -fPIC
gfortran -shared -o invBis.so invBis.o /usr/lib64/liblapack.so.3
```

Après avoir créé le fichier `invBis.dll` (ou `invBis.so` sous Linux) au moyen des instructions précédentes, vous pouvez lancer R et y taper les instructions suivantes :

```
> dyn.load(paste("invBis",.Platform$dynlib.ext,sep=""))
> A <- matrix(rnorm(4),nrow=2)
> B <- matrix(0,nrow=2,ncol=2)
> .Fortran("invF",A,res=B,2L)$res
           [,1]     [,2]
[1,] -0.1145578 1.770077
[2,]  0.8990205 2.157801
> solve(A)
           [,1]     [,2]
[1,] -0.1145578 1.770077
[2,]  0.8990205 2.157801
```

Astuce

Sous Linux, il est possible et assez aisé d'accéder à des sous-routines `LAPACK` depuis **R**. Par exemple, pour utiliser la sous-routine `zhpevx` qui calcule les valeurs/vecteurs propres d'une matrice hermitienne, on peut taper les instructions suivantes :

```
dyn.load(paste(.Library,"/../modules/lapack.so",sep=""))
is.loaded("zhpevx")"
```

6.5.3.4 Mélanger des librairies `C/C++` et `Fortran`

Il est possible d'appeler des fonctions `C/C++` depuis un code `Fortran`, en utilisant l'instruction `F77_SUB(name)`. Voyons ceci sur un exemple, qui permet de générer deux observations indépendantes : l'une de loi $\mathcal{N}(0,1)$ et l'autre de loi uniforme. Le programme `Fortran` ci-dessous utilise les fonctions (en `C`) `GetRNGstate`, `PutRNGstate`, `rnorm` et `runif` de l'API **R**, que nous avions déjà utilisées dans la section 6.5.3.1. Sauvegardez-le dans un fichier nommé `random.f`.

```
1       SUBROUTINE random(x,y)
2       real*8 normrnd, unifrnd, x, y
3       CALL rndstart()
4       x = normrnd()
5       y = unifrnd()
6       CALL rndend()
7       RETURN
8       END
```

Il vous faut ensuite créer le fichier `random.c` suivant :

```
1 #include <R.h>
2 #include <Rmath.h>
3 void F77_SUB(rndstart)(void) {GetRNGstate();}
4 void F77_SUB(rndend)(void) {PutRNGstate();}
5 double F77_SUB(normrnd)(void) {return rnorm(0,1);}
6 double F77_SUB(unifrnd)(void) {return runif(0,1);}
```

Pour compiler le tout afin de créer votre fichier `DLL`, utilisez les instructions suivantes :

```
gfortran -c random.f -o randomf.o
gcc -c random.c -o randomc.o  -I"C:\Program Files\R\R-3.1.0\include"
gfortran -shared randomf.o randomc.o -o random.dll ^
      -L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Linux

Sous Linux, utilisez :

```
gfortran -c random.f -o randomf.o -fPIC
gcc -c random.c -o randomc.o -I"/usr/lib/R/include" -fPIC
gfortran -shared randomf.o randomc.o -o random.so
```

Vous pouvez maintenant appeler votre code depuis R au moyen des instructions suivantes :

```
> dyn.load(paste("random",.Platform$dynlib.ext,sep=""))
> .Fortran("random", as.double(1), as.double(1))
[[1]]
[1] 2.100142
[[2]]
[1] 0.9682872
```

Il est également possible d'appeler des fonctions Fortran depuis un code C/C++, en se servant des instructions suivantes :

F77_NAME(name) pour déclarer une routine Fortran en C
F77_CALL(name) pour appeler une routine Fortran depuis C
F77_COMDECL(name) pour déclarer un bloc COMMON FORTRAN en C
F77_COM(name) pour accéder à un bloc COMMON FORTRAN depuis C

Voyons ceci sur un petit exemple (avec du Fortran77 pour changer). Enregistrez le code ci-dessous dans un fichier nommé combnCF.cpp :

```
1 #include <R.h>
2 #include <Rmath.h>
3 extern "C" {
4 void combnCF(int *combmat, int *n, int *m) {
5 // Attention! Il ne faut pas de majuscules dans le nom
6 // de la sous-routine.
7 void  F77_NAME(combnf)(int *combmat, int *n, int *m);
8 F77_CALL(combnf)(combmat,n,m);
9  }
10 }
```

Tapez ensuite les instructions suivantes dans une fenêtre de commandes MS-DOS afin de créer la librairie qui sera appelée depuis R :

```
g++ -c combnCF.cpp -o combnCF.o -I"C:\Program Files\R\R-3.1.0\include"
gfortran -c combn.f90 -o combn.o
g++ -shared -o combnCF.dll combnCF.o combn.o  ^
    -L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Linux

Sous Linux :

```
g++ -c combnCF.cpp -o combnCF.o -I"/usr/lib/R/include" -fPIC
gfortran -c combn.f90 -o combn.o -fPIC
g++ -shared -o combnCF.so combnCF.o combn.o \
 -I"/usr/lib/R/include" -L"/usr/lib" -lR
```

Modifiez à présent le code de la fonction `combnRC()` présenté en page 260 :
- changez le nom de cette fonction pour `combnRCF()` ;
- remplacez `"combn"` et `"combnC"` par `"combnCF"`.

Enregistrez ce nouveau code dans un fichier nommé `combnCF.R`. Puis tapez les deux instructions ci-dessous dans la console de R :

```
> source("combnCF.R")
> combnRCF(5,3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2     3
[2,]    2    2    2    3    3    4    3    3    4     4
[3,]    3    4    5    4    5    5    4    5    5     5
```

6.5.4 Appel d'un code R depuis un programme C/C++ appelé par R

Nous avons vu comment il était possible d'appeler une routine `C/C++` (ou `Fortran`) depuis R. Il est également possible d'appeler des fonctions R depuis un programme `C/C++`. Pour cela, il faudra utiliser un certain type de pointeur appelé `SEXP` (pour *Simple EXPression*) ainsi que la fonction `.Call()`. Dans cette sous-section, nous nous contentons de fournir un simple exemple. Le lecteur pourra s'en inspirer pour élaborer des exemples plus compliqués.

Renvoi

Le lecteur pourra consulter avec profit le site `http://cran.r-project.org/doc/manuals/R-exts.html#Handling-R-objects-in-C`.

Nous allons montrer dans l'exemple suivant comment il est possible d'appeler la fonction `pmvt()` du *package* `mvtnorm` depuis un code `C/C++`, lui-même appelé par R. La fonction `pmvt()` calcule la probabilité qu'un vecteur aléatoire suivant une loi de Student multivariée produise une valeur dans un pavé de $\mathbb{R}^n$ à spécifier.

Contrairement aux exemples des sections précédentes qui utilisaient la fonction `.C()`, nous allons ici devoir utiliser la fonction `.Call()`. Par ailleurs,

le code C/C++ devra consister en une fonction (nommée pmvtC ci-après) qui renvoie une structure de type SEXP et qui prend aussi comme arguments des SEXP. Le code ci-dessous, disponible à l'adresse http://biostatisticien.eu/springeR/pmvt.cpp, sera celui qui, une fois transformé en fichier DLL, sera appelé par la fonction .Call().

```
#include <R.h>
#include <Rdefines.h>
#include "Rmath.h"
#include <R_ext/Rdynload.h>
extern "C" {
  SEXP pmvtCR(SEXP Rupper,SEXP Rcorr,SEXP Rdf,
                SEXP Rpmvt,SEXP Renv,SEXP Rres) {
    SEXP R_fcall;
    if(!isFunction(Rpmvt) & (Rpmvt != R_NilValue))
             error("Rpmvt doit être une fonction");
    if(!isEnvironment(Renv))
             error("Renv doit être un environnement");
    PROTECT(R_fcall = lang4(Rpmvt,Rupper,Rcorr,Rdf));
    REAL(Rres)[0] = REAL(eval(R_fcall, Renv))[0];
    UNPROTECT(1);
    return(Rres);
  }
}
```

Pour compiler ce fichier, il faut utiliser les instructions suivantes :

```
g++ -c pmvt.cpp -o pmvt.o -I"C:\Program Files\R\R-3.1.0\include"
g++ -shared -o pmvt.dll pmvt.o ^
    -L"C:\Program Files\R\R-3.1.0\bin\i386" -lR
```

Linux

Sous Linux, on utilisera les instructions :

```
g++ -m64 -I/usr/include/R  -I/usr/local/include -fpic  \
      -c pmvt.cpp -o pmvt.o
R CMD SHLIB pmvt.cpp
# ou bien:
g++ -m64 -shared -L/usr/local/lib64 -o pmvt.so pmvt.o \
      -L/usr/lib64/R/lib -lR
```

Vous pouvez maintenant utiliser cette fonction depuis R de la façon suivante. Téléchargez le fichier http://biostatisticien.eu/springeR/pmvt.R contenant le code suivant :

```
pmvtRCR <- function(upper,corr,df) {
 res <- 0.0
 Rpmvt <- function(upper,corr,df) {
   d <- length(upper)
   pmvt(lower=rep(-Inf,d),upper=upper,delta=rep(0,d),
   corr=matrix(corr,ncol=d),df=df)}
 dyn.load(paste("pmvt",.Platform$dynlib.ext,sep=""))
 res<- .Call("pmvtCR",as.double(upper),as.double(corr),
          as.double(df),Rpmvt,new.env(),as.double(res))
 dyn.unload(paste("pmvt",.Platform$dynlib.ext,sep=""))
return(res) }
```

Puis tapez les instructions suivantes :

```
> require("mvtnorm")
> corr <- diag(3)
> set.seed(1)
> source("pmvt.R")
> pmvtRCR(c(2,3,2),corr,c(1,1,1))
[1] 0.706062
> set.seed(1)
> pmvt(lower=rep(-Inf,3),upper=c(2,3,2),corr=corr,df=c(1,1,1))[1]
[1] 0.706062
```

Astuce

Si un objet `SEXP` contient un vecteur (par exemple `SEXP x`) ou une matrice (par exemple `SEXP M`), vous pouvez utiliser les instructions `R_len_t n = length(x)` et `R_len_t p = nrows(M)` pour créer des entiers contenant la longueur `n` du vecteur `x` ou le nombre de lignes `p` de la matrice `M`. Le fichier Rinternals.h contient la liste de très nombreuses fonctions utiles de ce genre.

6.5.5 Appel d'un code R depuis un programme Fortran

Nous vous conseillons d'utiliser le logiciel Open-Source `RFortran` disponible à l'adresse `http://www.rfortran.org`.

6.5.6 Quelques fonctions utiles

Nous présentons ci-dessous quelques fonctions qui pourraient vous être utiles. Les fonctions suivantes s'utilisent dans une fenêtre terminal `MS-DOS` (ou `Cygwin`, voir la page 282) :

- `nm` : liste les symboles des fichiers objets (ex : `nm random.dll`) ;

- `objdump` : affiche de l'information sur les fichiers objets (ex : `objdump -x random.dll`) ;
- `ldd` : affiche les bibliothèques partagées nécessaires (ex : `ldd random.dll`).

Les fonctions suivantes s'utilisent dans R :

- `getLoadedDLLs()` : affiche une liste de toutes les DLLs chargées dans la session courante (ex : `getLoadedDLLs()`) ;
- `is.loaded()` : vérifie si une librairie a été chargée (ex. : `is.loaded(random.dll)`, ou, sous Linux, essayez `dyn.load(paste(.Library,"/../modules/lapack.so",sep="")) ; is.loaded("zhpevx"))`.

SECTION 6.6

† Débogage de fonctions

Nous allons présenter dans cette section diverses options qui s'offrent à vous pour déboguer une fonction afin d'y repérer une erreur. L'erreur pourrait se trouver soit dans le code R, soit dans un code `C/C++` ou `Fortran` qui serait appelé par votre fonction R.

Renvoi

Vous pourriez consulter avec profit la page http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR.

6.6.1 Débogage de fonctions en R pur

Nous présentons quelques fonctions de débogage, très utiles lors de l'écriture de code R.

La fonction `browser()`

Une fonction de debogage qui peut se révéler utile lorsque l'on programme dans R est la fonction `browser()`. En effet, il vous suffit d'insérer l'instruction `browser()` dans le code source de votre fonction et, quand vous l'exécuterez, le programme s'arrêtera à l'endroit où vous avez inséré cette ligne.

Voici un exemple montrant l'utilisation de `browser()` dans une fonction nommée `mc()` qui calcule l'estimateur des moindres carrés des paramètres inconnus dans un modèle de régression linéaire simple (voir le chapitre 12 pour plus de détails).

```
1 mc <- function(X,Y,intercept=TRUE) {
2    X <- as.matrix(X)
3    Y <- as.matrix(Y)
4    plot(X,Y)
```

```
5     nbindiv <- nrow(X)
6  browser()
7     if (intercept==TRUE) X <- cbind(rep(1,nbindiv),X)
8     betachap <- solve(t(X) %*% X) %*% t(X) %*% Y
9     curve(betachap[1]+betachap[2]*x,add=TRUE)
10
11 return(betachap)
12 }
```

Sourcez le fichier contenant le bout de code précédent (par exemple au moyen de l'instruction `source(file.choose())`), puis tapez :

```
mc(X=cars[,2],Y=cars[,1])
```

Vous constatez alors que le programme stoppe et que l'on peut examiner le contenu des variables locales situées avant `browser()`. Par exemple, tapez `nbindiv`.

Remarque

En entrant la lettre `n` (pour *next*) de façon successive, on peut inspecter le code et le contenu des variables locales de façon séquentielle. Pour quitter le mode d'inspection, tapez `Q`.

Voici un aperçu d'une possible session de débogage :

```
mc(X=cars[,2],Y=cars[,1])
Called from: mc(X = cars[, 2], Y = cars[, 1])
Browse[1]>nbindiv
[1] 50
Browse[1]> betachap
Erreur: Objet "betachap" non trouvé
Browse[1]> n
debug: if (intercept == T) X <- cbind(rep(1, nbindiv), X)
Browse[1]> n
debug: betachap <- solve(t(X) %*% X) %*% t(X) %*% Y
Browse[1]> n
debug: curve(betachap[1] + betachap[2] * x, add = T)
Browse[1]> betachap
          [,1]
[1,] 8.2839056
[2,] 0.1655676
Browse[1]> Q
>
```

Remarque

En entrant la lettre `c` (pour *continue*), le code est exécuté jusqu'à la fin, à moins que la commande `browser()` ne soit rencontrée à nouveau.

La fonction `debug()`

Une autre fonction intéressante est la fonction `debug()` qui revient à placer une instruction `browser()` au tout début du code d'une fonction. Ainsi `debug(var)` va marquer la fonction `var` comme étant débogable. Tout appel subséquent de cette fonction lancera le débogueur en ligne.

```
debug(var)
var(1:3)
```

Pour supprimer ce marquage, on utilise la fonction `undebug()`.

```
undebug(var)
```

6.6.2 Erreur dans le code R

Commencez par modifier la ligne 7 du fichier combn.R (voir page 260) en remplaçant la flèche d'affectation `<-` par le seul signe `<`. Ainsi nous incorporons une erreur qui consiste en une omission d'un symbole (le symbole `-`) :

```
combmat < matrix(out$res,nrow=m,byrow=F)
```

Sauvez le fichier résultant, puis sourcez-le et lancez l'instruction suivante :

```
> combnRC(5,3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    0    0    0    0    0    0    0    0     0
[2,]    0    0    0    0    0    0    0    0    0     0
[3,]    0    0    0    0    0    0    0    0    0     0
```

Comme on peut le constater, il y a une erreur dans le résultat. Et l'erreur que nous avons introduite délibérément dans le code pourrait être difficile à détecter s'il s'était agi d'une omission accidentelle. Voyons comment nous pourrions procéder pour tenter de détecter d'où vient l'erreur. Il suffit d'installer, puis de charger en mémoire le *package* `debug`. On utilise ensuite la fonction `mtrace()` contenue dans ce *package*, de la façon suivante :

```
mtrace(combnRC)
combnRC(5,3)
```

Vous devriez alors voir apparaître la fenêtre de débogage avec le code source de la fonction `combnRC()`. Un appui répété sur la touche `ENTRÉE` évaluera séquentiellement les différentes lignes de votre code source jusqu'à l'affichage suivant (survenant lors de l'évaluation de la ligne que nous avons modifiée) :

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

Ceci nous mettra très certainement la puce à l'oreille qu'il y a une erreur à cet endroit. Nous pourrons alors corriger l'erreur, par exemple en utilisant l'instruction `fix(combnRC)`.

Au passage, vous avez sans doute remarqué que l'utilisation de la fonction `mtrace()` ne nous a pas permis de rentrer plus profondément dans l'appel suivant :

```
.C("combnC",res=as.integer(combmat),as.integer(n),
                as.integer(m))
```

6.6.3 Erreur dans le code C/C++ ou Fortran

Nous allons à présent voir comment il est possible d'effectuer le même genre d'opération de débogage, pour une partie de code écrite en langage `C/C++` ou `Fortran`. Il s'agira essentiellement d'utiliser l'option de compilation `-g` pour rajouter des informations sur le code source dans le fichier `DLL` créé, et ensuite d'utiliser un outil de débogage spécialisé.

Attention

Vous aurez besoin d'installer un outil de débogage. Nous conseillons l'utilisation du logiciel libre `GDB`. Téléchargez la version 7.4 (32 *bits*) à l'adresse `http://biostatisticien.eu/springeR/32/gdb.exe` et placez-la dans le dossier `C:\Rtools\bin`. Ce logiciel s'utilise en ligne de commande et est un peu austère. Il est donc utile de lui adjoindre une interface graphique (ou *Graphical User Interface* GUI), comme le *Data Display Debugger* (dont l'acronyme est `DDD`) ou `Emacs`. Une autre option intéressante, sous Microsoft Windows, est le logiciel `Insight`, contenu dans l'ensemble d'outils `MinGW`, disponible à l'adresse `http://sourceforge.net/projects/mingw/files/OldFiles/insight.exe/download`. Toutefois, ce dernier logiciel semble tomber en désuétude. Si vous essayez de l'utiliser, pensez à modifier la variable d'environnement système `Path` pour indiquer le chemin d'accès vers `Insight` (probablement `C:\insight\bin`), comme cela est expliqué en page 253.

Sous Microsoft Windows, vous devrez installer la version du logiciel `Emacs` disponible à l'adresse `http://vgoulet.act.ulaval.ca/emacs/windows`. Le logiciel `DDD` est un peu plus compliqué à utiliser sous Windows. Vous devez lancer le programme d'installation de `Cygwin` (disponible à l'adresse `http://cygwin.com/install.html`), choisir comme dos-

sier d'installation C:\Rtools\bin et sélectionner les logiciels Devel: ddd et Math: gnuplot (et accepter d'installer les dépendances requises). Notez que si la liste de sites de téléchargement est vide, pous pouvez essayer d'utiliser l'URL suivante : http://cygwin.mirrorcatalogs.com. Pour pouvoir utiliser DDD, il faut aussi installer un portage sous Windows du système de fenêtrage X de Linux. Le logiciel Xming, disponible à l'adresse http://biostatisticien.eu/springeR/Xming-6-9-0-31-setup.exe, s'avère être un bon choix. Vous pourriez aussi envisager d'utiliser MobaXterm (http://mobaxterm.mobatek.net), ou encore le serveur Xorg de Cygwin (sélectionner X11: xorg-server: X.Org servers lors de l'installation).

6.6.4 Débogage avec GDB

Lancez une fenêtre de commandes MS-DOS à partir du menu Démarrer de Windows (tapez cmd) dans laquelle vous taperez :

```
cd chemin vers le dossier contenant inv.cpp
g++ -IC:\newmat -o inv.o -c inv.cpp -g
g++ -shared -o inv.dll inv.o -IC:\newmat C:/newmat/newmat.a
```

Cela va créer le fichier inv.dll avec des informations de débogage (voir page 270 pour la création de la librairie newmat).

Astuce

Afin de pouvoir également déboguer des fonctions présentes dans la librairie newmat, vous devrez auparavant créer cette librairie de façon à ce qu'elle contienne des informations de débogage :

```
cd \
cd newmat
g++ -c *.cpp -Wno-deprecated -g
ar cr newmatdebug.a *.o
ranlib newmatdebug.a
cp newmatdebug.a newmatdebug.dll
```

Ensuite tapez :

```
gdb Rgui
(gdb) run
```

Cela devrait avoir pour effet de lancer R, où vous taperez :

```
> setwd("chemin vers le fichier inv.dll")
> dyn.load("inv.dll")
```

Allez ensuite dans le menu `Misc/Interrompre vers le débuggueur` de R, ce qui vous redonnera la main dans GDB (fenêtre noire), où vous pourrez taper :

```
(gdb) info share
(gdb) break inv.cpp:1
(gdb) signal 0
```

La première instruction (`info share`) vous permettra de constater que votre librairie (`inv.dll`) a bien été chargée, la deuxième (`break inv.cpp:1`) permet d'ajouter un point d'arrêt sur la première ligne (exécutable) du fichier `inv.cpp`, la dernière (`signal 0`) vous permet de sortir de GDB pour revenir dans R. Tapez-y :

```
> A <- matrix(rnorm(4),nrow=2)
> source("inv.R") # Fichier créé en page 271.
> inv(A)
```

Lorsque le point d'arrêt va être recontré par le processeur, l'exécution du code va être suspendue. Vous pouvez maintenant taper les instructions ci-après dans GDB. La première (`list`) affiche les prochaines lignes devant être exécutées, la deuxième (`next`) permet de passer (successivement) à la ligne suivante, la troisième (`print nrow[0]`) affiche la valeur de `nrow[0]`, et la dernière continue l'exécution du code jusqu'à la fin ou bien jusqu'au prochain point d'arrêt.

```
(gdb) list
(gdb) next
(gdb) print nrow[0]
(gdb) continue
```

Vous êtes de nouveau dans R et vous voyez s'afficher le résultat de l'appel `inv(A)`. Vous pouvez taper les instructions suivantes pour vérifier que le résultat est identique avec la fonction `solve()`, et enfin pour quitter R.

```
> solve(A)
> q()
```

Linux

Sous Linux, on lance dans un terminal la commande :

```
R -d gdb
```

à la place de `gdb Rgui`.
Une autre façon de procéder consiste à utiliser les deux instructions suivantes :

```
export R_HOME=/usr/lib64/R
gdb /usr/lib64/R/bin/exec/R
```

Par ailleurs, pour revenir de R vers GDB, on utilise la combinaison de touches `CTRL+C`. De plus, notez que pour aller de GDB vers R, après avoir tapé `signal 0` (ou de façon équivalente `c`), il faut presser la touche `ENTRÉE`.

Astuce

Notez que GDB peut être appelé avec certaines options. Par exemple :

```
--directory=DIR    Chercher des fichiers source dans DIR.
--pid=PID          Attacher au processus actif PID.
```

Renvoi

Vous pourrez consulter avec profit la documentation de GDB, disponible à l'adresse http://sourceware.org/gdb/current/onlinedocs/gdb.

Astuce

Il est possible d'installer/compiler un *package* (appelons-le PKG par la suite) avec des informations de débogage (équivalent à l'utilisation du *flag* -g vu précédemment). Pour cela, il faut commencer par créer un fichier nommé Makevars.win (Makevars sous Linux) dans un sous-dossier nommé .R/ de son répertoire %HOME%. Ce fichier devra contenir les lignes suivantes :

```
## for C++ code
CXXFLAGS=-g
```

Pour cela, vous pouvez par exemple taper : WINDOWS+R, cmd, ENTRÉE, cd %HOME%, ENTRÉE, mkdir .R, ENTRÉE, cd .R, ENTRÉE, echo CXXFLAGS=-g > Makevars.win, ENTRÉE. Ensuite, il suffira de créer le *package* PKG (à partir des sources au moyen de la commande R CMD INSTALL --build --debug PKG), de l'installer, puis d'utiliser l'une des méthodes de débogage présentées précédemment. Notez que le fichier NAMESPACE de votre *package* PKG devra contenir la ligne : useDynLib("PKG") afin que le fichier DLL (ou .so) soit automatiquement chargé lorsque vous lancerez dans R la commande : require("PKG"). En cas d'échec de cette procédure, vous pouvez toujours utiliser la fonction dyn.load() pour charger « à la main » votre librairie depuis l'endroit où elle a été installée.

Astuce

Il est également possible de faire afficher le contenu d'un objet de type SEXP (notons s cet objet). Pour cela, vous pouvez soit inclure dans votre code C/C++ l'instruction PrintValue(s);. Ainsi, dès que cette instruction sera rencontrée pendant l'exécution de votre code, le contenu de l'objet s sera affiché dans la console de R. Vous pouvez aussi utiliser l'instruction

`p Rf_PrintValue(s)` depuis la console de GDB. Notez que dans ce dernier cas, il est possible que l'affichage du contenu de l'objet `s` dans la console de R soit différé jusqu'au moment où R reprendra la main sur GDB.

6.6.4.1 Débogage avec Emacs

Nous avons vu comment déboguer du code à l'aide de GDB. Voyons comment il est possible d'effectuer le même genre d'opération avec la combinaison des logiciels Emacs (et son excellent module ESS, *Emacs Speaks Statistics*) et GDB. Notez que vous devez avoir installé GDB comme cela est expliqué en page 282. Notez aussi que vous devrez avoir créé, depuis une fenêtre MS-Dos, le fichier combn.dll avec des informations de débogage (*flag* `-g`) au moyen des commandes suivantes :

```
g++ -g -c combn.cpp -o combn.o
g++ -shared -o combn.dll combn.o
```

Remarque

Sous Emacs, l'écriture `M-x` signifie que vous devez utiliser la combinaison simultanée des touches `ALT` et `X`, tandis que `C-x` désigne l'utilisation simultanée des touches `CTRL` et `X`, alors que `[RET]` indique un retour chariot, c'est-à-dire l'appui sur la touche `ENTRÉE` (ou `RETURN`).

Pour cela, ouvrez Emacs (voir en page 282 pour l'adresse de téléchargement de ce logiciel) puis exécutez les commandes qui suivent. Par exemple, la première ligne ci-dessous est effectuée en tapant simultanément `ALT` et `X`, puis `R` (qui affichera `M-x R` dans le bandeau horizontal inférieur d'Emacs), puis `ENTRÉE` (qui affichera dans le bandeau inférieur `ESS [S(R): R (newest)] starting data directory? ~/`), puis encore `ENTRÉE` (qui aura pour effet de lancer R dans Emacs).

```
M-x R [RET] [RET]
M-x gdb [RET] gdb -i=mi --annotate=3 [RET]
```

Votre fenêtre Emacs devrait alors être séparée en deux, avec R en haut et GDB en bas. Si ce n'est pas le cas, allez dans le menu `File/Split Window` ou `File/New Window Below` (`C-x 2`), puis allez dans le menu `Buffer` pour sélectionner `*R\{*} *`.

Attention

La variable d'environnement système `Path` doit contenir l'entrée `C:\Rtools\bin` en premier, afin que la version de GDB appelée soit la 7.4.

Il faut ensuite récupérer le numéro de processus de R. Pour cela, on utilise (sous le système d'exploitation Windows) la combinaison de touches CTRL+ALT+Suppr pour ouvrir le gestionnaire des tâches. On sélectionne ensuite l'onglet des Processus. Dans le menu Affichage/Sélectionner des colonnes..., on coche la boîte PID (identificateur de processus), ce qui aura pour effet de rajouter une colonne PID dans le gestionnaire des tâches. On cherche alors le numéro du processus (PID) dont le nom est Rterm.exe *32 (par exemple 5404). Une autre possibilité, plus rapide, est de taper Sys.getpid() dans la fenêtre R supérieure d'Emacs.

Linux

Sous Linux, on peut récupérer le numéro du processus de R, en tapant directement dans Emacs :

```
M-! Shell command: pgrep R [RET]
```

On tape ensuite dans Emacs les instructions suivantes :

```
(gdb) attach 5404 [RET]
(gdb) signal 0 [RET]
```

Cliquez sur le panneau (appelé *Buffer* dans Emacs) intitulé *R*, et entrez les instructions suivantes :

```
> setwd("chemin vers le fichier combn.R")
> source("combn.R")
> dyn.load(paste("combn",.Platform$dynlib.ext,sep=""))
```

Cliquez sur la sous-fenêtre inférieure (*Buffer* *gud*).

```
C-c C-c
(gdb) b combn.cpp:1 [RET]
(gdb) c [RET]
```

Cliquez sur la sous-fenêtre supérieure (*Buffer* *R*),

```
> combnRC(5,3)
```

```
C-g
M-x gdb-many-windows
```

Passez la fenêtre d'Emacs en plein écran. Votre fenêtre Emacs devrait maintenant être divisée en six panneaux comme on peut le voir sur la figure 6.2. Cliquez éventuellement sur certaines entrées du menu Buffer en cas de besoin.

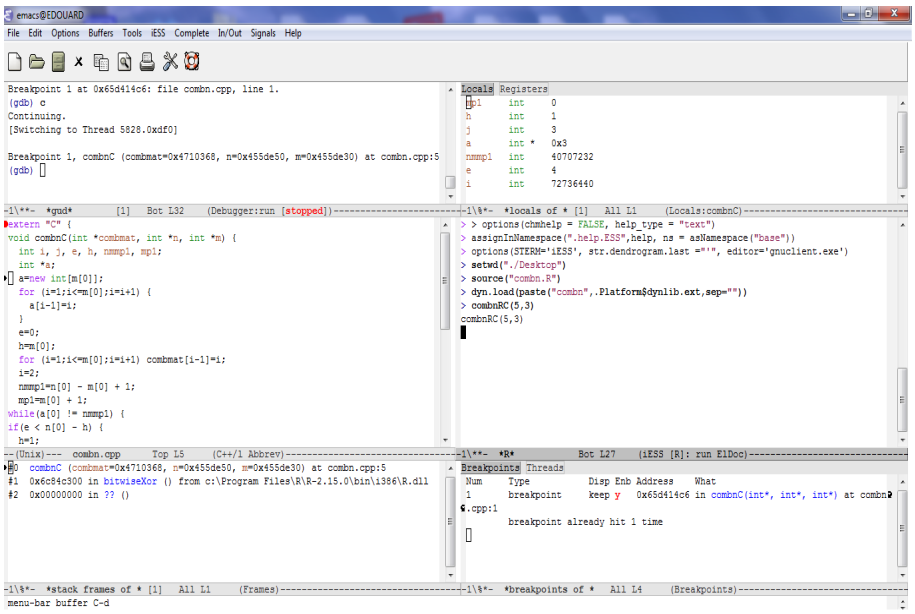


FIGURE 6.2 – Emacs et GDB.

Cliquez sur le panneau en bas à droite intitulé `*breakpoints of*`. Sélectionnez le menu `Buffers/*R\{*} *`.

Cliquez maintenant sur la fenêtre `combn.cpp`. Vous verrez alors de nouveaux icônes apparaître dans la partie supérieure d'Emacs. Vous pouvez par exemple cliquer sur le symbole de `Next Line` (à droite du `GO`) pour effectuer le code `C/C++` pas à pas.

Prise en main

- Modifiez la ligne 32 du fichier integ.cpp afin qu'elle devienne `limit[0] = -1;`. Recompilez ce programme et appelez-le depuis R comme nous l'avons vu précédemment : `.C("testintegral",val=0.0)$val`. Vous devriez constater un « plantage » du logiciel R. Supposons que vous ne vous souveniez plus d'avoir effectué cette modification. Utilisez alors ce que vous avez appris pour détecter d'où vient l'erreur.
- Déboguez le fichier pmvt.cpp vu en 6.5.4. Tapez l'instruction `p Rf_PrintValue(Rpmvt)` depuis la console de `GDB` pour faire afficher (dans la console de R) le contenu de l'objet `Rpmvt`.

6.6.4.2 Débogage avec DDD

Il faut commencer par lancer Xming (ou un outil équivalent), ce qui devrait entraîner l'affichage de son icone dans la barre des tâches. On lance ensuite la fenêtre de terminal de Cygwin  , dans laquelle on tape les instructions suivantes :

```
$ export DISPLAY=localhost:0.0
$ cd vers le repertoire contenant les fichiers sources et les DLLs
$ ddd Rgui
```

Vous devrez possiblement être patient pour que DDD finisse par s'ouvrir.

Linux

Sous Linux, la seule chose à faire est de taper l'instruction `R -d ddd`.

On tape alors les instructions suivantes dans GDB (panneau inférieur) :

```
(gdb) dir $cwd
(gdb) run
```

La première instruction indique à GDB de rechercher les fichiers sources dans le répertoire courant (celui qui serait donné par la commande `pwd`), contournant ainsi un problème de gestion hasardeuse des chemins Windows. La deuxième instruction (on peut aussi cocher la case : `Program/Run in Execution Window`, et cliquer sur `Program/Run`, puis sur `Run`) lance R, dans lequel on tape :

```
> dyn.load("inv.dll")
```

Notez que le fichier inv.dll a été créé avec des informations de débogage comme cela est indiqué à la page 283. On va alors dans le menu `Misc/Interrompre vers le débugger`, ce qui nous redonne la main vers le logiciel DDD. On va alors dans le menu `File/Open source...` et on ouvre le fichier inv.cpp. On coche aussi l'entrée *Data Window* du menu `View` (et éventuellement l'entrée *Display Local Variables* du menu `Data`, si vous êtes patient !). On place ensuite un (ou plusieurs) point(s) d'arrêt dans le code à déboguer (en double-cliquant au début d'une ligne ou par un clic droit), par exemple en face de l'instruction `M << values;`. Ceci a pour effet d'afficher un signe de stop. Puis on tape `continue` (ou simplement `c`) dans la partie inférieure `(gdb)`. Ceci nous ramène vers R. Tapez-y :

```
> A <- matrix(rnorm(4),nrow=2)
> source("inv.R") # Fichier créé en page 271.
> inv(A)
```

Lorsque le (premier) point d'arrêt va être rencontré par le processeur, l'exécution du code va être suspendue. Vous pouvez maintenant utiliser l'outil graphique DDD pour déboguer pas à pas votre code.

Notez qu'il est possible d'afficher plusieurs valeurs d'un tableau (*array*). Pour cela, on peut par exemple taper dans la fenêtre inférieure `(gdb)` l'instruction suivante :

```
graph display values[0] @ 4
```

pour afficher les 4 (premières) valeurs du tableau `values`.

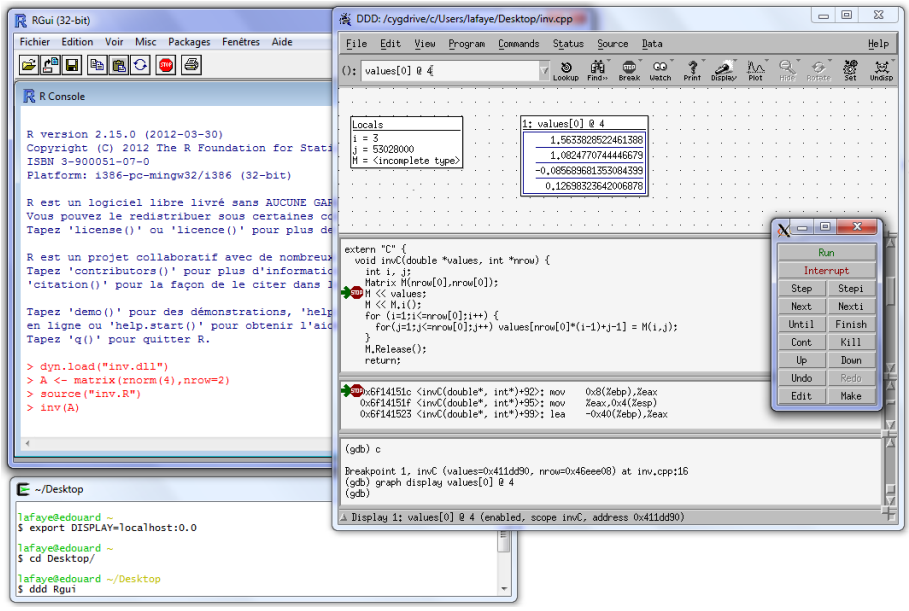


FIGURE 6.3 – DDD et GDB.

6.6.4.3 Débogage avec Insight

Le logiciel **Insight** semble avoir des difficultés à fonctionner sur certaines versions de Windows. Nous présentons tout de même son fonctionnement pour ceux qui auraient une version de Windows compatible, ou encore dans l'éventualité où une nouvelle version du logiciel **Insight** verrait le jour après la publication de ce livre.

Recompilons notre fichier en utilisant le *flag* `-g` (et éventuellement `-fPIC`) qui indique au compilateur `C++` d'ajouter des informations sur le code source directement dans le fichier compilé.

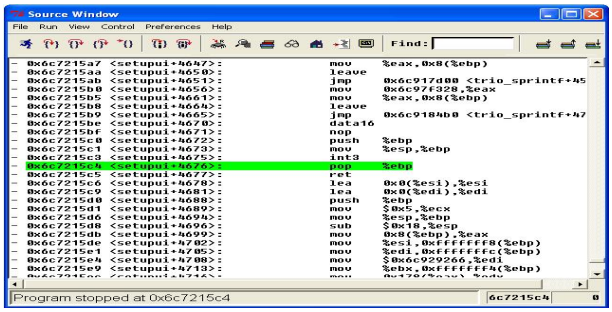
```
g++ -c combn.cpp -o combn.o -g
g++ -shared -o combn.dll combn.o
```

Ensuite, depuis la fenêtre `MS-DOS`, on lance la commande `insight Rgui.exe` puis on clique sur le bouton `Run`  .

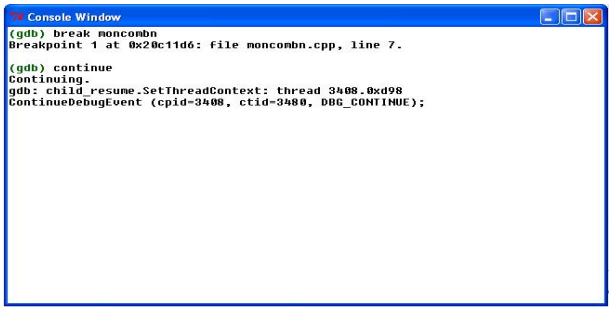
On tape alors les commandes suivantes dans la console R qui s'affiche :

```
> source("combn.R")
> dyn.load(paste("combn",.Platform$dynlib.ext,sep=""))
```

On va dans le menu R intitulé `Misc`, puis `Interrompre vers le debuggueur`. On se retrouve alors dans la fenêtre `Insight`.



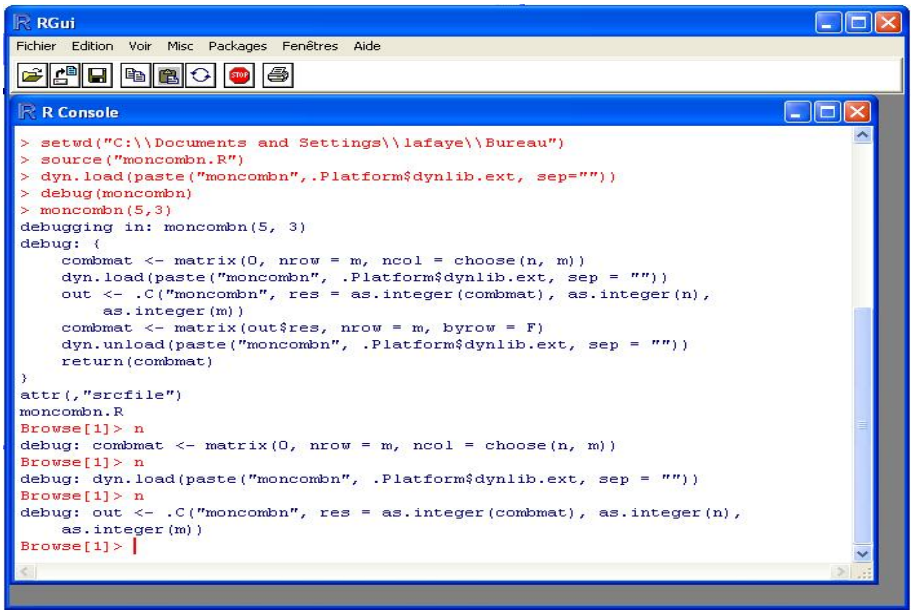
Puis, depuis `Insight`, on sélectionne le menu `View - Console [CTRL+N]`. Ceci a pour effet d'ouvrir la fenêtre de commandes du débogueur `GDB`. Nous allons ensuite ajouter un point d'arrêt sur la fonction `combnC` en tapant :
`break combnC`
Ensuite on tape :
`continue`
Ceci a pour effet de nous ramener vers le logiciel R, et dès qu'un appel à la fonction `combnC` sera effectué, le débuggueur reprendra la main.



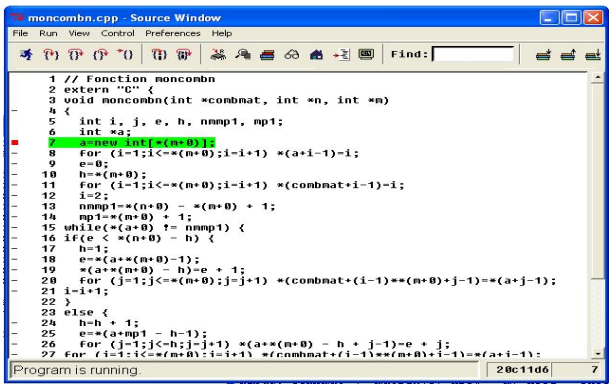
Maintenant, nous tapons les instructions suivantes dans la console de R :

```
> debug(combnRC)
> combnRC(5,3)
```

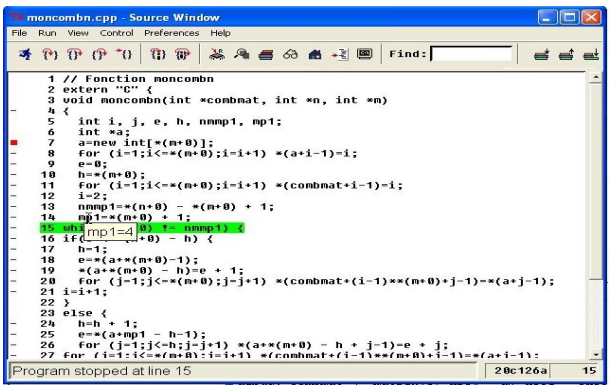
Puis on utilise l'instruction `n` (pour *next*) pour sauter d'une instruction à la suivante de notre code R, jusqu'à atteindre l'appel à la fonction écrite en `C++`.



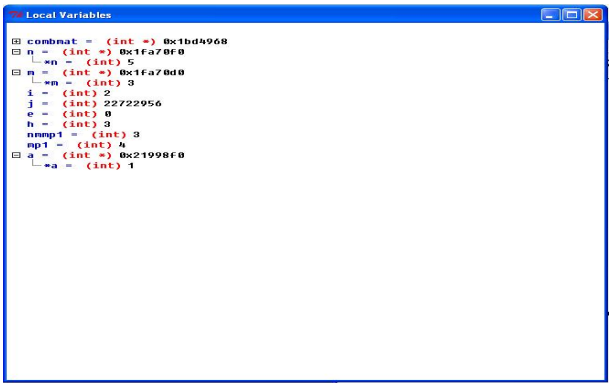
Le point d'arrêt que nous avions fixé est alors détecté et nous nous retrouvons de retour dans `Insight`.



On peut alors cliquer sur l'icone  pour exécuter pas à pas les lignes de code `C++` et voir les valeurs des différentes variables lors de l'exécution du code.

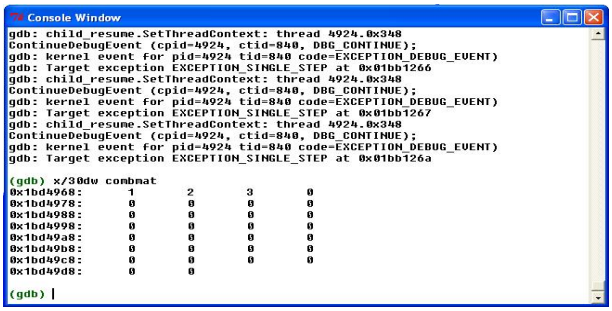


La fenêtre `Local Variables` (obtenue via le menu `View -> Local Variable [CTRL+L]`) permet d'afficher la totalité des variables locales et leur contenu en cours d'exécution du programme.



Notez que si l'on souhaite voir le contenu d'un tableau de valeurs `C++` (matrice ou vecteur dans **R**), il suffit d'aller dans la console `GDB` et de taper par exemple :

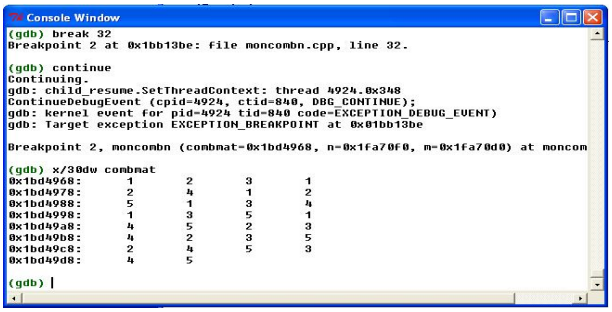
```
x/30dw combmat
```



On peut également faire un affichage graphique de ce tableau de valeurs, en le sélectionnant et en cliquant sur `plot`.

Maintenant, on peux taper les instructions suivantes dans la console GDB afin de rajouter un point d'arrêt à la ligne 32 de notre code C++, puis on relance l'exécution du code, et lorsque le point d'arrêt est rencontré, le code s'arrête à nouveau et on peut demander à ré-afficher le contenu de l'*array* x :

```
break 32
continue
x/30dw combmat
```



6.6.4.4 Détection de fuites de mémoire

L'affichage de l'un des messages `Segmentation fault` (ou encore `segfault`), `invalid next size`, `std::bad_alloc` (que vous rencontrerez certainement sous Linux !) ou encore des résultats incohérents, ou de façon plus radicale, le plantage complet de R, indique souvent un problème de mémoire (accès à une adresse non réservée ou non initialisée, utilisation d'une mémoire libérée, etc.). Ces fuites de mémoire se produisent souvent lorsque l'on a oublié d'utiliser l'instruction `delete[] ptr;` pour effacer de la mémoire un pointeur `ptr` introduit dans notre fonction C/C++. Ce problème est aussi parfois visible dans le gestionnaire des tâches lorsque l'on fait tourner une grosse simulation dans R et que l'on voit que le processus de R prend de plus en plus de mémoire, alors qu'il ne devrait pas.

Linux

Sous Linux, l'affichage de l'utilisation mémoire par les différents processus s'obtient grâce à la commande (tapée dans un terminal) `watch -d free` pour l'utilisation globale ou bien `top -p PID` pour un processus en particulier (utilisez `ps au` pour trouver le `PID` du processus souhaité). On peut aussi utiliser l'outil graphique `ksysguard`.

Une autre erreur classique consiste à chercher à manipuler la n-ième case d'un tableau de dimension strictement inférieure à n (accès à une mémoire non définie). Il peut alors devenir délicat de détecter d'où vient le problème. Le logiciel `Dr Memory` (`http://code.google.com/p/drmemory`), et éventuellement les

logiciels duma (http://duma.sourceforge.net) ainsi qu'electric-fence-win32 (http://code.google.com/p/electric-fence-win32), pourront s'avérer de précieux outils dans cette situation.

Linux

Sous Linux, on pourra utiliser les logiciels Valgrind ou Electric Fence.

Voyons sur un exemple comment utiliser le logiciel Dr. Memory que vous installerez dans le dossier C:\drmemory (choisir l'entrée Add Dr. Memory to the system PATH for all users lors de l'installation).

Le petit code suivant contient plusieurs erreurs, qui ne sont pas forcément faciles à détecter pour un débutant. Vous pouvez le télécharger à l'adresse http://biostatisticien.eu/springeR/memoire.cpp.

```
1 extern "C" {
2   void testmemoire(int *M, double *a) {
3     double *ptr1, *ptr2;
4     int i;
5     ptr1 = new double[10000];
6     ptr2 = new double[M[0]];
7     ptr1[0] = 1.0;
8     for (i=1;i<10000;i++) {
9       ptr1[i] = (double)i;
10      ptr2[i] = ptr1[i-1] * (double)i; }
11    delete[] ptr2;
12    for (i=0;i<10;i++) a[i] = ptr2[i];
13    return;
14  }
15 }
```

Commençez par fabriquer le fichier DLL associé au moyen des instructions suivantes tapées dans une fenêtre de commandes Ms-Dos :

```
cd dossier où est le fichier memoire.cpp
g++ -o memoire.o -c memoire.cpp -g
g++ -shared -o memoire.dll memoire.o
```

Linux

Sous Linux, on utilise les instructions suivantes :

```
g++ -o memoire.o -c memoire.cpp -g -fPIC
g++ -shared -o memoire.so memoire.o
```

Ensuite, tapez `drmemory.exe -- Rgui` dans votre fenêtre de commandes (soyez patient), puis dans la console de R, tapez les instructions suivantes :

```
> dyn.load("memoire.dll")
> .C("testmemoire",10000L,3.0)
> q()
```

Recherchez maintenant les occurences de `testmemoire` dans le fichier texte qui s'est ouvert. Cela vous indiquera les différentes lignes pouvant contenir des erreurs. Ainsi, on peut par exemple détecter qu'il y a une erreur en ligne 12. On s'aperçoit en fait que le tableau `a` est de longueur 1 (et contient au départ la seule valeur 3.0) alors que l'on cherche à écrire dans les cases 0 à 9 de ce tableau. En outre, le pointeur `ptr2` a été supprimé à la ligne précédente.

Vous pouvez aussi essayer l'instruction R ci-après, et vous constaterez dans le gestionnaire des tâches que la quantité de RAM utilisée par R augmente de façon très importante. Ceci est dû à l'oubli de l'instruction `delete[] ptr1;` dans le code `C/C++` ci-dessus.

```
> for (i in 1:10000) .C("testmemoire",10000L,as.double(1:10))
```

Linux

L'équivalent de `Dr Memory` sous Linux s'appele `Valgrind`. Pour détecter d'où vient la fuite, on peut utiliser l'instruction :

```
R -d 'valgrind --leak-check=full'

> dyn.load("memoire.so")
> .C("testmemoire",10000L,3.0)
> q()
```

Dans la sortie de `valgrind`, il faut alors chercher les erreurs indiquées ainsi que les numéros de ligne du code source de `memoire.cpp`. Les instructions ci-dessous pourront vous donner d'autres types d'erreurs affichées par R, et qui seront détectées par `Valgrind` :

```
> # Fonctionne une seule fois!
> # Après R plante avec un: "caught segfault":
> .C("testmemoire",10000L,c(3.0,5.0))
> # R se ferme: "invalid next size":
> .C("testmemoire",10000,c(3.0,5.0))
> # R se ferme: "std::bad_alloc":
> .C("testmemoire",10^12,c(3.0,5.0))
> # Fonctionne alors que ptr2 n'est plus défini:
> .C("testmemoire",10000L,as.double(1:10))
```

SECTION 6.7

Calcul parallèle et calculs sur cartes graphiques

6.7.1 Calcul parallèle

Il est possible d'accélérer vos calculs en les faisant tourner sur plusieurs processeurs simultanément, ces processeurs pouvant même se trouver sur des ordinateurs différents. Il existe plusieurs *packages* spécialisés pour cela, mentionnés sur le `CRAN Task View: High-Performance and Parallel Computing with R`, que vous pouvez consulter à l'adresse `http://cran.r-project.org/web/views/HighPerformanceComputing.html`.

Le plus simple d'utilisation est le *package* `parallel` avec le protocole de communication `PSOCK`, dont nous allons décrire succinctement l'utilisation ci-dessous sur un exemple.

Astuce

L'utilisation du protocole `MPI` (*Message Passing Interface*), via le *package* `Rmpi`, offre plus de flexibilité que le protocole `PSOCK`, mais il nécessite l'installation préalable d'autres logiciels (`OpenMPI` ou `mpich2` par exemple).

Renvoi

Le lecteur pourra consulter avec profit les sites `http://www.divms.uiowa.edu/~luke/R/cluster/cluster.html`, `http://www.sfu.ca/~sblay/R/snow.html` et `http://cran.r-project.org/web/packages/snowfall/vignettes/snowfall.pdf`.

Le code R suivant évalue numériquement (par une simulation de Monte Carlo) le niveau empirique du test de normalité de Shapiro-Wilks pour un niveau nominal de 5 % :

```
> myfunc <- function(M=1000) {
+  decision <- 0
+  for (i in 1:M) {
+   x <- rnorm(100)
+   if (shapiro.test(x)$p < 0.05) decision <- decision + 1
+  }
+ return(decision)
+ }
```

Affichons le temps de calcul résultant de l'utilisation de ce code avec $M = 60\,000$ boucles de Monte Carlo :

```
> system.time({
+  M <- 60000
+  decision <- myfunc(M)
+  print(decision/M)
+ })
[1] 0.04893333
   user  system elapsed
 16.839   0.000  16.843
```

Voyons maintenant comment il est possible de paralléliser l'exécution de ce code en utilisant le *package* `parallel`, ainsi que le gain résultant en termes de temps de calcul. Nous avons utilisé 6 processeurs (6 cœurs en fait).

Astuce

Pour connaître le nombre de cœurs sur votre ordinateur, tapez l'instruction `devmgmt.msc` dans le menu `Démarer/Exécuter`. Regardez ensuite le nombre de lignes dans l'entrée intitulée `Processeurs`. Sous Linux, tapez `top` dans une fenêtre de terminal, puis `1`. Vous pourrez voir le nombre de cœurs. Une autre possibilité est d'utiliser, depuis **R**, la fonction `detectCores()` du package `parallel`.

```
> require("parallel")
> system.time({
+  nbcores <- 6 # Ne devrait pas dépasser detectCores()-1.
+  M <- 60000
+  cl <- makeCluster(nbcores, type = "PSOCK")
+  out <- clusterCall(cl, myfunc, round(M/nbcores))
+  stopCluster(cl)
+  decision <- 0
+  for (clus in 1:nbcores) {
+   decision <- decision + out[[clus]]
+  }
+  print(decision/(round(M/nbcores)*nbcores))
+ })
[1] 0.04803333
   user  system elapsed
  0.006   0.011   5.808
```

Astuce

Il est possible de connaître le numéro de processus (*PID*) de chaque nœud (*node*) de la grappe de calcul (*cluster*) :

```
> require("parallel")
> Sys.getpid() # PID du Master.
[1] 7994
```

```
> cl <- makeCluster(4, type = "PSOCK")
> (out <- clusterCall(cl, Sys.getpid)) # PIDs des nœuds du
                                       # cluster.
[[1]]
[1] 8019
[[2]]
[1] 8028
[[3]]
[1] 8037
[[4]]
[1] 8046
```

6.7.2 Calcul sur cartes graphiques

Le processeur, ou CPU (de l'anglais *Central Processing Unit*, Unité centrale de traitement), est le composant de l'ordinateur qui est chargé d'exécuter les instructions contenues dans les programmes informatiques. Cependant, depuis quelques années, il est maintenant également possible d'exécuter des calculs sur un GPU (de l'anglais *Graphical Processing Unit*, Unité graphique de traitement), autrement dit sur une carte graphique. Les cartes graphiques permettant cette opération sont notamment développées par la compagnie Nvidia, et elles peuvent contenir plusieurs centaines de processeurs de calculs travaillant en parallèle. Le gain en termes de temps de calcul peut alors s'avérer appréciable. Toutefois, pour utiliser cette technologie, il faut connaître le langage de programmation CUDA, développé par Nvidia. Quelques développeurs R se sont lancés dans l'aventure et ont commencé à regrouper quelques fonctions dans le *package* gputools, qui n'est pour l'instant toutefois disponible que pour la plateforme Linux.

Nous présentons ci-dessous un petit exemple d'utilisation de ce *package*. La carte graphique utilisée est une NVIDIA GeForce GTX 480.

```
> require("gputools")
> A <- matrix(runif(40000),nrow=200,ncol=200)
> B <- matrix(runif(40000),nrow=200,ncol=200)
> system.time(cor(A, B, method="kendall")) # Calcul sur CPU.
   user  system elapsed
 28.074   0.004  28.137

> system.time(gpuCor(A, B, method="kendall")) # Calcul sur GPU.
   user   system elapsed
  0.836    0.052   0.891
```

Renvoi

Le lecteur désirant approfondir ses connaissances sur le sujet pourra consulter avec profit les sites suivants : `http://cran.r-project.org/web/packages/gputools/gputools.pdf` et `http://developer.nvidia.com/object/cuda_training.html`.

Termes à retenir

`function(<par1>,<par2>,...,<parN>) <corps>` : déclare un objet fonction
`"{"()` : permet de définir un bloc d'instructions et retourne la dernière évaluée
`class(), "class<-"()` : extrait, affecte la classe de l'objet
`missing()` : teste si un paramètre effectif a été fourni
`attributes(), "attributes<-"()` : extrait, affecte tous les attributs sous forme de liste
`attr(), "attr<-"()` : extrait, affecte un seul attribut
`expression()` : crée un objet expression
`parse()` : convertit du texte en une expression
`eval()` : évalue une expression
`"~"()` : crée un objet formule
`new.env()` : crée un environnement
`local()` : permet d'exécuter du code localement dans un environnement

Exercices

6.1- Pour chacune des lignes de commandes suivantes :
- `function(nom) {nom}`
- `(function(nom) {nom})("Ben")`
- `(function(nom) {cat(nom,"\n")})("Ben")`
- `(function(nom) {invisible(nom)})("Ben")`

indiquez quel est le type (ou classe) de l'objet R retourné. Quel est aussi l'affichage produit lors de l'exécution de ces lignes de commandes ?

6.2- Y a-t-il une différence entre :
- `nom <- function(nom) nom` et `nom <- function(nom) {nom}`
- `nom <- function(nom) {nom}` et `nom <- function(nom) {return(nom)}`
- `nom <- function(nom) {nom}` et `(function(nom) {nom}) -> nom`

6.3- Y a-t-il une différence lors de l'exécution de `nom()` et `nom("Peter")` lorsque :
- `nom <- function(nom="Peter") nom`
- `nom <- function(nom="Peter") nom2 <- nom`

Pour ces deux déclarations de la fonction `nom()`, y a-t-il une différence dans la nature de l'objet R `res` obtenu par `res <- nom("Ben")` ?

6.4- Quel est l'objet R retourné lors de l'exécution de `nom()` lorsque :

```
nom <- function(nom="Peter") {
  nom
  # La dernière instruction est un commentaire!
}
```

6.5- Lorsque `nom <- function(prénom="Peter",nom="L") { paste(prénom,nom)}`, quel est l'objet **R** retourné par :
- `nom(prénom="Ben")`
- `nom(pr="Ben")`
- `nom(n="D",p="R")`

6.6- Réécrivez la déclaration de la fonction suivante toujours en une seule ligne de commande, mais sans utiliser le séparateur de commande « ; » :
```
nom <- function(nom) { if(missing("nom"))
nom <- "Peter"; cat(nom,"\n") }
```

6.7- Quelle est la sortie lors de l'exécution de `noms("peteR","Ben","R")` lorsque :
- `noms <- function(...) c(...)`
- `noms <- function(...) list(...)`
- `noms <- function(...) for(nom in c(...)) print(nom)`
- `noms <- function(...) for(nom in list(...)) print(nom)`

Même question lors de l'exécution de
`noms(c("peteR","L"),c("Ben","L"),c("R","D"))`

6.8- Lorsque `noms <- function(noms=c("Ben","R"),...) c(noms,...)`, quels sont les objets **R** retournés par `noms("PeteR")`, `noms(nom="PeteR")` et `noms(noms="PeteR")` ?
Même question lorsque
`noms <- function(...,noms=c("Ben","R")) c(noms,...)`.

6.9- Créez une fonction constructeur `Homme()` générant un objet de classe `"Homme"` avec pour champs `prénom` et `nom` (placé dans un objet du type `list`). Créez la méthode `bonjour.Homme()` dont le but est uniquement d'afficher `"bonjour Monsieur PRENOM NOM!"` (sans oublier `"\n"` en fin d'affichage !) pour un objet ayant respectivement les valeurs `"PRENOM"` et `"NOM"` pour les champs `prénom` et `nom`. Lorsque `homme <- Homme("Ben","L")`, que produisent les exécutions des commandes suivantes :
`bonjour.Homme(homme)` et `bonjour(homme)` ? Que faut-il éventuellement exécuter pour que les deux résultats soient identiques ?

6.10- Créez les fonctions analogues pour la classe `"Femme"` (indice : ne pas oublier de mettre à jour le genre dans `bonjour.Femme()`). Lorsque `femme <- Femme("Dominique","L")`, que produisent les exécutions des commandes suivantes : `bonjour.Homme(femme)`, `bonjour.Femme(femme)` et `bonjour(femme)` ?

6.11- Lorsque `bienvenu <- function(...) for(individu in list(...)){ bonjour(individu)}`, que renvoie `bienvenu(homme,femme)` ?
Et lorsque `bienvenu <- function(...) for(individu in c(...)){ bonjour(individu)}` ?
Même question lorsque `bonjour.default <- function(obj){ cat("Bonjour",obj,"!\n")}`.

Fiche de TP

Programmation de fonctions et programmation R orientée objet

Avant même de lire les travaux pratiques proposés ci-dessous, il est fortement conseillé de reprendre ceux proposés dans les chapitres précédents (et notamment celui sur « les graphiques avancés ») afin de réorganiser leurs solutions en autant de fonctions que nécessaire.

A- Gestion d'un compte bancaire

L'objectif est de créer trois fonctions minimalistes permettant de gérer des comptes bancaires. Les comptes seront stockés dans des objets `data.frame` tous nommés `compte` et sauvergardés dans autant de fichiers .RData différents. Tous ces fichiers seront localisés dans un même répertoire dont le nom (ainsi que son chemin d'accès) est supposé avoir été enregistré dans la variable R `.repertoire.comptes` et accessible dans toutes les fonctions à développer.

6.1- L'instruction `file.path(.repertoire.comptes,paste(nom,".RData", sep=""))` fournit le chemin du fichier associé au compte `nom`. Créez la fonction `chemin.compte()`, avec pour unique paramètre formel le paramètre `nom` (représentant le nom du compte à traiter), permettant de retourner le chemin complet du fichier (contenant l'objet `compte` de classe `data.frame`) ayant pour extension `.RData`.

6.2- Sachant que `factor(levels=c("Débit","Crédit"))`, `numeric(0)` et `character(0)` fournissent respectivement des vecteurs vides dont les types sont explicites, quelle instruction génère une matrice de données vide contenant les champs prédéfinis `somme`, `mode`, `date` et `remarque` ? Créez alors la fonction `compte()` (à ne pas confondre avec la variable `compte` appelée dans son corps) avec, pour paramètre unique, le paramètre `nom` qui permet de créer un nouveau compte.

6.3- Créez alors les fonctions `débite()` et `crédite()` permettant respectivement de débiter et de créditer une somme `somme` (deuxième paramètre) dans le compte nommé `nom` (premier paramètre). Le troisième paramètre est un commentaire quelconque à placer éventuellement dans le paramètre `remarque`. Un quatrième paramètre pourra représenter la date avec pour valeur par défaut `format(Sys.time(),"%d/%m/%Y")` (c'est-à-dire la date de la saisie). Pensez à utiliser les fonctions `load()` et `save()` pour charger et sauvegarder la variable `compte` à l'intérieur du corps de chaque fonction.

6.4- Si `compte` désigne la matrice de données contenant les informations sur le compte, que renvoie `sum(compte[compte$mode=="Crédit","somme"])` ?

Modifiez la fonction `compte()` pour qu'elle permette de fournir l'état courant du compte uniquement dans le cas où le fichier fourni par `chemin.compte(nom)` est existant (on utilisera la fonction `file.exists()` pour tester l'existence d'un fichier).

6.5- Complétez à votre guise la gestion des comptes par la création de fonctions supplémentaires.

6.6- **Question optionnelle :** puisque la plupart des utilisations du R se font à l'aide des objets R, adaptez les fonctions précédentes de façon à respecter la philosophie de la programmation R orientée objets. On pourra s'inspirer des travaux pratiques qui suivent.

B- Organisation d'objets graphiques

Si l'on y prête attention, on peut remarquer que l'utilisation des graphiques sous R ne respecte pas vraiment l'esprit objets R dans le sens où un graphique n'est pas considéré, comme la plupart des autres entités R, comme un objet enregistrable (éventuellement modifiable) sur lequel il est possible d'appliquer certaines méthodes. Nous allons tenter de proposer un prototype très basique permettant de dessiner un graphique sur lequel seront représentés des cercles et des rectangles (et donc aussi des carrés). Vous pourrez ultérieurement enrichir cette bibliothèque d'objets graphiques selon votre inspiration. L'objectif est de maintenir une liste d'objets graphiques avec la possibilité de modifier à tout moment l'un de ses éléments.

6.1- Les fonctions R `plot.new()` et `plot.window()` permettent d'initialiser un graphique. Le paramètre `asp` fixé à 1 permet notamment de produire des graphiques respectant les bonnes unités pour les abscisses et les ordonnées. Proposez un objet `Fenêtre` qui offre à l'utilisateur l'option d'enregistrer les dimensions de la fenêtre d'affichage du graphique. L'utilisateur pourra alors appeler la fonction (ou méthode) constructeur `Fenêtre()` (ayant par exemple le même nom que la classe) ayant pour paramètres `x` (c'est-à-dire abscisse du centre), `y` (ordonnée du centre), `largeur` (taille en abscisse), `hauteur` (taille en ordonnée) et éventuellement `log` (transformation logarithmique). Toutes ces quantités seront stockées dans un objet `list` qui sera retourné par la fonction constructeur `Fenêtre()` en ayant au préalable affecté sa classe à `"Fenêtre"`.

6.2- De la même manière, proposez les fonctions constructeurs pour les objets des classes `Cercle` et `Rectangle`. Les champs `x`, `y` représentent les coordonnées du centre de ces objets, `rayon` représente le rayon d'un cercle et `largeur` et `hauteur` représentent les dimensions d'un rectangle.

6.3- Proposez à présent les méthodes d'affichage `plot.Fenêtre()`, `plot.Rectangle()` et `plot.Cercle()` qui pourront s'inspirer des traitements R suivants, permettant l'affichage d'un nouveau graphique contenant un cercle et un carré centrés en l'origine et de diamètre et de côté fixé à 1 :

```
plot.new()
plot.window(xlim=c(-1,1),ylim=c(-1,1),asp=1)
rect(-.5,-.5,.5,.5)
symbols(0,0,circle=.5,inches=FALSE,add=TRUE)
```

6.4- Testez vos développements en exécutant le code :

```
fenêtre <- Fenêtre(0,0,2,2)
cercle <- Cercle(0,0,.5)
rectangle <- Rectangle(0,0,1,1)
plot(fenêtre);plot(cercle);plot(rectangle)
```

Si tout se passe bien, vous devriez voir apparaître une fenêtre graphique représentant un cercle à l'intérieur d'un carré.

6.5- Il nous reste maintenant à développer les méthodes associées à la classe `Graphe` qui contiendra la liste de tous les objets graphiques. Proposez tout d'abord la fonction constructeur `Graphe()` qui initialise un objet à `list(objets=list())` (où `objets` est le champ contenant la liste des objets graphiques), lui affecte la classe `"Graphe"` puis la retourne.

6.6- Proposez une méthode `ajout.Graphe()` qui permet d'ajouter des objets graphiques. N'oubliez pas de fournir une fonction générique `ajout()` afin de lancer toutes les méthodes associées. En utilisant la fonctionnalité de la liste complémentaire de paramètres `...` et la fonction `c()`, faites en sorte que la méthode `ajout.Graphe()` puisse initialiser autant d'objets graphiques que souhaité par l'utilisateur. Proposez la méthode `plot.Graphe` qui permet simplement d'exécuter les méthodes `plot()` pour tous les objets graphiques. L'utilisateur est alors en mesure de saisir les lignes suivantes pour obtenir le même résultat que précédemment :

```
graphe <- Graphe()
graphe <- ajout(graphe,Fenêtre(0,0,2,2),Cercle(0,0,.5),
                Rectangle(0,0,1,1))
plot(graphe)
```

6.7- Pour afficher un graphique, il est donc nécessaire qu'un objet du type `Fenêtre` soit initialisé et placé au moins en première position dans la liste des objets graphiques de la classe `Graphe`. Il est alors peut-être souhaitable de directement l'initialiser à l'intérieur de la fonction constructeur `Graphe()`. Les paramètres de la fonction `Fenêtre()` pourront être directement proposés pour la fonction `Graphe()`. Une autre idée est aussi de proposer à l'utilisateur une liste d'objets graphiques à la création d'un objet de classe `Graphe`. Comme nous l'avons fait pour la méthode `ajout.Graphe()`, on pourra utiliser la liste complémentaire de paramètres `...` que l'on prendra soin de placer en premier paramètre de la fonction `Graphe()` de sorte que l'on pourra obtenir le résultat précédent en seulement deux lignes de commandes :

```
graphe <- Graphe(Cercle(),Rectangle())
plot(graphe)
```

Notez cependant que dans la première ligne ci-dessus, il est aussi supposé que les valeurs par défaut des paramètres des fonctions `Fenêtre()`, `Cercle()` et `Rectangle()` ont été fixées de manière appropriée.

6.8- Pour finaliser cet exercice, on peut aussi, par pur souci d'élégance, proposer la fonction générique `affiche()` (ou tout autre nom de votre choix) qui permettra de lancer les méthodes `plot()` et donc de proposer l'utilisation française ci-dessous :

```
graphe <- Graphe(Cercle(),Rectangle())
affiche(graphe)
```

6.9- Le projet est donc lancé avec la mise en place de ce premier prototype. Il ne vous reste plus qu'à le compléter en le façonnant à votre goût. Si vous manquez d'inspiration, pensez à proposer la gestion de la liste des objets graphiques (par exemple, la suppression et la modification), la gestion des styles d'affichage, l'affichage d'éventuels axes...

C- Création d'une classe `lm2` pour la régression linéaire avec deux régresseurs

L'objectif est ici de reproduire la même démarche que celle utilisée par nos deux compères pour la régression simple. La représentation graphique sera rendue possible grâce à l'excellent *package* `rgl` qui est une interface d'OpenGL pour le système **R**. Compte tenu de la difficulté technique de ce chapitre, nous proposons ici le développement de fonctions (en fait de méthodes). Compte tenu de la technicité de certains points, l'objectif se limitera ici à comprendre toutes les étapes du développement des fonctions suivantes. Cet exercice s'adresse plutôt aux utilisateurs un peu plus avancés.
La fonction suivante fournit un objet de la classe `lm2` tout en héritant de la classe standard `lm`.

```
1 lm2 <- function(...) {
2   obj <- lm(...)
3   if(ncol(model.frame(obj))!=3)
4     stop("Deux variables indépendantes sont requises!")
5   class(obj) <- c("lm2",class(obj)) # ou c("lm2","lm")
6   obj
7 }
```

À titre d'exemple, exécutons les lignes suivantes :

```
> n <- 20
> x1 <- runif(n,-5,5)
> x2 <- runif(n,-50,50)
> y <- 0.3+2*x1+2*x2+rnorm(n,0,20)
> reg2 <- lm2(y~x1+x2)
```

```
> summary(reg2)
Call:
lm(formula = ..1)
Residuals:
    Min      1Q  Median      3Q     Max
-37.846  -5.482   1.677   9.209  38.639
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -8.9753     5.3733  -1.670    0.113
x1            0.4831     1.8083   0.267    0.793
x2            1.7774     0.1676  10.604 6.51e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 20.63 on 17 degrees of freedom
Multiple R-squared:  0.8784,        Adjusted R-squared:  0.8641
F-statistic: 61.42 on 2 and 17 DF,  p-value: 1.663e-08
```

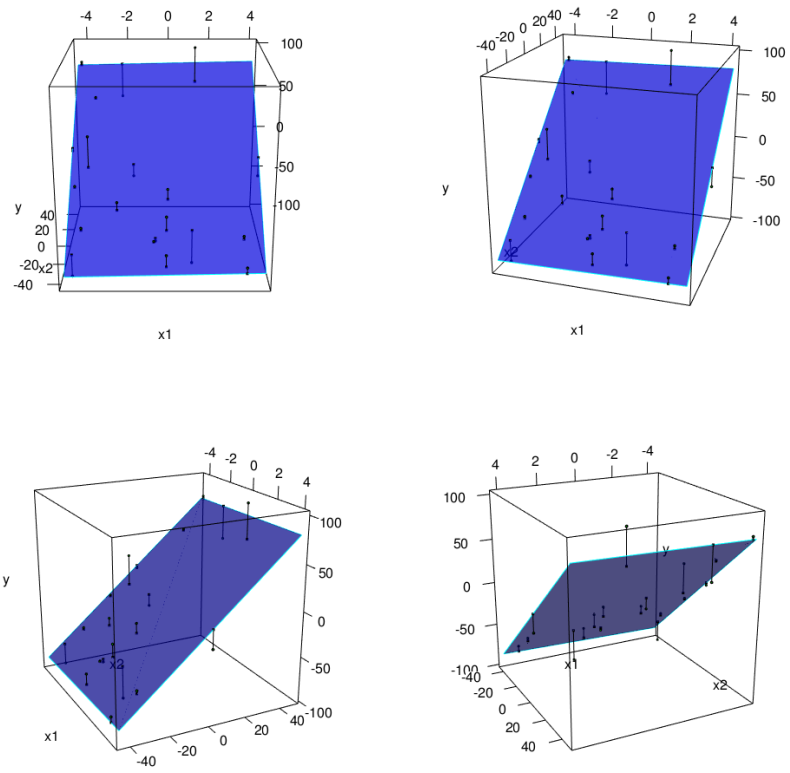
Rien de très surprenant ne se produit puisque la sortie R du résumé est obtenue grâce à la méthode `summary.lm()`. Les deux utilisateurs sont alors intéressés par une représentation tridimensionnelle du nuage de points et du plan de régression obtenu par la méthode des moindres carrés (ordinaires).

```
plot3d.lm2 <- function(obj,radius=1,lines=TRUE,
                           windowRect,...) {
  matreg <- model.frame(obj)
  colnames(matreg) <- c("y","x1","x2")
  predlim <- cbind(c(range(matreg[,2]),
                     rev(range(matreg[,2]))),
  rep(range(matreg[,3]),c(2,2)))
  predlim <- cbind(predlim,apply(predlim,1,
    function(l) sum(c(1,l)*coef(obj))
  ))
  if(missing(windowRect)) windowRect=c(2,2,500,500)
  open3d(windowRect=windowRect,...)
  bg3d(color = "white")
  plot3d(formula(obj),type="n")
 spheres3d(formula(obj),radius=radius,specular="green")
  quads3d(predlim,color="blue",alpha=0.7,shininess=128)
  quads3d(predlim,color="cyan",size=5,front="lines",
                back="lines",lit=F)
  if(lines) {
    matpred <- cbind(matreg[2:3],
                model.matrix(obj)%*%coef(obj))
    points3d(matpred)
    colnames(matpred) <- c("x1","x2","y")
    matlines <- rbind(matreg[,c(2:3,1)],matpred)
```

```
25      nr <- nrow(matreg)
26      matlines <- matlines[rep(1:nr,rep(2,nr))+c(0,nr),]
27      segments3d(matlines)
28    }
29 }
```

Voici une application directe de cette méthode avec quatre illustrations graphiques pour quatre angles de vue différents.

```
> require("rgl")
> plot3d(reg2)
```



Chapitre 7

Maintenance des sessions

Pré-requis et objectif

- Lecture des chapitres précédents.
- Ce chapitre décrit les diverses procédures à mettre en œuvre pour gérer vos sessions sous R. Il faut s'imposer une discipline assez rigoureuse et adopter une méthodologie spécifique au logiciel R pour **être sûr de sauvegarder efficacement son travail**. Les commandes permettant de sauvegarder votre travail, que ce soient les **objets** créés, les **instructions** tapées, les **graphiques** effectués, vous sont présentées. Par ailleurs, nous présentons également quelques autres commandes utiles et nous offrons une brève introduction à la création de *packages*.

SECTION 7.1

Les commandes R, les objets et leur stockage

• Stockage d'objets

Les commandes élémentaires consistent soit en des expressions, soit en des affectations obtenues au moyen de la flèche `<-` ou `->`. Si une expression est tapée, elle est évaluée, le résultat est affiché puis perdu. Une affectation évalue aussi une expression, mais n'affiche pas forcément le résultat. Ce résultat est alors stocké dans un objet.

```
> 2*9 # Le résultat est affiché puis perdu.
[1] 18
> Mon.Poids <- 75 ; Ma.Taille <- 1.90 # Ces deux résultats sont
                                      # stockés.On peut donc
                                      # les réutiliser.
> Mon.IMC <- Mon.Poids/Ma.Taille^2
```

```
> Mon.IMC
[1] 20.77562
```

• Lister des objets

Lorsque l'on a créé des objets R, il est possible d'en obtenir la liste en utilisant la fonction `ls()` ou la fonction `objects()` qui en est un synonyme.

```
> ls()
 [1] "A"          "B"          "cl"         "combnRC"
 [5] "combnRCF"   "corr"       "e"          "inv"
 [9] "lm2"        "Ma.Taille"  "Mon.IMC"    "Mon.Poids"
[13] "myfunc"     "n"          "out"        "plot3d.lm2"
[17] "pmvtRCR"    "reg2"       "x1"         "x2"
> objects()
 [1] "A"          "B"          "cl"         "combnRC"
 [5] "combnRCF"   "corr"       "e"          "inv"
 [9] "lm2"        "Ma.Taille"  "Mon.IMC"    "Mon.Poids"
[13] "myfunc"     "n"          "out"        "plot3d.lm2"
[17] "pmvtRCR"    "reg2"       "x1"         "x2"
```

• Supprimer des objets

Pour effacer des objets, on utilise la fonction `rm()`.

```
> rm(Ma.Taille) # Efface l'objet Ma.Taille.
> ls()
 [1] "A"          "B"          "cl"         "combnRC"
 [5] "combnRCF"   "corr"       "e"          "inv"
 [9] "lm2"        "Mon.IMC"    "Mon.Poids"  "myfunc"
[13] "n"          "out"        "plot3d.lm2" "pmvtRCR"
[17] "reg2"       "x1"         "x2"
> rm(list=ls()) # Efface tous les objets de l'environnement de
                # travail courant.
> ls()
character(0)
```

Expert

Il est possible d'utiliser des expressions régulières pour supprimer uniquement les objets dont le nom vérifie un certain critère. Par exemple, l'instruction suivante ne supprimera que les objets dont le nom est `a?b`, où *?* représente un seul caractère :
`rm(list=ls(pattern=glob2rx("a?b")))`
Nous ne rentrerons pas plus dans les détails et conseillons au lecteur intéressé de consulter l'aide en ligne de la fonction `glob2rx()`.

SECTION 7.2

Environnement de travail : les fichiers d'extension .RData

Lorsque l'on travaille avec le logiciel R, on est amené à créer un certain nombre d'objets : vecteurs, matrices, fonctions, etc. Ces objets sont physiquement enregistrés dans un fichier présent sur le disque dur appelé **fichier d'environnement de travail** (ou *workspace* en anglais) dont l'extension du nom est imposée : `.RData` (ou anciennement `.rda`). Le nom par défaut est juste `.RData` et c'est ce fichier qui est créé lorsque l'on quitte R avec l'instruction `q("yes")`.

Il est possible (et fortement souhaitable) de créer plusieurs (au moyen de la fonction `save.image()`) fichiers d'extension `.RData` : un pour chaque projet sur lequel on doit travailler. Il faut alors créer ces fichiers d'extension `.RData` dans des dossiers appropriés distincts. Par exemple, supposons que l'on travaille sur deux projets statistiques différents : l'un en relation avec des automobiles et l'autre en relation avec le climat, on pourra alors créer un dossier nommé Automobile contenant un fichier auto.RData et un autre dossier nommé Climat contenant un fichier nommé climat.RData (qui contiendront les objets R correspondant à chacune des deux études).

La fonction `save.image()` permet d'enregistrer un fichier d'environnement de travail et il faut utiliser la fonction `load()` pour en charger un existant. Sous l'environnement Microsoft Windows, il est possible d'accéder à ces fichiers d'extension `.RData` depuis le menu suivant (pour les sauvegarder) : `Fichier/Sauver l'environnement de travail...` et depuis cet autre menu : `Fichier/Charger l'environnement de travail...` (pour les charger en mémoire).

Mac

Sous MacOS, il est possible d'accéder à ces fichiers d'extension `.RData` depuis le menu `Espace de travail/Charger fichier d'espace de travail` (pour les sauvegarder) et `Espace de travail/Enregistrer le fichier d'espace de travail` (pour les charger en mémoire). Le menu `Espace de travail` permet également d'explorer le contenu de l'espace de travail (ouvrir une fenêtre listant l'ensemble des objets, leur type et leur dimension) et d'en éditer les objets.

Notez que la fonction `save()` permet de n'enregistrer que les objets que vous aurez choisis dans l'environnement de travail.

Remarque

L'environnement de travail par défaut fourni par R se trouve dans le dossier dont le chemin est renvoyé par l'instruction `getwd()` tapée dès que vous avez lancé R.

Il est utile de noter qu'il est possible, au moyen de la commande `getwd()`, de connaître le répertoire courant. La commande `setwd()` permet, quant à elle, de changer le répertoire courant.

Prise en main

Lancez R puis tapez :

```
x <- 3 # Création du premier x, regardez ce que contient x.
x <- 4 # Création du second x écrasant la valeur du premier x.
```

Maintenant, à l'extérieur de R, créez dans un même répertoire deux sous-dossiers : l'un nommé Automobile et l'autre nommé Climat. Tapez ensuite dans la fenêtre de commandes de R les instructions suivantes :

```
rm(list=ls()) # On commence par effacer tous les objets
         # présents dans l'environnement de travail courant.
ls()          # Renvoie character(0) indiquant bien qu'il
              # ne reste plus d'objets.
x <- c("FIAT","VOLVO","RENAULT","PEUGEOT") # Créez le premier x.
ls()          # Renvoie x.
setwd("/chemin/vers/Automobile") # On se place dans le
                                 # dossier Automobile.
save.image("auto.RData") # Créez le fichier auto.RData dans
                         # le dossier Automobile.
```

Vous venez ici de créer un objet nommé `x`, contenant des noms de constructeurs automobiles. Cet objet est enregistré (sous forme binaire) dans l'environnement de travail auto.RData dans le dossier Automobile.

Tapez maintenant les instructions suivantes :

```
# Créons le deuxième x:
x <- c("tempete","ouragan","tornade","typhon")
setwd("/chemin/vers/Climat") # On se place dans le
                             # dossier Climat.
save.image("evtclim.RData")  # Créez le fichier evtclim.RData
                             # dans le dossier Climat.
```

Vous venez de créer un objet nommé `x` contenant des noms d'événements climatiques. Notez qu'il y a eu écrasement des valeurs du premier `x` dans l'environnement de travail courant. Ce nouvel objet `x` est enregistré dans l'environnement de travail evtclim.RData dans le dossier Climat.
Quittez maintenant le logiciel R grâce à la fonction `q("yes")`. Lancez de nouveau R puis tapez les instructions suivantes :

```
ls()        # Renvoie x, regardez ce que contient x.
load(file.choose()) # Ouvrez le fichier auto.RData dans
                    # le dossier Automobile.
ls()        # Renvoie x, regardez ce que contient x.

load(file.choose()) # Ouvrez le fichier evtclim.RData dans
                    # le dossier Climat.
ls()        # Renvoie x, regardez ce que contient x.
q("no")     # Ferme R.
```

On voit bien ici l'intérêt de disposer de plusieurs espaces de travail, permettant de conserver plusieurs objets portant le même nom, mais ne contenant pas les mêmes informations. Sans cette possibilité, la création du deuxième `x` écraserait le premier !

Astuce

Lorsque l'on quitte une session R par la commande `q()` (ou bien en cliquant sur la croix en haut à droite de la fenêtre de R pour les utilisateurs Windows, ou sur le bouton rouge en haut à gauche pour les utilisateurs Macintosh), la question suivante nous est posée :

Sauver une image de la session?

Si l'on répond Oui (ou `y` pour *yes* sous Linux), un fichier d'environnement de travail nommé .RData (contenant les objets créés durant la session courante) et un fichier de l'historique des commandes nommé `.Rhistory` (présenté dans la prochaine section) sont enregistrés dans le répertoire courant.

Remarque

La fonction `attach()` joue un rôle sensiblement similaire à la fonction `load()`. Nous verrons un peu plus loin en quoi ces deux fonctions diffèrent.

SECTION 7.3

Historique des commandes : les fichiers d'extension .Rhistory

R fournit un mécanisme pour rappeler et réexécuter les commandes précédemment tapées. Les flèches verticales `haut` et `bas` du clavier peuvent être utilisées pour parcourir en arrière et en avant un historique des commandes tapées. Une fois qu'une commande est localisée de cette façon, le curseur peut être déplacé à l'intérieur de la commande en utilisant les flèches horizontales du clavier, et des caractères peuvent être effacés avec la touche `SUPPR` (ou `DEL`), ajoutés ou modifiés avec d'autres touches.

De la même façon que l'on peut sauvegarder nos objets dans des environnements de travail dédiés à l'aide de la commande `save.image()`, il est aussi possible de garder la trace de toutes les commandes tapées au clavier. La succession de ces commandes est sauvée dans un fichier dont l'extension du nom est imposée par le logiciel : `.Rhistory` (anciennement `.rhi`).

Là encore, il est bon de sauvegarder un fichier d'extension `.Rhistory` pour chaque projet sur lequel on travaille. Ces commandes pourront alors de nouveau être rendues disponibles de façon interactive depuis la ligne de commande de R en utilisant les flèches du clavier.

Pour sauvegarder l'historique des commandes de la session en cours, il faut utiliser la commande `savehistory()`. Pour recharger l'historique des commandes d'une session précédente, il faut utiliser la commande `loadhistory()`. Notons que sous Microsoft Windows, on peut effectuer les mêmes opérations depuis le menu `File/Save History...` et `File/Load History...`

Mac

Les utilisateurs Mac pourront compter sur `R.app` qui offre une barre latérale permettant de visionner, de naviguer dans et de manipuler l'historique. Elle est activée en cliquant sur l'icône `Afficher/Cacher l'historique` de la console R.

Astuce

La commande `history()` affiche dans une nouvelle fenêtre la liste de l'historique de toutes les commandes passées de la session en cours.

Prise en main

Lancez **R** et tapez les instructions suivantes :

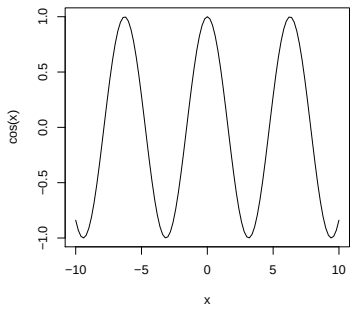
```
Mvoit <- "Ma voiture"
# À l'aide de la flèche vers le haut, modifiez
# la commande précédente en:
Tvoit <- "Ta voiture"
# Enregistrez un fichier nommé auto.Rhistory
# dans le dossier Automobile.
savehistory("/chemin/vers/Automobile/auto.Rhistory")
q("no") # Quittez R.
# Rouvrez R.
# Constatez que la flèche du haut ne vous permet
# pas de retrouver les deux commandes précédentes.
# Ouvrez alors le fichier auto.Rhistory présent
# dans le dossier AUTOMOBILE.
loadhistory(file.choose())
# Maintenant la flèche du haut vous permet de nouveau
# de retrouver les commandes déjà mentionnées.
q("no")
```

SECTION 7.4

Sauvegarder des graphiques

Une autre chose que l'on peut vouloir sauvegarder sont les différents graphiques que l'on produit sous **R**. Les instructions de cette section ont déjà été présentées au chapitre 5, mais nous les rappelons ici pour mémoire. Par exemple, pour sauvegarder le graphique suivant :

```
> curve(cos(x),xlim=c(-10,10))
```



il suffit de taper la commande :

```
dev.print(png,file="mongraphe.png",width=480,height=480)
```

Une autre possibilité pour enregistrer un graphique est de commencer par rediriger la fenêtre graphique dans un fichier puis d'exécuter la commande générant le graphique.

```
png(file="monographebis.png",width=480,height=480)
curve(cos(x),xlim=c(-10,10))
dev.off()
```

Attention

Ne pas oublier d'utiliser à la fin de la procédure la fonction `dev.off()` qui permet de fermer le *device* (« périphérique ») et d'écrire le graphique dans le fichier. Autrement, le fichier créé restera vide.

Plusieurs autres commandes sont disponibles pour enregistrer des images dans différents formats. Voir par exemple les fonctions `jpeg()`, `png()`, `bitmap()`, `postscript()`, `pdf()` ...

On peut aussi, sous Microsoft Windows, utiliser le menu `Fichier/Enregistrer sous...` ou encore faire un copier-coller du graphique dans un autre logiciel. Il faut au préalable avoir cliqué sur la fenêtre graphique pour la rendre active.

Mac

On peut utiliser le menu `Ficher/enregistrer` ou `Fichier/sauver`. Le graphique enregistré/copié est au format PDF. Notons que les fenêtres graphiques disposent d'un « historique ». On peut parcourir l'historique des différents graphiques générés en utilisant la combinaison de touches `COMMAND` + les flèches gauche et droite du clavier.

SECTION 7.5

La gestion des *packages*

Un *package* est un ensemble de données et de fonctions regroupées autour d'un même thème.
Lorsqu'on installe le logiciel R, celui-ci vient avec certaines fonctionnalités de base. Mais il est possible d'étendre les fonctionnalités du logiciel en lui ajoutant des librairies que l'on appelle des *packages*. Il faut d'abord commencer par installer le *package* désiré sur le disque dur de l'ordinateur puis on le chargera (on

l'activera) dans la mémoire de R seulement lorsque l'on en aura besoin (voir l'annexe A pour plus de détails).

Au préalable, vous pouvez utiliser la fonction `search()` donnant la liste des *databases* (ensemble de librairies ou *packages* R) attachées au système, c'est-à-dire auxquelles on peut avoir accès. La fonction `searchpaths()` renvoie la même chose, mais en y ajoutant le chemin d'accès au fichier correspondant.

Nous vous rappelons l'existence de la fonction `library()` qui fournit la liste de tous les *packages* disponibles dans la bibliothèque `C:/PROGRAM FILES/R/R-3.1.0/library`.

Prise en main

```
search()  # Renvoie la liste des databases attachées
          # au système.
library() # Renvoie la liste des packages enregistrés
          # sur le disque.
```

Installez le *package* `R2HTML` et tapez les instructions suivantes :

```
library() # Le package R2HTML est bien présent sur le disque.
search()  # Le package R2HTML n'est pas chargé en mémoire.
require("R2HTML")  # Permet d'activer le package R2HTML.
search()  # Le package R2HTML est maintenant
          # chargé en mémoire.
```

Remarque

Les instructions `require("package")` et `library("package")` ont un comportement similaire. La fonction `require()` est prévue pour être utilisée à l'intérieur d'autres fonctions ; elle renvoie `FALSE` et affiche un avertissement (plutôt qu'une erreur comme la fonction `library()`) si le *package* n'existe pas.

SECTION 7.6

La gestion des chemins d'accès aux objets R

À la section précédente, nous avons vu l'utilité de la fonction `search()` présentant la liste des *databases* (qui sont numérotées) attachées au système.

Nous avons aussi vu comment rajouter un *package* au moyen de la fonction `require()`. Il est possible d'attacher une *database* grâce à la fonction `attach()` et de détacher une *database* au moyen de la fonction `detach()`. Nous illustrerons leur fonctionnement plus en détail dans la partie des travaux pratiques de ce chapitre.

Astuce

Supposons que l'on a créé un *data.frame* (tableau individus × variables) nommé `donnees`. Alors `attach(donnees)` permet d'attacher le *data.frame* `donnees`, ce qui permettra d'avoir accès aux variables du *data.frame* `donnees` directement en tapant leurs noms dans la console.

Les quelques instructions suivantes permettent de s'interroger sur le fonctionnement de la fonction `attach()`.

```
# Lancez R.
attach(file.choose()) # Ouvrez le fichier auto.RData
                      # dans le dossier Automobile.
ls() # x ne s'affiche pas.
x    # Étrange, car x affiche son contenu alors que ls()
     # ne le mentionne pas.
rm(x)
Warning message:
In rm(x) : variable "x" introuvable
x    # Et pourtant x est bien là!
```

La commande `ls(pos=`n`)`, où l'on remplace n par un nombre, renvoie la liste des objets présents dans la *database* placée en n-ième position de la liste fournie par `search()`.
Ainsi, `ls(pos=2)` renvoie les objets pour le module placé en deuxième position, `ls(pos=3)` ceux placés en troisième position, et ainsi de suite.
Notons que la position 1 est réservée. Ainsi `ls()` est équivalent à `ls(pos=1)` et donne la liste de tous les objets de l'environnement de travail courant (*workspace*).

```
search()
ls(pos=2) # Affiche les objets contenus dans la
          # database auto.RData.
```

Prise en main

```
require("datasets") # Charge plusieurs jeux de données
                    # en mémoire.
warpbreaks
mesdonnees <- warpbreaks
fix(mesdonnees)     # Regardez toutes les données et les noms
                    # des variables.
breaks              # Renvoie un message d'erreur, car cet
                    # objet n'est pas défini.
search()            # Affiche l'ensemble des databases
                    # attachées au système.
searchpaths()       # Idem avec le chemin complet.
position <- match("package:datasets",search())
                    # Récupère la position de datasets dans
                    # la liste fournie par search().
ls(pos=position)    # Donne la liste de tous les jeux de
                    # données dans datasets.
data()              # Donne une description de ces
                    # jeux de données.
attach(mesdonnees)
search()
searchpaths()
ls(pos=2)
breaks # Le message d'erreur a disparu.
```

Maintenant, on peut accéder directement aux colonnes de `mesdonnees` : `breaks`, `wool` et `tension`.

SECTION 7.7

† Autres commandes utiles

Nous présentons ici quelques commandes intéressantes pour gérer votre travail :

- sous Microsoft Windows, le menu `File/Save to file...` permet de sauver dans un fichier texte (nommé lastsave.txt par défaut et créé dans le dossier courant) tout ce qui est affiché dans la console (y compris les éventuels messages d'erreurs). La taille de ce contenu est limitée par certains paramètres pouvant être modifiés dans le menu `Édition/Préférences...`
- la fonction `sink(file="sortie.txt")` redirige toutes les sorties R normalement affichées dans la console vers le fichier sortie.txt. Pour arrêter cette fonctionnalité, il faut taper `sink()` dans la console ;

– le menu `Fichier/Sourcer du code R...` permet de transférer une suite d'instructions R (contenues dans un fichier) directement dans la console. Cette commande présente l'avantage additionnel de vérifier la syntaxe du code R contenu dans le fichier avant de le transférer. Il est équivalent de taper `source(file.choose())` dans la console ;
– le logiciel R contient de nombreuses fonctions de manipulation de fichiers et de répertoires présents sur le disque :
`file.create()`, `file.exists()`, `file.remove()`, `file.rename()`, `file.append()`, `file.copy()`, `file.symlink()`, `dir.create()`, `Sys.chmod()`, `Sys.umask()`, `file.info()`, `file.access()`, `file.path()`, `file.show()`, `list.files()`, `unlink()`, `basename()`, `path.expand()`.
Par exemple, la commande `list.files()` renvoie un vecteur de chaînes de caractères des noms des fichiers présents dans le répertoire spécifié. La commande `file.exists()` permet de savoir si un fichier existe dans un répertoire donné. La consultation du fichier d'aide de toutes ces fonctions permettra de s'informer du détail de leur fonctionnement.

SECTION 7.8

† La gestion de la mémoire

Dans cette section, nous allons nous intéresser à la gestion de la mémoire par l'ordinateur en général et par R en particulier. Nous allons essayer de comprendre pourquoi surviennent des messages d'erreur renvoyés par R comme :

```
ne peut allouer un vecteur de taille xxx
```

Nous verrons également comment il est possible de travailler avec des vecteurs ou des matrices de grande dimension.

Avant de fournir des indications sur la façon de gérer des problèmes de mémoire en R, nous devons commencer par donner quelques explications succinctes sur le fonctionnement interne d'un ordinateur. Lorsqu'un programme R est exécuté, les composants internes de l'ordinateur qui sont sollicités sont :
– le disque dur (qui contient les fichiers de code ou de données) ;
– le processeur (qui exécute les calculs ; il existe des processeurs dits 32 *bits* et des processeurs dits 64 *bits*) ;
– et la mémoire vive, encore appelée RAM (pour *Random Access Memory*), qui contient de façon temporaire les données qui seront utilisées par le processeur pour effectuer les calculs.

Dans ce qui suit, on va surtout s'intéresser à la RAM, même si on parlera un peu du processeur. Le principal intérêt de la RAM par rapport au disque dur est qu'elle est d'un accès très rapide. Dans un ordinateur, le processeur accède aux instructions du programme à exécuter ainsi qu'aux données nécessaires à son exécution depuis la mémoire vive.

7.8.1 Organisation de la mémoire vive

La mémoire vive est organisée comme une succession ordonnée de cases, chacune de ces cases pouvant contenir un chiffre dit binaire : 0 ou 1. L'information contenue dans une case, plus petite quantité d'information que peut contenir la mémoire, est appelée un *bit* (pour *binary digit*). À ce stade, il convient de noter que l'information est en fait souvent organisée par « paquets » (ou blocs) de 8 cases. Dans cette optique, une autre unité de mesure que le *bit* a été introduite ; il s'agit de l'octet (*byte* en anglais), qui vaut 8 *bits*.

Attention

1 octet = 8 *bits*.

Notons également que chacun de ces blocs-mémoire est numéroté ; le numéro du bloc (de 8 cases) est appelé l'*adresse mémoire* (du bloc en question). Une adresse mémoire est donc un identifiant qui désigne une zone particulière de la mémoire physique où des données (ou bien des instructions à exécuter) peuvent être lues et stockées. Cet identifiant est usuellement un nombre entier, souvent exprimé en codage hexadécimal (base $b = 16$, voir section 3.9).

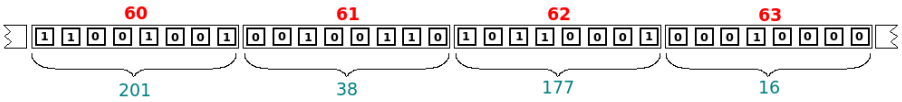


FIGURE 7.1 – Illustration du stockage de valeurs dans la mémoire. Chacune des petites cases contient un chiffre binaire (0 ou 1). Chaque nombre en bleu-vert donne la valeur en écriture décimale du même nombre exprimé en écriture binaire présent dans le bloc situé au-dessus. Chaque nombre en rouge donne l'adresse (exprimée ici dans une base décimale) du bloc-mémoire de 8 cases situé au-dessous. Notons que l'on aurait pu écrire les adresses mémoire dans une base hexadécimale ($b = 16$), ce qui aurait donné : 3C, 3D, 3E et 3F.

7.8.2 Accéder à la mémoire

Remarque

Chaque processus tournant sur l'ordinateur n'a en général pas un accès direct à la mémoire vive, mais plutôt accès à ce que l'on appelle la *mémoire virtuelle*. Les adresses mémoire utilisées par R sont ainsi des adresses de la mémoire virtuelle, adresses qui seront ensuite mises en correspondance (par le système d'exploitation) avec les véritables adresses de la mémoire vive. Nous ne faisons pas la distinction entre ces deux types de mémoire.

Pour accéder à une certaine zone de la mémoire, R utilise (de façon transparente et cachée pour l'utilisateur) ce que l'on appelle un *pointeur* (quantité qui « pointe » vers la zone de mémoire désirée). Un pointeur est donc une variable qui contient une adresse mémoire. À l'adresse contenue dans un pointeur donné, on pourra par exemple trouver une donnée. Notons que chaque donnée possède un certain type, tels que *integer*, *double*, etc. (voir chapitre 1). Notons aussi qu'un *integer* est codé sur 4 octets, un *double* sur 8 octets, un *character* sur 1 octet, un *logical* sur 4 octets, un *complex* sur 16 octets, pour ne citer que les types de variables les plus courants. Ceci est vrai sur un processeur 32 *bits* aussi bien que sur un processeur 64 *bits*. Examinons maintenant les instructions R suivantes :

```
> x <- 3L # création de la valeur 3, de type integer,
> # ou de façon équivalente:
> x <- as.integer(3)
```

D'après ce qui vient d'être expliqué, on peut supposer que s'opère simultanément la réservation (on parle aussi d'allocation) d'un espace mémoire de 32 cases contiguës (4 octets, de 8 *bits* chaque) et la création d'un pointeur qui contiendra l'adresse de (la première de) ces cases. Le pointeur en question doit en fait non seulement contenir l'adresse de la variable `x`, mais également son type pour savoir sur combien de cases cette variable est stockée. Pour cette raison, les pointeurs sont dit « typés ». Ainsi, quand on incrémente un pointeur typé (i.e. que l'on cherche à augmenter d'une unité l'adresse qu'il contient), il n'est en fait pas forcément incrémenté de un, mais de la taille du type pointé.

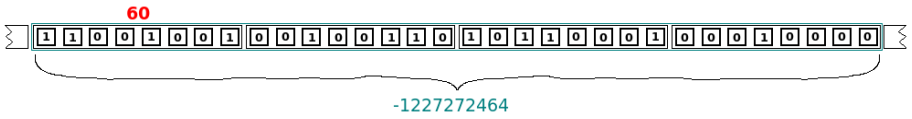


FIGURE 7.2 – Illustration du stockage par R d'un *integer* (signé) dans la mémoire. Chacune des petites cases contient un chiffre binaire (0 ou 1). Le nombre en vert donne la valeur en écriture décimale du même nombre (entier) exprimé en écriture binaire présent dans les quatre blocs situés au-dessus. Le nombre en rouge donne l'adresse (exprimée ici dans une base décimale) du premier bloc-mémoire de 8 cases situé au-dessous. Notons qu'ici un nombre est stocké sur 32 cases, et plus seulement sur 8 comme cela était illustré sur la figure 7.1. De plus, la première case permet de spécifier le signe du nombre, négatif ici.

7.8.2.1 Problèmes causés par la gestion mémoire des entiers

Puisqu'un entier (signé) est codé sur 4 octets, à savoir 32 *bits*, l'entier maximal que l'on peut représenter est 2 147 483 647. En effet, si l'on réserve le premier *bit* pour le signe, il ne reste que 31 cases disponibles, soit 2^{31} arrangements (avec répétitions) possibles. En comptant le 0, le plus grand entier disponible est donc $2^{31} - 1$, soit :

```
> as.integer(2^31-1)
[1] 2147483647
> .Machine$integer.max
[1] 2147483647
```

Le résultat ci-dessous n'est donc pas étonnant.

```
> as.integer(2^31)
[1] NA
```

Le chiffre 2^{31} ne pourra donc être manipulé par **R** que codé comme un *double* :

```
> 2^31
[1] 2147483648
> is.double(2^31)
[1] TRUE
```

Attention

Le plus long vecteur que l'on peut allouer dans **R** est de taille $2^{31} - 1 \sim 2 \cdot 10^9$, que ce soit sur un processeur 32 *bits* ou 64 *bits*. Ceci se comprend aisément au vu des résultats précédents puisque cela correspond au plus grand entier que **R** peut définir, et que la longueur d'un vecteur (nombre de ses éléments) est stockée comme un entier (signé).

Au passage, cette connaissance plus fine du comportement de **R** nous permet par exemple de bien comprendre la sortie ci-dessous.

```
> 46360*46360 # 46360 est stocké comme un double.
[1] 2149249600
> 46360L*46360L # 46360L est stocké comme un integer.
[1] NA
> sum(1:304) # 1:304 est stocké comme un integer.
[1] 46360
> sum(1:304)* sum(1:304)
[1] NA
Message d'avis :
In sum(1:304) * sum(1:304) :
  NA produit par débordement d'entier par le haut
> 46360^2 # Le résultat est stocké comme un double.
[1] 2149249600
> sum(1:304)^2 # Le résultat est stocké comme un double.
[1] 2149249600
```

Le *Message d'avis* ci-dessus vient du fait que `sum(1:304)` est un entier. Notez que la fonction puissance (`^`) transforme ses arguments en réels et renvoie un nombre réel.

7.8.2.2 Allocation consécutive de la mémoire

Il faut en fait savoir que la plus petite portion de mémoire virtuelle qui peut être allouée (réservée) par R est de 8 octets (= 64 *bits*). La mémoire est donc allouée dans R par paquets de 8 octets **consécutifs** (que ce soit sur les processeurs 32 ou 64 *bits*). Ainsi, dans l'instruction `x <- 3L`, c'est en fait 64 cases (d'1 *bit*) qui sont réservées, desquelles seules les 32 premières sont utilisées pour stocker la valeur entière +3. Les 64 cases seraient par exemple toutes utilisées pour allouer un *double* par l'instruction :

```
> x <- 3.0
```

Remarque

Les 64 cases sont réparties en 1 case pour le signe, 11 cases pour l'exposant et 52 cases pour la mantisse, dans la réprésentation du nombre en virgule flottante (voir la section 3.9.2).

Nous avons intégré au *package* associé à ce livre deux fonctions `getaddr()` et `writeaddr()` permettant respectivement de récupérer l'adresse mémoire d'une variable contenant un nombre, et d'écrire une valeur à une adresse mémoire. Notez que sur un système d'exploitation 64 *bits*, une adresse mémoire peut très bien être stockée sur 64 *bits*. Et puisque les entiers dans R ne sont stockés que sur 32 *bits*, récupérer cette adresse sous la forme d'un (seul) entier dans R pourrait poser problème.

```
> x <- c(8L,9L)
> x
[1] 8 9
> addr <- getaddr(x) # Récupère l'adresse de la première case du
                     # bloc de 64 cases où est stockée x.
> addr
<vector: 0x5fd0a80>
> addr$zero          # Adresse exprimée en valeur entière,
                     # stockée sur deux entiers.
[1] 100469376         0
> writeaddr(addr,6L) # Écrit l'entier 6 à cette adresse.
> x
[1] 6 9
> writeaddr(addr+4L,7L) # Un entier est codé sur 4 octets, donc
                        # on incrémente l'adresse de 4 pour
                        # passer à x[2].
> x
[1] 6 7
```

Voyons ce qu'il se passe avec un vecteur de *doubles*.

```
> x <- c(12.8,4.5)
> x
```

```
[1] 12.8  4.5
> addr <- getaddr(x)      # Récupère l'adresse de la première case
                          # du bloc de 128 cases où est stockée
                          # x.
> writeaddr(addr,6.2)
> x
[1] 6.2 4.5
> writeaddr(addr+8L,7.1) # Un double est codé sur 8 octets.
> x
[1] 6.2 7.1
```

Expert

R ne peut accéder qu'aux cases mémoires qu'il a lui-même allouées, et les autres logiciels ou programmes extérieurs à R ne peuvent pas accéder aux zones mémoires réservées par R. Et c'est bien heureux ! Car sinon, les données de nos calculs pourraient être altérées par des programmes externes à R. Même une autre session de R ne peut accéder aux zones mémoires réservées par la première. Ainsi, tapons dans une première session de R :

```
> require("LeLogicielR")
> x <- 1L
> (addr <- getaddr(x)) ; save(addr,file="addr.RData")
<vector: 0x4972c90>
```

qui est l'adresse (hexadécimale) du bloc-mémoire contenant la valeur entière 1. Puis tapons dans une deuxième session de R :

```
> require("LeLogicielR")
> load("addr.RData")
> addr
<vector: 0x4972c90>
> writeaddr(addr,7L)
```

Alors (si la deuxième session de R ne plante pas !) on pourra vérifier que la valeur de x dans la première session n'a pas été modifiée. Revenons à la première session de R et tapons :

```
> x # On constate que la tentative de modifier la valeur de x
    # dans une autre session de R est vouée à l'échec
[1] 1
> # Tentons cette modification dans la même session:
> writeaddr(addr,7L)
> # Là ça marche!
> x
[1] 7
```

7.8.3 Taille des objets dans R

Une fonction R, utile par la suite, permet de renseigner sur la taille d'un objet créé dans R. Il s'agit de la fonction `object.size()`. D'après ce qui précède, on s'attend à ce que le résultat de l'instruction `object.size(3L)` renvoie `8 bytes`, mais il n'en est rien.

```
> object.size(3L) # (Sur un processeur 64 bits).
48 bytes
```

En fait, chaque objet R contient (même lors de la déclaration d'un simple entier par `x <- integer(3)` une entête (*header* en anglais) qui occupe un peu d'espace dans la mémoire vive : 24 octets sur un processeur 32 *bits* et 40 octets sur un processeur 64 *bits*. Cette entête sert à sauvegarder des informations sur l'objet créé : son type (*integer*, *double*, *complex*, etc.), sa longueur, etc ..

Astuce

Pour connaître le processeur sur lequel R est en train de tourner, utilisez l'instruction :

```
> .Machine$sizeof.pointer
[1] 8
```

La valeur 8 est renvoyée pour un processeur 64 *bits* et la valeur 4 pour un processeur 32 *bits*. Une autre possibilité est d'utiliser l'instruction `sessionInfo()`.

Les valeurs renvoyées par les instructions suivantes sont donc maintenant assez claires :

```
> # Sur processeur 32 bits:
> object.size(3L) - 24
8 bytes
> # Sur processeur 64 bits:
> object.size(3L) - 40
8 bytes
```

Expert

L'allocation de la mémoire dans R se fait de façon différente pour les petits et les gros vecteurs d'entiers. Les petits vecteurs sont définis en 6 classes, dépendamment de leur longueur (inférieure ou égale à 2, 4, 8, 12, 16 ou 32), et sont capables de stocker des données de 8, 16, 32, 48, 64 ou 128 octets respectivement. Comme un *integer* occupe seulement 4 octets, il est possible respectivement de stocker 2, 4, 8, 12, 16 et 32 *integer* dans ces 6 classes. Un vecteur d'entiers de longueur $n > 32$ occupera une place de taille $4n+40$ si n est pair et de $4(n+1)$ si n est impair, à laquelle il faudra ajouter (pour

l'entête) 24 octets sur un processeur 32 *bits*, et 40 octets sur un processeur 64 *bits*. Le code suivant renvoie la taille occupée dans la mémoire par des vecteurs de taille croissante.

```
> N <- 50
> V <- vector(length = 50)
> for (L in 1:N) {
+       z <- sample(N, L, replace = TRUE)
+       V[L] <- object.size(z)
+   }

> V - 24 # Sur processeur 32 bits.
 [1]   8   8  16  16  32  32  32  32  48  48  48  48  64  64
[15]  64  64 128 128 128 128 128 128 128 128 128 128 128 128
[29] 128 128 128 128 136 136 144 144 152 152 160 160 168 168
[43] 176 176 184 184 192 192 200 200

> V - 40 # Sur processeur 64 bits.
 [1]   8   8  16  16  32  32  32  32  48  48  48  48  64  64
[15]  64  64 128 128 128 128 128 128 128 128 128 128 128 128
[29] 128 128 128 128 136 136 144 144 152 152 160 160 168 168
[43] 176 176 184 184 192 192 200 200
```

7.8.4 Quantité totale de mémoire utilisée par R

La totalité de la mémoire virtuelle allouée par R au cours d'une session se répartit en :

- la mémoire utilisée pour stocker les valeurs des objets (leur contenu) ;
- la mémoire utilisée pour stocker les entêtes des différents objets.

Cette information est accessible au moyen de la fonction `gc()`, dans laquelle `Ncells` représente le nombre de cellules utilisées pour le *header*, et `Vcells` le nombre de blocs pour les valeurs.

Prise en main

L'exemple suivant permet d'en illustrer l'utilisation.

```
> rm(list=ls()) # On supprime tous les objets de la session.
```

Taper trois fois `gc()`. On remarque que les valeurs affichées se sont stabilisées. On tape maintenant l'instruction suivante :

```
> x <- as.integer(3)
```

Retaper encore plusieurs fois `gc()` jusqu'à stabilisation des résultats affichés. On peut constater que la valeur de `Vcells` a augmenté d'une unité, correspondant à 8 octets (taille minimale d'un bloc de données). Rappelons qu'un *integer* occupe 4 octets mais sera quand même stocké dans une zone mémoire de 8 octets.

La quantité totale de mémoire disponible pour R dépend de plusieurs facteurs :

- la RAM physiquement présente sur l'ordinateur ;
- la RAM déjà utilisée par le système d'exploitation et par les différents autres programmes en cours d'exécution sur le système (navigateur web par exemple s'il est ouvert) ;
- le type de processeur (32 *bits* ou 64 *bits*), puisque la RAM est limitée à 4 Go (1 Go = 1 024 Ko) pour les processeurs 32 *bits* (en réalité plutôt 3 Go ou 2 Go), alors qu'elle est (vraiment !) beaucoup plus large pour des processeurs 64 *bits*.

Remarque

Sur un processeur 32 *bits*, une adresse est codée sur 32 *bits*, la mémoire adressable est donc limitée à 2^{32} adresses mémoire de blocs (8 cases), soit 2^{32} octets (= 4 Go). Sur un processeur 64 *bits*, une adresse est codée sur 64 *bits*, la mémoire théoriquement adressable est donc « limitée » à 2^{64} octets, un nombre gigantesque (17 179 869 184 Go). En fait, elle est souvent beaucoup plus limitée, de par l'architecture du processeur. Cette information est disponible auprès du constructeur du processeur (sous l'appellation : `'Max Memory Size'`).

Par ailleurs, il faut noter que R alloue de la mémoire pour la création d'objets de grande taille, et vide la mémoire de ces objets (lorsqu'ils ne sont plus utilisés) par un processus appelé collection des déchets (*garbage collection*). On peut forcer cette collection de déchets au moyen de l'appel de la fonction `gc()`.

Attention

Lors de la création d'un objet de grande taille, la mémoire réservée par R doit l'être de façon contiguë (non fragmentée sur plusieurs cases).

Il est donc possible qu'il reste suffisament de mémoire vive (au total) disponible pour R, mais pas de « trou » assez grand pour y faire rentrer les données d'un seul objet de grande taille. En voici une illustration. Prenez garde toutefois que ces commandes pourraient entraîner un ralentissement conséquent de votre système, voire même une sortie brutale de R.

```
> # En premier lieu, nous avons tapé les instructions:
  # rm(list=ls()) ; gc() ; gc() # pour vider la mémoire.
> P <- 14000
> D <- matrix(rep(0, P*P), nrow=P)
Erreur : impossible d'allouer un vecteur de taille 1.5 Go
> # Alors que l'allocation suivante est certainement possible,
  # après avoir tapé gc();gc() # pour vider la mémoire.
> Q <- round(sqrt(P^2/2))
> D1 <- matrix(rep(0, Q*Q), nrow=Q)
> D2 <- matrix(rep(0, Q*Q), nrow=Q)
> # La somme des tailles de D1 et D2 fait approximativement 1.5 Go
> object.size(D1) + object.size(D2)
1567843440 bytes
```

Ainsi, dans l'exemple ci-dessus, il n'a pas été possible de créer un seul objet de taille 1.5 Go, alors qu'il a été possible de créer deux objets de même taille approximativement égale à 0.75 Go. C'est un problème de *fragmentation* de la mémoire.

Astuce

Notons que sur un processeur 64 *bits*, nous n'aurions probablement pas rencontré ce problème, même lors de la création d'un objet de taille approximativement égale à 3 Go :

```
> # En premier lieu nous avons tapé les instructions
  # rm(list=ls()) ; gc() ; gc() # pour vider la mémoire.
> P <- 20000
> D <- matrix(rep(0, P*P), nrow=P)
> object.size(D)
> 3200000200 bytes
```

7.8.5 Quelques recommandations

Une compréhension élémentaire de la gestion de la mémoire dans un ordinateur en général, et par R en particulier, pourra s'avérer très utile afin d'iden-

tifier l'origine de problèmes relatifs à la mémoire. La lecture des pages qui précèdent constitue donc notre première recommandation. Nous donnons ci-dessous quelques conseils supplémentaires.

Vous pourriez par exemple calculer la taille (approximative) d'une matrice avant sa création. Puisqu'un nombre réel occupe 8 octets, alors une matrice de réels de taille $n \times p$ occupera 8 np octets. Si vous devez travailler avec des matrices de très grande taille, l'utilisation d'un processeur 64 *bits* vous permettra d'allouer des blocs de mémoire de grande taille. Comparativement, un processeur 32 *bits* ne permettra généralement pas de dépasser 2 *gigabytes*. Si vous n'êtes pas capable de créer un objet de grande taille, pensez à effacer (au moyen de la fonction `rm()`) les autres gros objets inutiles (la fonction `object.size()` vous renseignera sur la taille de ces objets) et à libérer la mémoire en utilisant la fonction `gc()`. Vous pouvez également envisager de fermer certains logiciels tournant sur votre ordinateur afin de libérer de la mémoire, et en dernier ressort d'acheter de la mémoire physique supplémentaire.

Linux

Le logiciel `ksysguard` permet de visualiser en temps réel l'utilisation de la mémoire consommée par R, ainsi que par les autres processus tournant sur le système.

Une autre option pourrait consister à couper votre matrice en plusieurs sous-matrices et à trouver un moyen d'effectuer votre analyse sur celles-ci avant de combiner les résultats obtenus.

Astuce

Les packages `bigmemory`, `ff` et `RevoScaleR` pourraient vous être utiles.

Sous Windows, les fonctions `memory.size()` et `memory.limit()` permettent d'afficher certaines informations sur l'utilisation de la mémoire. Il peut aussi être utile de consulter l'aide en ligne : `help("Memory-limits")`.

Remarque

Il est possible, dans le futur, que ces problèmes trouvant leur origine dans la conception de R soient résolus. Le lecteur intéressé pourra par exemple consulter le document `http://www.divms.uiowa.edu/~luke/talks/uiowa11.pdf`.

SECTION 7.9

† Utiliser R en mode BATCH

Il est possible de lancer une succession d'instructions R en mode BATCH. Ce mode signifie que R se lance pour exécuter automatiquement une suite d'instructions en tâche de fond, puis lorsqu'il a terminé son travail, il se ferme.

• Ce mode s'obtient en lançant l'instruction suivante dans une fenêtre DOS (ou une fenêtre terminal LINUX ou Mac) :

```
R CMD BATCH monfichier.R sortie.out
```

Le fichier monfichier.R doit contenir la liste d'instructions R à traiter et le fichier sortie.out contiendra les éventuels messages et sorties affichés par R.

Attention

Notez que vous devrez régler la variable d'environnement système PATH afin qu'elle contienne le chemin vers l'exécutable Rgui.exe. Voir l'encadré Attention à la page 253.

• On peut aussi utiliser ce mode lorsque l'on veut lancer des simulations sur une station de travail UNIX/LINUX distante (accessible via la commande ssh). Dans ce cas précis, il faut alors rajouter la commande LINUX nohup.

```
nohup /chemin/vers/executable/R CMD BATCH monfichier.R sortie.out &
```

Linux

Sous Linux, pour trouver le /chemin/vers/executable/ de R, il suffit de taper l'instruction suivante dans un terminal : which R.

Il est aussi possible de créer un script R qui pourra être exécuté sans avoir besoin d'ouvrir R au préalable. Pour cela, téléchargez puis modifiez le fichier http://biostatisticien.eu/springeR/runthis.bat.

Renvoi

Le lecteur intéressé pourra consulter avec profit l'adresse http://cran.r-project.org/contrib/extra/batchfiles.

Linux

Sous Linux, créez un script nommé runthis et rendez-le exécutable (`chmod u+x runthis`). Ce script contiendra les lignes suivantes :

```
#!/bin/bash
  R --vanilla << "EOF" # Dirige les lignes suivantes vers R.
  ################ Mettez tout votre code R ici  ############
  X11(); plot(1:3)
  require("tcltk")
  tkmessageBox(message="hello")
  ############################ fin du code R ################
EOF
```

Astuce

Si vous voulez passer des arguments (en ligne de commandes) à `R CMD BATCH`, vous pouvez employer l'approche suivante. Tout d'abord, créer une fichier `test.R` contenant les lignes suivantes :

```
args <- commandArgs(trailingOnly = FALSE)
print(args)
q("no")
```

Ensuite, tapez dans une fenêtre de terminal :
`R CMD BATCH -q -4 -foo test.R`
Pour finir, tapez :
`cat test.Rout`

Dans le même contexte, veuillez également noter l'existence de l'utilitaire `Rscript` qui permet par exemple de lancer une expression R sans la rentrer dans un fichier.

SECTION 7.10

† Création d'un *package* R simplifié

Un *package* est un moyen commode de regrouper dans une même arborescence des jeux de données, des fonctions et des fichiers d'aide les décrivant. Cette arborescence est stockée dans un fichier d'extension `.zip` (ou `.tar.gz` sous Linux). Cette opération est tout de même délicate sous l'environnement Microsoft Windows, car elle nécessite l'installation préalable de nombreux outils non présents par défaut sur ce système d'exploitation.

Mac

Une documentation spécifique pour les utilisateurs Mac sera disponible sur le site internet associé à ce livre.

Vous devrez donc commencer par installer les trois logiciels suivants :
- la dernière version de Rtools disponible ici : `http://cran.r-project.org/bin/windows/Rtools` ;
- `http://www.biostatisticien.eu/springeR/htmlhelp.exe` ;
- une version complète de Tex Live. Pour cela, téléchargez le fichier `http://mirror.ctan.org/systems/texlive/tlnet/install-tl.zip` et décompressez-le dans un dossier temporaire. Puis double-cliquez sur le fichier install-tl-advance.bat. Une interface graphique d'installation est alors affichée qui vous guidera dans l'installation de Tex Live.

Voici à présent la procédure à suivre pour créer un *package* pour le logiciel R :
- lancez R ;
- importez dans l'espace de travail les jeux de données et les fonctions que vous voulez intégrer à votre *package* ;
- utilisez la fonction `package.skeleton()` afin de créer l'arborescence de votre *package*. Cette fonction comprend un certain nombre de paramètres qu'il est utile de renseigner :
  - `name` : une chaîne de caractères contenant le nom du *package*,
  - `list` : un vecteur de chaînes de caractères, spécifiant les différents objets (données et fonctions) à intégrer au *package*,
  - `path` : une chaîne de caractères contenant le chemin vers un dossier dans lequel l'arborescence de votre *package* sera créée ;
- l'appel de cette fonction permet de créer un dossier (enregistré dans votre répertoire courant) contenant les fichiers et les sous-dossiers constituant votre *package*. Il faut alors modifier certains de ces fichiers comme cela est décrit dans le fichier Read-and-delete-me présent dans le dossier de votre *package* ;
- il vous reste une dernière opération à effectuer qui consiste à construire le fichier d'extension `.zip` qui contiendra une version remaniée par R de votre arborescence. Pour cela, les commandes suivantes devront être exécutées dans une fenêtre de commandes MS-DOS :
  - `R CMD check` *NomPackage*
  - `R CMD INSTALL --build` *NomPackage*

Astuce

Si votre code appelle des fonctions `C/C++` ou `Fortran`, les fichiers contenant le code source de ces fonctions doivent être placés dans un dossier

nommé `src/`, lui-même situé dans le dossier dont le nom est spécifié par l'argument `name` de la fonction `package.skeleton()`.

Un exemple de création de *package* sera présenté dans la section des travaux pratiques en fin de chapitre.

Expert

Les utilisateurs Linux n'ayant pas accès à un système d'exploitation Microsoft Windows mais désirant construire un *package* pour ce système, peuvent utiliser le site internet `http://win-builder.r-project.org/` qui permet de téléverser le fichier source de son *package* (d'extension `.tar.gz`) créé sous Linux. Le *package* d'extension `.zip` fonctionnant sous Windows est alors envoyé à l'adresse email renseignée dans le champ `Maintainer` du fichier DESCRIPTION.

Termes à retenir

`ls(), objects()` : liste les objets disponibles dans l'environnement de travail
`rm()` : efface un objet nommé
`.RData` : extension des fichiers d'environnement de travail
`save.image()` : enregistre dans un fichier (nom.RData) les objets créés
`load()` : charge un fichier .RData contenant des objets créés
`.Rhistory` : extension des fichiers d'historique des commandes
`savehistory()` : sauvegarde l'historique des commandes (.Rhistory)
`loadhistory()` : charge l'historique des commandes
`dev.print()` : permet de sauvegarder un graphique
`search(), searchpaths()` : liste les *databases* attachées au système
`attach()` : attache une *database*
`detach()` : détache une *database*
`require()` : charge un *package* déjà enregistré sur le disque
`sink()` : redirige les sorties de R dans un fichier (.txt)
`source()` : importe une suite d'instructions R présentes dans un fichier vers la console
`package.skeleton()` : crée l'arborescence d'un *package*

Exercices

7.1- Donnez les noms des deux fonctions R permettant d'afficher la liste des objets de votre session.

7.2- Comment effacer l'objet `toto` ?

7.3- Quelle est la commande R permettant de connaître le répertoire courant ?

7.4- Quelle est la commande R permettant de changer le répertoire courant ?

7.5- À quoi sert la fonction `save.image()` ?

7.6- Quelles sont les quatre choses que vous pouvez sauvegarder avec R avant de terminer votre session ?

7.7- Quel est l'intérêt du mécanisme de sauvegarde de l'historique des commandes ? Quelles sont les touches du clavier à utiliser pour en bénéficier ?

7.8- À quoi sert la fonction `history()` ?

7.9- Donnez la liste des instructions R permettant d'obtenir un fichier nommé myplot.png qui contiendra un graphique de la droite $y = x^2$.

7.10- Quel est l'intérêt d'utiliser la fonction `attach()` sur un *data.frame* ?

7.11- Quelle est la fonction R permettant de charger en mémoire un *package* R ?

7.12- À quoi sert la fonction `source()` ?

Fiche de TP

Maintenance et création de *packages*

A- Utilisation des fonctions `attach()` et `detach()`

7.1- Téléchargez le fichier `http://www.biostatisticien.eu/springeR/imcenfant.xls`.

7.2- Affichez les noms des variables présentes dans le *data.frame*.

7.3- Tapez `SEXE`. Que constatez-vous ?

7.4- Tapez `ls()`. Voyez-vous la variable `SEXE` ?

7.5- Utilisez la fonction `attach()` sur votre *data.frame* puis tapez `SEXE`. Que constatez-vous maintenant ?

7.6- Tapez de nouveau `ls()`. Que constatez-vous ?

7.7- Utilisez la fonction `search()` pour voir en quelle position est attaché votre *data.frame*.

7.8- Utilisez le paramètre `pos` de la fonction `ls()` pour voir les objets présents à cette position.

7.9- Détachez votre *data.frame* et vérifiez au moyen de la fonction `search()` que tout s'est bien déroulé. Puis tapez de nouveau `SEXE` pour constater que cet objet a disparu.

7.10- Créez un objet nommé `SEXE` contenant la chaîne de caractères `"Homme"`. Affichez le contenu de cet objet.

7.11- Utilisez la fonction `attach()` sur votre *data.frame* puis tapez `SEXE`. Que constatez-vous maintenant ?

7.12- Pouvez-vous afficher le contenu de l'objet `SEXE` de votre *data.frame* ? De l'objet `poids` ?

7.13- Tapez `ls()`. Que constatez-vous ? Et avec `search()` ?

7.14- Réutilisez le paramètre `pos` de la fonction `ls()` pour vérifier l'existence de l'objet `SEXE` du *data.frame*.

7.15- Utilisez la fonction `get()` et son paramètre `pos` pour afficher le contenu de l'objet `SEXE` de votre *data.frame*. Quelles autres approches pouvez-vous proposer ?

B- Création d'un mini-*package*

• Les objets du *package*

7.1- Lancez R, puis changez le répertoire courant pour vous placer sur le bureau de Windows au moyen de l'instruction `setwd(choose.dir())`.

7.2- Créez les deux fonctions et les deux jeux de données suivants :

```
f <- function(x,y) x+y
g <- function(x,y) x-y
d <- data.frame(a=1,b=2)
e <- rnorm(1000)
```

• L'arborescence du *package*

7.3- Utilisez la fonction `package.skeleton()` pour créer l'arborescence de votre *package*.

```
package.skeleton(name="PetitPkgR",list=c("f","g","d","e"))
```

Un dossier nommé PetitPkgR est alors créé sur le bureau. Celui-ci contient trois dossiers (`data`, `man` et `R`) et deux fichiers (DESCRIPTION et Read-and-delete-me).
Le dossier data contient les fichiers d.RData et e.RData, qui contiennent (dans un format binaire) les jeux de données `d` et `e` précédemment importés depuis la console de R.
Le dossier R contient les fichiers f.R et g.R, qui contiennent le code source des fonctions `f` et `g` définies précédemment.
Le dossier man contient les fichiers d'aide pour tous les objets inclus dans le *package*.

7.4- Modifiez **impérativement** ces fichiers d'aide (fichiers d'extension `.Rd`) même s'ils sont préremplis, en vous aidant de la description du fichier d'aide sur la fonction `mean()` fourni dans le chapitre 4. Les champs à renseigner sont indiqués dans tous les fichiers d'aide par des phrases débutant par le signe %%. Remplacez ces lignes (y compris le signe %%) par les informations appropriées. Il ne faut pas modifier les phrases commençant par un simple signe %. En outre, pour les champs de la forme `keyword ~ kwd1`, il faut absolument remplacer `~ kwd1` par l'un des *keywords* (mots clefs) réservés du langage R dont la liste peut être obtenue au moyen de l'instruction `file.show(file.path(R.home("doc"), "KEYWORDS"))`.

7.5- Vous devez également modifier le fichier DESCRIPTION pour renseigner les champs le nécessitant. Il est par exemple **très important** de renseigner une adresse email valide.

7.6- Vous pouvez ensuite lire puis effacer le fichier Read-and-delete-me.

À ce stade, l'arborescence de votre *package* est créée.

• Création effective du fichier du *package*

7.7- Il vous reste une dernière opération à effectuer qui consiste à construire le fichier d'extension `.zip` qui contiendra une version remaniée par R de votre arborescence. Vous devez tout d'abord commencer par modifier quelques variables d'environnement du système. Pour cela, utilisez la combinaison de touches `WINDOWS+PAUSE` afin d'ouvrir la fenêtre des propriétés du système, puis allez dans les paramètres systèmes avancés. Cliquez ensuite sur `Variables d'environnement` et allez dans la partie `Variables système` pour modifier la variable `PATH`. Rajoutez alors au tout début de cette longue liste de chemins séparés par des points virgules (**en veillant bien à ne rien effacer !**) le chemin vers l'exécutable Rgui.exe et aussi le chemin vers l'exécutable hhc.exe.

7.8- Ensuite, dans une fenêtre de commandes MS-DOS (obtenue via le menu `Démarrer/Exécuter: command`), exécutez les instructions :
- `cd "C:\Documents and Settings\dupont\Bureau"` (permet de vous placer dans le dossier contenant l'arborescence de votre *package*).
- `R CMD check PetitPkgR`
  Assurez-vous qu'aucune erreur ou qu'aucun message d'avertissement ne subsiste ici. Dans le cas contraire, veuillez apporter les modifications suggérées.
- `R CMD build --binary --use-zip PetitPkgR`

S'il n'y a pas eu d'erreurs, le fichier de votre *package* PetitPkgR.zip sera créé.

7.9- Installez-le depuis le menu suivant :
`Packages/Installer le(s)` *`package`*`(s) depuis des fichiers zip...`
Consultez l'aide de votre *package*.

Vous pouvez à présent suivre cette procédure pour créer des *packages* plus élaborés que vous pourrez diffuser.

Seconde partie

Mathématiques et statistiques élémentaires

Chapitre 8

Mathématiques de base : calcul matriciel, intégration, optimisation

Objectif

Ce chapitre décrit les fonctions mathématiques de base. Il donne ensuite quelques calculs usuels appliqués sur des matrices ainsi que les décompositions les plus courantes. Nous présentons aussi quelques fonctions d'intégration et de dérivation numérique, et les principales fonctions d'optimisation.

SECTION 8.1

Les fonctions mathématiques de base

Le tableau suivant fournit une liste quasi exhaustive des fonctions mathématiques les plus classiques.

TABLE 8.1 – Tableau des fonctions mathématiques de base.

Nom R	Description	Exemple	Résultat
`x%%y`	Reste de la division de x par y	`10%%3`	`1`
`ceiling()`	Plus petit entier relatif supérieur ou égal à x	`ceiling(2.3)`	`3`
`floor()`	Plus grand entier relatif inférieur ou égal à x	`floor(2.3)`	`2`
`round()`	Arrondit les valeurs de son premier paramètre d'appel à un certain nombre de décimales spécifiées par le second paramètre d'appel	`round(2.375,2)`	`2.38`
`signif()`	Arrondit les valeurs de son premier paramètre d'appel à un certain nombre de chiffres significatifs	`signif(2.375,2)`	`2.4`
`trunc()`	Partie entière de x obtenue en supprimant tous les chiffres après la virgule	`trunc(1.37)`	`1`
`sign()`	Signe ±1	`sign(-2)`	`-1`
`abs()`	Valeur absolue $\|x\|$	`abs(-2)`	`2`
`exp()`	Exponentielle e^x	`exp(0)`	`1`
`log()`	Logarithme népérien	`log(1)`	`0`
`sqrt()`	Racine carrée $\sqrt{x}$	`sqrt(4)`	`2`
`range()`	Étendue	`range(2,5,1)`	`1 5`
`max()`	Maximum	`max(2,3)`	`3`
`min()`	Minimum	`min(2,3)`	`2`
`sum()`	Somme de ses paramètres effectifs	`sum(2,3,4)`	`9`
`prod()`	Produit de ses paramètres effectifs	`prod(2,4,2)`	`16`
`cummax()`	Maxima cumulés	`cummax(c(2,4,3))`	`2 4 4`
`cummin()`	Minima cumulés	`cummin(c(2,4,1))`	`2 2 1`
`cumsum()`	Sommes cumulées	`cumsum(c(2,3,4))`	`2 5 9`
`cumprod()`	Produits cumulés	`cumprod(c(2,4,3))`	`2 8 24`
`cos()`	Cosinus	`cos(pi)`	`-1`
`sin()`	Sinus	`sin(pi/2)`	`1`
`tan()`	Tangente	`tan(pi/4)`	`1`
`acos()`	Arc-cosinus	`acos(1)`	`0`
`asin()`	Arc-sinus	`asin(0)`	`0`
`atan()`	Arc-tangente	`atan(0)`	`0`
`cosh()`	Cosinus hyperbolique	`cosh(0)`	`1`
`sinh()`	Sinus hyperbolique	`sinh(0)`	`0`
`tanh()`	Tangente hyperbolique	`tanh(0)`	`0`
`acosh()`	Arc-cosinus hyperbolique	`acosh(1)`	`0`
`asinh()`	Arc-sinus hyperbolique	`asinh(0)`	`0`
`atanh()`	Arc-tangente hyperbolique	`atanh(0)`	`0`
`beta()`	Fonction bêta	`beta(1,2)`	`0.5`
`lbeta()`	Logarithme de la fonction bêta $B(a,b)$	`lbeta(1,1)`	`0`
`factorial()`	Factorielle $x!$	`factorial(6)`	`720`
`choose()`	Coefficients du binôme $\binom{n}{p} = \frac{n!}{p!(n-p)!}$	`choose(5,2)`	`10`
`gamma()`	Fonction gamma $\Gamma(x)$ ($\Gamma(n) = (n-1)!$, si $n \in \mathbb{N}$)	`gamma(4)`	`6`
`lgamma()`	Logarithme de la fonction gamma	`lgamma(2)`	`0`
`digamma()`	Première dérivée du logarithme de la fonction gamma	`digamma(2)`	`0.4227843`
`trigamma()`	Seconde dérivée du logarithme de la fonction gamma	`trigamma(2)`	`0.6449341`

Vous noterez que la plupart de ces fonctions peuvent prendre un vecteur comme paramètre d'appel.

Prise en main

- Vérifiez numériquement pour quelques valeurs la validité de la formule $\binom{n-1}{p-1} + \binom{n-1}{p} = \binom{n}{p}$.
- Calculez la somme des n premiers entiers pour chacune des valeurs $n = 1, \ldots, 10$. Vérifiez qu'elle est donnée par la formule $\frac{n(n+1)}{2}$.
- Calculez la somme des n premiers carrés d'entiers pour chacune des valeurs $n = 1, \ldots, 10$. Vérifiez qu'elle est donnée par la formule $\frac{n(n+1)(2n+1)}{6}$.
- Calculez, pour $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$ = (0.83,0.13,-1.16,-1.14,-0.68,0.73,-1.27)$^\mathsf{T}$, la valeur de

$$\hat{G}_n = \frac{n}{K} g_n(\widehat{m_e}, \hat{\theta})^2 \quad \text{où} \quad g_n(m_e, \theta) = 1 - \gamma - \frac{1}{n} \sum_{i=1}^{n} |y_i| \log |y_i|$$

avec $y_i = \frac{x_i - m_e}{\theta}$, $\widehat{m_e}$ = Médiane$(x_1, \ldots, x_n)$, $\hat{\theta} = \frac{1}{n} \sum_{i=1}^{n} |x_i - \widehat{m_e}|$, $K = \frac{\pi^2}{3} - 3$, $\gamma = 1 - \psi(2)$ la constante d'Euler et $\psi(\cdot)$ la fonction digamma. Vous utiliserez la fonction `median()`.

Note : Il suffit de vérifier que la valeur $\hat{G}_n$ est supérieure à la valeur critique `qchisq(1-`α`,df=1)`, pour un seuil α donné (en général 5 %), pour pouvoir décider, avec un risque d'erreur α de se tromper que les données n'ont pas été générées suivant une loi de Laplace. Nous reviendrons en détail sur ce type de procédure dans le chapitre 11.

Astuce

Le nombre π s'obtient en R au moyen de la commande `pi`.

SECTION 8.2

Calcul matriciel

Un certain nombre d'opérations classiques sur les matrices sont présentes dans la version de base de R. Avant de les présenter, définissons un scalaire λ et deux matrices réelles $\boldsymbol{\mathcal{A}}$ et $\boldsymbol{\mathcal{B}}$, et une matrice complexe $\boldsymbol{C}$ dont nous nous servirons par la suite. Le lecteur intéressé par ce sujet pourra consulter avec profit [36].

```
> lambda <- 2 # Création du scalaire λ.
> A <- matrix(c(2,3,5,4),nrow=2,ncol=2) # Matrice réelle.
> A
     [,1] [,2]
[1,]    2    5
[2,]    3    4
> B <- matrix(c(1,2,2,7),nrow=2,ncol=2) # Matrice réelle
                                        # symétrique.
> B
     [,1] [,2]
[1,]    1    2
[2,]    2    7
> C <- matrix(c(1,1i,-1i,3),ncol=2) # Matrice complexe
                                    # hermitienne.
> C
     [,1] [,2]
[1,] 1+0i 0-1i
[2,] 0+1i 3+0i
> I2 <- diag(rep(1,2)) # Matrice identité d'ordre 2.
```

Nous allons nous servir de ces objets pour illustrer les différentes opérations possibles sur les matrices.

Remarque

Pour des fonctions plus élaborées de manipulation de matrices, le lecteur pourra utiliser le *package* `Matrix`.

8.2.1 Opérations de base

Les opérations de base sur les matrices en R sont les suivantes :

- L'addition d'un scalaire : $\lambda + \mathcal{A}$

```
> lambda+A
     [,1] [,2]
[1,]    4    7
[2,]    5    6
```

- L'addition (terme à terme) : $\mathcal{A} + \mathcal{B}$

```
> A+B
     [,1] [,2]
[1,]    3    7
[2,]    5   11
```

- La soustraction (terme à terme) : $\mathcal{A} - \mathcal{B}$

```
> A-B
     [,1] [,2]
[1,]    1    3
[2,]    1   -3
```

- La multiplication par un scalaire : $\lambda\mathcal{A}$

```
> lambda*A
     [,1] [,2]
[1,]    4   10
[2,]    6    8
```

- La transposition : $\mathcal{A}^{\mathsf{T}}$

```
> t(A)
     [,1] [,2]
[1,]    2    3
[2,]    5    4
```

- La conjuguée : $\overline{\mathcal{C}}$

```
> Conj(C)
     [,1] [,2]
[1,] 1+0i 0+1i
[2,] 0-1i 3+0i
```

- La multiplication terme à terme :

```
> A*B
     [,1] [,2]
[1,]    2   10
[2,]    6   28
```

- La multiplication matricielle : $\mathcal{AB}$

```
> A%*%B
     [,1] [,2]
[1,]   12   39
[2,]   11   34
```

- La division terme à terme :

```
> A/B
     [,1]      [,2]
[1,]  2.0 2.5000000
[2,]  1.5 0.5714286
```

- L'inversion matricielle : $\mathcal{B}^{-1}$

```
> solve(B)
           [,1]       [,2]
[1,]  2.3333333 -0.6666667
[2,] -0.6666667  0.3333333
```

- La division matricielle : $\mathcal{A}^{-1}\mathcal{B}$

```
> solve(A)%*%B # Identique à: solve(A,B)
            [,1]       [,2]
[1,]  0.8571429  3.857143
[2,] -0.1428571 -1.142857
```

- Le produit avec transposition : $\mathcal{A}^{\mathsf{T}}\mathcal{B}$

```
> crossprod(A,B)  # t(A)%*%B
     [,1] [,2]
[1,]    8   25
[2,]   13   38
```

Prise en main

Soit les matrices suivantes :

$$\boldsymbol{\mathcal{M}} = \begin{bmatrix} 1 & 1 \\ 2 & 3 \\ 4 & 1 \end{bmatrix}, \quad \boldsymbol{\mathcal{N}} = \begin{bmatrix} 3 & 4 \\ 1 & 3 \\ 4 & 1 \end{bmatrix}, \quad \boldsymbol{\mathcal{O}} = \begin{bmatrix} 3 & 4 & 2 \\ 1 & 3 & 2 \end{bmatrix} \text{ et } \boldsymbol{\mathcal{P}} = \begin{bmatrix} 3 & 4 & 2 \\ 1 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Donnez les dimensions des matrices $\boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{O}}$ et $\boldsymbol{\mathcal{P}}$. Calculez $\boldsymbol{\mathcal{M}} + \boldsymbol{\mathcal{N}}$, $\boldsymbol{\mathcal{M}} - \boldsymbol{\mathcal{N}}$, $3\boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{M}}\boldsymbol{\mathcal{O}}$, $\boldsymbol{\mathcal{O}}\boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{M}}^{\mathsf{T}}$, $\boldsymbol{\mathcal{P}}^{-1}$. Vérifiez que $\boldsymbol{\mathcal{P}}\boldsymbol{\mathcal{P}}^{-1} = \boldsymbol{\mathcal{I}}_3 = \boldsymbol{\mathcal{P}}^{-1}\boldsymbol{\mathcal{P}}$.

Soit les matrices suivantes : $\boldsymbol{\mathcal{Q}} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$ et $\boldsymbol{\mathcal{R}} = \begin{bmatrix} 3 & 4 & 1 \end{bmatrix}$. Calculez $\boldsymbol{\mathcal{Q}}\boldsymbol{\mathcal{R}}$, $\boldsymbol{\mathcal{R}}\boldsymbol{\mathcal{Q}}$ et $\boldsymbol{\mathcal{Q}}^{\mathsf{T}}\boldsymbol{\mathcal{P}}\boldsymbol{\mathcal{Q}}$.

8.2.2 Produit extérieur

Le produit extérieur des vecteurs colonnes $\boldsymbol{x}$ et $\boldsymbol{y}$ est la matrice $\boldsymbol{x}\boldsymbol{y}^{\mathsf{T}}$ d'élément général $x_i y_j$.

```
> x <- seq(1,4)
> y <- seq(4,7)
> outer(x,y,FUN="*")
     [,1] [,2] [,3] [,4]
[1,]    4    5    6    7
[2,]    8   10   12   14
[3,]   12   15   18   21
[4,]   16   20   24   28
```

Astuce

La fonction `outer()` permet des opérations plus générales que le produit de termes. Ainsi, l'appel de la commande `outer(x,y,FUN=f)` sur les vecteurs $\boldsymbol{x} = (x_1, \ldots, x_n)^\mathsf{T}$, $\boldsymbol{y} = (y_1, \ldots, y_n)^\mathsf{T}$ via la fonction $f(x, y)$ produit la matrice suivante :

$$\begin{pmatrix} f(x_1,y_1) & \cdots & f(x_1,y_n) \\ \vdots & f(x_i,y_j) & \vdots \\ f(x_n,y_1) & \cdots & f(x_n,y_n) \end{pmatrix}.$$

8.2.3 Produit de Kronecker

Si $\mathcal{A}$ est une matrice $m \times n$ et $\mathcal{B}$ une matrice $p \times q$, alors le produit de Kronecker de la matrice $\mathcal{A}$ par la matrice $\mathcal{B}$ est la matrice $\mathcal{A} \otimes \mathcal{B} = \begin{bmatrix} a_{11}\mathcal{B} & \cdots & a_{1n}\mathcal{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathcal{B} & \cdots & a_{mn}\mathcal{B} \end{bmatrix}$ de taille $mp \times nq$.

```
> kronecker(A,B)
     [,1] [,2] [,3] [,4]
[1,]    2    4    5   10
[2,]    4   14   10   35
[3,]    3    6    4    8
[4,]    6   21    8   28
```

8.2.4 Matrices triangulaires

Il est parfois utile de récupérer les sous-matrices triangulaires inférieure et supérieure d'une matrice. Cela est possible au moyen des fonctions `lower.tri()` et `upper.tri()`.

```
> M <- matrix(1:16,nrow=4)
> lower.tri(M)
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE FALSE FALSE
[2,]  TRUE FALSE FALSE FALSE
[3,]  TRUE  TRUE FALSE FALSE
[4,]  TRUE  TRUE  TRUE FALSE
> upper.tri(M,diag=TRUE)
      [,1]  [,2]  [,3] [,4]
[1,]  TRUE  TRUE  TRUE TRUE
[2,] FALSE  TRUE  TRUE TRUE
```

```
[3,] FALSE FALSE  TRUE TRUE
[4,] FALSE FALSE FALSE TRUE
> M[lower.tri(M)] <- 0
> M
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    0    6   10   14
[3,]    0    0   11   15
[4,]    0    0    0   16
```

8.2.5 Opérateurs vec et demi-vec

L'opérateur matriciel *vec* appliqué à une matrice $\mathcal{A}$ consiste à fabriquer le long vecteur colonne *vec*($\mathcal{A}$) constitué de l'empilement des colonnes de $\mathcal{A}$. Il s'obtient en R au moyen de l'instruction suivante :

```
> vec <- function(M) as.matrix(as.vector(M))
> # ou de façon équivalente, mais à l'extérieur d'une fonction:
> # dim(A) <- c(prod(dim(A)),1)
> A
     [,1] [,2]
[1,]    2    5
[2,]    3    4
> vec(A)
     [,1]
[1,]    2
[2,]    3
[3,]    5
[4,]    4
```

L'opérateur matriciel *vech* (pour *vec half* ou demi-vec) appliqué à une matrice $\mathcal{A}$ consiste à fabriquer le long vecteur colonne *vech*($\mathcal{A}$) constitué de l'empilement des colonnes de $\mathcal{A}$, mais en excluant les éléments au-dessus de la diagonale de $\mathcal{A}$. Il s'obtient en R au moyen de l'instruction suivante :

```
> vech <- function(M) as.matrix(M[lower.tri(M,diag=TRUE)])
> vech(A)
     [,1]
[1,]    2
[2,]    3
[3,]    4
```

8.2.6 Déterminant, trace, nombre de conditionnement

La fonction `det()` permet de calculer le déterminant d'une matrice.

```
> det(A)
[1] -7
```

Il n'existe pas de fonction R permettant de calculer directement la trace d'une matrice, mais il est très facile de la calculer ainsi :

```
> sum(diag(A))
[1] 6
```

Attention

Il ne faut pas utiliser la fonction `trace()` pour calculer la trace d'une matrice. En effet, cette fonction sert dans le débogage de code R.

Le nombre de conditionnement est le rapport de la plus grande sur la plus petite valeur singulière non nulle. Une grande valeur du nombre de conditionnement indique de mauvaises propriétés numériques de la matrice. Il s'obtient au moyen de la fonction `kappa()`.

```
> kappa(A,exact=TRUE)
[1] 7.582401
```

8.2.7 Données centrées, données réduites

La fonction `scale()` permet de centrer et/ou de réduire une matrice. Le centrage consiste à retrancher à chacune des colonnes de la matrice la moyenne de cette colonne. La réduction consiste à diviser chacune des colonnes de la matrice par son écart type.

Attention

Notez que la fonction `sd()` calcule un écart type avec un numérateur égal à $n-1$.

Centrage

```
> scale(A,scale=FALSE)
     [,1] [,2]
[1,] -0.5  0.5
[2,]  0.5 -0.5
attr(,"scaled:center")
[1] 2.5 4.5
```

Réduction

```
> scale(A,center=FALSE,scale=apply(A,2,sd))
         [,1]     [,2]
[1,] 2.828427 7.071068
[2,] 4.242641 5.656854
attr(,"scaled:scale")
[1] 0.7071068 0.7071068
```

Attention

Si l'on veut obtenir une réduction fondée sur l'écart type de la population comme cela est préconisé par exemple dans l'analyse de données à la française, on pourra utiliser l'instruction :

```
> red <- sqrt((nrow(A)-1)/nrow(A))
> scale(A,center=FALSE,scale=apply(A,2,sd)*red)
     [,1] [,2]
[1,]    4   10
[2,]    6    8
attr(,"scaled:scale")
[1] 0.5 0.5
> # ou de façon équivalente: t(A/apply(A,2,sd))/red
```

8.2.8 Calcul des valeurs propres et vecteurs propres

On peut obtenir les valeurs propres et les vecteurs propres d'une matrice au moyen de la fonction `eigen()`.

```
> eigen(A)
$values
[1]  7 -1
$vectors
           [,1]       [,2]
[1,] -0.7071068 -0.8574929
[2,] -0.7071068  0.5144958
```

Astuce

Notez que pour une matrice hermitienne $\boldsymbol{C}$ (c'est-à-dire une matrice complexe égale à sa transconjuguée), la fonction `eigen()` permet d'obtenir la décomposition aux valeurs propres de cette matrice $\boldsymbol{C}$, c'est-à-dire $\boldsymbol{C} = \boldsymbol{\mathcal{V}}\boldsymbol{\mathcal{D}}\boldsymbol{\mathcal{V}}^*$ (où $\boldsymbol{\mathcal{V}}^*$ est la matrice adjointe de $\boldsymbol{\mathcal{V}}$) :

```
> C <- matrix(c(1,1i,-1i,3),ncol=2)
> e <- eigen(C,symmetric=TRUE)
> V <- e$vectors
> D <- diag(e$values)
> all.equal(C,V%*%D%*%t(Conj(V)))
[1] TRUE
```

8.2.9 Racine carrée d'une matrice hermitienne définie positive

Une racine carrée d'une matrice définie positive $\boldsymbol{C}$ est toute matrice $\boldsymbol{\mathcal{M}}$ qui vérifie $\boldsymbol{\mathcal{M}}^*\boldsymbol{\mathcal{M}} = \boldsymbol{C}$, où $\boldsymbol{\mathcal{M}}^*$ désigne la matrice adjointe (transposée conjuguée) de $\boldsymbol{\mathcal{M}}$. On la note en général $\boldsymbol{C}^{1/2}$ même si elle n'est pas unique. Lorsque $\boldsymbol{C}$ est

hermitienne (c'est-à-dire ou bien une matrice complexe égale à sa transconjuguée, ou bien une matrice réelle symétrique), on peut calculer $\boldsymbol{C}^{1/2}$ de la façon suivante :

```
> e <- eigen(C,symmetric=TRUE)
> V <- e$vectors
> V %*% diag(sqrt(e$values)) %*% t(Conj(V)) # C^{1/2},
                                            # qui est ici
                                            # hermitienne.
                      [,1]                  [,2]
[1,] 0.9238795+0.0000000i 0.000000-0.3826834i
[2,] 0.0000000+0.3826834i 1.689246+0.0000000i
```

La matrice $\boldsymbol{C}^{-1/2}$ peut se calculer ainsi :

```
> V %*% diag(1/sqrt(e$values)) %*% t(Conj(V)) # C^{-1/2},
                                              # qui est ici
                                              # hermitienne.
                     [,1]                   [,2]
[1,] 1.194478+0.0000000i 0.0000000+0.2705981i
[2,] 0.000000-0.2705981i 0.6532815+0.0000000i
```

8.2.10 Décomposition en valeurs singulières

On cherche à écrire $\boldsymbol{C} = \boldsymbol{\mathcal{U}}\boldsymbol{\mathcal{D}}\boldsymbol{\mathcal{V}}^*$ où $\boldsymbol{\mathcal{D}}$ est la matrice diagonale des valeurs singulières de $\boldsymbol{C}$, $\boldsymbol{\mathcal{U}}$ (respectivement $\boldsymbol{\mathcal{V}}$) est la matrice des vecteurs singuliers à gauche (respectivement à droite) de $\boldsymbol{C}$. Pour cela, il faut utiliser la fonction `svd()`.

```
> res <- svd(C)
> res
$d
[1] 3.4142136 0.5857864
$u
                      [,1]                  [,2]
[1,] -0.3826834+0.0000000i 0.9238795+0.0000000i
[2,]  0.0000000-0.9238795i 0.0000000-0.3826834i
$v
                      [,1]                  [,2]
[1,] -0.3826834+0.0000000i 0.9238795+0.0000000i
[2,]  0.0000000-0.9238795i 0.0000000-0.3826834i
> D <- diag(res$d)
> U <- res$u
> V <- res$v
> all.equal(C,U%*%D%*%t(Conj(V))) # A = UDV*.
[1] TRUE
```

Astuce

Pour calculer l'inverse (généralisée) de Moore-Penrose d'une matrice (non inversible), on peut utiliser la fonction suivante :

```
> mpinv <- function(M,eps=1e-13) {
+  s <- svd(M)
+  e <- s$d
+  e[e>eps] <- 1/e[e>eps]
+ return(s$v%*%diag(e)%*%t(s$u))
+ }
```

8.2.11 Décomposition de Cholesky

Pour une matrice réelle symétrique définie positive $\mathcal{B}$, on cherche à écrire $\mathcal{B} = \mathcal{U}^T\mathcal{U} = \mathcal{L}\mathcal{L}^T$, où $\mathcal{U}$ (respectivement $\mathcal{L}$) est une matrice triangulaire supérieure (respectivement inférieure). Vous aurez noté que cela implique que $\mathcal{U}$ est une racine carrée de $\mathcal{B}$. Pour obtenir cette décomposition, il faut utiliser la fonction `chol()`.

```
> U <- chol(B) # C'est un autre moyen d'obtenir
               # B^{1/2}.
> L <- t(U)
> U
     [,1]     [,2]
[1,]    1 2.000000
[2,]    0 1.732051
> all.equal(B,t(U)%*%U) # B = U^T U.
[1] TRUE
```

Notez également que vous pouvez utiliser la fonction `chol2inv()` pour calculer l'inverse $\mathcal{B}^{-1}$ d'une matrice carrée symétrique définie positive $\mathcal{B}$, à partir de sa décomposition de Cholesky.

```
> B
     [,1] [,2]
[1,]    1    2
[2,]    2    7
> chol2inv(U)   # C'est B^{-1}.
           [,1]       [,2]
[1,]  2.3333333 -0.6666667
[2,] -0.6666667  0.3333333
> all.equal(chol2inv(U),solve(B))
[1] TRUE
```

Notez enfin que la fonction `chol()` peut être utilisée pour calculer $\mathcal{B}^{-1/2}$ de la façon suivante :

```
> solve(chol(B))# C'est une version de B^{-1/2}.
     [,1]       [,2]
[1,]    1 -1.1547005
[2,]    0  0.5773503
```

8.2.12 Décomposition QR

On cherche à écrire $\mathcal{A} = \mathcal{QR}$, où $\mathcal{Q}$ est une matrice orthogonale ($\mathcal{QQ}^\mathsf{T} = \mathcal{Q}^\mathsf{T}\mathcal{Q} = \mathcal{I}$) et $\mathcal{R}$ est une matrice triangulaire supérieure.

```
> res <- qr(A)
> Q <- qr.Q(res)
> Q
           [,1]       [,2]
[1,] -0.5547002 -0.8320503
[2,] -0.8320503  0.5547002
> all.equal(I2,Q%*%t(Q)) # I_2 = QQ^T.
[1] TRUE
> R <- qr.R(res)
> R
          [,1]      [,2]
[1,] -3.605551 -6.101702
[2,]  0.000000 -1.941451
> all.equal(A,Q%*%R) # A = QR.
[1] TRUE
```

Astuce

Notez que `qr(A,tol=1e-07)$rank` renvoie le rang de la matrice stockée dans `A`, en utilisant l'argument de tolérance `tol` pour détecter des dépendances linéaires dans les colonnes de `A`.

SECTION 8.3

Intégration numérique

Le logiciel **R** sait faire du calcul numérique d'intégrales à l'aide de la fonction `integrate()`.

Quelques exemples permettront d'illustrer simplement l'utilisation de cette fonction. Ainsi, supposons que l'on veuille vérifier numériquement que $\int_{-\infty}^{\infty} \frac{\exp(-x^2/2)}{\sqrt{2\pi}} dx = 1$. Il suffit de procéder de la façon suivante :

```
> myf <- function(x) {exp(-x^2/2)/sqrt(2*pi)}
> integrate(myf,lower=-Inf,upper=Inf)$value
[1] 1
```

Notez qu'il est aussi possible d'intégrer des fonctions de plusieurs variables. Ainsi, supposons maintenant que l'on veuille vérifier numériquement que $\int_{x=0}^{x=1}\int_{y=2}^{y=3}\cos(x+y)dydx = 2\cos(3)-\cos(4)-\cos(2)$. On procèdera ainsi :

```
> myf <- function(x) {
+  res <- vector("integer",length(x))
+  for (i in 1:length(x)) {
+    res[i] <- integrate(f=function(y,x){cos(x+y)},lower=2,
+                        upper=3,x=x[i])$value
+                       }
+ return(res)
+ }
> integrate(myf,lower=0,upper=1)$value
[1] -0.9101945
> 2*cos(3)-cos(4)-cos(2)
[1] -0.9101945
```

Remarque

Notez que la fonction `integrate()` renvoie également la précision du calcul numérique effectué.

Prise en main

Montrez numériquement que la fonction $f_X(x) = \frac{1}{2}\left(1+\frac{(x-1)}{8}\right)^{-5}\mathbf{1}_{[1,+\infty]}(x)$ est bien une densité, c'est-à-dire que son intégrale vaut 1. Il s'agit en fait de la densité d'une loi de Pareto généralisée de paramètres (1,2,1/4).

SECTION 8.4

Dérivation

8.4.1 Dérivation symbolique

Les capacités de calcul symbolique avec R sont très limitées et nous ne nous étendrons pas sur ce point. Soulignons quand même que R possède des fonctions de dérivation symbolique : `D()` et `deriv()`. Par exemple :

```
> D(expression(sin(cos(x + y^2))), "x")     # Dériver par rapport
                                            # à x.
-(cos(cos(x + y^2)) * sin(x + y^2))
> D(expression(sin(cos(x + y^2))), "y")     # Dériver par rapport
```

```
                                                    # à y.
-(cos(cos(x + y^2)) * (sin(x + y^2) * (2 * y)))
> f <- deriv(~ x^2, "x", TRUE)          # Dériver x^2
                                        # pour trouver 2x.
> f(3)  # Renvoie 3^2 et 2*3
[1] 9
attr(,"gradient")
     x
[1,] 6
```

Expert

Le *package* `Ryacas` permet d'interfacer **R** avec le logiciel de calcul formel `Yacas` disponible à l'adresse `http://yacas.sourceforge.net`. Essayez, après avoir installé et chargé le *package* `Ryacas`, de taper la commande suivante : `vignette("Ryacas")`.

8.4.2 Dérivation numérique

Il est possible d'effectuer des dérivations numériques en utilisant la fonction `grad()` du *package* `numDeriv`.

```
> require("numDeriv")
> f <- function(x) x^2  # Fonction d'une variable.
> grad(f,c(2,1,3,5))    # Calcule la dérivée en plusieurs points
                        # scalaires.
[1]  4  2  6 10
```

On trouve également dans ce *package* la fonction `hessian()` permettant d'obtenir des dérivées secondes, mais uniquement en un seul point vectoriel.

```
> g <- function(x) x[1]*x[2]^2 # Fonction de deux variables.
> grad(g,c(2,1))        # Calcule la dérivée en un seul point
                        # vectoriel.
[1] 1 4
> hessian(g,c(2,1))     # Calcule la dérivée seconde en un seul
                        # point vectoriel.
             [,1] [,2]
[1,] 4.210428e-14    2
[2,] 2.000000e+00    4
```

La fonction `numericDeriv()`, plus délicate d'utilisation, permet d'obtenir le gradient d'une fonction de plusieurs variables en plusieurs points vectoriels. Par exemple, les instructions suivantes permettent de calculer le vecteur gradient de la fonction xy^2 aux points $(2,1)$ et $(3,4)$. Les résultats sont $(1,4)$ et $(16,24)$.

```
> h <- function(x,y) x*y^2 # Fonction de deux variables.
> x <- c(2,3)
> y <- c(1,4)
> attributes(numericDeriv(quote(h(x,y)),c("x","y")))$gradient
     [,1] [,2] [,3] [,4]
[1,]    1    0    4    0
[2,]    0   16    0   24
```

SECTION 8.5

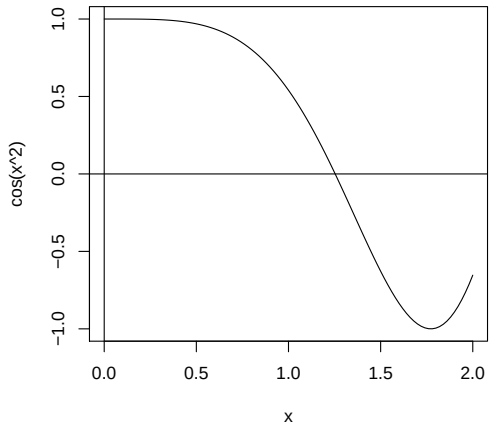
Optimisation

8.5.1 Fonctions d'optimisation

L'optimisation d'une fonction consiste à trouver l'endroit où elle atteint son maximum ou son minimum. Le logiciel R dispose de plusieurs fonctions pour cela, fondées sur divers algorithmes : `optimize()`, `optim()`, `nlm()`, `constrOptim()`, `nlminb()`.

- **Optimisation unidimensionnelle**

  La fonction `optimize()`, qui s'utilise uniquement dans un contexte unidimensionnel, nous semble être la plus simple à manipuler. Donnons-en un exemple. Supposons que l'on veuille calculer le minimum, ainsi que l'endroit où il est atteint, de la fonction `cos(x^2)` sur l'intervalle $[0, 2]$, représentée sur la figure suivante :



  La fonction `optimize()` permet de résoudre ce problème de façon numérique.

```
> optimize(f=function(x){cos(x^2)},lower=0,upper=2)
$minimum
[1] 1.772453
$objective
[1] -1
```

Le minimum vaut donc -1 et il est atteint en $x = 1.772453$.

Astuce

Le paramètre `maximum=TRUE` de la fonction `optimize()` permet de calculer le maximum plutôt que le minimum d'une fonction.

- **Optimisation multidimensionnelle**

  Voyons maintenant comment il est possible, à l'aide de la fonction `nlm()` conçue pour des problèmes de minimisation, de trouver le maximum d'une fonction de deux variables, d'équation $f(x,y) = 10\frac{\sin\left(\sqrt{(x-3)^2+(y-4)^2}\right)}{((x-3)^2+(y-4)^2)^{\alpha/2}}$ avec $\alpha = 1.1$, représentée sur la figure 8.1 au moyen des instructions R suivantes :

```
> y <- x <- seq(-10,10,length = 30)
> f <- function(x,y,alpha=1.1) { r <- sqrt((x-3)^2+(y-4)^2)
+                       return(10 * sin(r)/r^alpha) }
> z <- outer(x, y, f)
> z[is.na(z)] <- 1
> op <- par(bg = "white")
> persp(x,y,z,theta=30,phi=30,expand=0.5,
+                       col="lightblue",ticktype="detailed")
```

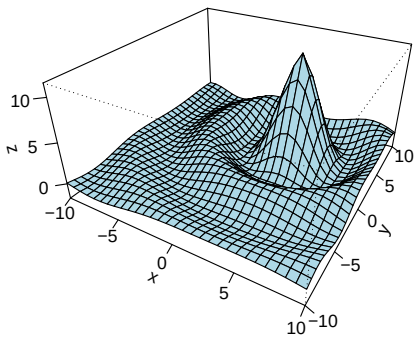


FIGURE 8.1 – Fonction sinc modifiée.

Puisque trouver le maximum d'une fonction f est équivalent à trouver le minimum de la fonction $-f$, nous utiliserons `nlm()` sur la fonction $-f$.

```
> f <- function(z,alpha=1.1) {
+  x <- z[1]
+  y <- z[2]
+  r <- sqrt((x-3)^2+(y-4)^2)
+  return(- 10 * sin(r)/r^alpha)
+ }
> res <- nlm(f,c(0,0))  # Le deuxième paramètre effectif
                        # correspond aux valeurs initiales.
> res
$minimum
[1] -1.046464
$estimate
[1] -1.627385 -2.169848
$gradient
[1] -1.534979e-07  1.139977e-07
$code
[1] 1
$iterations
[1] 7
```

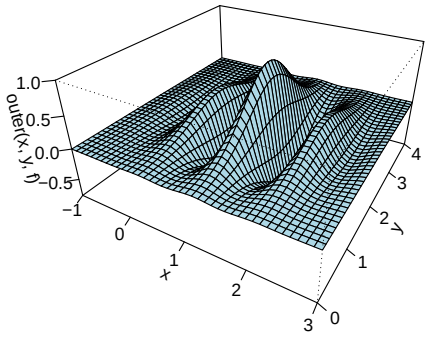
Le maximum vaut $(-1)\times$`res$minimum`$=1.046464$ et se produit en `res$estimate`$=(-1.627,-2.169)$.

Astuce

Voici un exemple de passage de paramètre dans `nlm` pour la fonction à minimiser. Ainsi, pour la maximisation de f avec $\alpha = 2$, vous pouvez exécuter l'instruction suivante : `nlm(f,c(0,0),alpha=2)`

- **Optimisation sous contrainte**
  Pour effectuer une optimisation avec des contraintes simples du type $-a \leq x \leq b$ et $-c \leq y \leq d$, vous pouvez utiliser la fonction `nlminb()` et ses paramètres `lower` et `upper`. Par exemple, recherchons les trois maxima de la surface suivante donnée par l'équation $e^{-(x-1.2)^2-(y-2)^2}\cos(2\pi(x-1.2))$ sur le domaine $[-1,3]\times[0,4]$, dont le tracé peut être obtenu au moyen des instructions suivantes :

```
> f <- function(x,y) exp(-(x-1.2)^2-(y-2)^2)*cos((x-1.2)*pi*2)
> x <- seq(-1,3,0.1)
> y <- seq(0,4,0.1)
> persp(x,y,outer(x,y,f),theta=30,phi=30,expand=0.5,
+                        col="lightblue",ticktype="detailed")
```



Pour cela, on commence par effectuer une recherche du maximum global que l'on peut cibler visuellement sur le sous-domaine $[0.5, 1.5] \times [0, 4]$.

```
> f <- function(x) -exp(-(x[1]-1.2)^2-(x[2]-2)^2)*
+                           cos(2*pi*(x[1]-1.2))
> nlminb(c(0.8,0),f,lower=c(0.5,0),upper=c(1.5,4))
$par
[1] 1.2 2.0
$objective
[1] -1
$convergence
[1] 0
$iterations
[1] 19
$evaluations
function gradient
      27       42
$message
[1] "relative convergence (4)"
```

Le maximum (global) vaut $(-1)\times$`res$objective`$=1$ et se produit en `res$par`$=(1.2, 2)$.

On termine par une recherche sur les sous-domaines $[-1, 0.5] \times [0, 4]$ et $[1.5, 3] \times [0, 4]$, en tapant les instructions suivantes :

```
nlminb(c(0,0),f,lower=c(-1,0),upper=c(0.5,4))
nlminb(c(2,0),f,lower=c(1.5,0),upper=c(3,4))
```

Remarque

Dans le cas de contraintes linéaires du type suivant :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \geq 0 \Leftrightarrow \begin{cases} ax + by - c_1 \geq 0 \\ cx + dy - c_2 \geq 0 \end{cases}$$

on peut utiliser la fonction `constrOptim()` avec les paramètres `ui`$=\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ et `ci`$=\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$.

8.5.2 Racines d'une fonction

Les racines d'une fonction f sont définies comme étant les solutions de l'équation $f(x) = 0$.

- **Cas d'une racine unique**
  Dans le cas d'une racine unique, on utilise la fonction `uniroot()` effectuant une recherche unidimensionnelle sur un intervalle donné. Voilà par exemple comment on peut déterminer la valeur qui annule la fonction $\cos(x^2)$ (vue lors de la présentation de l'optimisation unidimensionnelle) sur l'intervalle $[0, 2]$. La solution analytique à ce simple problème est $x = \sqrt{\pi/2} = 1.253314$.

```
> uniroot(f=function(x){cos(x^2)},lower=0,upper=2,
+                       tol=0.00001)$root
[1] 1.253314
```

- **Cas des racines d'un polynôme**
  La fonction `polyroot()` sert à trouver toutes les racines d'un polynôme (possiblement complexes). Par exemple, trouvons les racines du polynôme $P(x) = 3 - 8x + x^2$.

```
> polyroot(c(3,-8,1)) # Les valeurs exactes sont:
                      # (8+c(-1,1)*sqrt(54))/2.
[1] 0.3944487+0i 7.6055513+0i
```

Termes à retenir

`round()` : permet d'arrondir des chiffres
`abs()` : valeur absolue
`sqrt()` : racine carrée
`exp()`, `log()` : exponentielle, logarithme
`max()`, `min()` : maximum, minimum
`sum()`, `prod()` : somme, produit
`cummax()`, `cummin()`, `cumprod()`, `cumsum()` : maxima, minima, produits et sommes cumulés
`cos()`, `sin()` : cosinus, sinus
`%*%` : opérateur de produit matriciel
`t()`, `solve()` : transposition, inversion matricielle
`outer()`, `kronecker()` : produit extérieur, produit de Kronecker
`det()` : déterminant d'une matrice
`eigen()` : calcul des valeurs et vecteurs propres d'une matrice
`svd()`, `chol()`, `qr()` : décomposition en valeurs singulières, de Cholesky, QR d'une matrice
`integrate()` : intégration numérique
`D()`, `deriv()` : dérivation symbolique
`grad()`, `hessian()` : dérivation numérique du *package* `numDeriv`
`optimize()`, `nlm()`, `nlminb()` : optimisation d'une fonction
`uniroot()`, `polyroot()` : calcul de racine d'une fonction

Exercices

8.1- Quelle fonction permet de calculer les coefficients binomiaux ?

8.2- Donnez l'instruction permettant de calculer la somme des n premiers entiers.

8.3- Quelle fonction renvoie l'étendue d'un échantillon de valeurs ?

8.4- Que renvoie cette instruction :
`matrix(c(1,0,0,1),nrow=2)*matrix(1:4,nrow=2)` ?

8.5- Quel est le symbole de multiplication matricielle ?

8.6- Quelle fonction permet d'inverser une matrice ? De transposer une matrice ?

8.7- Donnez l'instruction permettant de créer la matrice identité d'ordre 5.

8.8- Donnez l'instruction permettant de calculer le déterminant d'une matrice. La trace d'une matrice.

8.9- Donnez l'instruction permettant de centrer-réduire la matrice $\mathcal{A}$.

8.10- Quelle fonction permet de calculer les valeurs et vecteurs propres d'une matrice ?

8.11- Donnez l'instruction permettant d'intégrer numériquement la fonction $3x^2 + 2$ sur $[-1, 1]$.

8.12- Donnez l'instruction permettant de trouver le maximum de la fonction $\sin^2(x)$ sur $[0, 2]$.

8.13- Quelle fonction permet de savoir où une fonction s'annule ? Où un polynôme s'annule ?

Fiche de TP

Calcul matriciel, optimisation, intégration

A- Un premier problème d'optimisation

L'objectif est de retrouver les valeurs propres de la matrice suivante de plusieurs façons différentes.

```
> A
     [,1] [,2]
[1,]    2    5
[2,]    3    4
```

8.1- Programmez une fonction nommée `myf()` qui permet de calculer la valeur du polynôme caractéristique $P(x) = \det(\mathcal{A} - x\mathcal{I}_2)$ au point x (indice : utiliser la fonction `det()`). Nous rappelons que les valeurs propres d'une matrice sont les racines de son polynôme caractéristique.

8.2- Modifiez la fonction `myf()` pour la rendre à valeurs vectorielles.

8.3- Tracez la courbe de la fonction `myf()` sur l'intervale $[-10, 10]$. Rajoutez les axes du repère.

8.4- Utilisez la fonction `uniroot()` à deux reprises pour trouver les deux racines de cette fonction.

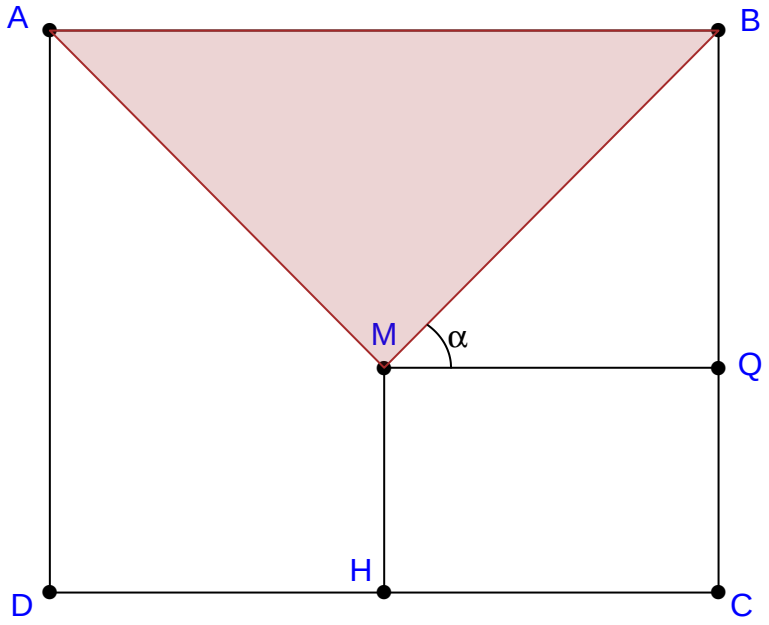
8.5- Trouvez les coefficients du polynôme $P(x)$, puis utilisez la fonction `polyroot()` pour calculer les racines de ce polynôme.

8.6- Utilisez la fonction `eigen()` pour vérifier vos résultats.

B- Un second problème d'optimisation

Voici les informations connues sur la figure ci-dessous :
- $ABCD$ est un rectangle
- H est le milieu du segment $[DC]$
- M est un point quelconque de la médiatrice du segment $[DC]$, situé à l'intérieur du rectangle $ABCD$
- Q est le projeté orthogonal de M sur (BC)
- $g(\alpha) = MA + MB + MH$ avec $\alpha = \widehat{BMQ} \in]0; \pi/2]$

– $AB = 10$ et $BC = 6$



L'objectif est de trouver l'angle α pour lequel $g(\alpha)$ est minimal.

On rappelle les formules trigonométriques classiques suivantes :
– $\cos(\theta) = \frac{\text{longueur côté adjacent}}{\text{longueur hypoténuse}}$;
– $\sin(\theta) = \frac{\text{longueur côté opposé}}{\text{longueur hypoténuse}}$;
– $\tan(\theta) = \frac{\text{longueur côté opposé}}{\text{longueur côté adjacent}} = \frac{\sin(\theta)}{\cos(\theta)}$;
– $\sin(\pi/2 - \theta) = \cos(\theta)$.

8.1- Reproduisez sous **R** la figure ci-dessus.

8.2- Montrez analytiquement que $g(\alpha) = 5(2 - \sin(\alpha))/\cos(\alpha) + 6$.

8.3- Programmez sous **R** cette fonction `g`.

8.4- Calculez numériquement le minimum de $g(\alpha)$ et la valeur pour laquelle il est atteint.

8.5- Calculez analytiquement $g'(\alpha)$.

8.6- Retrouvez ce résultat par dérivation symbolique.

8.7- Programmez sous **R** cette fonction que vous nommerez `gprime`.

8.8- Calculez numériquement la racine de $g'(\alpha)$, et constatez que vous retrouvez le résultat précédent.

C- Table de la loi normale centrée-réduite

Nous allons utiliser la fonction `integrate()` pour construire la table statistique des probabilités de la loi $\mathcal{N}(0,1)$.

Notons $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$ la fonction de répartition de la loi $\mathcal{N}(0,1)$. Il est bien connu que $\Phi(-x) = 1 - \Phi(x)$. Par conséquent, on peut se contenter de construire la table pour les valeurs positives de x.

8.1- Programmez la fonction `phi()` qui prend un vecteur $\boldsymbol{x}$ de longueur n comme paramètre d'entrée et renvoie le vecteur des valeurs $\frac{1}{\sqrt{2\pi}} e^{-\frac{x_i^2}{2}}$, $i = 1, \ldots, n$.

8.2- Utilisez la fonction `integrate()` pour calculer $\Phi(x)$ pour tous les x du vecteur suivant : `quantiles <- seq(0,5.5,by=0.1)`. Vous les stockerez dans un vecteur nommé `probs`.

8.3- Comparez, au moyen de la fonction `all.equal()`, les résultats que vous avez obtenus avec ceux donnés par la fonction `pnorm()`.

8.4- Tracez la courbe des valeurs de $\Phi(x)$ pour tous les x et tous les $-x$ dans `quantiles` (indice : utilisez la fonction `rev()`).

8.5- Ajoutez à ce graphique, en bleu, la courbe de la fonction `pnorm()`. Vérifiez que ces deux courbes sont parfaitement superposées.

D- Une analyse en composantes principales

La méthode d'analyse en composantes principales permet de décrire des proximités entre des individus sur lesquels plusieurs caractères quantitatifs ont été mesurés. C'est une méthode qui nécessite de nombreux calculs matriciels, ce qui permettra d'illustrer les notions présentées au début de ce chapitre.

Des données ont été récoltées dans la région viticole de Bordeaux. Elles consistent en :

- quatre caractéristiques météorologiques :
  - `TEMPER` : somme des températures moyennes journalières (en degrés Celcius),
  - `SOLEIL` : durée d'ensoleillement (en heures),
  - `CHALEUR` : nombre de jours de grande chaleur,
  - `PLUIE` : hauteur des pluies (en mm) ;
- la qualité du vin (`QUALITE`) déterminée, sous forme d'appréciation, par des œnologues :
  1 = bon vin, 2 = vin moyen, 3 = vin médiocre.

Nous allons représenter le nuage de points des données quantitatives précédentes après les avoir projetées sur un sous-espace (un plan) choisi de façon à limiter la perte d'information ainsi engendrée par cette projection.

8.1- Importez dans la variable `climatvin` le contenu du fichier de données http://www.biostatisticien.eu/springeR/climatvin.csv

8.2- Stockez dans la matrice `X` les variables `TEMPER, SOLEIL, CHALEUR, PLUIE` (indice : `as.matrix()`).

8.3- Calculez le centre (de gravité) `g` du nuage de points des individus présents dans `X` (vecteur des moyennes des colonnes) en utilisant la fonction `colMeans()`. Affichez-le avec deux chiffres après la virgule.

8.4- Utilisez la fonction `scale()` pour calculer la matrice $\dot{\boldsymbol{X}} = (\dot{x}_{ij})$ des données centrées que vous stockerez dans la variable nommée `Xpoint`.

8.5- Calculez l'`inertie` globale du nuage de points, qui représente la dispersion des points : $\mathcal{I} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p}\dot{x}_{ij}^2$, où n est le nombre d'individus présents dans $\boldsymbol{X}$.

8.6- La contribution de l'individu i à l'inertie totale est donnée par $\frac{1}{n\mathcal{I}}\sum_{j=1}^{p}\dot{x}_{ij}^2$, où p est le nombre de variables (ici quatre).
Calculez le vecteur `contrinertie` des contributions à l'inertie totale de tous les individus.

8.7- Créez la matrice colonne `unn` de longueur n, ne contenant que des 1.

8.8- Vérifiez que vous pouvez recalculer le centre de gravité `g` du nuage de points au moyen de la formule $\boldsymbol{g} = \frac{1}{n}\boldsymbol{X}^{\mathsf{T}}\mathbf{1}_n$.

8.9- Vérifiez que vous pouvez recalculer la matrice `Xpoint` des données centrées au moyen de la formule $\dot{\boldsymbol{X}} = \boldsymbol{X} - \mathbf{1}_n\boldsymbol{g}^{\mathsf{T}}$.

8.10- Calculez la matrice `S` des covariances au moyen de la formule $\boldsymbol{S} = \frac{1}{n}\dot{\boldsymbol{X}}^{\mathsf{T}}\dot{\boldsymbol{X}}$. Essayez de retrouver ce résultat au moyen de la fonction `cov()`.

8.11- Utilisez la matrice `S` pour calculer la matrice diagonale `Dunsurs` contenant les inverses des écarts types de la population : $\mathcal{D}_{1/s} = diag(1/s_1, \ldots, 1/s_p)$.

8.12- Calculez la matrice `Z` des données centrées-réduites au moyen de la formule $\mathcal{Z} = \dot{\boldsymbol{X}}\mathcal{D}_{1/s}$.

8.13- Calculez la matrice des corrélations `R` au moyen de la formule $\mathcal{R} = \frac{1}{n}\mathcal{Z}^{\mathsf{T}}\mathcal{Z}$. Essayez de la retrouver au moyen de la fonction `cor()`.

8.14- Calculez la matrice diagonale `Lambda` des valeurs propres ($\boldsymbol{\Lambda}$) et la matrice `W` des vecteurs propres ($\mathcal{W}$) de la matrice des corrélations $\mathcal{R}$. La matrice `W` contient les coordonnées d'une nouvelle base dans laquelle nous représenterons les individus.

8.15- Tracez le cercle des corrélations, qui consiste en un cercle de rayon 1, sur lequel sont superposées des flèches partant de l'origine du repère et dont les coordonnées des pointes sont indiquées dans les deux premières colonnes de la matrice $\mathcal{W}\boldsymbol{\Lambda}^{1/2}$. Vous indiquerez également sur ce graphique les noms des variables au bout de chacune des quatre flèches.

8.16- L'inertie totale du nuage des données centrées-réduites est égale au nombre de variables (ici quatre). Celle-ci se décompose en une somme d'inerties apportées par chacun des quatre axes de la base de $\boldsymbol{\mathcal{W}}$ et qui sont données sur la diagonale de $\boldsymbol{\Lambda}$. Calculez le vecteur des pourcentages d'inertie totale expliquée par les k premiers axes ($1 \leq k \leq p$). Quel est le pourcentage d'inertie expliquée par les deux premiers axes ?

8.17- Calculez la matrice `CW` des composantes principales au moyen de la formule $\boldsymbol{C_W} = \boldsymbol{\mathcal{Z}\mathcal{W}}$. Les composantes principales sont les coordonnées des individus dans la nouvelle base décrite dans `W`.

8.18- Ouvrez une nouvelle fenêtre graphique et représentez-y les individus projetés sur le premier plan principal, c'est-à-dire les points dont les coordonnées sont indiquées dans les deux premières colonnes de $\boldsymbol{C_W}$. Vous indiquerez sur le graphique le nom de chacun des individus.

8.19- Retracez le graphique précédent, mais en coloriant en rouge (respectivement en bleu, en vert) les bons vins (respectivement les vins moyens, les vins médiocres). Vous ajouterez une légende.

8.20- La qualité de représentation de l'individu i sur le premier plan principal est donnée par $\frac{c_{i1}^2+c_{i2}^2}{\sum_{j=1}^{p} c_{ij}^2}$, où c_{ij} est le terme de la ligne i/colonne j de la matrice $\boldsymbol{C_W}$. Calculez le vecteur `QLT` des qualités de représentation des individus sur le premier plan principal. Affichez `QLT` avec deux chiffres significatifs.

8.21- Nous allons explorer rapidement le *package* `ade4` qui permet d'effectuer des ACP. Installez puis chargez ce *package*.

8.22- Tapez les instructions suivantes :

```
rownames(X) <- climatvin[,1]
res <- dudi.pca(X) # Répondre 2 à la question posée.
scatter(res)
s.class(res$li,as.factor(climatvin[,6]))
```

Chapitre 9

Statistique descriptive

Objectif

Ce chapitre décrit les différentes commandes à taper sous R pour structurer vos variables, tracer des résumés graphiques classiques de vos données et calculer des résumés numériques statistiques simples sur un jeu de données. Les données utilisées pour illustrer ce chapitre sont celles du jeu de données NUTRIAGE. Quelques exemples de fonctions permettant d'obtenir des graphiques de qualité esthétique supérieure pouvant servir dans des présentations ou des rapports sont également fournis.

SECTION 9.1

Introduction

Nous allons fonder tous les exemples de ce chapitre sur le fichier de données nutriage.xls que vous pouvez importer dans R en suivant l'une des méthodes exposées dans le chapitre 2. Vous pourriez par exemple utiliser les instructions suivantes (pensez à installer le fichier `http://www.biostatisticien.eu/springeR/Rtools.exe` si vous travaillez sous l'environnement Microsoft) :

```
> require("gdata") # Donne accès à la fonction read.xls()
> nutriage <- read.xls(
+ "http://www.biostatisticien.eu/springeR/nutriage.xls")
```

```
> attach(nutriage)
> head(nutriage)
  sexe situation the cafe taille poids age viande poisson
1    2         1   0    0    151    58  72      4       3
2    2         1   1    1    162    60  68      5       2
3    2         1   0    4    162    75  78      3       1
4    2         1   0    0    154    45  91      0       4
5    2         1   2    1    154    50  65      5       3
6    2         1   2    0    159    66  82      4       2
  fruit_crus fruit_legume_cuits chocol matgras
1          1                  4      5       6
2          5                  5      1       4
3          5                  2      5       4
4          4                  0      3       2
5          5                  5      3       2
6          5                  5      1       3
```

Les données de ce tableau consistent en la mesure de treize variables sur 226 individus.

Nous supposerons, dans tout ce chapitre, que les données utilisées dans les différents exemples ont été structurées comme cela est indiqué dans la section suivante.

SECTION 9.2

Structuration des variables suivant leur type

Les treize variables du jeu de données NUTRIAGE peuvent être classées suivant leur type, en suivant l'algorithme décrit dans la figure suivante. Rappelons que le type d'une variable X se détermine par rapport à l'ensemble E_X des modalités **susceptibles d'être observées** et non pas sur la base des modalités réellement observées. En fait, le type d'une variable est défini (fait partie intégrante) du modèle mathématique que l'on choisit, parce qu'en l'état actuel des connaissances, c'est celui qui « représente » le mieux le phénomène que l'on veut étudier.

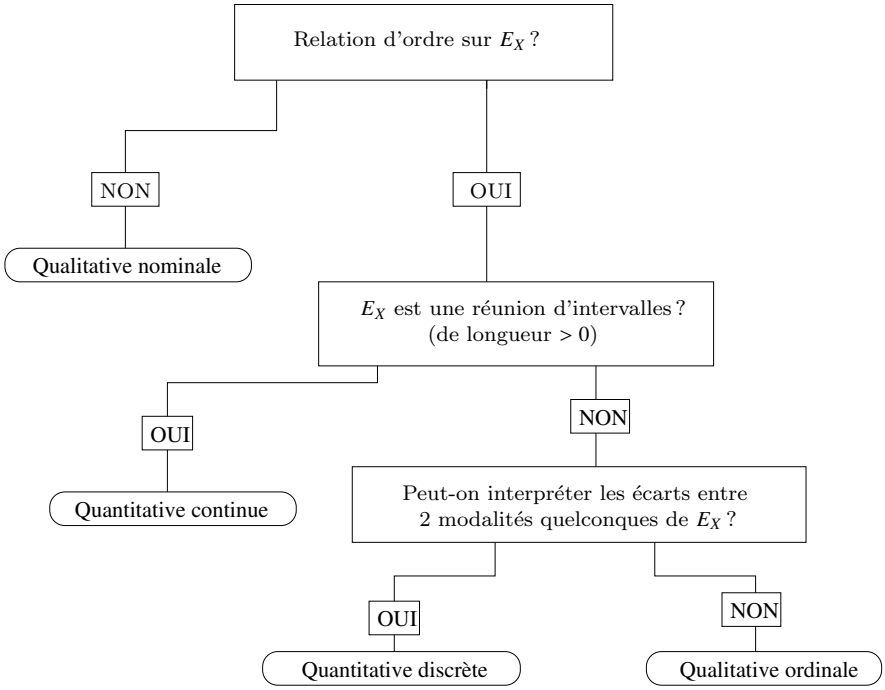


FIGURE 9.1 – Algorithme de détermination du type d'une variable.

En suivant cet algorithme, nous obtenons le tableau récapitulatif suivant :

Variables qualitatives	`sexe`, `situation` et `matgras`
Variables ordinales	`viande`, `poisson`, `fruit_crus`, `fruit_legume_cuits` et `chocol`
Variables quantitatives discrètes	`the` et `cafe`
Variables quantitatives continues	`taille`, `poids` et `age`

Nous allons commencer par imposer une structure **R** adaptée au type de chacune des variables précédentes.

9.2.1 Structurer les variables qualitatives

Pour les variables qualitatives, la structure est imposée au moyen de la fonction `as.factor()`. Il peut éventuellement être intéressant d'utiliser aussi la fonction `levels()` pour recoder les modalités d'une variable qualitative. Notez que l'association entre les codes et les *levels* se fait par ordre alphabétique (et pas par ordre de saisie).

Effectuons ces opérations sur les variables qualitatives de notre jeu de données.

```
> sexe <- as.factor(sexe)
> levels(sexe) <- c("Homme","Femme")
> situation <- as.factor(situation)
> levels(situation) <- c("seul","couple","famille","autre")
> matgras <- as.factor(matgras)
> levels(matgras) <- c("beurre","margarine","arachide",
+             "tournesol","olive","Isio4","colza","canard")
```

Notez que dans le cas où une variable serait codée en présence/absence, il est aussi possible d'utiliser une structuration R sous la forme d'un vecteur de logiques :

```
> fumeur <- c(1,0,0,1,0,1,0,1,0,0) # 10
                                   # fumeurs(=1)/non-fumeurs(=0).
> fumeur
 [1] 1 0 0 1 0 1 0 1 0 0
> fumeur <- as.logical(fumeur)
> fumeur
 [1]  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
[10] FALSE
```

Si en outre vous souhaitez associer un nom à vos modalités, vous pouvez procéder ainsi :

```
> fumeur <- c(fume=fumeur)
> fumeur
 fume1  fume2  fume3  fume4  fume5  fume6  fume7  fume8  fume9
  TRUE  FALSE  FALSE   TRUE  FALSE   TRUE  FALSE   TRUE  FALSE
fume10
 FALSE
```

Nous déconseillons toutefois le codage en logique des variables du type présence/absence, car cela nuit à leur utilisation subséquente dans certaines fonctions R, notamment pour les représentations graphiques.

Expert

Si vous partez d'une variable déjà structurée en `factor` telle que celle-ci :

```
> fumeur <- as.factor(c("Fumeur","NonFumeur","NonFumeur",
+                 "Fumeur","NonFumeur","Fumeur","NonFumeur",
+                 "Fumeur","NonFumeur","NonFumeur"))
> fumeur
 [1] Fumeur    NonFumeur NonFumeur Fumeur    NonFumeur
 [6] Fumeur    NonFumeur Fumeur    NonFumeur NonFumeur
Levels: Fumeur NonFumeur
```

vous pouvez procéder ainsi pour la coder en logiques :

```
> fumeur <- c(fume=as.logical(2-as.integer(fumeur)))
> fumeur
fume1  fume2  fume3  fume4  fume5  fume6  fume7  fume8  fume9
 TRUE  FALSE  FALSE   TRUE  FALSE   TRUE  FALSE   TRUE  FALSE
fume10
 FALSE
```

9.2.2 Structurer les variables ordinales

Pour les variables ordinales, la structure est imposée au moyen de la fonction `as.ordered()`. Il peut éventuellement être intéressant d'utiliser aussi la fonction `levels()` pour recoder les modalités d'une variable ordinale.

Effectuons ces opérations sur les variables ordinales de notre jeu de données :

```
> viande <- as.ordered(viande)
> poisson <- as.ordered(poisson)
> fruit_crus <- as.ordered(fruit_crus)
> fruit_legume_cuits <- as.ordered(fruit_legume_cuits)
> chocol <- as.ordered(chocol)
> niveaux <- c("jamais","< 1/sem.","1/sem.","2-3/sem.",
+              "4-6/sem.","1/jour")
> levels(chocol) <- levels(fruit_legume_cuits) <-
+                   levels(fruit_crus)  <- niveaux
> levels(poisson) <- levels(viande) <- niveaux
```

9.2.3 Structurer les variables quantitatives discrètes

Pour une variable discrète, la structure est imposée au moyen de la fonction `as.integer()`.

```
> the <- as.integer(the)
> cafe <- as.integer(cafe)
```

Cela n'est toutefois valable que si les données observées sont des entiers.

9.2.4 Structurer les variables quantitatives continues

Pour une variable continue, la structure est imposée au moyen de la fonction `as.double()`.

```
> taille <- as.double(taille)
> poids <- as.double(poids)
> age <- as.double(age)
```

SECTION 9.3

Tableaux de données

Pour un jeu de données particulier, les graphiques et les résumés numériques qu'il est possible de réaliser dépendent étroitement de la structure du tableau de données, et du type de la ou des variables en jeu. Dans ce chapitre, nous détaillons les possibilités offertes par R en ce qui a trait à l'organisation des données sous forme de tableaux.

9.3.1 Tableaux des données individuelles

Il s'agit du type d'organisation le plus courant. On dispose des mesures d'une ou de plusieurs variables pour chacun des N individus constitutifs d'une certaine population. Vous êtes priés de vous reporter au chapitre 2 pour savoir comment importer des données dans R. Les données sont en général organisées dans un *data.frame*.

9.3.2 Tableaux des effectifs ou des fréquences d'une variable

Il est souvent intéressant de représenter un tableau de données individuelles (ou tableau de données brutes) sous une forme plus condensée. Ainsi, le tableau des effectifs ou des fréquences (appelé aussi tri à plat) permet d'appréhender plus facilement la distribution d'une variable, notamment qualitative ou ordinale. Il s'obtient au moyen de la fonction `table()`.

```
> tpe <- table(matgras)       # Tri à plat en effectifs.
> tpe
matgras
   beurre margarine  arachide tournesol     olive     Isio4
       15        27        48        68        40        23
    colza    canard
        1         4
> tpf <- tpe/length(matgras)  # Tri à plat en fréquences.
> tpf
matgras
     beurre   margarine    arachide   tournesol       olive
0.066371681 0.119469027 0.212389381 0.300884956 0.176991150
      Isio4       colza      canard
0.101769912 0.004424779 0.017699115
> levels(matgras)             # Modalités.
[1] "beurre"    "margarine" "arachide"  "tournesol" "olive"
[6] "Isio4"     "colza"     "canard"
> nlevels(matgras)            # Nombre de modalités.
[1] 8
```

9.3.3 Tableaux de données regroupées en classes

Il est parfois intéressant de représenter un tableau de données individuelles (ou tableau de données brutes), récoltées sur une ou plusieurs variables quantitatives, sous une forme plus condensée. On utilise pour cela un tableau de données regroupées en classes, en notant les effectifs (ou les fréquences) de différentes classes préalablement déterminées.

Notez que vous pouvez utiliser la fonction `hist()` en spécifiant le vecteur des bornes des classes dans son paramètre `breaks` qui renvoie alors les effectifs de chacune de ces classes. La valeur par défaut du paramètre `breaks` (qui vaut `"Sturges"`) effectue un calcul automatique des classes.

```
> res <- hist(taille,plot=FALSE)
> nn <- as.character(res$breaks)
> x <- as.table(res$counts)
> dimnames(x) <- list(paste(nn[-length(nn)],nn[-1],sep="-"))
> x
140-145 145-150 150-155 155-160 160-165 165-170 170-175
      1       7      37      50      46      31      27
175-180 180-185 185-190
     17       4       6
> # Ou bien en utilisant la fonction cut():
> table(cut(taille,res$breaks,include.lowest=TRUE))
[140,145] (145,150] (150,155] (155,160] (160,165] (165,170]
        1         7        37        50        46        31
(170,175] (175,180] (180,185] (185,190]
       27        17         4         6
```

9.3.4 Tableaux croisant deux variables

9.3.4.1 Tableaux de contingence

Lorsque l'on dispose du tableau des données individuelles, on peut utiliser la fonction `table()` pour obtenir le tableau de contingence observé (encore appelé tri croisé en effectifs) du couple (X, Y).

```
> matable <- table(sexe,situation)
> matable
        situation
sexe     seul couple famille autre
  Homme    20     63       2     0
  Femme    78     56       7     0
```

Si l'on veut rajouter les marges à ce tableau, on peut utiliser la fonction `addmargins()`.

```
> table.complète <- addmargins(matable,FUN=sum,quiet=TRUE)
> table.complète
       situation
sexe     seul couple famille autre sum
  Homme    20     63       2     0  85
  Femme    78     56       7     0 141
  sum      98    119       9     0 226
```

Astuce

- Pour changer l'intitulé « sum » des marges des tableaux précédents, vous pouvez définir la fonction `Total()` (ainsi : `Total <- sum`) puis remplacer `sum` par `Total` dans l'appel de `addmargins()` ci-dessus.
- Lorsque le tableau de contingence est déjà fourni dans un fichier, nous avons vu dans le chapitre 2 comment importer ce type de fichier au moyen de la fonction `read.ftable()` et l'afficher au moyen de la fonction `ftable()`.

9.3.4.2 Distribution conjointe

Le tableau de la distribution conjointe (encore appelé tri croisé en fréquences relatives) du couple (X, Y) s'obtient à partir du tableau de contingence `matable` précédent.

```
> tableaufreq <- matable/sum(matable)
> tableaufreq
       situation
sexe           seul      couple     famille       autre
  Homme 0.088495575 0.278761062 0.008849558 0.000000000
  Femme 0.345132743 0.247787611 0.030973451 0.000000000
```

Si l'on veut aussi obtenir les marges, on peut utiliser l'une des instructions suivantes :

```
> Total <- sum
> addmargins(tableaufreq,FUN=Total,quiet=TRUE)
       situation
sexe           seul      couple     famille       autre
  Homme 0.088495575 0.278761062 0.008849558 0.000000000
  Femme 0.345132743 0.247787611 0.030973451 0.000000000
  Total 0.433628319 0.526548673 0.039823009 0.000000000
       situation
sexe          Total
  Homme 0.376106195
  Femme 0.623893805
  Total 1.000000000
```

```
> tableaufreqtotal <- table.complète/table.complète[
+                         length(table.complète)]
> tableaufreqtotal
       situation
sexe            seul      couple     famille       autre
  Homme 0.088495575 0.278761062 0.008849558 0.000000000
  Femme 0.345132743 0.247787611 0.030973451 0.000000000
  sum   0.433628319 0.526548673 0.039823009 0.000000000
       situation
sexe             sum
  Homme 0.376106195
  Femme 0.623893805
  sum   1.000000000
```

9.3.4.3 Distributions marginales

L'obtention des marges d'une table de distribution `tableaufreq` (ou d'une table de contingence) s'obtient au moyen de la fonction `margin.table()`.

```
> margin.table(tableaufreq,1) # Marge de droite.
sexe
    Homme     Femme
0.3761062 0.6238938
> margin.table(tableaufreq,2) # Marge du bas.
situation
      seul     couple    famille      autre
0.43362832 0.52654867 0.03982301 0.00000000
```

Attention

Lorsque la table de distribution (ou de contingence) est complète, c'est-à-dire qu'elle contient déjà les marges (comme pour `tableaufreqtotal`), il ne faut pas utiliser la fonction `margin.table()` pour les extraire, mais plutôt procéder ainsi :

```
> tableaufreqtotal[,ncol(tableaufreqtotal)] # Marge de droite.
    Homme     Femme       sum
0.3761062 0.6238938 1.0000000
> tableaufreqtotal[nrow(tableaufreqtotal),] # Marge du bas.
      seul     couple    famille      autre        sum
0.43362832 0.52654867 0.03982301 0.00000000 1.00000000
```

9.3.4.4 Distributions conditionnelles

Les tableaux des distributions conditionnelles s'obtiennent au moyen de la fonction `prop.table()`.

• Distributions conditionnelles de `situation` sachant les valeurs de `sexe` :

```
> prop.table(matable,1)
       situation
sexe          seul     couple    famille      autre
  Homme 0.23529412 0.74117647 0.02352941 0.00000000
  Femme 0.55319149 0.39716312 0.04964539 0.00000000
> addmargins(prop.table(matable,1),margin=2,FUN=sum) # Ajout du
                                                     # Total.
       situation
sexe          seul     couple    famille      autre        sum
  Homme 0.23529412 0.74117647 0.02352941 0.00000000 1.00000000
  Femme 0.55319149 0.39716312 0.04964539 0.00000000 1.00000000
```

• Distributions conditionnelles de `sexe` sachant les valeurs de `situation` :

```
> prop.table(matable,2)
       situation
sexe          seul    couple   famille autre
  Homme 0.2040816 0.5294118 0.2222222
  Femme 0.7959184 0.4705882 0.7777778
> addmargins(prop.table(matable,2),margin=1,FUN=sum) # Ajout du
                                                     # Total.
       situation
sexe          seul    couple   famille autre
  Homme 0.2040816 0.5294118 0.2222222
  Femme 0.7959184 0.4705882 0.7777778
  sum   1.0000000 1.0000000 1.0000000
```

SECTION 9.4

Résumés numériques

Nous présentons tous les résumés numériques sur le vecteur $\boldsymbol{x} = (x_1, \ldots, x_N)^\mathsf{T}$ pour plus de simplicité. Ce vecteur est l'ensemble des N valeurs de la variable X mesurées sur une population d'effectif N (cas standard de la statistique descriptive). Lorsque le vecteur $\boldsymbol{x}$ sera plutôt considéré comme un échantillon extrait, nous noterons sa longueur n. Les exemples d'application seront principalement fondés sur la série de données du vecteur `taille`.

Attention

Les résumés numériques ne peuvent être calculés en présence de données manquantes (`NA`). Si cela est nécessaire, il est possible d'utiliser la fonction `na.omit()` pour les retirer lors du calcul.

```
> x <- na.omit(taille) # Inutile ici car taille n'a pas de NA.
```

9.4.1 Résumés de position d'une distribution

9.4.1.1 Le (ou les) mode(s)

Les modes sont les valeurs de la variable X qui apparaissent le plus fréquemment. Ils peuvent se calculer pour une variable de n'importe quel type, bien que, pour une variable quantitative continue, on présente plutôt la classe modale. Notez que le mode peut être unique, auquel cas on parle de distribution unimodale, par opposition à des variables multimodales.

```
> tabthe <- table(the)
> names(which.max(tabthe))                    # Obtention d'un mode
                                              # unique.
[1] "0"
> names(tabthe)[max(tabthe)==tabthe] # Obtention de tous les
                                    # modes.
[1] "0"
```

Ici, la variable `the` (nombre de tasses de thé par jour) est unimodale.

Astuce

Dans le cas d'une variable quantitative, vous pouvez utiliser la fonction `as.numeric()` sur les résultats ci-dessus pour récupérer des valeurs numériques.

9.4.1.2 La médiane

La médiane d'une série statistique est la valeur m_e de la variable X qui partage cette série statistique en deux parties (inférieure et supérieure à m_e) de même effectif, les valeurs du caractère étant rangées dans l'ordre croissant. Il s'agit d'un critère de position qui ne se calcule évidemment pas pour des variables purement qualitatives. Pour la calculer, on distingue deux cas :

- l'effectif total N de la série est impair. Dans ce cas, la médiane est la valeur située à la position $\frac{N+1}{2}$;
- l'effectif total N de la série est pair. Dans ce cas, n'importe quelle valeur comprise entre les valeurs aux positions $\frac{N}{2}$ et $\frac{N}{2}+1$ peut être considérée comme une médiane de la série. En pratique, la médiane est généralement la moyenne de ces deux valeurs (cela exclut donc les caractères ordinaux de ce dernier calcul).

La fonction R permettant de calculer une médiane **uniquement pour des données numériques** est `median()`.

```
> median(x)
[1] 163
```

Astuce

Nous proposons le code suivant qui permettra de calculer la médiane pour des données **individuelles** ordinales ou numériques :

```
> ma.mediane <- function(x) {
+  if (is.numeric(x)) return(median(x))
+  if (is.ordered(x)) {
+   N <- length(x)
+   if (N%%2) return(sort(x)[(N+1)/2]) else {
+    inf <- sort(x)[N/2]
+    sup <- sort(x)[N/2+1]
+    if (inf==sup) return(inf) else return(c(inf,sup))
+   }}
+ stop("Calcul de médiane impossible pour ce type")
+ }
> ma.mediane(poisson)
[1] 2-3/sem.
6 Levels: jamais < < 1/sem. < 1/sem. < ... < 1/jour
```

Supposons maintenant que l'on dispose non pas des données individuelles, mais uniquement du tableau des fréquences observées $f_1, \ldots, f_k, \ldots, f_K$ dans les K classes $]e_0, e_1], \ldots,]e_{k-1}, e_k], \ldots,]e_{K-1}, e_K]$ d'une variable **quantitative**. Il faut alors procéder par interpolation linéaire entre les deux valeurs dont les fréquences cumulées encadrent 50 %. Notons e_k et e_{k+1} les deux valeurs consécutives des bornes de classe telles que $PFC_X(e_k) < 0.5 \leq PFC_X(e_{k+1})$, où $PFC_X(e_j) = f_1 + f_2 + \cdots + f_j$, $j = 1, \ldots, K$ ($PFC_X(x)$ est la valeur du polygone des fréquences cumulées au point x). Alors on a :

$$m_e = e_k + (e_{k+1} - e_k)\frac{0.5 - PFC_X(e_k)}{PFC_X(e_{k+1}) - PFC_X(e_k)}.$$

Nous pouvons proposer le programme suivant permettant de calculer la médiane dans ce cas :

```
> mediane.sur.freq <- function(x) {
+  # x est le tableau des fréquences.
+  tmp <- suppressWarnings(as.double(names(x)))
+  if (!(is.null(tmp) | all(is.na(tmp)))) {
+   tab.freq.cum <- cumsum(x)
+    index <- order(tab.freq.cum < 0.5)[1]
+    fc1 <- tab.freq.cum[index]
+    fc2 <- tab.freq.cum[index-1]
+    x1 <- as.numeric(names(fc1))
+    x2 <- as.numeric(names(fc2))
+    mex <- as.numeric(x1 + (x2-x1)*(0.5-fc1)/(fc2-fc1))
+    } else {mex <- NA}
```

```
+ return(mex)
+ }
```

Voici le calcul de la médiane pour les données du vecteur `x` regroupées en classes au moyen de la fonction `hist()` :

```
> res <- hist(x,plot=FALSE,breaks=c(130,150,160,170,180,190))
> tab.x <- table(rep(res$breaks[-1],res$counts))
> tab.x
150 160 170 180 190
  8  87  77  44  10
> mediane.sur.freq(tab.x/sum(tab.x))
[1] 162.3377
```

9.4.1.3 La moyenne

Elle se calcule uniquement pour des variables quantitatives.

```
> mean(x)       # μ_X = 1/N ∑_{i=1}^n x_i
[1] 163.9602
```

9.4.1.4 Les fractiles

Le fractile d'ordre p ($0 < p < 1$) est la valeur q_p de la variable X qui coupe l'échantillon en deux portions, l'une ayant un nombre d'éléments égal à p % du nombre total d'éléments dans $\boldsymbol{x}$ (ce sont les éléments inférieurs à q_p), l'autre à $(1-p)$ % (ce sont les éléments supérieurs à q_p). Il ne se calcule pas pour des variables purement qualitatives.

Fractiles d'ordre 10 % et 90 % :

```
> quantile(x,probs=c(0.1,0.9))
10% 90%
153 176
```

Quartiles $q_{1/4}, q_{1/2}, q_{3/4}$ (aussi notés $q_1, q_2 = m_e, q_3$) :

```
> quantile(x,probs=c(0.25,0.5,0.75))
25% 50% 75%
157 163 170
```

Déciles :

```
> quantile(x,probs=1:10/10)
  10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
153.0 156.0 158.5 160.0 163.0 165.0 168.0 172.0 176.0 188.0
```

Attention

La fonction `quantile()` contient un argument `type` pouvant prendre une valeur entière (entre 1 et 9) sélectionnant l'un des neuf algorithmes qui sont détaillés dans le fichier d'aide de cette fonction.

Astuce

La fonction `summary()` appliquée à un vecteur de données quantitatives permet de calculer le minimum, le maximum, la moyenne et les trois quartiles.

9.4.2 Résumés de dispersion d'une distribution

Ces résumés peuvent être calculés uniquement pour des variables quantitatives. Nous les présentons dans le tableau ci-dessous après avoir défini les trois fonctions R suivantes :

```
> # Variance σ² de la population σ²(x) = 1/N Σ_{i=1}^N (x_i − μ_X)² .
> var.pop <- function(x) var(x)*(length(x)-1)/length(x)
> # Écart type σ de la population σ(x) = √(1/N Σ_{i=1}^N (x_i − μ_X)²) .
> sd.pop <- function(x) sqrt(var.pop(x))
> # Coefficient de variation:
  # c_v = σ/μ_X .
> co.var <- function(x) sd.pop(x)/mean(x)
```

Nom	Formule mathématique	Instruction R	Résultat
Étendue	$\max_{1\leq i\leq N}(x_i) - \min_{1\leq i\leq N}(x_i)$	`max(x)-min(x)` `diff(range(x))`	48.0
Intervalle inter-quartiles	$q_{3/4} - q_{1/4}$	`IQR(x)`	13.0
Variance	$\sigma^2_{Pop}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^N (x_i - \overline{x})^2$	`var.pop(x)`	80.702
Écart type	$\sigma_{Pop}(\boldsymbol{x}) = \sqrt{\frac{1}{N}\sum_{i=1}^N (x_i - \overline{x})^2}$	`sd.pop(x)`	8.983
Coefficient de variation	$c_v = \frac{\sigma_{Pop}}{\mu_X}$	`co.var(x)`	0.055
Déviation absolue par rapport à la médiane	$\frac{1}{N}\sum_{i=1}^N \lvert x_i - me_X\rvert$	`mad(x)`	10.378
Écart moyen	$\frac{1}{N}\sum_{i=1}^N \lvert x_i - \overline{x}\rvert$	`mean(abs(x-mean(x)))`	7.312

Remarque

Une estimation sans biais $\hat{\sigma}^2$ de la variance σ^2 de la population, fondée sur un échantillon de taille n, est calculée au moyen de la fonction `var()`. L'écart type $\hat{\sigma}$ correspondant est calculé au moyen de la fonction `sd()`.

```
> var(x)          # σ̂²(x) = 1/(n-1) Σ_{i=1}^n (x_i - x̄)²
                  # avec x̄ = 1/n Σ_{i=1}^n x_i .
[1] 81.06063
> sd(x)           # σ̂(x) = √(1/(n-1) Σ_{i=1}^n (x_i - x̄)²)
                  # avec x̄ = 1/n Σ_{i=1}^n x_i .
[1] 9.003368
```

9.4.3 Résumés de forme d'une distribution

Ces résumés peuvent être calculés uniquement pour des variables quantitatives. Nous pouvons mentionner les coefficients d'asymétrie (*skewness*) et d'aplatissement (*kurtosis*) dont nous fournissons le code **R** ci-dessous.

```
> skew <- function(x) mean((x-mean(x))^3)/sd.pop(x)^3
> skew(x)     # γ1 = μ3/σ³
              # avec μ3 = 1/N Σ_{i=1}^N (x_i - μ_X)³ .
[1] 0.4256203
> kurt <- function(x) mean((x-mean(x))^4)/sd.pop(x)^4
> kurt(x)     # β2 = μ4/σ⁴
              # avec μ4 = 1/N Σ_{i=1}^N (x_i - μ_X)⁴ .
[1] 2.778185
```

Astuce

Les fonctions `skewness()` et `kurtosis()` du *package* `moments` permettent d'effectuer les mêmes opérations.

SECTION 9.5

Mesures d'association

9.5.1 Mesures de liaison entre deux variables qualitatives

9.5.1.1 La statistique du χ^2 de Pearson

Les résultats de cette partie s'obtiennent au moyen de la fonction `chisq.test()`. Nous présentons le calcul du tableau de contingence des effectifs observés O_{ij}, du tableau de contingence des effectifs théoriques E_{ij} $(1 \leq i \leq p, 1 \leq j \leq q)$ (encore appelé tableau d'indépendance), du tableau des contributions au χ^2 ainsi que le calcul du χ^2.

```
> sexematgras <- table(sexe,matgras)    # Tableau de contingence
                                        # observé des
```

```
                                          # O_ij.
> tab.ind <- chisq.test(sexematgras)$expected  # Les E_ij.
> round(tab.ind)
       matgras
sexe    beurre margarine arachide tournesol olive Isio4 colza
  Homme      6        10       18         26    15     9     0
  Femme      9        17       30         42    25    14     1
       matgras
sexe    canard
  Homme      2
  Femme      2
> # (sexematgras-tab.ind)^2/tab.ind:
  # (O_ij-E_ij)^2/E_ij.
> tab.contr <- chisq.test(sexematgras)$residuals^2
> tab.contr
       matgras
sexe         beurre   margarine    arachide   tournesol
  Homme 3.367083116 0.002361810 0.233489502 0.818473834
  Femme 2.029801879 0.001423786 0.140756083 0.493406212
       matgras
sexe          olive       Isio4       colza      canard
  Homme 1.632483082 1.540468053 0.376106195 1.486777720
  Femme 0.984121007 0.928650954 0.226730685 0.896284441
> khi2 <- chisq.test(sexematgras)$statistic   # sum(tab.contr)
> khi2              # X^2 = sum_ij (O_ij-E_ij)^2/E_ij.
X-squared
 15.15842
```

Astuce

Une autre manière de calculer la statistique du χ^2 repose sur l'utilisation de la fonction `summary()`.

```
> khi2 <- summary(table(sexe,matgras))$statistic
> khi2
[1] 15.15842
```

9.5.1.2 Φ^2, V de Cramér et coefficient de contingence de Pearson

Tous les indicateurs du tableau suivant se calculent à partir du coefficient du χ^2.

```
> N <- sum(sexematgras)
> p <- nrow(sexematgras)
> q <- ncol(sexematgras)
```

Indicateur	Formule	Instruction R	Résultat
Φ^2	$\Phi^2 = \frac{\chi^2}{N}$	`Phi2 <- khi2/N`	0.067
V^2 de Cramér	$V^2 = \frac{\Phi^2}{\min(p-1,q-1)}$	`Phi2/(min(p,q)-1)`	0.067
Coefficient de contingence de Pearson	$C = \sqrt{\frac{\chi^2}{(N+\chi^2)}}$ $= \sqrt{\frac{\Phi^2}{(1+\Phi^2)}}$	`sqrt(khi2/(N+khi2))`	0.251

Astuce

La fonction `cramer.v()` du *package* `rgrs` permet aussi de calculer le V de Cramér.

```
> require("rgrs")
> cramer.v(sexematgras)^2
[1] 0.06707265
```

9.5.2 Mesures de liaison entre des variables ordinales (ou des rangs)

9.5.2.1 Le τ et le τ_b de Kendall

Ce coefficient est fondé sur la notion de concordance entre individus. Pour deux individus i et j et pour les variables ordinales X et Y ayant p et q modalités respectivement, on dit que les paires (x_i, y_i) et (x_j, y_j) sont concordantes si $\text{sign}(x_j - x_i) = \text{sign}(y_j - y_i)$ et discordantes si $\text{sign}(x_j - x_i) = -\text{sign}(y_j - y_i)$. Si $x_i = x_j$ ou $y_i = y_j$ (ou les deux), la paire correspondante n'est ni concordante ni discordante, et nous disons qu'il s'agit d'*ex æquo*. S'il y a n_c paires concordantes, n_d paires discordantes et n_e *ex æquo*, alors $n_c + n_d + n_e = \frac{1}{2}N(N-1)$. On calcule alors le τ_b de Kendall par la formule :

$$\tau_b = \frac{2(n_c - n_d)}{\sqrt{(N^2 - \sum_{k=1}^{p} n_{k\bullet}^2)(N^2 - \sum_{l=1}^{q} n_{\bullet l}^2)}}$$

avec $2(n_c - n_d) = \sum_{k=1}^{p}\sum_{l=1}^{q} \text{sign}(x_k - x_l)\text{sign}(y_k - y_l)$, $n_{k\bullet} = \sum_{l=1}^{q} n_{kl}$ et $n_{\bullet l} = \sum_{k=1}^{p} n_{kl}$, où n_{kl} est le nombre d'individus ayant à la fois la modalité k pour X et l pour Y.

Lorsqu'il n'y a pas d'*ex æquo*, cette formule se simplifie en ce que l'on appelle le τ de Kendall :

$$\tau = \frac{2(n_c - n_d)}{N(N-1)}.$$

Ces deux quantités se calculent en R au moyen de la fonction `cor()`.

```
> cor(as.numeric(viande),as.numeric(poisson),method="kendall")
[1] -0.1583088
```

Astuce

Notez qu'il est possible de programmer soi-même ces coefficients en traduisant simplement la formule de τ_b en instructions R :

```
> Kendall.taub <- function(x,y) {
+  a1 <- sign(outer(as.numeric(x),as.numeric(x),"-"))
+  a2 <- sign(outer(as.numeric(y),as.numeric(y),"-"))
+  num <- sum(a1*a2)
+  N <- length(x)
+  b1 <- sum(margin.table(table(x,y),1)^2)
+  b2 <- sum(margin.table(table(x,y),2)^2)
+  denom <- sqrt((N^2-b1)*(N^2-b2))
+  taub <- num / denom
+ return(taub)
+ }
> Kendall.taub(viande,poisson)
[1] -0.1583088
```

9.5.2.2 Coefficient ρ de corrélation des rangs de Spearman

Il faut commencer par calculer les rangs (fonction `rank()`) des individus pour la variable X (notés x_i) et pour la variable Y (notés y_i). En cas d'*ex æquo*, il faut assigner comme même rang aux valeurs égales la moyenne de leurs positions dans l'ordre croissant des valeurs (ce que fait la fonction `rank()` par défaut).

Lorsqu'il n'y a pas d'*ex æquo*, le coefficient de corrélation des rangs de Spearman est donné par la formule suivante :

$$\rho = 1 - \frac{6\sum_{i=1}^{N} d_i^2}{N(N^2-1)} \quad \text{avec } d_i = x_i - y_i.$$

En présence de valeurs *ex æquo*, il faut utiliser le coefficient de corrélation classique de Pearson **entre les rangs** :

$$\rho = \frac{N(\sum_{i=1}^{N} x_i y_i) - (\sum_{i=1}^{N} x_i)(\sum_{i=1}^{N} y_i)}{\sqrt{N(\sum_{i=1}^{N} x_i^2) - (\sum_{i=1}^{N} x_i)^2}\sqrt{N(\sum_{i=1}^{N} y_i^2) - (\sum_{i=1}^{N} y_i)^2}}.$$

Pour l'obtenir sous R, on peut donc utiliser les fonctions `rank()` et `cor()`, ou bien directement la fonction `cor()` avec la valeur d'entrée `"spearman"` de son paramètre `method`.

```
> cor(rank(matgras),rank(situation))
[1] 0.008787643
> cor(as.numeric(matgras),as.numeric(situation),method="spearman")
[1] 0.008787643
```

9.5.3 Mesures de liaison entre deux variables quantitatives

9.5.3.1 Covariance et coefficient de corrélation de Pearson

L'indicateur de liaison approprié dans le cas de deux variables quantitatives est la corrélation. Il est défini comme le rapport entre la covariance des deux variables et le produit de leurs écarts types respectifs. Il se calcule au moyen de la fonction `cor()`.

```
> cor(taille,poids)
[1] 0.6306576
```

La covariance se calcule au moyen de la fonction `cov()`.

```
> cov(taille,poids)
[1] 68.32596
```

Astuce

Ce coefficient peut aussi se calculer avec la fonction `cor.test()`.

```
> cor.test(taille,poids)$estimate
      cor
0.6306576
```

Attention

Cet indicateur est pertinent si la relation unissant les deux variables est de type linéaire. Dans le cas contraire, il faut utiliser une mesure de dépendance plus générale, comme par exemple la *distance correlation* disponible dans le *package* `IndependenceTests`.

9.5.4 Mesures de liaison entre une variable quantitative et une variable qualitative

9.5.4.1 Le rapport de corrélation $\eta^2_{Y|X}$

Le rapport de corrélation $\eta^2_{Y|X}$ indique dans quelle mesure les variations d'une variable quantitative Y sont expliquées par les modalités d'une variable qualitative X à p modalités. En effet, on peut considérer que la variable X définit des groupes dans la population. Le rapport de corrélation est alors défini comme le rapport entre la variance inter-groupes et la variance intra-groupe. Il se calcule au moyen de la formule suivante :

$$\eta^2_{Y|X} = \frac{\sum_{k=1}^{p} n_k(\bar{y}_k - \bar{y})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

dans laquelle n_k désigne le nombre d'observations y_i correspondant à la k-ième modalité de X.

Voici le programme R permettant de le calculer :

```
> eta2 <- function(x, gpe) {
+  moyennes <- tapply(x, gpe, mean)
+  effectifs <- tapply(x, gpe, length)
+  varinter <- (sum(effectifs * (moyennes - mean(x))^2))
+  vartot <- (var(x) * (length(x) - 1))
+  res <- varinter/vartot
+ return(res)
+ }
```

Calculons-le par exemple pour les variables `poids` et `sexe`.

```
> eta2(poids,sexe)
[1] 0.3325501
```

Astuce

Vous pouvez aussi utiliser l'instruction R suivante pour calculer $\eta^2_{Y|X}$:

```
> summary(lm(poids~sexe))$r.squared
[1] 0.3325501
```

SECTION 9.6

Représentations graphiques

Rappelons qu'il convient toujours de choisir adéquatement le mode de représentation graphique d'une variable adapté à son type. En effet, **le type d'une variable est souvent traduit par des caractéristiques particulières d'un graphique** donné. Par exemple, l'ajout d'une pointe de flèche au bout d'un axe indique un ordre sur les modalités correspondantes. Il conviendra donc d'indiquer le sens de l'axe des modalités pour toutes les variables sauf pour celles qui sont du type qualitatif. Nous avons par conséquent défini, et intégré au *package* associé à ce livre, la fonction `fleches()` qui nous permettra d'ajouter, lorsque cela sera nécessaire, une flèche sur les axes d'un graphique. Par ailleurs, nous avons choisi de présenter côte à côte, pour quelques graphiques, une version basique et une version esthétiquement plus élaborée de ceux-ci. Bien que nous ne conseillions pas nécessairement l'utilisation de ces versions élaborées dans une phase d'exploration des données, ces dernières ont l'avantage d'illustrer les grandes possibilités offertes par R pour modifier à sa guise un graphique.

9.6.1 Graphiques pour les variables qualitatives

9.6.1.1 Diagramme en croix

Le diagramme en croix affiche pour chaque observation une petite barre horizontale dans la colonne de la modalité correspondante. Il n'est pas intégré au logiciel **R**, mais nous pouvons le programmer au moyen de la fonction `plot()` et de son paramètre `pch`. La fonction obtenue, que nous avons nommée `diagcroix()`, est intégrée au *package* associé au livre.

```
> diagcroix(situation,col=c("orange","darkgreen","black","tan"))
```

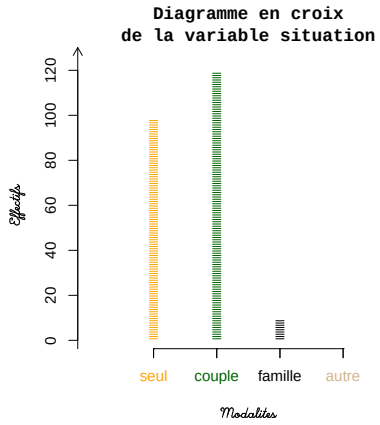


FIGURE 9.2 – Diagramme en croix pour une variable qualitative.

Notons au passage l'existence de la fonction `dotchart()` dont l'utilisation, couplée avec la fonction `table()`, offre un type d'affichage légèrement similaire, si ce n'est qu'il représente plutôt le tableau des effectifs de la variable.

```
> dotchart(table(situation),col=c("orangered","darkgreen",
+  "turquoise","tan"),pch=15,main=paste("Diagramme en points des
+   effectifs","de la variable situation",sep="\n"),lcolor="red")
```

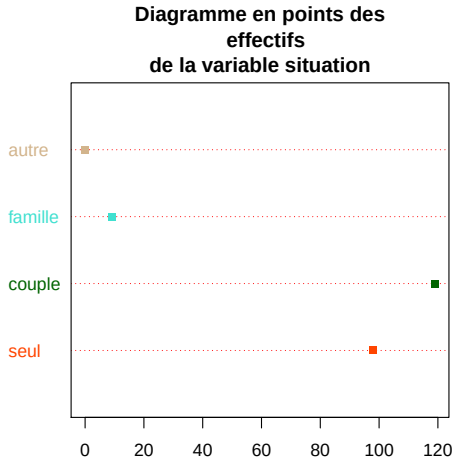


FIGURE 9.3 – Diagramme en points pour une variable qualitative.

9.6.1.2 Diagramme en tuyaux d'orgue

Son obtention se fait au moyen de la fonction `barplot()`. Afin d'améliorer un peu la qualité esthétique du graphique, nous proposons la fonction `tuyauxorgue()`, présente dans le *package* associé à ce livre.

```
> col <- c("gray","orangered","lightgoldenrodyellow","red")
> barplot(table(situation),col=col)
```

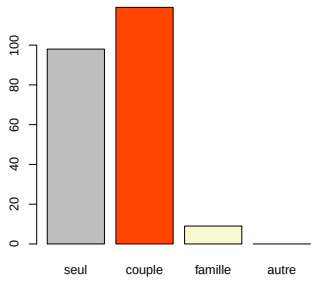
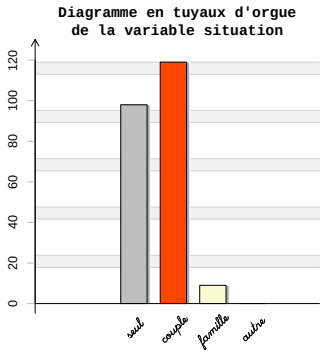


FIGURE 9.4 – Diagramme en tuyaux d'orgue pour une variable qualitative.

```
> tuyauxorgue(situation,col)
```



9.6.1.3 Diagramme de Pareto

Son obtention se fait également au moyen de la fonction `barplot()`, puisqu'il s'agit d'un diagramme en tuyaux d'orgue dont les tuyaux sont représentés par hauteur décroissante.

```
> col <- c("yellow","yellow2","sandybrown","orange",
+    "darkolivegreen","green","olivedrab2","green4")

> barplot(sort(table(matgras),TRUE),col=col)
```

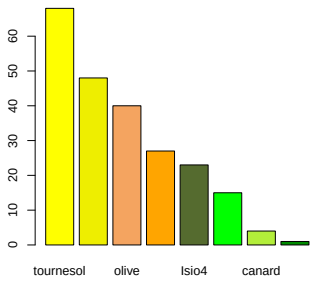
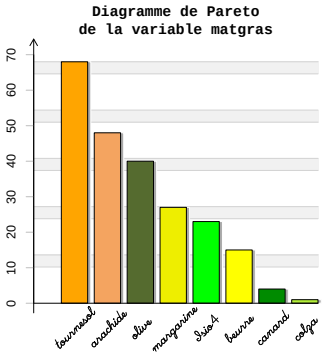


FIGURE 9.5 – Diagramme de Pareto pour une variable qualitative.

```
> tuyauxorgue(matgras,col,pareto=TRUE)
```



9.6.1.4 Diagramme empilé

Il s'obtient au moyen de la fonction `barplot()` en fournissant un objet du type `matrix` comme premier paramètre effectif.

```
> nbh <- table(sexe)[1]
> nbf <- table(sexe)[2]
> tabfreq.matgras.hommes <- table(matgras[sexe=="Homme"])/nbh
> tabfreq.matgras.femmes <- table(matgras[sexe=="Femme"])/nbf
> barplot(cbind(tabfreq.matgras.hommes,tabfreq.matgras.femmes),
+          main="Diagramme empilé de la variable matgras",col=
+          c("yellow","yellow2","sandybrown","orange",
+          "darkolivegreen","green","olivedrab2","green4"),xlim=
+          c(0,1),width=0.15,space=1,names.arg=
+          c("Hommes","Femmes"),legend=TRUE,density=40)
> fleches(x=FALSE,y=TRUE)
```

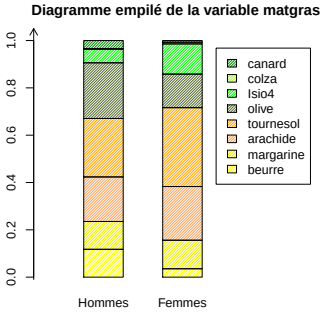


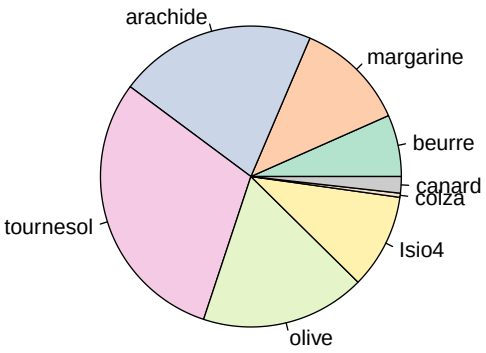
FIGURE 9.6 – Diagramme empilé pour une variable qualitative.

9.6.1.5 Diagramme circulaire

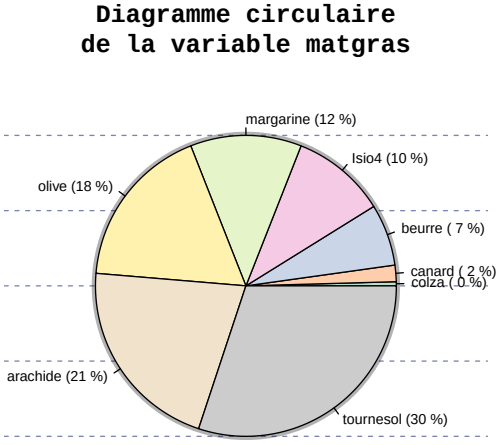
Il s'obtient au moyen de la fonction `pie()`. Afin d'améliorer un peu la qualité esthétique du graphique, nous proposons d'utiliser la fonction `camembert()`, présente dans le *package* associé à ce livre.

```
> require("RColorBrewer")
> col <- brewer.pal(8,"Pastel2")

> pie(table(matgras),col=col)
```




```
> camembert(matgras,col)
```



9.6.2 Graphiques pour les variables ordinales

9.6.2.1 Diagramme en tuyaux d'orgue avec courbe des fréquences cumulées

Il s'obtient au moyen des fonctions `barplot()` et `points()`. Ici encore, l'utilisation de la fonction `tuyauxorgue()` présente dans le *package* associé à ce livre permet une représentation différente (avec notamment la courbe des fréquences cumulées apparaissant en relief avec une ombre projetée).

```
> require("RColorBrewer")
> col <- brewer.pal(6,"Blues")
> tx <- table(poisson)
> tx <- tx/sum(tx)
> r <- barplot(tx,ylim=c(0,1),col=col)
> points(r,cumsum(tx),type="l")
```

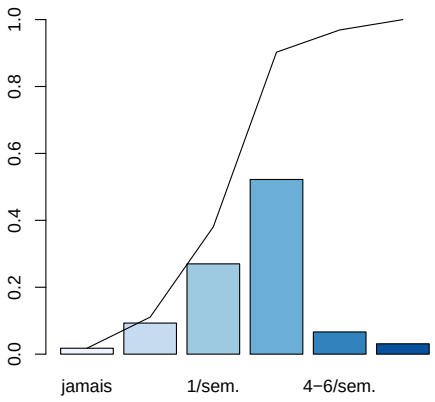


FIGURE 9.7 – Tuyaux d'orgue avec courbe des fréquences cumulées pour une variable ordinale.

9.6.3 Graphiques pour les variables quantitatives discrètes

9.6.3.1 Diagramme en croix

Il s'obtient au moyen de la fonction `diagcroix()`, présente dans le *package* associé à ce livre.

Renvoi

Voir la partie sur les variables qualitatives, page 387.

9.6.3.2 Diagramme en bâtons

Il s'obtient au moyen de la fonction `plot()` appliquée à une table de contingence.

```
> plot(tabthe/length(unique(the)),ylab="",
+          col="darkolivegreen",lwd=5,
+   main="Diagramme en bâtons de la variable thé")
> fleches()
```

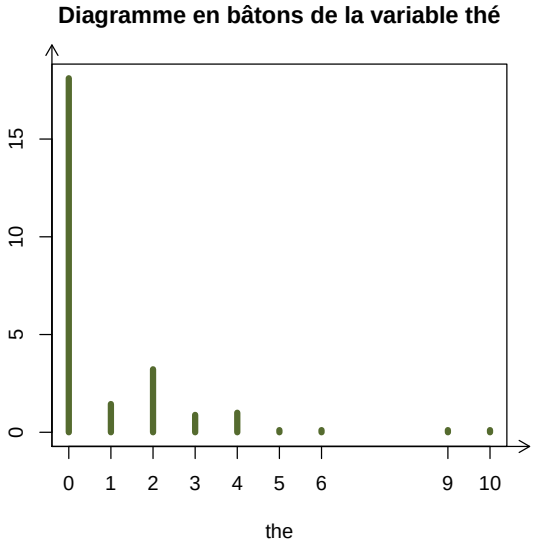


FIGURE 9.8 – Diagramme en bâtons pour une variable quantitative discrète.

9.6.3.3 Graphe de la fonction de répartition empirique

Il s'obtient au moyen des fonctions `plot()` et `ecdf()`.

```
> plot(ecdf(na.omit(cafe)),main=paste("Fonction de répartition
+   empirique","de la variable café",sep="\n"),verticals=TRUE,
+   ylab=expression(F[n](x)),col.01line="#89413A",col.points=
+   "#6D1EFF",col.hor='
> fleches(x=TRUE,y=TRUE)
```

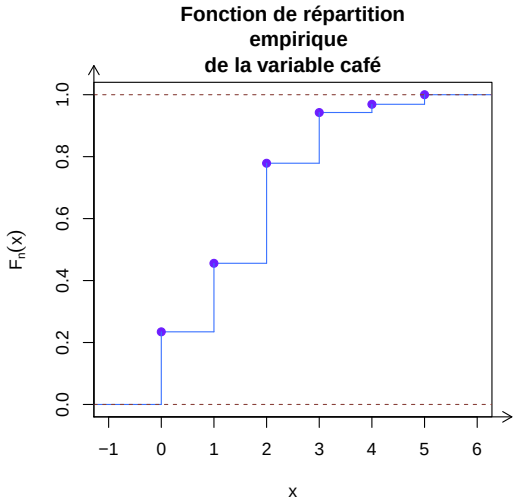


FIGURE 9.9 – Graphe de la fonction de répartition empirique pour une variable quantitative discrète.

9.6.3.4 Diagramme en tiges et feuilles

Il s'obtient au moyen de la fonction `stem.leaf()` du *package* `aplpack`.

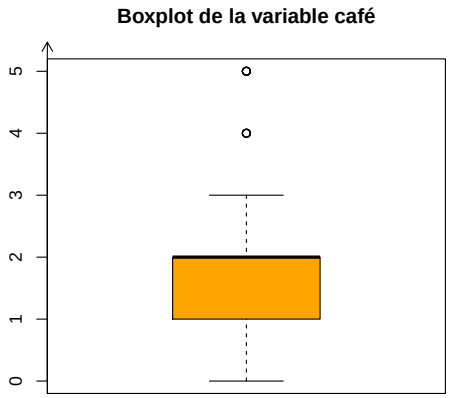
Renvoi

Voir la partie sur les variables quantitatives continues, page 397.

9.6.3.5 Boîte à moustaches (*boxplot*)

Afin de tracer un diagramme en boîte à moustaches, il faut utiliser la fonction `boxplot()` qui produit le graphique ci-après. Le schéma en explicite la lecture.

```
> boxplot(cafe,col="orange",
+ main="Boxplot de la variable café")
> fleches(x=FALSE,y=TRUE)
```



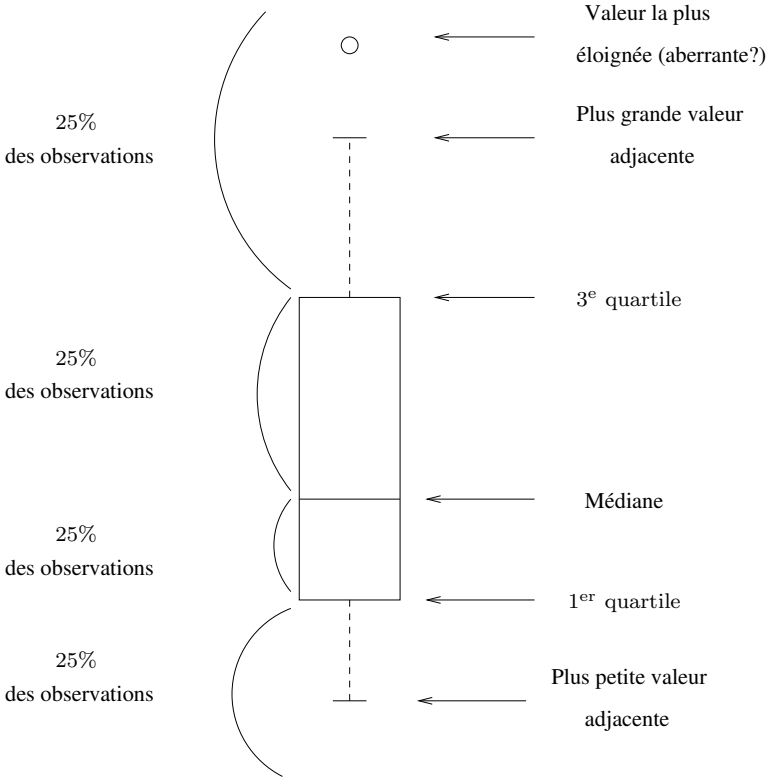


FIGURE 9.10 – Boîte à moustaches et explications associées.

La boîte est tracée en se servant des valeurs des trois quartiles. Notez que les valeurs repérées par des petits cercles sont des valeurs hors norme, éventuellement suspectes ou aberrantes (`boxplot(cafe)$out`). Ces valeurs extrêmes sont celles qui se situent à l'extérieur de la boîte, au-delà d'une distance de 1.5 fois l'intervalle inter-quartiles (le paramètre `range` permet de modifier cette valeur par défaut 1.5). Notez également que les valeurs se situant à l'extérieur de la boîte, mais à une distance en deçà de 1.5 fois l'intervalle inter-quartile sont des valeurs dites adjacentes. Les deux moustaches sont tracées respectivement à la plus grande et à la plus petite valeur adjacente.

Astuce

Tapez `example(bxp)` ou `example(boxplot)` pour d'autres exemples de ce type de diagramme.

9.6.4 Graphiques pour les variables quantitatives continues

Nous présentons maintenant quelques graphiques utiles pour l'exploration de données quantitatives.

9.6.4.1 Graphe de la fonction de répartition empirique

Il faut utiliser les fonctions `plot()` et `ecdf()`.

```
> plot(ecdf(na.omit(age)),main=paste("Fonction de répartition
+  empirique","de la variable age",sep="\n"),col.hor='# 3971FF',
+  col.points='#3971FF')
> fleches()
```

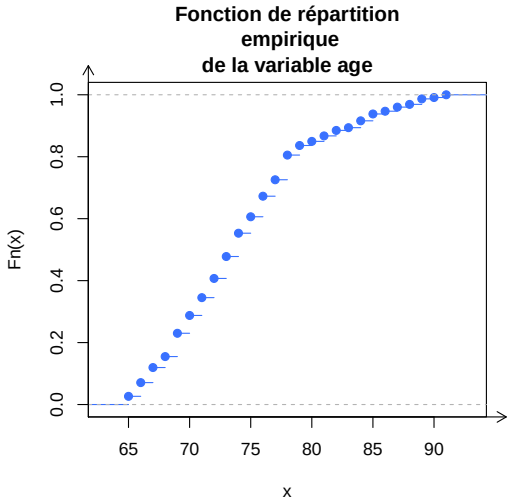


FIGURE 9.11 – Graphe de la fonction de répartition empirique pour une variable quantitative continue.

9.6.4.2 Diagramme en tiges et feuilles

On peut utiliser la fonction `stem()`, mais elle est assez limitée et nous conseillons plutôt l'utilisation de la fonction plus évoluée `stem.leaf()` du *package* `aplpack`. La construction du diagramme se fait en trois étapes :

- choix d'une unité de tige au moyen du paramètre `unit` ;
- nombre de parties (1, 2 ou 5) en lesquelles chaque tige sera divisée via le paramètre `m` ;
- choix d'un style de représentation en utilisant le paramètre `style` qui peut prendre les valeurs `"Tukey"` ou `"bare"`.

```
> require("aplpack")
> stem.leaf(taille,m=5,style="bare")
1 | 2: represents 12
 leaf unit: 1
            n: 226
    1     14 | 0
          14 |
          14 |
          14 |
    3     14 | 88
   12     15 | 000001111
```

```
  24    15 | 222223333333
  45    15 | 444444444444555555555
  60    15 | 666666666667777
  73    15 | 8888888899999
  97    16 | 00000000000000000000011
(22)    16 | 22222222223333333333
 107    16 | 444444445555555555555
  85    16 | 6666777
  78    16 | 88888888888999
  64    17 | 000000000011111
  49    17 | 2222222222333
  36    17 | 444555555
  27    17 | 6666666777
  17    17 | 88889
  12    18 | 0011
   8    18 | 22
       18 |
   6    18 | 666
   3    18 | 888
```

9.6.4.3 Boîte à moustaches

Elle s'obtient au moyen de la fonction `boxplot()`.

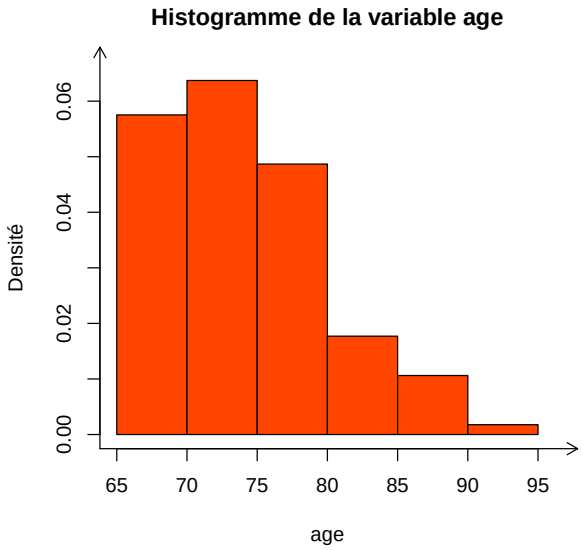
Renvoi

Voir la partie sur les variables quantitatives discrètes, page 394.

9.6.4.4 Histogramme en densité à amplitudes de classes égales ou inégales

Il faut utiliser la fonction `hist()`. Notez que par défaut, R construit des classes de la forme $]a;b]$ (« standard » anglo-saxon), alors qu'en France l'habitude est plutôt de faire des classes $[a;b[$.

```
> classes <- hist(age,right=TRUE,freq=FALSE,ylab="Densité",
+  main="Histogramme de la variable age",col="orangered")
>  fleches()
```




```
> classes <- hist(poids,right=TRUE,freq=FALSE,
+  main="Histogramme de la variable poids",
+  ylab="Densite",breaks=c(min(poids),50,80,
+            90,max(poids)),col="olivedrab")
> fleches()
```

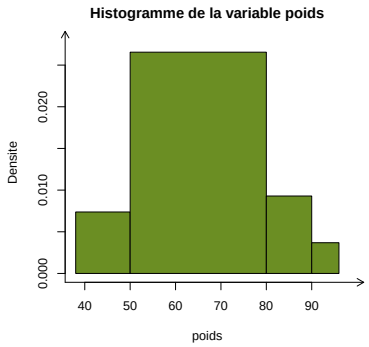


FIGURE 9.12 – Histogramme en densité à amplitudes de classes égales ou inégales.

9.6.4.5 Polygone des fréquences

On utilise les fonctions `hist()` et `segments()`.

```
> classes <- hist(taille,right=TRUE,freq=FALSE,
+ main=paste("Histogramme et polygone des fréquences",
+             "de la variable taille",sep="\n"),col="orangered")
> milieux <- classes$mid ; mlon <- length(milieux)
> densites <- classes$density
> segments(milieux[1:mlon-1],densites[1: mlon-1],
+          milieux[2:mlon],densites[2:mlon],col=
+          rgb(0.4196078,0.4196078,0.1372549,0.9),lwd=3)
> fleches()
```

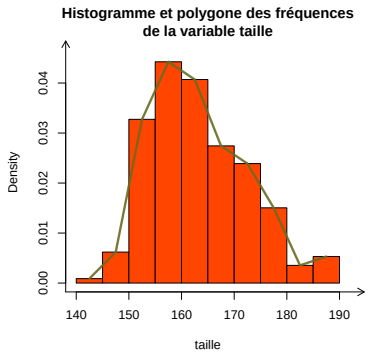


FIGURE 9.13 – Polygone des fréquences.

9.6.4.6 Polygone des fréquences cumulées

On utilise les fonctions `hist()`, `ecdf()` et `plot()`.

```
> bornes <- hist(taille,right=TRUE,plot=FALSE)$breaks
> plot(bornes,ecdf(taille)(bornes),type="l",main=paste("Polygone
+   des fréquences cumulées","de la variable taille",sep="\n"),
+   ylab="Fréquences",col="darkolivegreen",lwd=3)
> fleches()
```

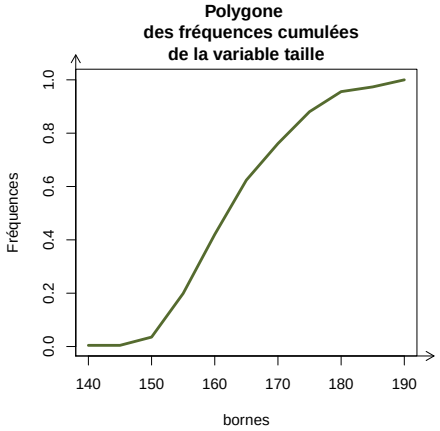


FIGURE 9.14 – Polygone des fréquences cumulées.

Astuce

Il est possible d'évaluer graphiquement la médiane sur ce graphique. Pour cela, il suffit de rajouter une ligne horizontale à la hauteur $h = 0.5$ au moyen de l'instruction `abline(h=0.5)`. Ensuite, vous entrez l'instruction `locator(1)$x` puis vous cliquez sur le point d'intersection entre cette ligne horizontale et la courbe du polygone des fréquences cumulées. On peut bien entendu obtenir les autres quantiles en remplaçant la valeur 0.5 correspondant à la médiane par la valeur désirée.

9.6.5 Représentations graphiques dans un cadre bivarié

Nous présentons ici quelques représentations utiles dans un cadre bivarié.

9.6.5.1 Croisement de deux variables qualitatives

Il est possible de superposer deux diagrammes en tuyaux d'orgue comme on peut le voir sur les deux figures suivantes.

```
> tss <- prop.table(table(sexe,situation),1)
```

```
> barplot(tss,bes=TRUE,leg=TRUE)
> title(paste("Diagrammes en tuyaux d'orgue de la situation",
+ "en fonction du sexe",sep="\n"))
> fleches(FALSE,TRUE)
```

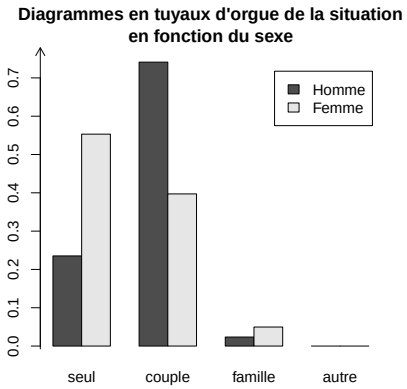


FIGURE 9.15 – Diagramme en tuyaux d'orgue pour deux variables qualitatives.

Le diagramme mosaïque peut aussi être utile pour le croisement de deux variables qualitatives.

```
> par(las=1) # Écriture horizontale des modalités.
> mosaicplot(sexe~matgras,color=brewer.pal(5,"Set1"),
+            main="Mosaicplot de matgras en fonction de age")
```

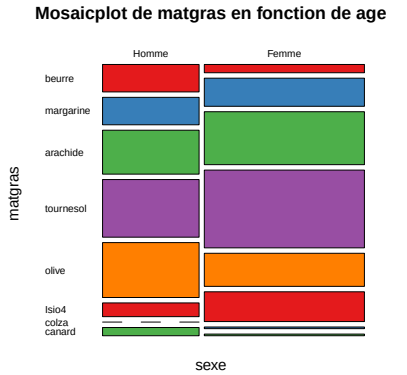


FIGURE 9.16 – Diagramme mosaïque pour le croisement de deux variables qualitatives.

Astuce

Une autre fonction intéressante dans ce contexte est la fonction `assocplot()` qui produit un graphique d'association de Cohen-Friendly indiquant les déviations par rapport à l'indépendance dans une table de contingence 2×2. Mentionnons aussi la fonction `spineplot()`.

```
> assocplot(table(sexe,matgras))
```

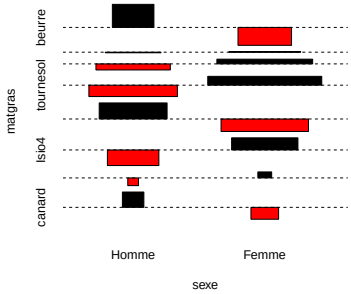


FIGURE 9.17 – Graphique d'association de Cohen-Friendly croisant deux variables qualitatives.

Une dernière fonction intéressante dans ce contexte est la fonction `table.cont()` du *package* `ade4`.

```
> require("ade4")
> sexematgras <- table(sexe,matgras)
> table.cont(sexematgras,row.labels=rownames(sexematgras),
+            col.labels=colnames(sexematgras))
```

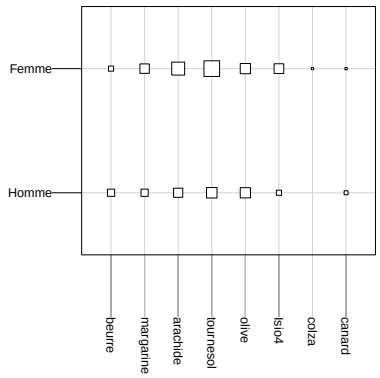
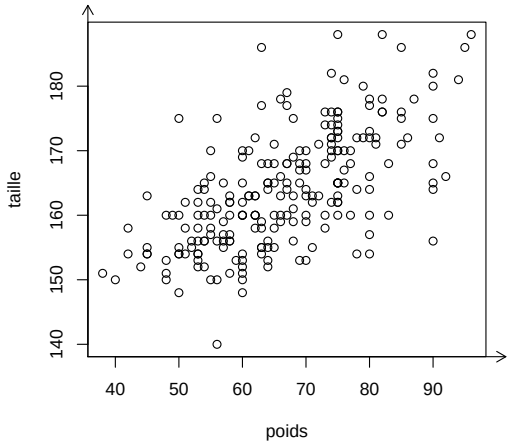


FIGURE 9.18 – Graphique `table.cont` croisant deux variables qualitatives.

9.6.5.2 Croisement de deux variables quantitatives

La fonction à utiliser dans ce contexte est la fonction `plot()`.

```
> plot(taille~poids)
> fleches()
```



Nous pouvons agrémenter le graphique ci-dessus en utilisant ce que nous avons vu dans le chapitre 5 pour créer une fonction `flashy.plot()`.

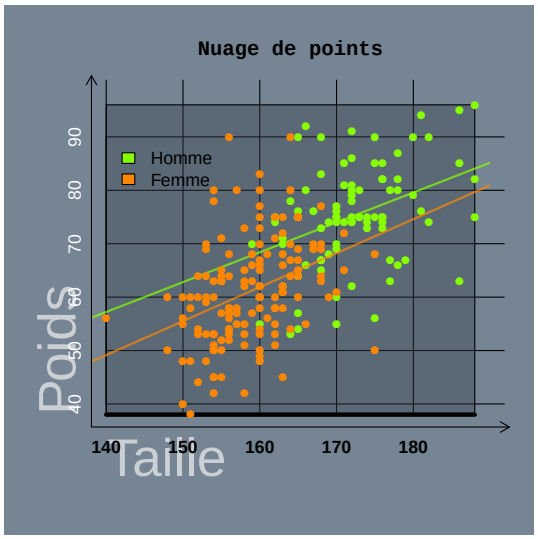


FIGURE 9.19 – Graphique croisant deux variables quantitatives.

9.6.5.3 Croisement d'une variable qualitative et d'une variable quantitative

Dans ce contexte, il est intéressant de tracer des diagrammes en boîte à moustaches (*boxplots*) de la variable quantitative pour chaque modalité de la variable qualitative. Si les variables ont été correctement structurées dans **R**, il suffit d'utiliser la fonction `plot()`.

```
> par(bty="n")
> plot(cafe~sexe,col=brewer.pal(5,"Set2"),notch=TRUE,varwidth=TRUE,
+         boxwex=0.3)
> title(paste("Boxplot de la consommation de café",
+  "en fonction du sexe",sep="\n"),family="Courier")
> fleches(FALSE,TRUE)
```

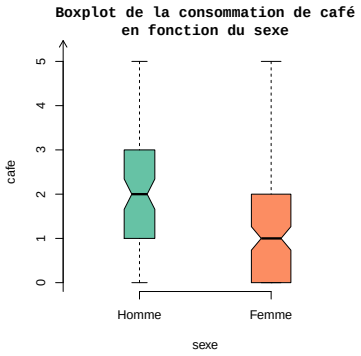


FIGURE 9.20 – *Boxplots* d'une variable quantitative selon les niveaux d'une variable qualitative.

Notons également l'existence de la fonction `stripchart()` qui permet d'obtenir le graphique suivant :

```
> stripchart(fruit_crus~age,data=nutriage)
> fleches(y=TRUE)
```

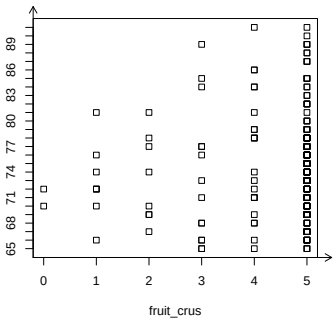


FIGURE 9.21 – Diagramme `stripchart` croisant une variable quantitative avec une variable qualitative.

Termes à retenir

`as.factor()` : transforme une variable en facteurs
`levels()` : affiche ou affecte les niveaux d'un facteur
`as.ordered()` : transforme une variable en facteurs ordonnés
`as.integer()` : pour structurer une variable discrète
`as.double()` : pour structurer une variable continue
`table()` : renvoie le tri à plat d'une variable ou crée une table de contingence entre deux variables
`addmargins()` : rajoute les marges à une table de contingence
`margin.table()` : renvoie les distributions marginales d'une table de contingence
`prop.table()` : calcule les distributions conditionnelles à partir d'une table de contingence
`na.omit()` : supprime les valeurs manquantes (`NA`) d'une variable
`median()` : renvoie la médiane d'un vecteur
`mean()` : renvoie la moyenne d'un vecteur
`quantile()` : renvoie les quantiles d'un vecteur
`summary()` : appliquée à une série numérique, renvoie minimum, maximum, quartiles et moyenne
`range()` : renvoie le minimum et le maximum
`IQR()` : renvoie l'intervalle inter-quartiles
`var()` : renvoie la variance de l'échantillon
`sd()` : renvoie l'écart type de l'échantillon
`mad()` : déviation absolue par rapport à la médiane
`chisq.test()` : calcul de la statistique du khi-2
`cor.test()` : calcul du coefficient de corrélation de Pearson, du τ de Kendall ou du ρ de Spearman
`cov()` : renvoie la covariance
`cor()` : renvoie la corrélation
`barplot()` : trace un diagramme en tuyaux d'orgue
`pie()` : trace un diagramme circulaire
`plot.ecdf()` : trace la courbe de la fonction de répartition empirique
`stem()` : affiche un diagramme en tiges et feuilles
`boxplot()` : trace un diagramme en boîte à moustaches
`hist()` : trace un histogramme
`plot()` : permet de tracer un nuage de points

Exercices

9.1- Donnez l'instruction permettant d'obtenir le tableau des fréquences de la variable qualitative `x`.

9.2- Donnez l'instruction permettant d'obtenir la table de contingence des variables qualitatives `x` et `y`.

9.3- Quelle est la fonction permettant d'obtenir les distributions marginales à partir d'une table de contingence ?

9.4- Quelle est la fonction permettant d'obtenir les distributions conditionnelles à partir d'une table de contingence ?

9.5- Donnez l'instruction permettant d'obtenir le mode d'une distribution.

9.6- Donnez l'instruction permettant de calculer l'étendue du vecteur `x`.

9.7- Donnez l'instruction permettant de calculer l'intervalle inter-quartiles du vecteur `x`.

9.8- Donnez l'instruction permettant de calculer la variance (non empirique) du vecteur `x`.

9.9- Donnez le code d'un programme permettant de calculer le coefficient de variation.

9.10- Donnez l'instruction permettant de calculer l'écart moyen.

9.11- Quel *package* contient des fonctions permettant de calculer le *skewness* et le *kurtosis* ?

9.12- Donnez l'instruction permettant de calculer le Φ^2 de Cramér.

9.13- Donnez le code d'un programme permettant de calculer le rapport de corrélation $\eta^2_{Y|X}$.

9.14- Quelle est la fonction à utiliser pour obtenir un diagramme de Pareto ?

9.15- Quelle est la fonction à utiliser pour obtenir un diagramme empilé ?

9.16- Quelle est la fonction à utiliser pour obtenir un diagramme circulaire ?

9.17- Quelle est la fonction à utiliser pour obtenir un diagramme en boîte à moustaches ?

9.18- Quelle est la fonction à utiliser pour obtenir un histogramme ?

Fiche de TP

Études descriptives de données

A - Réflexion sur l'indépendance en statistique descriptive

9.1- Importez le fichier `http://www.biostatisticien.eu/springeR/snee74.txt` dans un objet R nommé `snee`.

9.2- Affichez les premières et les dernières lignes de `snee` à l'aide des fonctions `head()` et `tail()`. Combien y a-t-il d'individus ? de variables ? De quel type sont les variables ?

9.3- Utilisez la fonction `attach()` sur votre *data.frame* puis vérifiez au moyen des fonctions `class()` et `levels()` que vos variables sont correctement structurées. Quelles sont les modalités des variables ?

9.4- Faites une étude descriptive univariée de chacune des trois variables prises séparément : résumés numériques et représentations graphiques adéquats.

9.5- Nous allons maintenant étudier la dépendance entre les variables `yeux` et `cheveux`. Créez le tableau de contingence `yeuxcheveux` (effectifs observés) des variables `yeux` et `cheveux`.

9.6- Calculez les fréquences de chacune des modalités de la variable `cheveux` (profils colonnes). Nous obtenons donc la répartition `fchev` des différentes couleurs de cheveux dans la population.

9.7- Faisons maintenant intervenir le deuxième caractère : la couleur des yeux. Calculez le nombre d'individus `nbleus` ayant les yeux bleus dans toute la population.

9.8- On suppose qu'il y a indépendance entre la couleur des yeux et la couleur des cheveux. Autrement dit, le fait que des individus ont les yeux de cette couleur (bleu) est sans relation avec la couleur de leurs cheveux. Par conséquent, parmi la sous-population des gens qui ont les yeux bleus, les proportions calculées en 9.6 devraient être respectées. Calculez alors maintenant, sous cette hypothèse d'indépendance, le nombre de personnes aux yeux bleus qui devraient avoir les cheveux blonds (respectivement marron, noirs et roux).

9.9- Faites de même pour chacune des autres couleurs d'yeux. Vous devez donc obtenir le tableau `tab.ind1` dit des effectifs théoriques sous l'hypothèse d'indépendance, entre la couleur des yeux et la couleur des cheveux.

9.10- Recommençons toute l'opération, mais en inversant les deux caractères, c'est-à-dire en partant cette fois-ci de la variable `yeux`. Calculez, à partir du tableau `yeuxcheveux`, les fréquences de chacune des modalités de la variable `yeux` (profils lignes). Nous obtenons donc la répartition `fyeux` des différentes couleurs d'yeux dans la population.

9.11- Faisons maintenant intervenir le deuxième caractère : la couleur des cheveux. Calculez le nombre d'individus `nblonds` ayant les cheveux blonds dans toute la population.

9.12- On suppose qu'il y a indépendance entre la couleur des cheveux et la couleur des yeux. Autrement dit, le fait que des individus ont les cheveux de cette couleur (blonds) est sans relation avec la couleur de leurs yeux. Par conséquent, parmi la sous-population des gens qui ont les cheveux blonds, les proportions calculées en 9.10 devraient être respectées. Calculez alors maintenant, sous cette hypothèse d'indépendance, le nombre de personnes qui devraient avoir les yeux bleus (respectivement marron, noisette et verts).

9.13- Faites de même pour chacune des autres couleurs de cheveux. Vous devez donc obtenir le tableau `tab.ind2` dit des effectifs théoriques sous l'hypothèse d'indépendance, entre la couleur des cheveux et la couleur des yeux.

9.14- Comparez, à l'aide de la fonction `all.equal()`, les deux tableaux d'effectifs théoriques obtenus. Que constatez-vous ?

9.15- Comparez maintenant le tableau des effectifs observés `yeuxcheveux` à celui des effectifs théoriques (pour chacune des cases, on calcule le carré de la différence).

9.16- Calculez le tableau des contributions au χ^2.

9.17- Calculez tous les indicateurs de liaison appropriés. Concluez.

9.18- L'indépendance est aussi définie (et c'est même sa définition première) comme étant l'égalité de toutes les distributions conditionnelles. Calculez la distribution conditionnelle de la variable `cheveux` sachant que la couleur des yeux est bleue (c'est-à-dire les répartitions des différentes couleurs de cheveux sachant que la couleur des yeux est fixée à bleue). Calculez les trois autres distributions conditionnelles de la variable `cheveux`. Calculez les distributions conditionnelles de la variable `yeux` connaissant chacune des modalités de la variable `cheveux`. Concluez.

9.19- Faites une étude descriptive de la variable `cheveux` en fonction de la variable `sexe`.

9.20- Faites une étude descriptive de la variable `yeux` en fonction de la variable `sexe`.

9.21- Analysez maintenant la dépendance entre la couleur des yeux et la couleur des cheveux pour le tableau de contingence suivant :

		Cheveux				
		Blonds	Bruns	Noirs	Roux	Total
Yeux	Bleus	1 768	807	189	47	2 811
	Gris-vert	946	1 387	746	53	3 132
	Bruns	115	438	288	16	857
	Total	2 829	2 632	1 223	116	6 800

B- Analyse descriptive du jeu de données NUTRIAGE

Nous vous proposons de résoudre ces questions diverses sur le jeu de données NUTRIAGE, dans lesquelles des erreurs ont été délibérément introduites.

9.1- Importez le fichier de données nutriage.xls.

9.2- Donnez le mode absolu de la variable `situation`, de la variable `chocol` et de la variable `taille`.

9.3- Choisissez des classes pour la variable `taille` et donnez la classe modale.

9.4- Calculez la médiane de la variable `chocol`.

9.5- Faites un tri à plat de la variable `chocol` et de la variable `fruit_crus`.

9.6- En vous fondant **uniquement** sur ces tris à plat, donnez la médiane de ces deux variables.

9.7- Calculez les quartiles de la variable `taille` en utilisant les classes choisies précédemment.

9.8- Tracez le polygone des fréquences cumulées de la variable `taille` et évaluez, sur ce graphique, les quartiles de la distribution.

9.9- Calculez en utilisant les données individuelles la moyenne des variables `taille`, `poids` et `age`.

9.10- Faites le tri à plat de la variable `the`, et en vous fondant sur ce tri à plat, calculez-en la moyenne.

9.11- Calculez la moyenne de la variable `taille` en utilisant les classes choisies précédemment.

9.12- Calculez l'étendue de la variable `poids`.

9.13- Tracez un *boxplot* de la variable `poids`.

9.14- Calculez, en utilisant les données individuelles, l'écart type de la variable `taille`.

9.15- En vous fondant **uniquement** sur le tri à plat de la variable `the`, calculez-en le coefficient de variation.

9.16- Calculez la variance totale, la variance *inter* et la variance *intra* de la variable `cafe` en considérant que la population est partitionnée en deux sous-groupes : les hommes et les femmes. Calculez le coefficient η^2.

C- Analyse descriptive de jeux de données

9.1- Faites une analyse de statistique descriptive du jeu de données POIDS.-NAISSANCE.

9.2- Faites une analyse de statistique descriptive du jeu de données INFARCTUS.

Chapitre 10

Variables aléatoires, lois et simulations : une meilleure compréhension grâce aux spécificités de R

Objectif

Nous utilisons les atouts propres au logiciel R, en termes de spécificités du langage, pour aborder de façon empirique les notions de variable aléatoire, de loi d'une variable aléatoire, la loi des grands nombres et le théorème de la limite centrale. Nous survolons quelques notions complexes de l'inférence statistique et nous discourons sur la fluctuation d'échantillonnage ainsi que sur les notions de biais et de variance d'un estimateur. Nous décrivons ensuite quelques méthodes classiques de simulation de lois. Les commandes permettant la génération d'observations issues des lois de probabilité usuelles, ainsi que le calcul de leurs fonctions de répartition, quantile et de densité sont fournies en fin de chapitre.

SECTION 10.1

Notions sur la génération de nombres au hasard

Considérons une urne remplie avec n boules numérotées de 1 à n. La fabrication de nombres au hasard peut s'imaginer facilement au travers d'une expérience consistant à piocher (on dit aussi tirer) avec remise, plusieurs fois

de suite, une seule boule à la fois dans cette urne. Cette opération produit une suite de nombres entiers dont l'ordre d'apparition est régi par une loi dite uniforme discrète (sur l'ensemble $\{1, \ldots, n\}$). Il devient ensuite naturel de se poser la question de la génération de nombres réels uniformément répartis sur un intervalle. Nous nous proposons ici de donner quelques éléments sur la génération de nombres selon une telle loi, dite uniforme, à l'aide d'un algorithme informatique.

La génération de nombres au hasard est un élément incontournable de la simulation. Cette production de nombres peut être fondée sur un algorithme mathématique qui cherche à imiter le hasard. Nous expérimentons ici un tel algorithme ([41]), fondé sur une congruence linéaire, de période $2^{31} - 1$.

L'algorithme est défini par le nombre premier $m = 2^{31} - 1$, par une valeur initiale x_1 choisie arbitrairement dans $\{1, \ldots, m-1\}$ (appelée la racine du générateur ou *seed* en anglais), et par la relation de récurrence suivante qui produit n valeurs (y compris la *seed*) :

$$x_{j+1} \quad \leftarrow \quad 48\,271 \times x_j \pmod{m}, \quad j = 1, \ldots, n-1$$

suivie par une étape de normalisation (à l'intervalle unité) :

$$x_{j+1} \quad \leftarrow \quad x_{j+1}/m, \quad j = 0, \ldots, n-1.$$

Rappelons que $x \pmod{m}$ signifie « reste de la division de x par m ».

Remarque

D'autres algorithmes du même type sont présentés dans le livre de Dodge et Melfi ([20]).

Prise en main

Donnez la suite d'instructions R qui permettent de coder cet algorithme pour fabriquer $n = 30$ valeurs. On se rappellera que $x \pmod{m}$ se calcule en R au moyen de l'instruction `x%%m`.

Le vecteur `x` suivant contient $n = 30$ valeurs générées suivant l'algorithme précédent (avec une graine égale à 598 074 196).

```
> x
 [1] 0.2785000 0.4735070 0.6554026 0.9377200 0.6813453
```

```
 [6] 0.2190502 0.7743987 0.9984000 0.7660816 0.5251324
[11] 0.6643606 0.3513046 0.8235011 0.2202823 0.2478704
[16] 0.9542908 0.5703867 0.1386948 0.9379738 0.9309887
[21] 0.7576328 0.6942503 0.1547888 0.8092940 0.4322561
[26] 0.4344264 0.1965560 0.9561434 0.9964051 0.4690752
```

On constate donc que cet algorithme fabrique des valeurs imprévisibles, mis à part le fait qu'elles sont comprises entre 0 et 1. Toutefois, puisqu'elles sont issues d'un algorithme mathématique déterministe, ces valeurs aléatoires seront plutôt appelées des nombres **pseudo-aléatoires**.

L'algorithme précédent est un générateur de nombres pseudo-aléatoires que l'on qualifiera d'**uniforme** sur $[0, 1]$ pour traduire le fait que les nombres qu'il génère se répartissent de façon uniforme sur cet intervalle.

Astuce

La génération de nombres pseudo-aléatoires est bien entendu implémentée dans le cœur de R. Une version plus élaborée de l'algorithme précédent est par exemple disponible au travers de la fonction `runif()` de R. Par ailleurs, la fonction `set.seed()` permet d'en fixer la racine.

SECTION 10.2

La notion de variable aléatoire

Dans la section précédente, nous avons proposé un algorithme permettant la génération de nombres pseudo-aléatoires. En probabilités et statistique, on parle plutôt de variable aléatoire pour décrire un tel processus. L'objet de cette section est de proposer une heuristique sur ce sujet, une variable aléatoire pouvant être assimilée à un algorithme de génération de nombres qui sont alors appelés les réalisations de la variable aléatoire.

10.2.1 Réalisations d'une variable aléatoire et loi de fonctionnement

Nous allons voir ici comment la syntaxe du logiciel R va nous permettre de mieux comprendre la différence qui existe entre une variable aléatoire et les réalisations de cette dernière.

Il est par exemple courant qu'un énoncé de statistique commence par l'une ou l'autre des deux affirmations suivantes : (1) « Soit X une variable aléatoire de loi $\mathcal{U}(0, 1)$ » ou bien encore (2) « Soit X une variable aléatoire de loi $\mathcal{N}(0, 1)$ ».

Une **variable aléatoire** peut être vue comme **une usine à fabriquer des nombres** ou, exprimé en langage mathématique, une fonction qui fabrique des nombres au hasard à chaque fois que l'on fait appel à elle. Par ailleurs, le processus de fabrication de valeurs générées par une variable aléatoire est gouverné par une **règle de fonctionnement** imposée à cette variable, que l'on appelle la **loi de la variable aléatoire**. Cela apparaît clairement dans le corps de la fonction qui la définit.

Il est facile en R, et qui plus est informatif, de créer de telles variables aléatoires. Ainsi l'instruction

```
> X <- function() runif(1)
```

permet de traduire l'affirmation (1). Quelques appels successifs de cette fonction fabriquent ce que l'on appelle des **réalisations** de la variable aléatoire X :

```
> X()
[1] 0.9220729
> X()
[1] 0.776064
> X()
[1] 0.1678566
```

De même, l'instruction suivante :

```
> X <- function() rnorm(1)
```

permet de traduire l'affirmation (2). Et là encore, quelques appels successifs de cette fonction produisent ce que l'on appelle des réalisations de la nouvelle variable aléatoire X :

```
> X()
[1] -0.08571164
> X()
[1] -0.3881452
> X()
[1] 1.398255
```

Il nous semble qu'il est ainsi plus facile de percevoir la différence existant entre une **variable aléatoire** (la fonction) et ses **réalisations** (les nombres obtenus). On constate en effet qu'une variable aléatoire est une « machine » (un procédé) à fabriquer des valeurs, que l'on nomme réalisations. Elle est dite aléatoire dans le sens que les réalisations obtenues changent à chaque appel, et qu'il est impossible de prévoir à l'avance la valeur particulière qui va survenir.

Attention

Notez bien que les deux variables aléatoires précédentes sont construites de manière identique (mêmes instructions) sauf pour la **règle de fonctionnement** ou **loi** qui les régit (`runif` ou `rnorm` dans le corps de la fonction). Un synonyme (anglicisme) souvent utilisé pour remplacer le mot **loi** est **distribution** (qui est en fait la traduction anglaise du mot **répartition**). Cela vient du fait que la fonction de répartition (qui sera vue plus tard), qui décrit comment les valeurs produites par une variable aléatoire se répartissent sur son support, caractérise entièrement la loi de cette variable.

10.2.2 Variables aléatoires *i.i.d.*

Un autre énoncé courant en statistique est le suivant : « Soit n variables aléatoires $X_1, \ldots, X_n$ indépendantes et identiquement distribuées (*i.i.d.*) de même loi $\mathcal{N}(0,1)$ ». Nous pouvons traduire ce concept en R de manière simple. En prenant par exemple $n = 4$, nous obtenons :

```
> X <- function() rnorm(1) # v.a. de loi (de fonctionnement)
                           # rnorm (c'est-à-dire N(0,1)).
> X4 <- X3 <- X2 <- X1 <- X     # Traduit le fait que les
                                # X_i sont i.i.d.
```

Attention

Prenez garde au fait que `X1` et `X2` ont la même loi `rnorm`. On dit qu'elles sont identiquement distribuées. En revanche, elles ne sont pas identiques dans le sens qu'elles ne produisent pas les mêmes valeurs. En outre, l'indépendance entre `X1` et `X2` vient du fait que la production de valeurs par `X1` n'est pas influencée par le fonctionnement de `X2` :

```
> X1
function() rnorm(1) # v.a. de loi (de fonctionnement)
                    # rnorm (c'est-à-dire N(0,1)).
> X2
function() rnorm(1) # v.a. de loi (de fonctionnement)
                    # rnorm (c'est-à-dire N(0,1)).
> c(X1(),X2())
[1] -0.6319022 -0.1029233
```

Notez que l'on peut aussi considérer le vecteur $\mathbf{X} = (X_1, \ldots, X_4)$, vecteur aléatoire constitué de quatre composantes, qui fabrique simultanément quatre réalisations « indépendantes » à chaque appel.

```
> vecX <- function() c(X1(),X2(),X3(),X4())
> vecX()
```

```
[1] -0.2492207  1.5451713 -0.2447986 -0.1663086
> # ou de façon équivalente en R:
> vecX <- function() rnorm(n = 4)
> vecX()
[1]  2.05249392 -0.08238655 -1.23540291 -1.04219309
```

Astuce

Si les n X_i sont *i.i.d.* de même loi que X, alors on peut fabriquer $\mathbf{X}_n = (X_1, \ldots, X_n)$ ainsi :

```
> vecXn <- function(n) replicate(n,X())
```

Le vecteur aléatoire $\mathbf{X}_n = (X_1, \ldots, X_n)$ est appelé l'**échantillon**.

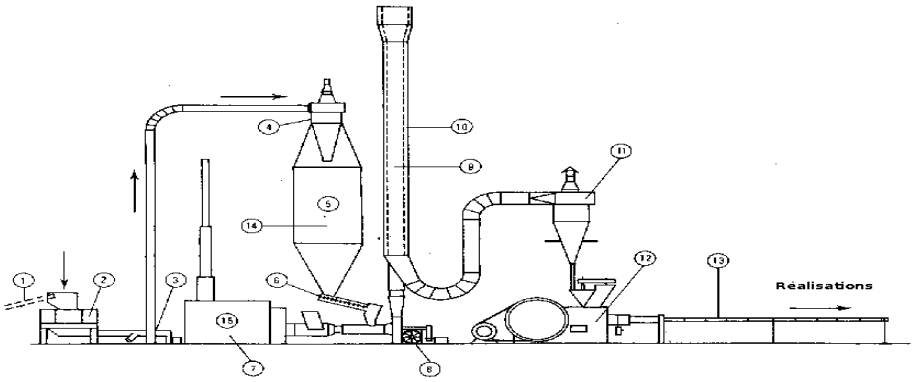
10.2.3 Caractériser la loi d'une variable aléatoire

Nous avons vu que le processus de fabrication de valeurs générées par une variable aléatoire est gouverné par une **règle de fonctionnement** que l'on appelle **la loi** de la variable aléatoire.

Remarque

Nous utiliserons le symbole usuel $\sim$ pour indiquer qu'une variable aléatoire X obéit à une certaine loi. Nous écrirons par exemple $X \sim \mathcal{U}(0,1)$ pour indiquer que la variable aléatoire X suit une loi uniforme sur l'intervalle $[0,1]$ ou encore $X \sim \mathcal{N}(0,1)$ pour indiquer que X suit une loi gaussienne standard.

Le dessin ci-après illustre bien l'idée qu'une variable aléatoire est une « machine » gouvernée par des **paramètres**, et qui fabrique des réalisations en bout de chaîne. Un plan de fabrication de la machine existe, et c'est celui-ci qui régit la façon dont sont fabriquées ces réalisations. Mais de légères variations imprévues dans le déroulement du processus conduisent à des variations sur les valeurs en sortie, qui ne seront par conséquent pas entièrement prédictibles. En fait, chaque réalisation particulière est effectivement imprévisible et soumise au hasard. Mais puisqu'une règle figée de fonctionnement de leur générateur existe, on peut tout de même décrire quelques propriétés globales sur ces nombres. On peut ainsi considérer qu'il existe plusieurs types de hasard, structurés par la loi de la variable aléatoire en jeu. Il conviendra alors, lorsque l'on parlera de « nombres au hasard », de spécifier de quel hasard il s'agit.



Le mathématicien a ainsi introduit des objets mathématiques décrivant cette structure : ce sont la densité ou fonction de masse, la fonction de répartition et la fonction quantile d'une variable aléatoire, pour n'en citer que quelques-uns. En fait, il se trouve même que dans la plupart des cas toutes ces fonctions caractérisent entièrement la loi de la variable aléatoire.

10.2.3.1 Densité, fonction de répartition, fonction quantile

Supposons que l'on observe de nombreuses valeurs, imprévisibles, produites par une certaine variable aléatoire X. On cherche à **décrire** ces valeurs, et au travers d'elles, à mieux appréhender ou spécifier le fonctionnement de X en tant que générateur de ces valeurs.

Voici quelques exemples de ce que l'on pourrait constater pour une certaine variable aléatoire X :
- il y a des valeurs positives et négatives ;
- elles s'agglutinent toutes autour de 7 ;
- autant inférieures à 7 que supérieures ;
- de façon symétrique ;
- etc.

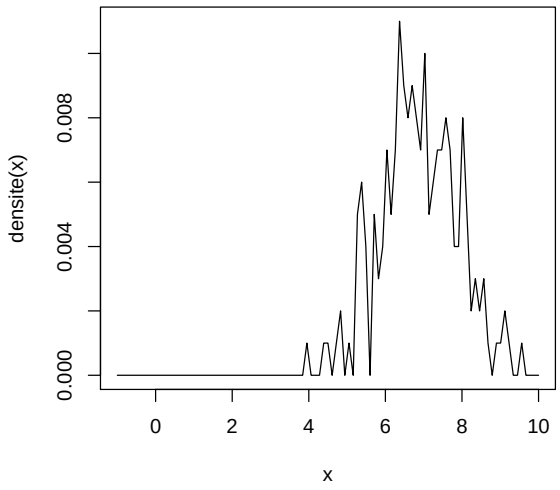
De façon plus globale, on peut aussi s'intéresser à la **densité** d'observations (dans le sens commun du terme, c'est-à-dire qu'une densité élevée correspond à des valeurs serrées les unes contre les autres) générées par X dans chacune des portions $[x-\epsilon, x+\epsilon]$ (où ϵ est un nombre réel positif très petit) de la plage des valeurs possibles. La plage des valeurs possibles est appelée le **support** de X. Le code de la fonction `densite()`, défini ci-dessous, est à visée pédagogique et ne cherche pas à être efficace, mais seulement à traduire la notion de densité d'observations que nous venons de présenter.

```
> # Le code (loi) de la v.a. X est pour le moment délibérément
  # caché.
> X <- function() un.certain.code(des,parametres)
```

```
> densite <- function(x,n=1000,eps=0.01) {
+  lx <- length(x)
+  res <- vector("integer",lx)
+  obs <- replicate(n,X()) # On génère des observations venant de
                          # X.
+  for (i in 1:lx) { res[i] <-
+    length(which((x[i]-eps) <= obs & obs <= (x[i]+eps)))/n
+  }
+  return(res)
+ }
```

Si l'on trace la courbe qui, à chaque portion du support de X, associe la valeur y de la concentration relative d'observations dans cette portion (c'est-à-dire la fréquence de valeurs appartenant à cette portion), on obtient la courbe ci-dessous.

```
> curve(densite,xlim=c(-1,10))
```



Si l'on augmente à la fois le nombre n d'observations générées par X tandis que l'on diminue ϵ, cette courbe deviendra de plus en plus lisse.

```
> curve(densite(x,n=1000000,eps=0.001),xlim=c(-1,10))
```

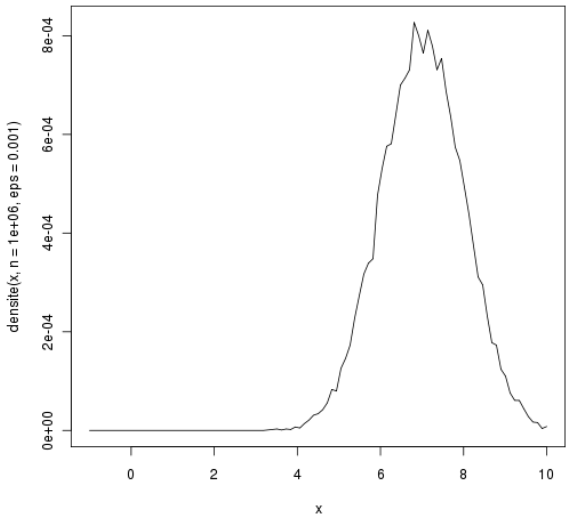


FIGURE 10.1 – Courbe approchant la densité de X.

À la limite, quand $n \to \infty$ et $\epsilon \to 0$, cette courbe deviendra parfaitement lisse et régulière. Correctement normalisée de façon à ce que son aire devienne égale à l'unité, elle est appelée **fonction de densité** de la variable aléatoire X et elle est notée f_X.

Remarque

Une variable aléatoire dont les valeurs possibles peuvent constituer un ensemble continu de valeurs (intervalle par exemple) est dite **continue**. Une variable aléatoire ne pouvant prendre qu'un nombre fini ou dénombrable de valeurs distinctes (encore appelées modalités) est dite **discrète**. Dans le cas d'une variable aléatoire discrète, on ne parlera pas de la densité (quoiqu'étant tout de même une densité au sens de la dérivée de Radon-Nykodym par rapport à une mesure de dénombrement !), mais plutôt de la **fonction de masse** qui donne la probabilité de survenue de chacune des modalités.

Pour conclure, on peut dire que décrire la **loi** d'une variable aléatoire X consiste ainsi à fournir deux informations :

1) la plage des valeurs possibles ou **support** de X (celle-ci pouvant être discrète ou continue) ;
2) pour chaque portion (infinitésimale) de cette plage, la valeur de la **densité** de X dans le cas d'une variable aléatoire continue ou la valeur de la **fonction de masse** de X pour une variable aléatoire discrète.

Remarque

La **fonction de répartition** $F_X(x)$, qui cumule les densités d'observations jusqu'au point x (ou qui cumule les probabilités d'observation des modalités inférieures ou égales à x pour une variable discrète), est une autre façon de caractériser la loi d'une variable aléatoire. On peut montrer qu'elle représente l'aire sous la courbe de densité f_X jusqu'au point x et traduit la probabilité que la variable aléatoire X fabrique des observations plus petites que x :

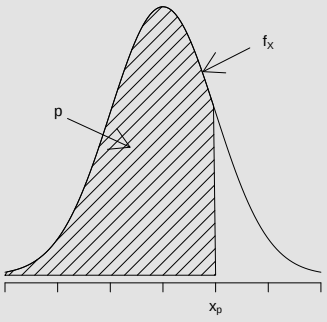
$$F_X(x) = P[X \leq x] = \int_{-\infty}^{x} f_X(t)dt.$$

La **fonction quantile** F_X^{-1}, fonction réciproque de la fonction de répartition, permet également de caractériser la loi d'une variable aléatoire.

Pour chaque valeur de probabilité $p \in [0, 1]$, la valeur $x_p = F_X^{-1}(p)$ est appelée le **fractile ou quantile d'ordre** p de la variable aléatoire X. Il est donc défini par :

$$x_p = F_X^{-1}(p) \Leftrightarrow F_X(x_p) = P[X \leq x_p] = p.$$

Ainsi, la probabilité que les réalisations de la variable aléatoire X soient inférieures ou égales à la valeur x_p vaut p. Le graphique suivant illustre cette notion.



10.2.4 Paramètres de la loi d'une variable aléatoire

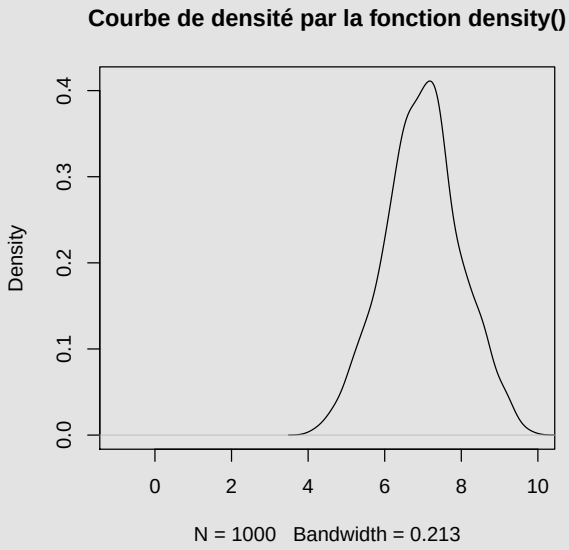
Nous avions délibérément caché le corps de la fonction définissant la variable aléatoire X. Nous pouvons à présent révéler que la densité représentée sur la figure 10.1 est la densité d'une variable aléatoire X de loi $\mathcal{N}(\mu = 7, \sigma^2 = 1)$.

```
> X <- function() rnorm(1,7,1)
```

Astuce

La fonction R `density()` permet d'obtenir la courbe de densité d'un échantillon de valeurs observées.

```
> plot(density(rnorm(1000,7,1)),xlim=c(-1,10),
+  main="Courbe de densité par la fonction density()")
```



Dans cette définition, on voit apparaître explicitement les quantités $\mu = 1$ et $\sigma^2 = 7$ qui sont appelées les **paramètres** de cette loi.

Soit donc maintenant n variables aléatoires $X_1, \ldots, X_n$ indépendantes et identiquement distribuées (noté *i.i.d.*) de loi normale de moyenne $\mu = 7$ et de variance $\sigma^2 = 1$. Définissons également $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$.

Nous avons remarqué qu'il est parfois délicat de comprendre la différence qu'il peut y avoir (et elle est fondamentale) entre :
- la variable aléatoire $\overline{X}_n$;
- les réalisations $\overline{x}_n$ de cette dernière ;
- et le **paramètre** $\mu = \mathbb{E}(X)$, qui en est l'espérance théorique.

Il est vrai que, par abus de langage, on nomme souvent incorrectement ces trois objets d'une même appellation par : *la moyenne*.

Il nous semble que le logiciel **R** offre la possibilité de mieux appréhender et différencier ces concepts. Ainsi la variable aléatoire $\overline{X}_4 = \frac{1}{4}\sum_{i=1}^{4} X_i$ (pour $n = 4$) pourrait être définie par :

```
> X1 <- function() rnorm(1, mean = mu <- 7,sd = 1)
> X4 <- X3 <- X2 <- X1
> Xbarre4 <- function() (X1()+X2()+X3()+X4())/4
```

Expert

Il est possible de définir de façon automatique (pour n petit) la variable aléatoire $\overline{X}_n$ ainsi :

```
> n <- 10
> eval(parse(text=paste(paste("X",1:n," <- ",sep=""),
+                               collapse=""),"X")))
> eval(parse(text=paste("Xbarre",n," <- function()
+  (",paste("X",1:(n-1),"()+",sep="",collapse=""),
+  "X",n,"())/",n,sep="")))
> Xbarre10
function()
 (X1()+X2()+X3()+X4()+X5()+X6()+X7()+X8()+X9()+X10())/10
```

On peut utiliser cette fonction autant de fois que désiré pour fabriquer plusieurs réalisations, que l'on pourrait par exemple noter $\overline{x}_4^{(i)}, i = 1, 2, \ldots$:

```
> xbarre4.1 <- Xbarre4()
> xbarre4.1
[1] 7.092584
> xbarre4.2 <- Xbarre4()
> xbarre4.2
[1] 5.808099
> xbarre4.3 <- Xbarre4()
> xbarre4.3
[1] 6.61183
```

Ainsi, on voit clairement que des appels successifs de la même fonction `Xbarre4()` produisent plusieurs réalisations successives imprévisibles et **fluctuant** autour de 7 (les $\overline{x}_4^{(i)}$). Par conséquent, on se rend compte que :

- $\overline{X}_4$ est bien une variable aléatoire, c'est-à-dire une machine fabriquant des réalisations ;
- ces réalisations $\overline{x}_4^{(1)}, \overline{x}_4^{(2)}, \overline{x}_4^{(3)}, \ldots$ sont toutes différentes, imprévisibles et issues de la variable $\overline{X}_4$;
- la machine $\overline{X}_4$ est constituée des variables aléatoires individuelles $X_1, \ldots, X_4$, toutes régies par une même loi de fonctionnement ;

– cette loi de fonctionnement dépend d'un paramètre `mu` (affecté au paramètre formel `mean` de la fonction `rnorm()`) qui est fixé lors de sa création à la valeur $\mu = 7$, et qui ne varie jamais lors des appels successifs de la fonction ;
– ce paramètre **μ est donc une caractéristique intrinsèque** de chacune des variables X_i.

Remarque

Puisque $\overline{X}_4$ est une variable aléatoire, elle possède donc une loi comme toute variable aléatoire. La théorie mathématique nous dit que $\overline{X}_4 \sim \mathcal{N}(\mu = 7, \sigma^2 = 1/4)$. Si l'on s'intéresse uniquement au comportement en loi de `Xbarre4()`, nous pouvons la définir directement par :

```
> Xbarre4 <- function() rnorm(1,mean = 7,sd = sqrt(1/4))
```

SECTION 10.3

Loi des grands nombres et théorème de la limite centrale

Soit $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ la variable aléatoire moyenne constituée de n variables aléatoires X_i *i.i.d.*, chacune ayant la même loi $\mathcal{L}$ qui n'est pas forcément connue, d'espérance $\mathbb{E}(X_i) = \mu$ et de variance $\mathbb{V}\text{ar}(X_i) = \sigma^2 < \infty$.

10.3.1 Loi des grands nombres

La **loi des grands nombres** stipule que lorsque n tend vers l'infini, la variable aléatoire moyenne $\overline{X}_n$ tend (on dit converge en probabilité) vers $\mathbb{E}(X_1)$, c'est-à-dire la moyenne théorique μ. On écrit alors $\overline{X}_n \xrightarrow{P} \mathbb{E}(X_1)$. Cela peut facilement être constaté avec R, par exemple lorsque $\mathcal{L} = \mathcal{U}(0, 1)$:

```
> mean(runif(1))
[1] 0.8043387
> mean(runif(10))
[1] 0.6478526
> mean(runif(100))
[1] 0.5574017
> mean(runif(1000))
[1] 0.5024962
> mean(runif(10000))
[1] 0.5001176
> mean(runif(100000))
[1] 0.4999408
```

```
> mean(runif(1000000))
[1] 0.4999134
```

On constate bien à l'aide de R, en augmentant la valeur de la taille de l'échantillon n, que ces différents chiffres se rapprochent de la moyenne théorique $\mu = 0.5$ des n variables aléatoires *i.i.d.* $\mathcal{U}(0,1)$.

Remarque

Cela se révèlera très utile pour approcher (on dit **estimer**) les paramètres inconnus d'une loi. Nous reviendrons sur ce point dans la section 10.4.

10.3.2 Théorème de la limite centrale

On peut également s'intéresser à la variable aléatoire $Y_n = \sqrt{n}\left(\frac{\overline{X}_n - \mu}{\sigma}\right)$ qui sert de pivot à la construction de certains intervalles de confiance et tests d'hypothèses. Comme toute variable aléatoire, Y_n possède une loi (de fonctionnement). Les mathématiques permettent parfois de la calculer de façon explicite. Cette loi dépend en général de la taille n de l'échantillon. On peut alors se poser la question de l'évolution de cette loi pour des valeurs grandissantes de n. C'est ce que l'on appelle l'étude de la convergence en loi d'une variable aléatoire. Le **théorème central limite** énonce que Y_n converge en loi vers une variable aléatoire limite de loi $\mathcal{N}(0,1)$. En d'autres termes, générer des observations à partir de Y_∞ revient à générer des observations à partir d'une variable aléatoire suivant une loi $\mathcal{N}(0,1)$ (si tant est que l'on ne s'intéresse qu'à la distribution des valeurs de Y_∞ et pas à ses réalisations individuelles).

Ce mode de convergence, ainsi que les autres modes de convergence classiques (convergence en probabilité, presque sûre et en moyenne r), sont très bien expliqués dans [31].

Le *package* `ConvergenceConcepts`, décrit dans [30], est un outil qui permet de visualiser graphiquement cette évolution. Vous pouvez par exemple essayer les instructions suivantes :

```
require("ConvergenceConcepts")
investigate() # Sélectionnez Example 3.
```

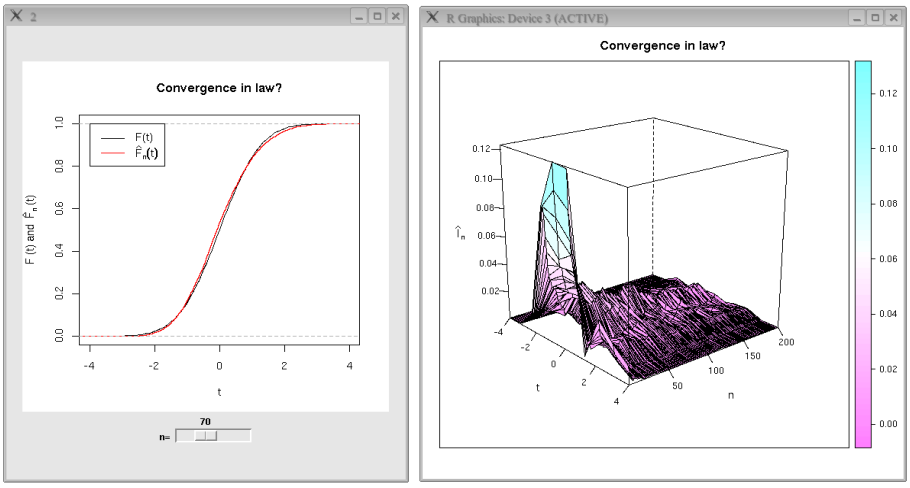


FIGURE 10.2 – Convergence en loi en action sur un exemple de données simulées. Gauche : la fonction de répartition d'une $\mathcal{N}(0,1)$ est tracée en noir tandis que la fonction de répartition empirique $\hat{F}_{y_n}$ (voir section 10.4.2) de l'échantillon Y_n ($n = 70$) fondée sur $M = 5\ 000$ réalisations est tracée en rouge. Droite : graphique tridimensionnel de $|\hat{F}_{y_n}(t) - F(t)|$ en tant que fonction de n et t.

SECTION 10.4

La statistique inférentielle

Nous avons vu dans la section précédente comment la loi des grands nombres peut être utilisée pour approcher un paramètre fixe (notons le θ) d'une loi par la réalisation d'une variable aléatoire (par exemple le paramètre μ par une réalisation de $\overline{X}_n$). Cela sera évidemment très utile lorsque le paramètre sera inconnu. La variable aléatoire « approximante » sera alors appelée **un estimateur** de ce paramètre. Les réalisations de cet estimateur seront appelées des **estimations** (du paramètre inconnu θ). Faire de l'**inférence** consiste à estimer des paramètres inconnus sur la base d'un échantillon généré par des variables aléatoires dont la loi dépend de ces paramètres inconnus.

10.4.1 Estimation (ponctuelle) de paramètres

Mais qu'est-ce donc qu'un estimateur ? Puisqu'il s'agit de proposer, sur la base d'un échantillon généré suivant une certaine loi, une valeur plausible du paramètre inconnu de cette loi, on peut donc considérer que cette valeur plausible est une fonction de l'échantillon. Si θ est la valeur inconnue du paramètre à deviner (estimer), on pourra noter $\hat{\theta}(x_1, \ldots, x_n)$ notre proposition, spécifiant ainsi explicitement qu'elle dépend des valeurs de notre échantillon. Nous l'appellerons une estimation de θ.

On peut alors parler de $\hat{\theta}(X_1, \ldots, X_n)$ comme étant l'estimateur de θ. Cet estimateur est une variable aléatoire, fonction des variables aléatoires ayant généré l'échantillon, qui a été construit afin de se rapprocher dans un certain sens de θ.

Il existe plusieurs techniques pour proposer un estimateur d'un paramètre θ. Retenons par exemple que la loi des grands nombres nous permet de proposer $\overline{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$ comme estimateur de l'espérance théorique $\mu = \mathbb{E}(X_1)$. Notez que l'échantillon $(X_1, \ldots, X_n)$ a été généré en suivant une loi dont l'un des paramètres est justement cette valeur inconnue μ.

```
> theta <- 7 # Valeur supposée inconnue.
> mean(rnorm(10000,mean=theta)) # On se sert de l'échantillon
                                # observé (x_1,...,x_10000).
[1] 7.003201
```

En suivant la même idée fondée sur la loi des grands nombres, on peut proposer $\overline{X_n^2} - \left(\overline{X}_n\right)^2 = \frac{1}{n}\sum_{i=1}^n X_i^2 - \left(\frac{1}{n}\sum_{i=1}^n X_i\right)^2$ comme estimateur du paramètre (supposé inconnu) $\sigma^2 = \mathbb{V}\text{ar}(X_1) = \mathbb{E}(X_1^2) - [\mathbb{E}(X_1)]^2$. On ne se sert ainsi que d'un échantillon observé de variables aléatoires (qui suivent une loi dépendant du paramètre inconnu σ) pour estimer σ^2.

Remarque

L'estimateur ainsi proposé est biaisé (voir section 10.4.4). L'estimateur sans biais de σ^2 est $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^n \left(X_i - \overline{X}_n\right)^2$.

```
> theta <- 2 # Var(X_1), supposée inconnue.
> vecx <- rnorm(10000,sd=sqrt(theta))
> mean(vecx^2)-(mean(vecx))^2 # On se sert uniquement de
                              # l'échantillon observé.
[1] 2.032261
```

Astuce

Notez que l'approche que nous venons de décrire permet de calculer des intégrales. En effet, on a par exemple

$$\int_a^b g(x)dx = \int_{\mathbb{R}} g(x)\mathbf{1}_{[a,b]}(x)dx = \mathbb{E}[g(X)],$$

où X est une variable aléatoire qui suit une loi uniforme $\mathcal{U}(a,b)$ sur l'intervalle $[a,b]$.

Il suffit donc de générer un échantillon $(x_1, \ldots, x_n)$ de grande taille n provenant de variables aléatoires suivant une loi $\mathcal{U}(a, b)$, puis d'estimer $\mathbb{E}[g(\mathrm{X})]$ par

$$\frac{1}{n}\sum_{i=1}^{n} g(x_i).$$

C'est ce que l'on appelle le calcul d'intégrale par **simulation de Monte-Carlo**.

10.4.2 La fonction de répartition empirique

Une variable aléatoire suivant une loi de Bernoulli de paramètre p est définie comme un générateur pouvant fabriquer uniquement des 1 et des 0, avec une probabilité p de fabriquer un 1, et $1-p$ de fabriquer un 0. Si l'on se donne un échantillon $(x_1, \ldots, x_n)$, de taille n, de réalisations de variables aléatoires ayant cette loi, on peut alors calculer la proportion (notée $\hat{p}$) de valeurs égales à 1 parmi toutes les valeurs de l'échantillon. Cette proportion n'est rien d'autre que la moyenne $\overline{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$.
Par la loi des grands nombres, cette moyenne se rapproche, quand n augmente de l'espérance de X, soit $\mathbb{E}(\mathrm{X}) = 1 \times P[\mathrm{X}=1] + 0 \times P[\mathrm{X}=0] = P[\mathrm{X}=1] = p$. C'est ce que l'on appelle l'**approche fréquentiste des probabilités** qui stipule que la probabilité d'un événement est définie comme la limite de la fréquence d'apparition de cet événement. On notera $\hat{\mathrm{p}}$ l'estimateur de la proportion p.

Soit X une variable aléatoire et x une valeur réelle (ici x n'est pas une réalisation de X). On peut alors fabriquer la nouvelle variable aléatoire $\mathbf{1}_{[\mathrm{X}\leq x]}$ qui prend les valeurs 1 ou 0 suivant que X prend des valeurs inférieures à x ou pas. En suivant la même idée, et en se fondant sur un échantillon $\mathbf{X}_n = (\mathrm{X}_1, \ldots, \mathrm{X}_n)^\intercal$, on peut fabriquer la variable aléatoire $\hat{\mathrm{F}}_n(x) := \hat{\mathrm{F}}_{\mathbf{X}_n}(x) := \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{[\mathrm{X}_i\leq x]}$. La loi des grands nombres nous permet d'affirmer que

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{[\mathrm{X}_i\leq x]} \xrightarrow{P} \mathbb{E}(\mathbf{1}_{[\mathrm{X}_1\leq x]}).$$

On peut montrer théoriquement que $\mathbb{E}(\mathbf{1}_{[\mathrm{X}_1\leq x]}) = P[\mathrm{X}_1 \leq x] = F_{\mathrm{X}_1}(x)$. Ainsi, on a

$$\hat{\mathrm{F}}_{\mathbf{X}_n}(x) \xrightarrow{P} F_{\mathrm{X}_1}(x).$$

La variable aléatoire $\hat{\mathrm{F}}_{\mathbf{X}_n}(x)$, vue comme une fonction de x, s'appelle la **fonction de répartition empirique** de l'échantillon $\mathbf{X}_n = (\mathrm{X}_1, \ldots, \mathrm{X}_n)$. Nous reviendrons sur cette fonction lors de la présentation de la méthode du *bootstrap* à la section 10.6.

La fonction R `ecdf()` permet de créer simplement cette fonction de répartition empirique :

```
> X <- function() rnorm(1)
> vecXn <- function(n) replicate(n,X()) # Échantillon de v.a.
                                        # de loi N(0,1).
> Fnchap <- function(n,x,X) ecdf(X(n))(x) # Création de la
                                          # fonction (v.a.) F̂_n.
> Fnchap(n=10,x=0,X=vecXn)    # Un premier appel de F̂_n(x)
                              # pour x = 0.
[1] 0.7
> Fnchap(n=1000,x=0,X=vecXn) # Un deuxième appel de F̂_n(x)
                              # pour x = 0.
[1] 0.515
```

10.4.3 Estimation par la méthode du maximum de vraisemblance

Le problème est le suivant. Nous observons dans la nature un certain phénomène et récoltons des données en lien avec ce phénomène. Nous disposons donc d'un échantillon observé de données $(x_1, \ldots, x_n)$ de taille n. Nous pouvons alors supposer que ces données ont été fabriquées par « Mère Nature » (appelons là ainsi), qui les a générées au moyen d'une variable aléatoire suivant une loi disons $\mathcal{N}(\theta^\bullet, 1)$, pour un certain paramètre $\theta^\bullet$ dont la valeur exacte nous est inconnue, mais qui est élément de l'ensemble $\{0, 1, 2, \ldots, 9\}$.

L'objectif de la **méthode du maximum de vraisemblance** est de deviner (estimer) la valeur de $\theta^\bullet$ uniquement sur la base de cet échantillon observé, qui constitue ainsi la seule **information** disponible sur le **processus de génération** de données (si l'on fait abstraction de l'hypothèse sur la loi de la variable aléatoire ayant généré les données). On se demande en fait quelle est la valeur la plus plausible pour $\theta^\bullet$ (parmi $\{0, 1, \ldots, 9\}$) ; celle étant la plus susceptible d'avoir conduit à la génération des données que nous avons effectivement observées.

Nous allons utiliser une approche par **simulation informatique**, qui va nous permettre de mieux comprendre le fonctionnement de cette méthode du maximum de vraisemblance.

Pour cela, on revêt un temps l'habit de « Mère Nature » (ou du « Grand Architecte ») afin de fixer une valeur $\theta^\bullet$ dans l'ensemble $\{0, 1, 2, ..., 9\}$, appelée la vraie valeur du paramètre θ. Pour nous (« Mère Nature ») permettre de choisir la valeur de $\theta^\bullet$ tout en ayant la possibilité de nous (« Statisticien ») la laisser temporairement inconnue, on utilisera la fonction `runif()`.

```
> theta.point <- as.integer(runif(1)*10)
> # Ne pas faire afficher pour le moment la valeur de
  # theta.point.
```

Maintenant, toujours en jouant le rôle de « Mère Nature », simulons un échantillon $(x_1, \ldots, x_n)$ de taille n provenant d'une loi $\mathcal{N}(\theta^\bullet, 1)$.

```
> n <- 1000
> x1...xn <- rnorm(n,mean=theta.point)
```

Attention

Comme nous l'avons vu dans la partie I, il est possible de mettre des points (.) dans le nom d'une variable. Ainsi, `x1...xn` représente ici le nom d'une variable R.

À présent, nous pouvons quitter l'habit de « Mère Nature » pour revenir dans notre enveloppe de simple statisticien. Le but du jeu est alors d'estimer le $\theta^\bullet$ inconnu par une valeur numérique (une estimation) que nous noterons $\hat{\theta}(x_1, \ldots, x_n)$. Pour cela, on va construire au moyen de la fonction R d'optimisation `nlminb()`, une fonction $\hat{\theta}(\mathrm{X}_1, \ldots, \mathrm{X}_n)$ qui fabrique cette estimation. Celle-ci sera construite comme la valeur de θ qui maximise la vraisemblance notée $\mathcal{V}(\theta; \mathrm{X}_1, \ldots, \mathrm{X}_n)$ de l'échantillon ou, ce qui est équivalent, qui minimise $-\mathrm{Log}\mathcal{V}(\theta; \mathrm{X}_1, \ldots, \mathrm{X}_n)$. Cette fonction sera appelée l'**estimateur du maximum de vraisemblance** de $\theta^\bullet$. Notez que la vraisemblance (évaluée en θ) est en quelque sorte une mesure de plausibilité d'observer l'échantillon si l'on suppose que $\theta^\bullet = \theta$.

On définit

```
> theta.chapeau <- function(X1...Xn) {
+  start <- 0.5
+  nlminb(start,moins.log.vraisemblance,X1...Xn=X1...Xn)$par
+ }
```

où la fonction `moins.log.vraisemblance()` est définie par

```
> moins.log.vraisemblance <- function(theta,X1...Xn) {
+  res <- -sum(log(dnorm(X1...Xn,theta)))
+ return(res)
+ }
```

Maintenant, on peut utiliser notre estimateur sur les observations $x_1, \ldots, x_n$ fournies par « Mère Nature » pour obtenir notre estimation.

```
> theta.chapeau(x1...xn)
[1] 5.974491
```

Puisque nous avions supposé que la valeur $\theta^\bullet$ vit dans l'ensemble $\{0, 1, \ldots, 9\}$, on peut donc proposer comme estimation de $\theta^\bullet$ la valeur $\widehat{\theta^\bullet} = 6$.

On peut maintenant retourner dans la peau de « Mère Nature » afin de vérifier si l'estimation obtenue est proche de la vraie valeur inconnue $\theta^\bullet$ que l'on avait temporairement cachée.

```
> theta.point
[1] 6
```

Notez tout de même que nous avions supposé ici que $\theta^\bullet$ vivait dans un ensemble discret (fini ou dénombrable), ce qui a permis de retrouver sa valeur exacte par estimation. Cela ne sera pas possible avec des valeurs continues du paramètre à estimer.

Attention

L'un des intérêts majeurs de l'approche par simulation en statistique est donc de pouvoir se placer simultanément (ou en tout cas de façon alternée) des deux côtés de la barrière :

« Mère Nature » | Statisticien

10.4.4 Fluctuation d'échantillonnage et qualités d'un estimateur

- **Fluctuation d'échantillonnage**

Dans la partie sur les travaux pratiques, nous verrons comment créer un outil permettant de simuler un lancer de dé. Utilisons dès à présent un tel outil pour lancer vingt dés virtuels :

```
> n <- 20
> res <- lancer.le.dé(n)
> res
 [1] 2 3 4 6 2 6 6 4 4 1 2 2 5 3 5 3 5 6 3 5
```

Nous avons vu à la section 10.4.2 que nous pouvions estimer la probabilité p d'obtenir un 4 par la proportion du nombre de fois où l'on a obtenu 4 dans l'échantillon ci-dessus : $\hat{p} = 0.15$.

Puisqu'un dé possède six faces, la valeur attendue est $1/6 \approx 0.1667$. Nous constatons que ce n'est pas tout à fait la valeur que nous avons obtenue. Nous pouvons essayer de lancer de nouveau vingt dés virtuels pour voir ce qu'il se passe.

```
> res <- lancer.le.dé(n)
> res
 [1] 2 5 3 2 4 4 1 2 4 4 4 4 4 4 6 5 1 5 6 2
```

Cette fois-ci nous estimons la probabilité p d'obtenir un 4 par $\hat{p} = 0.4$. Nous nous rendons compte que l'estimation a changé avec ce nouvel échantillon.

C'est ce que l'on appelle la **fluctuation d'échantillonnage**. Ainsi, lorsque l'on cherche à estimer un paramètre θ inconnu, les estimations obtenues varient en fonction des échantillons. Chaque nouvel échantillon observé donne lieu à une estimation différente de θ.

Si maintenant on augmente la taille de l'échantillon (par exemple à n = 10 000), on observe une fluctuation beaucoup moins importante entre deux lancers de 10 000 dés.

```
> n <- 10000
> res <- lancer.le.dé(n)
> sum(res==4)/n
[1] 0.1725
> res <- lancer.le.dé(n)
> sum(res==4)/n
[1] 0.1678
```

- **Qualités d'un estimateur**

Comment savoir alors si un estimateur $\hat{\theta}(X_1, \ldots, X_n)$ est assez précis pour estimer un paramètre θ inconnu ? Nous pouvons pour cela définir deux critères de qualité d'un estimateur :

- son **biais** (pour estimer θ) $\mathbb{B}\left[\hat{\theta}(X_1, \ldots, X_n); \theta\right] = \mathbb{E}\left[\hat{\theta}(X_1, \ldots, X_n)\right] - \theta$ qui mesure si l'estimateur vise juste « en moyenne » ;
- et sa **variance** $\mathbb{V}\text{ar}\left[\hat{\theta}(X_1, \ldots, X_n)\right]$ qui mesure la variabilité de l'estimateur.

Vous aurez noté que le biais et la variance de $\hat{\theta}(X_1, \ldots, X_n)$ dépendent (en théorie) de n.

Maintenant, si l'on dispose d'un générateur permettant de simuler le comportement des variables aléatoires $X_1, \ldots, X_n$, c'est-à-dire si l'on est en mesure de fabriquer un grand nombre d'échantillons $(X_1, \ldots, X_n)$ (disons M = 10 000 ou même plus suivant le contexte), alors il devient possible, en utilisant l'approche par simulation de Monte-Carlo, d'estimer les deux quantités précédentes. En effet, pour chaque échantillon $(x_{1,i}, \ldots, x_{n,i})$ observé, on calculera l'estimation correspondante $\hat{\theta}(x_{1,i}, \ldots, x_{n,i})$. On se retrouvera alors avec M valeurs $\hat{\theta}_1, \ldots, \hat{\theta}_M$ qui permettront d'estimer $\mathbb{E}\left[\hat{\theta}(X_1, \ldots, X_n); \theta\right]$ par $\overline{\hat{\theta}} = \frac{1}{M}\sum_{i=1}^{M}\hat{\theta}_i$ et la variance $\mathbb{V}\text{ar}\left[\hat{\theta}(X_1, \ldots, X_n)\right]$ par $\frac{1}{M}\sum_{i=1}^{M}\left(\hat{\theta}_i - \overline{\hat{\theta}}\right)^2$.

Dans l'exemple ci-après, nous cherchons à estimer par Monte-Carlo le biais et la variance de l'estimateur $\hat{p}$ (fréquence d'apparition de 4 pour n lancers de dés) du paramètre connu $p = \theta = 1/6$. Notez que p est le paramètre de la variable aléatoire X représentant l'apparition ou non d'un 4 lors du lancer d'un dé. La loi de X est une loi de Bernoulli de paramètre $p = 1/6$.

```
> n <- 20
> M <- 100000
> vec.theta.chap <- replicate(M,{res <- lancer.le.dé(n)
+  theta.chap <- sum(res==4)/n})
> mean(vec.theta.chap)-1/6 # Estimation du biais.
[1] -0.00006716667
> var(vec.theta.chap)      # Estimation de la variance.
[1] 0.006969351
```

Pour cet exemple, il est possible de montrer que $n\hat{\theta}(X_1,\ldots,X_n)$ est une variable aléatoire de loi binomiale $\mathcal{B}in(n,p)$, d'espérance np et de variance $np(1-p)$ (avec ici $p = 1/6$). Ainsi, l'estimateur $\hat{\theta}(X_1,\ldots,X_n)$ est sans biais pour estimer θ et sa variance vaut $\frac{p(1-p)}{n}$.

On peut le vérifier numériquement :

```
> p <- 1/6
> p*(1-p)/n
[1] 0.006944444
```

Renvoi

La technique du *bootstrap*, que nous verrons à la section 10.6, permettra d'approcher le biais et la variance d'un estimateur donné, uniquement sur la base d'un seul échantillon $(X_1,\ldots,X_n)$ (comme cela est souvent le cas dans la vie réelle), et non pas sur la base de M échantillons comme cela est possible en simulation de Monte-Carlo où nous disposons d'un générateur des données.

SECTION 10.5

Quelques techniques de simulation (d'une loi)

Commençons par répondre à la question « Pourquoi faire des simulations ? » :
- pour « vérifier », à l'aide de l'ordinateur, un résultat mathématique déjà connu ;
- pour « conjecturer », à l'aide de l'ordinateur, un résultat mathématique que l'on ne parvient pas à démontrer rigoureusement ;
- cela peut nous guider dans la démonstration d'un résultat mathématique difficile ;
- c'est souvent exigé lorsque l'on essaye de publier un résultat dans une revue scientifique ;
- cela permet de mieux raisonner comme un statisticien, comme nous avons déjà pu le voir dans ce chapitre.

Ceci étant dit, comment bien faire des simulations ? Il faut :

- bien cerner le (cœur du) problème ;
- décomposer le phénomène étudié en une suite d'expériences plus simples, simulables avec les outils mathématiques et informatiques dont on dispose ;
- écrire la trame d'un algorithme permettant de résoudre le problème ;
- transcrire cet algorithme dans un programme écrit dans un langage interprété comme R ;
- tester ce programme pour s'assurer qu'il fonctionne correctement ;
- transcrire (éventuellement) le programme en utilisant un langage de plus bas niveau (compilé) comme `C/C++` ou `Fortran`.

Toute simulation nécessite d'être en mesure de simuler des variables aléatoires d'une loi donnée. Nous avons vu dans les sections précédentes comment simuler quelques lois avec R (`rnorm()`, `runif()`). Lorsque la loi à simuler n'est pas implémentée dans R, il est possible d'utiliser l'une des méthodes présentées dans cette section.

10.5.1 Simuler à partir d'une autre loi

Il existe parfois des formules simples exprimant la variable aléatoire X de loi $\mathcal{L}$, dont on veut simuler des observations, en fonction d'une ou de plusieurs variables aléatoires de lois usuelles. Il devient alors aisé de fabriquer un générateur d'observations de loi $\mathcal{L}$ au moyen de cette formule. L'exemple très simple suivant permet d'illustrer ce point. On se rappellera par exemple que l'on peut fabriquer une variable aléatoire de loi χ^2_1 en prenant le carré d'une variable aléatoire gaussienne standard.

```
> rkhi2.1 <- function() rnorm(1)^2 # X ~ N(0,1) ⇒ X² ~ χ²₁.
```

Prise en main

Générez des observations suivant une loi $TU(2)$ (voir la définition de cette loi dans la section 10.7), page 440.

10.5.2 Méthode de la transformation inverse

Supposons que l'on connaisse la fonction de répartition réciproque F_X^{-1} de la variable aléatoire X et que l'on veuille simuler des observations ayant la même loi que celle de X. Cela peut alors très facilement se faire à partir d'un générateur U de loi $\mathcal{U}(0,1)$, au moyen de la formule suivante :

$$\tilde{X} = F_X^{-1}(U).$$

En effet, il se trouve que la variable aléatoire $\tilde{X}$ a pour fonction de répartition F_X. Cette propriété est connue sous le nom de *transformation intégrale de probabilité* et a été découverte par R.A. Fisher ([23]).

Prise en main

On rappelle que la fonction de répartition d'une variable aléatoire X de loi exponentielle $\mathcal{E}(\lambda)$ est donnée par

$$F_X(x;\lambda) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

Calculez F_X^{-1} puis utilisez la fonction `runif()` pour générer des observations d'une loi $\mathcal{E}(2)$.

10.5.3 Méthode du rejet

Supposons que l'on connaisse la fonction de densité f_X de la variable aléatoire X, mais pas sa fonction de répartition inverse. On veut simuler des observations ayant la même loi que celle de X. La méthode du rejet consiste à générer des données suivant une distribution de fonction de densité de référence g proche de celle désirée (dans le sens que $f_X(x) \leq c \times g(x)$ pour un $c > 0$) et ensuite à éliminer une certaine proportion de ces données de manière à se ramener à des données qui suivent la distribution attendue.

L'algorithme à utiliser est le suivant :

1) générer une donnée y en utilisant une variable aléatoire Y de densité g ;
2) générer u en utilisant une variable aléatoire U de densité $\mathcal{U}(0,1)$;
3) si $c \times g(y) \times u \leq f(y)$, alors on garde y comme donnée générée, sinon on l'écarte et on recommence l'algorithme au départ.

Les différentes valeurs obtenues par cet algorithme peuvent être considérées comme ayant été générées par une variable aléatoire de densité f_X. La meilleure valeur de la constante c à utiliser pour minimiser le nombre de rejets est $c = \sup_x f_X(x)/g(x)$.

Prise en main

Nous allons utiliser la méthode du rejet pour générer des observations de loi $\mathcal{N}(0,1)$ en prenant comme fonction de référence la densité d'une loi exponentielle de paramètre $\lambda = 1$. Vous prendrez la valeur $c = \sqrt{e^1/(2\pi)}$ et utiliserez la fonction `rbinom()` pour affecter un signe positif ou négatif aux valeurs fabriquées par l'algorithme du rejet.

10.5.4 Simulation de variables aléatoires discrètes

On veut simuler un échantillon provenant d'une variable aléatoire discrète X satisfaisant $P(\mathrm{X} = x_i) = p_i$ pour tout i dans $\mathbb{N}$ (ou l'un de ses sous-ensembles). On définit U comme étant une variable aléatoire de loi $\mathcal{U}(0,1)$ qui pourra être obtenue à l'aide de la fonction `runif()`, et on utilise l'algorithme suivant :

$$\begin{cases} \mathrm{X} = x_0 & \text{si } 0 < \mathrm{U} \leq p_0; \\ \mathrm{X} = x_i & \text{si } \sum_{j=0}^{i-1} p_j < \mathrm{U} \leq \sum_{j=0}^{i} p_j. \end{cases}$$

Prise en main

Générez des observations de loi uniforme discrète sur $\{0,1,2,3,4,5\}$ en utilisant cet algorithme.

SECTION 10.6

La méthode du *bootstrap*

Les techniques de rééchantillonnage, appelées aussi méthodes du *bootstrap*, consistent à utiliser l'information présente dans un échantillon de valeurs observées $(x_1, \ldots, x_n)$ pour approcher la loi des variables aléatoires *i.i.d.* $\mathrm{X}_1, \ldots, \mathrm{X}_n$ ayant généré cet échantillon. Nous avons vu à la section 10.2.3.1 que la loi d'une variable aléatoire pouvait être décrite au moyen de la fonction de répartition F_{X} de ces variables, et nous avons vu à la section 10.4.2 comment il était possible d'approcher cette fonction de répartition par la fonction de répartition empirique $\hat{\mathrm{F}}_{\mathbf{X}_n}$. L'idée du *bootstrap* consiste alors à générer plusieurs jeux d'observations $\boldsymbol{x}_1^* = (x_{1,1}^*, \ldots, x_{n,1}^*), \ldots, \boldsymbol{x}_B^* = (x_{1,B}^*, \ldots, x_{n,B}^*)$ en suivant la loi décrite par $\hat{\mathrm{F}}_{\mathbf{X}_n}$ (qui est connue) et non pas par F_{X} (qui est inconnue en pratique). Nous considérerons alors que les nouvelles données que nous sommes capables de générer ainsi auront sensiblement les mêmes propriétés que les observations que nous obtiendrions si nous pouvions utiliser à notre guise le générateur fondé sur F_{X} (ce qui n'est pas le cas en pratique). Nous pourrons ensuite utiliser des techniques de Monte-Carlo, comme cela a été présenté à la section 10.4.1, pour estimer par exemple le biais et la variance de l'estimateur $\hat{\theta}(\mathrm{X}_1, \ldots, \mathrm{X}_n)$ d'un paramètre inconnu θ par respectivement $\frac{1}{B}\sum_{b=1}^{B} \hat{\theta}(x_{1,b}^*, \ldots, x_{n,b}^*) - \hat{\theta}(x_1, \ldots, x_n)$ et $\frac{1}{B}\sum_{b=1}^{B} \left(\hat{\theta}(x_{1,b}^*, \ldots, x_{n,i}^*) - \frac{1}{B}\sum_{j=1}^{B} \hat{\theta}(x_{1,i}^*, \ldots, x_{n,i}^*)\right)^2$. La question à laquelle il reste maintenant à répondre est celle de savoir comment générer des observations à partir de $\hat{\mathrm{F}}_{\mathbf{X}_n}$. C'est en fait très simple puisqu'il suffit de tirer avec remise n observations parmi l'échantillon de départ. La procédure du *bootstrap* consistera donc à générer B tels échantillons $\boldsymbol{x}_b^*$ et à s'en servir comme cela est illustré sur l'exemple ci-dessous. C'est la fonction **R** `sample()` qui permettra d'effectuer l'opération de tirage avec remise.

Reprenons l'exemple de la section 10.4.4 où nous cherchions à estimer le biais et la variance de l'estimateur $\hat{p}$ (fréquence d'apparition du chiffre 4 pour n lancers de dés) du paramètre $p = \theta$ de la variable aléatoire X représentant l'apparition ou non d'un 4 lors du lancer d'un dé.

```
> n <- 20 ; xvec <- lancer.le.dé(n) ; sum(xvec==4)/n
[1] 0.15
> # Tirons avec remise un éch. de taille n dans xvec.
> sample(xvec,n,replace=TRUE)
 [1] 6 6 4 5 6 2 5 2 5 2 5 4 6 6 5 5 6 2 1 4
> B <- 10000
> vec.theta.etoile <- replicate(B,sum(
+                 sample(xvec,n,replace=TRUE)==4)/n)
> mean(vec.theta.etoile) - sum(xvec==4)/n
[1] 0.00009
> ((B-1)/B)*var(vec.theta.etoile) ; sd(vec.theta.etoile)
[1] 0.006377992
[1] 0.07986632
```

Dans ce cas très simple, la théorie nous dit que le biais est nul et que la variance vaut $p(1-p)/n = 0.00694$.

Astuce

Notez l'existence du *package* `boot` qui facilite la pratique du *bootstrap* : `boot(xvec,function(x,w) sum(x[w]==4)/n,B)`

SECTION 10.7

Lois usuelles et moins usuelles

10.7.1 Lois usuelles

Les lois de probabilité courantes sont implémentées dans R. Nous donnons, dans les tableaux 10.1 et 10.2, les fonctions permettant de calculer la densité (ou la fonction de masse), la fonction de répartition et la fonction quantile de ces lois. Nous donnons également l'instruction permettant de générer des nombres pseudo-aléatoires issus de ces lois.

TABLE 10.1 – Lois discrètes usuelles. Fonctions **R** pour la fonction de masse (`d-`), de répartition (`p-`) et quantile (`q-`). Instruction pour générer (`r-`) des nombres pseudo-aléatoires issus de ces lois.

Lois discrètes	**Fonctions R**	**Espérance** **Variance**	**Fonction** **de masse** $P(X = x)$

Binomiale(m, α)	`dbinom(x,size=`m`,prob=`α`)`
	`pbinom(q,size=`m`,prob=`α`)`
	`qbinom(p,size=`m`,prob=`α`)`
	`rbinom(n,size=`m`,prob=`α`)`
		$m\alpha$
		$m\alpha(1-\alpha)$
			$\binom{m}{x}\alpha^x(1-\alpha)^{m-x}$

Poisson(λ)	`dpois(x,lambda=`λ`)`
	`ppois(q,lambda=`λ`)`
	`qpois(p,lambda=`λ`)`
	`rpois(n,lambda=`λ`)`
		λ
		λ
			$e^{-\lambda}\frac{\lambda^x}{x!}$

Géométrique(α)	`dgeom(x,prob=`α`)`
	`pgeom(q,prob=`α`)`
	`qgeom(p,prob=`α`)`
	`rgeom(n,prob=`α`)`
		$\frac{1}{\alpha}$
		$\frac{1-\alpha}{\alpha^2}$
			$(1-\alpha)^{x-1}\alpha$

Hyper-géométrique(m, n, k)	`dhyper(x,m=`m`,n=`n`,k=`k`)`
	`phyper(q,m=`m`,n=`n`,k=`k`)`
	`qhyper(p,m=`m`,n=`n`,k=`k`)`
	`rhyper(nn,m=`m`,n=`n`,k=`k`)`
		$\frac{nm}{N}$ (avec $N = n + m$)
		$\frac{n(m/N)(1-(m/N))(N-n)}{(N-1)}$
			$\frac{\binom{m}{x}\binom{n}{k-x}}{\binom{m+n}{k}}$

Binomiale négative(m, α)	`dnbinom(x,size=`m`,prob=`α`)`
	`pnbinom(q,size=`m`,prob=`α`)`
	`qnbinom(p,size=`m`,prob=`α`)`
	`rnbinom(n,size=`m`,prob=`α`)`
		$m\frac{1-\alpha}{\alpha}$
		$m\frac{1-\alpha}{\alpha^2}$
			$\binom{x+m-1}{m-1}\alpha^m(1-\alpha)^x$

Uniforme discrète$\{1, \ldots, m\}$	`(x %in% 1:`m`)/`m
	`sum(1:`m`<=q)/`m
	`match(1,1:`m`/`m`>=p)`
	`sample(x=1:`m`,size=n,TRUE)`
		$\frac{m+1}{2}$
		$\frac{m^2-1}{12}$
			$\frac{1}{m}\mathbf{1}_{\{1,\ldots,m\}}(x)$

TABLE 10.2 – Lois continues usuelles. Fonctions **R** pour la fonction de densité (`d-`), de répartition (`p-`) et quantile (`q-`). Instruction pour générer (`r-`) des nombres pseudo-aléatoires issus de ces lois (notations : $B(\cdot,\cdot)$: fonction bêta, $I(\cdot)$: fonction de Bessel modifiée, $\Gamma(\cdot)$: fonction gamma, $P(\cdot;\lambda)$: fonction de masse d'une Poisson(λ), $I'_x(\cdot,\cdot)$: dérivée de la fonction bêta incomplète, $\text{sech}(x) = \frac{2}{e^x+e^{-x}}$).

Lois continues	**Fonctions R**	**Espérance** **Variance**	**Densité**

Normale(μ,σ^2)	`dnorm(x,mean=`μ`,sd=`σ`)`
	`pnorm(q,mean=`μ`,sd=`σ`)`
	`qnorm(p,mean=`μ`,sd=`σ`)`
	`rnorm(n,mean=`μ`,sd=`σ`)`
		μ
		σ^2
			$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Student(ν,μ)	`dt(x,df=`ν`,ncp=`μ`)`
	`pt(q,df=`ν`,ncp=`μ`)`
	`qt(p,df=`ν`,ncp=`μ`)`
	`rt(n,df=`ν`,ncp=`μ`)`
		$\mu\sqrt{\frac{\nu}{2}}\frac{\Gamma((\nu-1)/2)}{\Gamma(\nu/2)}$
		$(\nu>1)$
		$\frac{\nu(1+\mu^2)}{\nu-2} - \frac{\mu^2\nu}{2}\times$
		$\left(\frac{\Gamma((\nu-1)/2)}{\Gamma(\nu/2)}\right)^2$, $(\nu>2)$
			$\frac{\nu^{\nu/2}e^{-\nu\mu^2/2(x^2+\nu)}}{\sqrt{\pi}\Gamma(\nu/2)2^{(\nu-1)/2}(x^2+\nu)^{(\nu+1)/2}}$
			$\times\int_0^\infty t^\nu e^{\frac{\mu x}{\sqrt{x^2+\nu}}t - \frac{t^2}{2}}dt$

Khi-deux(k,λ)	`dchisq(x,df=`k`,ncp=`λ`)`
	`pchisq(q,df=`k`,ncp=`λ`)`
	`qchisq(p,df=`k`,ncp=`λ`)`
	`rchisq(n,df=`k`,ncp=`λ`)`
		$k+\lambda$
		$2(k+2\lambda)$
			$\frac{1}{2}e^{-(x+\lambda)/2}\left(\frac{x}{\lambda}\right)^{k/4-1/2}$
			$\times I_{k/2-1}(\sqrt{\lambda x})$

Fisher(ν_1,ν_2,λ)	`df(x,df1=`ν_1`,df2=`ν_2`,ncp=`λ`)`
	`pf(q,df1=`ν_1`,df2=`ν_2`,ncp=`λ`)`
	`qf(p,df1=`ν_1`,df2=`ν_2`,ncp=`λ`)`
	`rf(n,df1=`ν_1`,df2=`ν_2`,ncp=`λ`)`
		$\frac{\nu_2(\nu_1+\lambda)}{\nu_1(\nu_2-2)}$
		$(\nu_2>2)$
		$2\frac{(\nu_1+\lambda)^2+(\nu_1+2\lambda)(\nu_2-2)}{(\nu_2-2)^2(\nu_2-4)}$
		$\times\left(\frac{\nu_2}{\nu_1}\right)^2$, $(\nu_2>4)$
			$\sum_{k=0}^{\infty}\frac{e^{-\lambda/2}(\lambda/2)^k}{B\left(\frac{\nu_2}{2},\frac{\nu_1}{2}+k\right)k!}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}+k}$
			$\times\left(\frac{\nu_2}{\nu_2+\nu_1 x}\right)^{\frac{\nu_1+\nu_2}{2}+k} x^{\frac{\nu_1}{2}-1+k}$

Exponentielle(λ)	`dexp(x,rate=`λ`)`
	`pexp(q,rate=`λ`)`
	`qexp(p,rate=`λ`)`
	`rexp(n,rate=`λ`)`
		$\frac{1}{\lambda}$
		$\frac{1}{\lambda^2}$
			$\lambda e^{-\lambda x}\mathbf{1}\{x\geq 0\}$

Uniforme(a,b)	`dunif(x,min=`a`,max=`b`)`
	`punif(q,min=`a`,max=`b`)`
	`qunif(p,min=`a`,max=`b`)`
	`runif(n,min=`a`,max=`b`)`
		$\frac{a+b}{2}$
		$\frac{(b-a)^2}{12}$
			$\frac{1}{b-a}\mathbf{1}\{a\leq x\leq b\}$

Bêta(α,β,λ)	`dbeta(x,shape1=`α`,shape2=`β`,ncp=`λ`)`
	`pbeta(q,shape1=`α`,shape2=`β`,ncp=`λ`)`
	`qbeta(p,shape1=`α`,shape2=`β`,ncp=`λ`)`
	`rbeta(n,shape1=`α`,shape2=`β`,ncp=`λ`)`
		$\approx 1-\frac{\beta}{C}\left(1+\frac{\lambda}{2C^2}\right)$
		avec $C=\alpha+\beta+\frac{\lambda}{2}$
		$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$ si $\lambda=0$
			$\sum_{i=0}^{\infty}P(i;\frac{\lambda}{2})I'_x(\alpha+i,\beta)$

Cauchy(x_0,γ)	`dcauchy(x,location=`x_0`,scale=`γ`)`
	`pcauchy(q,location=`x_0`,scale=`γ`)`
	`qcauchy(p,location=`x_0`,scale=`γ`)`
	`rcauchy(n,location=`x_0`,scale=`γ`)`
		Non définie
		Non définie
			$\frac{1}{\pi}\left[\frac{\gamma}{(x-x_0)^2+\gamma^2}\right]$

Logistique(μ,s)	`dlogis(x,location=`μ`,scale=`s`)`
	`plogis(q,location=`μ`,scale=`s`)`
	`qlogis(p,location=`μ`,scale=`s`)`
	`rlogis(n,location=`μ`,scale=`s`)`
		μ
		$\frac{\pi^2}{3}s^2$
			$\frac{1}{4s}\text{sech}^2\left(\frac{x-\mu}{2s}\right)$

Log-Normale(μ,σ)	`dlnorm(x,meanlog=`μ`,sdlog=`σ`)`
	`plnorm(q,meanlog=`μ`,sdlog=`σ`)`
	`qlnorm(p,meanlog=`μ`,sdlog=`σ`)`
	`rlnorm(n,meanlog=`μ`,sdlog=`σ`)`
		$e^{\mu+\sigma^2/2}$
		$(e^{\sigma^2}-1)e^{2\mu+\sigma^2}$
			$\frac{1}{x\sigma\sqrt{2\pi}}e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$

Gamma(α,β)	`dgamma(x,shape=`α`,rate=`β`)`
	`pgamma(q,shape=`α`,rate=`β`)`
	`qgamma(p,shape=`α`,rate=`β`)`
	`rgamma(n,shape=`α`,rate=`β`)`
		$\alpha\beta$
		$\alpha\beta^2$
			$x^{\alpha-1}\frac{e^{-x/\beta}}{\beta^\alpha\Gamma(\alpha)}\mathbf{1}_{x>0}$

Weibull(λ,k)	`dweibull(x,shape=`λ`,scale=`k`)`
	`pweibull(q,shape=`λ`,scale=`k`)`
	`qweibull(p,shape=`λ`,scale=`k`)`
	`rweibull(n,shape=`λ`,scale=`k`)`
		$\lambda\Gamma\left(1+\frac{1}{k}\right)$
		$\lambda^2\Gamma\left(1+\frac{2}{k}-\mu^2\right)$
			$\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}e^{-(x/\lambda)^k}\mathbf{1}_{x\geq 0}$

Gumbel(μ,β)
(*Package* `evd`)
	`dgumbel(x,loc=`μ`,scale=`β`)`
	`pgumbel(q,loc=`μ`,scale=`β`)`
	`qgumbel(p,loc=`μ`,scale=`β`)`
	`rgumbel(n,loc=`μ`,scale=`β`)`
		$\mu+\beta$
		$\frac{\pi^2}{6}\beta^2$
			$\frac{ze^{-z}}{\beta}$ avec $z=e^{-\frac{x-\mu}{\beta}}$

10.7.2 † Lois moins usuelles

Nous fournissons dans les tableaux ci-dessous les formules permettant de générer un échantillon de quelques lois moins classiques.

Le tableau 10.3 présente les lois suivantes : *Rademacher* Rad, *Irwin-Hall* Irw(n), *Kumaraswamy* Kum(a,b), *gaussienne inverse* GI(μ,λ), *Lévy* Levy(c), *Log-logistique* Log-Logis(α,β), *Rayleigh* Ray(σ^2), *Rice* Rice(σ,ν), *multinomiale* $\mathcal{M}(n,p_1,\ldots,p_k)$.

TABLE 10.3 – Lois moins usuelles (notations : $B_{1/2} \sim$ Bernoulli(1/2), $\mathrm{Y}_{1,b} \sim \mathcal{B}\mathrm{eta}(1,b)$, $\mathrm{Z} \sim \mathcal{N}(0,1)$, $\mathrm{U}_k, \mathrm{U} \sim \mathcal{U}(0,1)$, $\mathrm{G}(\alpha,\beta) \sim$ Gamma(α,β), $L_{1/2}(x) = e^{x/2}\left[(1-x)I_0\left(\frac{-x}{2}\right) - xI_1\left(\frac{-x}{2}\right)\right]$, $I_\alpha(\cdot)$: fonctions de Bessel modifiées, $B(\alpha,\beta)$: fonction bêta).

Loi	**Densité**	**Génération**	**Espérance** **Variance**
Rad	$1/2$ si $k = \pm 1$	$2B_{1/2} - 1$	0 1
Irw(n)	$\frac{1}{2(n-1)!}\sum_{k=0}^{n}(-1)^k\binom{n}{k}$ $\times(x-k)^{n-1}\,sgn(x-k)$	$\sum_{k=1}^{n}\mathrm{U}_k$	$n/2$ $n/12$
Kum(a,b)	$abx^{a-1}(1-x^a)^{b-1}$	$\mathrm{X}_{a,b} = \mathrm{Y}_{1,b}^{1/a}$	$bB(1+1/a,b)$ $bB(1+\frac{2}{a},b) - b^2B^2(1+\frac{1}{a},b)$
GI(μ,λ)	$\left[\frac{\lambda}{2\pi x^3}\right]^{1/2}\exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right)$	$\mathrm{X} = \mu + \frac{\mu^2}{2\lambda}\left[\mathrm{Z}^2 - \frac{\lvert\mathrm{Z}\rvert}{\mu}\sqrt{4\mu\lambda + \mu^2\mathrm{Z}^2}\right]$ X si $\mathrm{U} \le \frac{\mu}{\mu+\mathrm{X}}$, sinon $\frac{\mu^2}{\mathrm{X}}$	μ $\frac{\mu^3}{\lambda}$
Levy(c)	$\sqrt{\frac{c}{2\pi}}\,\frac{e^{-c/2x}}{x^{3/2}}$	$\mathrm{X} = \frac{1}{\mathrm{G}(1/2,c/2)}$	∞ ∞
Log-Logis(α,β)	$\frac{(\beta/\alpha)(x/\alpha)^{\beta-1}}{[1+(x/\alpha)^\beta]^2}$	$\mathrm{X} = \exp(\mathit{Logistic}(\log(\alpha),\beta))$	$\frac{\alpha\pi/\beta}{\sin(\pi/\beta)}$ si $\beta > 1$ $\alpha^2(\frac{2b}{\sin 2b} - \frac{b^2}{\sin^2 b})$ si $\beta > 2$
Ray(σ^2)	$\frac{x}{\sigma^2}\exp\left(\frac{-x^2}{2\sigma^2}\right)$	$\mathrm{X} = \sigma\sqrt{-2\log(\mathrm{U})}$	$\sigma\sqrt{\frac{\pi}{2}}$ $\frac{4-\pi}{2}\sigma^2$
Rice(σ,ν)	$\frac{x}{\sigma^2}\exp\left(\frac{-(x^2+\nu^2)}{2\sigma^2}\right)I_0\left(\frac{x\nu}{\sigma^2}\right)$	$\mathrm{R} = \sqrt{\mathrm{X}^2+\mathrm{Y}^2}$ avec $\mathrm{X} \sim \mathcal{N}(1,\sigma^2)$ et $\mathrm{Y} \sim \mathcal{N}(0,\sigma^2)$	$\sigma\sqrt{\pi/2}L_{1/2}(-\nu^2/2\sigma^2)$ $2\sigma^2 + \nu^2 - \frac{\pi\sigma^2}{2}L_{1/2}^2\left(\frac{-\nu^2}{2\sigma^2}\right)$
$\mathcal{M}(n,p_1,\ldots,p_k)$	$\frac{n!}{x_1!\cdots x_k!}p_1^{x_1}\ldots p_k^{x_k}$ si $\sum_{i=1}^{k}x_i = n$	$\mathrm{Y}_j = \arg\min_{j'=1}^{k}\left(\sum_{i=1}^{j'}p_i \ge \mathrm{U}\right)$ $\mathrm{X} = \sum_{j=1}^{n}\mathrm{Y}_j$ (Y_j i.i.d)	$\mathbb{E}(X_i) = p_i$ $\mathbb{V}\mathrm{ar}(X_i) = np_i(1-p_i)$

Le tableau 10.4 ci-après donne les formules permettant de générer un échantillon des autres lois moins classiques suivantes : *skew-normale* $SN(\xi,\omega^2,\alpha)$, *Laplace* $Lp(\mu,b)$, *shifted exponentielle* $SE(l,b)$, *generalized Pareto* $GP(\mu,\sigma,\xi)$, *generalized error distribution* $GED(\mu,\sigma,p)$, *Johnson SU* $JSU(\mu,\sigma,\nu,\tau)$, *symmetrical Tukey* $TU(l)$, *scale contaminated* $SC(p,d)$, *location contaminated* $LC(p,m)$, *Johnson SB* $SB(g,d)$, *stable* $S(\alpha,\beta)$.

TABLE 10.4 – Lois moins usuelles (notations : $U \sim \mathcal{U}(0.1)$, $Z \sim \mathcal{N}(0,1)$, $Z_\mu \sim \mathcal{N}(\mu,1)$, $V_d \sim \mathcal{N}(0,d)$, $E \sim \mathcal{E}(1)$, $B_p \sim$ Bernoulli(p), $G_p \sim$ Gamma$(1/p,p)$, $F_\lambda^{-1}(p) = p^\lambda - (1-p)^\lambda$, $\Theta = \pi(U - \frac{1}{2})$, $\Theta_\beta = \frac{\pi}{2} + \beta\Theta$).

Loi	Densité	Génération	Espérance Variance
$SN(\xi,\omega^2,\alpha)$	$\frac{2}{\omega\sqrt{2\pi}} e^{-\frac{(x-\xi)^2}{2\omega^2}} \int_{-\infty}^{\alpha\left(\frac{x-\xi}{\omega}\right)} \frac{e^{-\frac{t^2}{2}}}{\sqrt{2\pi}} dt$	$X = \xi + \omega Y$ $Y = W$ si $Z_0 \geq 0$; $-W$ sinon $W = \delta Z_0 + \sqrt{1-\delta^2} V_1$ $\delta = \alpha/\sqrt{1+\alpha^2}$	$\xi + \omega\sqrt{2/\pi}\delta$ $\omega^2(1 - 2\delta^2/\pi)$
$Lp(\mu,b)$	$\frac{1}{2b}\exp\left(-\frac{\lvert x-\mu\rvert}{b}\right)$	$X = \mu - b.sgn(U - \frac{1}{2})\ln(1 - 2\lvert U - \frac{1}{2}\rvert)$	μ $2b^2$
$SE(l,b)$	$b\exp\{-(x-l)b\}, x \geq l$	$X = \frac{-\ln U}{b} + l$	$l + \frac{1}{b}$ $\frac{1}{b^2}$
$GP(\mu,\sigma,\xi)$	$\frac{1}{\sigma}\left(1 + \frac{\xi(x-\mu)}{\sigma}\right)^{\left(-\frac{1}{\xi}-1\right)}$ $x \geq \mu$ si $\xi \geq 0$ $x \leq \mu - \frac{\sigma}{\xi}$ si $\xi < 0$	$X = \mu + \frac{\sigma(U^{-\xi}-1)}{\xi}$	$\mu + \frac{\sigma}{1-\xi}$ $(\xi < 1)$ $\frac{\sigma^2}{(1-\xi)^2(1-2\xi)}$, $(\xi < \frac{1}{2})$
$GED(\mu,\sigma,p)$	$\frac{1}{2p^{1/p}\Gamma(1+1/p)\sigma} e^{\left[-\frac{1}{p\sigma^p}\lvert x-\mu\rvert^p\right]}$	$\mu + \sigma G_p^{1/p} sgn(U - 1/2)$	μ
$JSU(\mu,\sigma,\nu,\tau)$	$\frac{1}{c\sigma}\frac{\tau}{\sqrt{z^2+1}}\frac{1}{\sqrt{2\pi}}e^{-r^2/2}$ $r = -\nu + \tau\sinh^{-1}(z)$ $z = \frac{x-(\mu+c\sigma\sqrt{w}\sinh(\Omega)}{c\sigma}$ $c = \sqrt{((w-1)(w\cosh(2\Omega)+1)/2)}$ $w = e^{1/(\tau^2)}$ et $\Omega = -\nu/\tau$	$X = \mu + c\sigma\times$ $\left[\sqrt{w}\sinh(\Omega) + \sinh\left(\frac{Z_0+\nu}{\tau}\right)\right]$	μ σ^2
$TU(\lambda)$	$\frac{1}{F_A(x)^{\lambda-1}+(1-F_A(x))^{\lambda-1}}$	$X = U^\lambda - (1-U)^\lambda$	0 $\frac{2}{2\lambda+1} - 2\frac{\Gamma^2(\lambda+1)}{\Gamma(2\lambda+2)}$
$SC(p,d)$	$\frac{1}{\sqrt{2\pi}}\left[\frac{p}{d}e^{-\frac{x^2}{2d^2}} + (1-p)e^{-\frac{x^2}{2}}\right]$	$B_p V_d + (1 - B_p)Z_0$	0 $pd^2 + 1 - p$
$LC(p,m)$	$\frac{1}{\sqrt{2\pi}}\left[pe^{-\frac{(x-m)^2}{2}} + (1-p)e^{-\frac{x^2}{2}}\right]$	$B_p Z_\mu + (1 - B_p)Z_0$	mp $1 - (mp)^2$
$SB(g,d)$	$\frac{d}{\sqrt{2\pi}}\frac{1}{x(1-x)}e^{-\frac{1}{2}\left(g+d\log\frac{x}{1-x}\right)^2}$, $d > 0$	$X = \left(1 + e^{-\frac{Z_0-g}{d}}\right)^{-1}$	Pas d'expression explicite
$S(\alpha,\beta)$ avec $-1 \leq \beta \leq 1$ $0 < \alpha \leq 2$	Pas de forme analytique	$\theta_0 = -\frac{\pi}{2}\beta(\alpha - 1 + sgn(1-\alpha))/\alpha$ $X_1 = \sin\{\alpha(\Theta - \theta_0)\}$ $X_2 = \{\cos(\Theta)\}^{1/\alpha}$ $X_3 = \left[\frac{\cos\{(\alpha-1)\Theta - \alpha\theta_0\}}{E}\right]^{\frac{1-\alpha}{\alpha}}$ Si $\alpha \neq 1$: $X = (X_1/X_2)(X_3)$, sinon : $X = \Theta_\beta\tan\Theta - \beta\log\left(\frac{E\cos\Theta}{\Theta_\beta}\right)$	Non définie si $\alpha \leq 1$ 0 si $1 < \alpha \leq 2$ Non définie

SECTION 10.8

Modélisation d'un phénomène

Supposons que l'on observe uniquement les $n = 500$ valeurs suivantes produites par un certain phénomène étudié.

```
> xvec
  [1] 1 1 1 2 0 0 1 1 1 1 2 1 0 1 1 1 0 1 1 2 2 1 2 1 1 2 2 1 2
 [30] 1 1 0 0 3 1 2 2 3 2 1 1 0 0 2 1 0 0 1 0 2 1 3 1 1 0 1 1 1
 [59] 1 1 0 0 1 1 2 2 1 1 0 0 0 1 2 1 0 1 1 3 1 0 2 1 0 1 0 0 2
 [88] 1 0 0 1 2 1 1 1 2 1 1 2 0 1 0 1 1 3 2 3 1 2 1 2 2 0 1 2 3
[117] 1 2 1 2 0 1 1 1 0 0 2 1 2 3 2 0 2 0 1 1 2 2 0 1 2 0 0 3 0
```

```
[146] 2 1 1 0 3 2 0 0 0 1 0 0 1 3 0 2 1 1 1 2 1 2 3 2 1 1 2 2 2
[175] 4 1 0 1 1 1 1 0 1 1 1 1 2 2 2 0 0 0 0 1 2 1 1 1 2 2 0 1 2
[204] 2 0 2 2 4 1 2 2 2 2 2 1 0 2 2 1 2 0 0 2 2 2 1 0 1 2 2 1 1 1
[233] 2 0 2 1 2 1 2 2 1 1 1 0 1 2 0 2 2 2 0 2 1 0 0 2 1 1 0 2 3
[262] 1 2 1 0 1 1 1 1 2 0 4 2 0 2 4 2 2 2 0 0 4 0 3 0 3 3 1 2 2
[291] 2 3 2 4 1 1 3 0 1 0 1 0 1 1 2 2 0 1 0 2 0 1 2 1 2 0 0 0 0
[320] 1 2 1 1 4 2 1 1 1 1 3 1 1 2 0 0 2 1 2 0 3 0 2 1 0 1 0 2 2
[349] 1 2 3 3 1 2 1 1 2 2 2 2 2 1 2 1 0 2 1 1 2 3 3 1 1 0 1 1 2
[378] 1 1 0 1 1 2 2 1 1 0 1 0 0 1 2 0 2 0 2 0 1 0 3 2 2 1 2 3 1
[407] 2 0 0 1 2 2 2 1 0 0 1 0 0 1 1 1 2 2 1 3 0 5 2 2 0 0 2 0 1
[436] 0 1 1 0 2 1 4 1 0 2 1 1 3 1 0 2 3 1 0 3 1 2 1 3 0 1 0 0 1
[465] 1 1 1 4 2 1 2 2 0 1 0 2 0 1 0 0 3 0 2 1 2 3 2 2 2 1 1 1 1
[494] 1 1 1 1 1 2 1
```

À la vue de ces données, le statisticien va tenter de décrire (de façon mathématique) le processus de génération qui a conduit à la production de ces valeurs. Les valeurs observées semblent surgir de façon imprévisible, et il apparaît difficile de trouver une suite logique (et déterministe) pouvant les expliquer. Mais peut-on tenter de les décrire ?

La **statistique inférentielle** inductive permet de remonter des faits observés sur l'échantillon à la loi de probabilité dans la population. Pour cela, le statisticien va par exemple considérer, dans une première approche simplifiée, que chacune des observations ci-dessus est la réalisation d'une même variable aléatoire X et que ces observations sont produites indépendamment les unes des autres. C'est ce qu'il traduit dans son langage mathématique par « Soit $\boldsymbol{x}_n = (x_1, \ldots, x_n)$ un échantillon (observé) du vecteur aléatoire $\mathbf{X}_n = (\mathrm{X}_1, \ldots, \mathrm{X}_n)$ constitué de $n = 500$ variables aléatoires indépendantes et identiquement distribuées (*i.i.d.*) ». En procédant de la sorte, il suppose donc que « Mère Nature » détient un générateur de nombres aléatoires `X` qui est connu d'elle seule et que le statisticien ne connaît évidemment pas. Et le but du jeu pour le statisticien est de tenter de deviner, dans la mesure du possible, quel est ce générateur. C'est ce que l'on appelle faire de la **modélisation statistique**.

Il va alors fouiller dans son arsenal de modèles et commencer par choisir celui qui lui paraît le plus simple (principe de parcimonie) et le plus adéquat. Les variables aléatoires en jeu étant discrètes, il a par exemple à sa disposition les modèles probabilistes suivants :

- la loi binomiale $\mathcal{B}in(m, \alpha)$;
- la loi de Poisson $\mathcal{P}(\lambda)$;
- etc.

Compte tenu des observations (nombres compris entre 0 et 4), le choix d'une loi binomiale s'impose.

Il lui reste alors à estimer les paramètres inconnus de ce modèle, c'est-à-dire à donner des valeurs plausibles pour les paramètres de la loi choisie (ici m et α). C'est la phase d'estimation des paramètres qui a été décrite à la section 10.4.1.

Remarque

Vous l'aurez peut-être deviné, les 500 données ci-dessus ont été simulées à l'aide d'un ordinateur. Nous nous sommes donc placés un temps de l'autre côté de la barrière (du côté de « Mère Nature ») pour générer ces données. Nous pouvons maintenant vous révéler que le générateur qui a été utilisé est le suivant :

```
X <- function() rbinom(500,5,1/4)
```

C'est l'un des gros intérêts de l'ordinateur de pouvoir se placer simultanément d'un côté ou de l'autre de la barrière pour mieux comprendre les phénomènes qui nous entourent. Cela n'est évidemment pas possible lorsque l'on traite des données réelles.

Les nombreuses lois que nous avons présentées dans ce chapitre peuvent être utilisées pour modéliser d'autres types de phénomènes. Le lecteur devrait ainsi être capable de proposer une loi parmi celles-ci pour la modélisation aléatoire d'un phénomène particulier.

Remarque

Nous pouvons par exemple noter les points suivants :

- la loi de Bernoulli (`rbinom(n,1,p)`) est utilisée lorsqu'une expérience aléatoire n'a que deux résultats possibles : le succès avec une probabilité p, et l'échec avec une probabilité $1-p$;
- la loi binomiale négative de paramètres (k, p) (`rnbinom(n,k,p)`) modélise le nombre d'observations jusqu'au k-ième succès (inclus) ;
- la loi de Poisson de paramètres λ (`rpois(n,`λ`)`) est la loi de la variable X qui compte le nombre de réalisations d'un certain événement rare, par exemple par unité de temps ou encore par unité de surface ;
- la loi exponentielle de paramètre λ (`rexp(n,`λ`)`) permet de modéliser l'instant où un système tombe en panne ou de façon équivalente la durée de vie d'un système ;
- la loi de Pareto est souvent appliquée pour la description de la distribution des revenus ;
- la loi de Cauchy permet de décrire les points d'impact de particules émises en faisceau ;
- la loi bêta permet d'ajuster des distributions dont le support est connu.

Il est également intéressant de noter que le site suivant `http://en.wikipedia.org/wiki/List_of_probability_distributions` fournit la description de nombreuses distributions.

Attention

Nous avons modélisé uniquement des phénomènes impliquant plusieurs copies indépendantes de la même variable aléatoire. Dans le chapitre 12, nous présenterons un outil statistique permettant de modéliser certains phénomèmes impliquant plusieurs variables aléatoires en interaction.

Termes à retenir

`d-` : fonction de masse ou de densité (ex. : `dnorm()`)
`p-` : fonction de répartition (ex. : `pchisq()`)
`q-` : fonction quantile (ex. : `qt()`, `qf()`)
`r-` : génération de nombres pseudo-aléatoires (ex. : `runif()`)

Exercices

10.1- Quelle est la fonction R permettant de générer des nombres suivant une loi $\mathcal{N}(0,1)$?

10.2- Quelle est la fonction R permettant de générer des nombres suivant une loi $\mathcal{N}(2,10)$?

10.3- Quelle est la fonction R permettant de calculer les quantiles d'une loi du χ^2 ?

10.4- Quelle est la fonction R permettant de calculer la densité d'une loi de Fisher ?

10.5- Quelle est la fonction R permettant de calculer les quantiles d'une loi de Student ?

10.6- Comment calculer la probabilité que X soit comprise entre 3 et 5 sachant que $X \sim \mathcal{N}(4,2)$?

10.7- Comment calculer le quantile d'ordre $p = 0.95$ d'une $\mathcal{N}(0,1)$?

Fiche de TP

Simulations

A- Étude de la loi $f(x) = \frac{3}{2}\sqrt{x}$ sur $[0,1]$

10.1- Vérifiez que $f(x)$ est une densité au moyen de la fonction `integrate()`.

10.2- Simulez un échantillon de taille 1 000 selon la loi définie par la densité $f(x) = \frac{3}{2}\sqrt{x}$ sur $[0,1]$.

10.3- Calculez les moyennes et variances empiriques.

10.4- Comparez avec les valeurs théoriques.

10.5- Calculez et comparez les probabilités théoriques et empiriques des classes suivantes :
$[0, 0.30]$, $]0.30, 0.50]$, $]0.50, 0.70]$, $]0.70, 0.85]$, $]0.85, 1]$.

B- Étude sur la loi *generalized Pareto*

Soit X une variable aléatoire de loi *generalized Pareto* $GP(\mu, \sigma, \xi)$. La densité de cette loi est

$$f_{\mathrm{X}}(x) = \frac{1}{\sigma}\left(1 + \frac{\xi(x-\mu)}{\sigma}\right)^{-\frac{1}{\xi}-1}$$

avec

$$x \geq \mu \text{ si } \xi \geq 0 \qquad \text{et} \qquad x \leq \mu - \sigma/\xi \text{ si } \xi < 0.$$

On donne

$$\mathbb{E}(\mathrm{X}) = \mu + \frac{\sigma}{1-\xi} \quad (\xi < 1)$$

et

$$\mathbb{V}\mathrm{ar}(\mathrm{X}) = \frac{\sigma^2}{(1-\xi)^2(1-2\xi)} \quad (\xi < 1/2).$$

On peut simuler X au moyen de la formule suivante :

$$\mathrm{X} = \mu + \frac{\sigma(\mathrm{U}^{-\xi} - 1)}{\xi}$$

où U est une variable aléatoire de loi uniforme sur $[0, 1]$.

10.1- Proposez le code R d'une fonction permettant de générer un échantillon de taille n d'une loi $GP(\mu, \sigma, \xi)$. Vous devez fournir un code source utilisant les variables suivantes : `n`, `mu`, `sigma` et `xi`.

10.2- Simulez un échantillon de taille $n = 1\,000$ d'une loi $GP(0, 1, 1/4)$.

10.3- Calculez-en les moyennes et variances empiriques.

10.4- Comparez avec les valeurs théoriques.

10.5- Recommencez les questions de 2 à 3 avec $n = 10\,000$.

10.6- Tracez en rouge l'histogramme en densité de l'échantillon obtenu. Vous prendrez 500 classes équidistantes et limiterez l'affichage de l'histogramme à l'intervalle $[0, 10]$ sur l'axe des abscisses.

10.7- Superposez à cet histogramme la courbe de densité de la $GP(0, 1, 1/4)$ (en bleu). Constatez que la courbe s'ajuste bien à l'histogramme.

C- Uniforme sur un carré

10.1- Simulez 1 000 observations de $(\mathrm{X}_1, \mathrm{X}_2)$ suivant la loi uniforme sur le carré $[0, 1] \times [0, 1]$.

10.2- Obtenez une approximation de la probabilité que la distance de $(\mathrm{X}_1, \mathrm{X}_2)$ au côté le plus proche soit inférieure à 0.25.

10.3- Même question pour la distance au sommet le plus proche.

10.4- Essayez d'identifier la loi théorique de la variable distance au côté le plus proche : espérance, variance, densité.

D- Vers la notion de modélisation

Le statisticien considère que dans le monde qui nous entoure, les phénomènes qui s'y produisent constituent un vaste enchevêtrement d'événements aléatoires, qui peuvent être **modélisés** de façon simplifiée par des variables aléatoires.

10.1- Nous pouvons commencer par évoquer l'exemple simple, et classique, du lancer d'une pièce de monnaie. Cette expérience peut être assimilée à l'observation du côté PILE ou FACE à chaque lancer. Et nous pouvons modéliser cela par une variable aléatoire X de loi (de fonctionnement) Bernoulli de paramètre 1/2. Cette expérience peut être recréée à l'intérieur de l'ordinateur. Créez une fonction nommée `X` permettant de simuler le lancer d'une pièce. Vous pouvez maintenant effectuer quelques lancers de votre pièce virtuelle.

10.2- Nous pouvons également proposer une **modélisation** de l'expérience consistant à lancer un dé. Celle-ci peut être assimilée à l'observation du nombre de points que l'on voit sur la face supérieure du dé à chaque lancé. Et nous pouvons modéliser tout cela (si le dé n'est pas truqué) par une variable aléatoire X de loi (de fonctionnement) uniforme discrète sur $\{1, 2, 3, 4, 5, 6\}$. Cette expérience peut ainsi être recréée à l'intérieur de l'ordinateur. Créez une fonction nommée `lancer.le.dé()` en utilisant la fonction `sample()`. Vous pouvez maintenant effectuer quelques lancers de votre dé virtuel.

10.3- Pour simuler le jeu du Yams, nous allons créer une fonction nommée `yams()` qui permet de lancer cinq dés virtuels. Créez cette fonction en utilisant le paramètre `size` et le paramètre `replace` de la fonction `sample()`.

10.4- Estimez la probabilité d'obtenir un *yams*, c'est-à-dire cinq dés identiques sur un même lancer de cinq dés (indice : utilisez les fonctions `apply()`, `replicate()` et `unique()`). Vous devriez trouver une valeur proche de $\frac{1}{6^4}$.

E- Théorème de Box et Muller

Soit U_1 et U_2 deux variables aléatoires uniformes et indépendantes sur l'intervalle $[0, 1]$. Les variables

$$Z_1 = \sqrt{-2\log(U_1)}\cos(2\pi U_2)$$

$$Z_2 = \sqrt{-2\log(U_1)}\sin(2\pi U_2)$$

sont alors deux variables aléatoires normales centrées réduites indépendantes.

10.1- Générez $n = 1\ 000$ couples d'observations $(z_1, z_2)_1, \ldots, (z_1, z_2)_n$ en utilisant cet algorithme.

10.2- Utilisez la fonction `kde2d()` du *package* `MASS` pour estimer la densité bivariée de ces données.

10.3- Utilisez les fonctions `spheres3d()` et `surface3d()` du *package* `rgl` pour représenter ces observations, ainsi que la surface estimée de la densité bivariée de ces données et également la surface de la densité d'une gaussienne standard bivariée. Constatez que l'on obtient une courbe en cloche caractéristique de la gaussienne bivariée.

Chapitre 11

Intervalles de confiance et tests d'hypothèses

Objectif

Ce chapitre se veut être un catalogue des fonctions R le plus couramment utilisées afin d'obtenir les intervalles de confiance observés pour les paramètres classiques : moyenne, proportion, variance, médiane et corrélation. Nous présentons également un catalogue des fonctions R permettant d'effectuer les tests d'hypothèses les plus classiques. Par ailleurs, quelques séances de travaux pratiques permettront au lecteur de bien comprendre la façon correcte d'interpréter un intervalle de confiance, ainsi que les différentes erreurs attachées aux procédures de tests d'hypothèses.

SECTION 11.1

Notations

Le tableau 11.1 présente les notations nécessaires à la définition des intervalles de confiance et des tests d'hypothèses introduits dans ce chapitre. Nous présentons dans le tableau 11.2 les notations des différents quantiles qui seront utilisés.

TABLE 11.1 – Notations sur les estimations de paramètres classiques.

Paramètre	Notation	Estimateur	Estimation	Fonction R
moyenne	μ	$\bar{\mathrm{X}}$	$\bar{x}$	`mean()`
variance	σ^2	$\hat{\sigma}^2$	$\hat{\sigma}^2$	`var()`
médiane	m_e	$\widehat{\mathrm{m}_e}$	$\widehat{m_e}$	`median()`
corrélation	ρ	$\hat{\rho}$	$\hat{\rho}$	`cor()`
proportion	p	$\hat{\mathrm{p}}$	$\hat{p}$	`mean()`

TABLE 11.2 – Notation des différents quantiles d'ordre p.

Loi	Notation	Fonction R
Normale : $\mathcal{N}(0,1)$	u_p	`qnorm(`p`)`
Student à n d.d.l. : $\mathcal{T}(n)$	t_p^n	`qt(`p`,df=`n`)`
Khi-deux à n d.d.l. : $\chi^2(n)$	q_p^n	`qchisq(`p`,df=`n`)`
Fisher à n et m d.d.l. : $\mathcal{F}(n,m)$	$f_p^{n,m}$	`qf(`p`,df1=`n`,df2=`m`)`

d.d.l. : degrés de liberté

SECTION 11.2

Intervalles de confiance

On dispose d'un échantillon $\mathbf{X}_n = (\mathrm{X}_1, \ldots, \mathrm{X}_n)^\mathsf{T}$ de variables aléatoires suivant une loi dépendant d'un certain paramètre θ inconnu que l'on cherche à estimer. Un intervalle de confiance aléatoire de niveau (de confiance) $1-\alpha$ pour θ est la donnée de deux variables aléatoires $\mathrm{A} := a(\mathbf{X}_n; \alpha)$ et $\mathrm{B} := b(\mathbf{X}_n; \alpha)$ telles que

$$P[\mathrm{A} \leq \theta \leq \mathrm{B}] = 1 - \alpha.$$

Les variables aléatoires A et B constituent les bornes de cet intervalle de confiance aléatoire, que l'on note en général

$$\mathrm{IC}_{1-\alpha}(\theta) = [\mathrm{A}, \mathrm{B}].$$

Lorsque l'échantillon est effectivement observé, et que l'on dispose des données $(x_1, \ldots, x_n)$, on notera

$$ic_{1-\alpha}(\theta) = [a, b]$$

l'intervalle de confiance réalisé résultant, où $a = a(x_1, \ldots, x_n; \alpha)$ et $b = b(x_1, \ldots, x_n; \alpha)$. Dans la suite de ce chapitre, nous nous permettrons un abus de langage en ne distinguant pas l'intervalle de confiance aléatoire de sa réalisation.

Notez pour finir que la façon correcte d'interpréter un intervalle de confiance (aléatoire et réalisé) sera décrite dans la partie des travaux pratiques. Ici, nous présentons uniquement un catalogue des intervalles de confiance classiques pour les paramètres usuels de moyenne, de proportion, de variance, de médiane et de coefficient de corrélation.

11.2.1 Intervalles de confiance pour une moyenne

- **Cas des grands échantillons ($n > 30$) ou des petits échantillons avec hypothèse de normalité**

  ▶ *Définition :* Un intervalle de confiance de niveau $(1-\alpha)$ pour la moyenne μ est :

  $$ic_{1-\alpha}(\mu) = \left[\bar{x} - t^{n-1}_{1-\alpha/2}\frac{\hat{\sigma}}{\sqrt{n}}; \bar{x} + t^{n-1}_{1-\alpha/2}\frac{\hat{\sigma}}{\sqrt{n}}\right].$$

  ▶ *Instruction* R *:* L'intervalle de confiance est obtenu grâce à la fonction `t.test()`.

  ▶ *Exemple d'application :* À partir de l'étude alimentaire, on s'intéresse à l'estimation par intervalle de confiance de la moyenne du poids des personnes âgées vivant à Bordeaux.

  ```
  > t.test(poids,conf.level=0.9)$conf.int
  [1] 65.16024 67.80436
  attr(,"conf.level")
  [1] 0.9
  ```

  Nous obtenons l'intervalle de confiance $[65.16, 67.80]$ de niveau de confiance 0.9.

- **Cas des petits échantillons**

  ▶ *Définition :* Dans le cas où aucune hypothèse n'est faite sur les données, nous conseillons d'utiliser une approche par *bootstrap*. Plusieurs types d'intervalles de confiance par *bootstrap* sont définis dans [19].

  ▶ *Instruction* R *:* Il est possible d'utiliser les fonctions `boot()` et `boot.ci()` disponibles dans le *package* `boot`.

  ▶ *Exemple d'application :* Nous disposons d'un échantillon, représentatif de la population féminine vivant en France, de dix femmes ayant les taux de cholestérol suivants (en g/l) :

  ```
  > taux <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
  ```

  Sans hypothèse gaussienne des données, nous proposons un intervalle de confiance de niveau 95 % du taux moyen de cholestérol des femmes vivant en France.

```
> require("boot")
> moyenne <- function(x,indices) mean(x[indices])
> taux.boot <- boot(taux, moyenne, R = 999, stype = "i",
+                   sim = "ordinary")
> boot.ci(taux.boot, conf = 0.95,type = c("norm","basic",
+                   "perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL :
boot.ci(boot.out = taux.boot, conf = 0.95, type =
c("norm", "basic", "perc", "bca"))
Intervals :
Level      Normal              Basic
95%   ( 1.787,  2.366 )   ( 1.770,  2.340 )
Level     Percentile            BCa
95%   ( 1.82,  2.39 )   ( 1.83,  2.41 )
Calculations and Intervals on Original Scale
```

11.2.2 Intervalles de confiance pour une proportion p

- **Cas des grands échantillons ($np \geq 5$ et $n(1-p) \geq 5$)**

  ▶ *Définition :* Un intervalle de confiance de niveau $(1-\alpha)$ pour la proportion inconnue p est :

$$ic_{1-\alpha}(p) = \left[\hat{p} - u_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}; \hat{p} + u_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right].$$

  ▶ *Conditions de validité :* On peut « vérifier » *a posteriori* les conditions requises ($np \geq 5$ et $n(1-p) \geq 5$) en remplaçant p par les bornes de l'intervalle de confiance. Si ces conditions ne sont pas satisfaites, on utilisera la méthode exacte présentée dans le cas des petits échantillons.

  ▶ *Instruction* R *:* Il est possible d'utiliser la fonction `binom.approx()` du *package* `epitools`.

  ▶ *Exemple d'application :* On s'intéresse à l'estimation par intervalle de confiance de la proportion d'hommes (codés 2 ci-dessous) chez les personnes âgées vivant à Bordeaux à partir de l'étude alimentaire.

```
> require("epitools")
> table(sexe) # Répartition de la variable sexe.
sexe
  1   2
 85 141
> binom.approx(141,226)[c("lower","upper")] # Calcul de
                                            # l'ic
                                            # avec n=226.
```

```
      lower     upper
1 0.5607393 0.6870483
```

Attention

La fonction `prop.test()` fournit également un intervalle de confiance de la proportion, fondé sur la statistique du score.

- **Cas des petits échantillons : calcul exact**

  - *Définition :* Un intervalle de confiance de niveau $(1-\alpha)$ pour la proportion p provient de
  $$n\hat{p} \sim \mathcal{B}in(n,p).$$
  - *Instruction* R *:* Il est possible d'utiliser la fonction `binom.test()`.
  - *Exemple d'application :* Reprenons le même exemple afin de déterminer de façon exacte l'intervalle de confiance de la proportion d'hommes chez les personnes âgées vivant à Bordeaux.

```
> binom.test(141,226)$conf # Calcul de l'ic avec n=226.
[1] 0.5572321 0.6872590
attr(,"conf.level")
[1] 0.95
```

Astuce

La fonction `binom.exact()` du *package* `epitools` renvoie le même intervalle de confiance.

11.2.3 Intervalles de confiance pour une variance

- **Cas des échantillons avec une hypothèse de normalité**

  - *Définition :* Un intervalle de confiance de niveau $(1-\alpha)$ pour la variance σ^2 est :
  $$ic_{1-\alpha}(\sigma^2) = \left[\frac{(n-1)\hat{\sigma}^2}{q^{n-1}_{1-\alpha/2}}; \frac{(n-1)\hat{\sigma}^2}{q^{n-1}_{\alpha/2}}\right].$$
  - *Instruction* R *:* La fonction à utiliser est `sigma2.test()`. Cette fonction se trouve dans le *package* associé à ce livre.
  - *Exemple d'application :* On s'intéresse à l'estimation par intervalle de confiance de la variance du poids des personnes âgées vivant à Bordeaux à partir de l'étude alimentaire.

```
> sigma2.test(poids,conf.level=0.9)$conf
[1] 124.8330 170.3277
attr(,"conf.level")
[1] 0.9
```

- **Cas des échantillons sans hypothèse de normalité**

  Dans le cas où aucune hypothèse n'est faite sur les données, nous conseillons d'utiliser une approche par *bootstrap* comme pour la moyenne.

  ▶ *Instruction* R *:* Il est possible d'utiliser les fonctions `boot()` et `boot.ci()` disponibles dans le *package* `boot`.

  ▶ *Exemple d'application :* Reprenons les données du taux de cholestérol des femmes et calculons un intervalle de confiance de la variance du taux de cholestérol sans hypothèse de normalité des données.

```
> taux <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
> require("boot")         # Charger le package boot.
> variance <- function(x,indices) var(x[indices])
> taux.boot <- boot(taux,variance,R=999,stype="i",
+                   sim="ordinary")
> boot.ci(taux.boot,conf=0.95,type=c("norm","basic",
+                   "perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL :
boot.ci(boot.out=taux.boot,conf=0.95,type=
                c("norm","basic","perc","bca"))
Intervals :
Level      Normal              Basic
95%   ( 0.1060,  0.4412 )   ( 0.1026,  0.4448 )
Level     Percentile            BCa
95%   ( 0.0521,  0.3943 )   ( 0.1201,  0.4670 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

Remarque

Notez que pour de grands échantillons sans hypothèse de normalité, vous pouvez utiliser une approche asymptotique. Cela est proposé dans le *package* `asympTest`. Voilà un exemple d'intervalle de confiance de la variance du poids des personnes âgées vivant à Bordeaux à partir de l'étude alimentaire.

```
> require("asympTest")
> asymp.test(poids,par="var")$conf
[1] 121.6842 167.9196
attr(,"conf.level")
[1] 0.95
```

11.2.4 Intervalles de confiance pour une médiane

- *Définition :* Un intervalle de confiance de niveau $(1-\alpha)$ pour la médiane m_e est :

$$ic_{1-\alpha}(m_e) = [x_{(m_1)}; x_{(m_2+1)}]$$

où les $x_{(i)}$ sont les valeurs ordonnées de l'échantillon de taille n, m_1 étant la plus petite valeur telle que $P(\mathrm{L} \leq m_1) \geq \alpha/2$, m_2 est la plus grande valeur telle que $P(\mathrm{L} \geq m_2) > \alpha/2$ avec $\mathrm{L} \sim \mathcal{B}in(n, 0.5)$.

- *Instruction* R *:* Un intervalle de confiance de la médiane est obtenu grâce à la fonction `qbinom()`.

- *Exemple d'application :* Reprenons l'exemple du taux de cholestérol et cherchons un intervalle de confiance à 95 % de la médiane du taux de cholestérol des femmes vivant en France.

```
> taux <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
> m1 <- qbinom(0.025,length(taux),0.5)
> m2 <- qbinom(1-0.025,length(taux),0.5)
> median.ic <- c(sort(taux)[m1],sort(taux[m2+1]))
> median.ic
[1] 1.6 2.0
```

Remarque

Il est toujours possible d'utiliser un intervalle de confiance du type *bootstrap*. La méthode percentile donne pour cet exemple :

```
> taux <- c(3,1.8,2.5,2.1,2.7,1.9,1.5,1.7,2,1.6)
> require("boot")
> mediane <- function(x,indices) median(x[indices])
> taux.boot <- boot(taux,mediane,R=999,stype="i",
+                   sim="ordinary")
> taux.int <- boot.ci(taux.boot,conf=0.95,type="perc")
> taux.int$perc[4:5]
[1] 1.7 2.6
```

Notez également qu'un intervalle de confiance non paramétrique est proposé dans la fonction `wilcox.test()`.

```
> wilcox.test(taux,conf.int=TRUE)$conf
[1] 1.70 2.45
attr(,"conf.level")
[1] 0.95
```

11.2.5 Intervalle de confiance pour un coefficient de corrélation

- *Définition :* Un intervalle de confiance de niveau $(1-\alpha)$ pour le coefficient de correlation ρ est :

$$ic_{1-\alpha}(\rho) = \left[\frac{\exp(2\hat{\theta}_{min})-1}{\exp(2\hat{\theta}_{min})+1}; \frac{\exp(2\hat{\theta}_{max})-1}{\exp(2\hat{\theta}_{max})+1}\right]$$

où $\hat{\theta}_{min} = \frac{1}{2}\ln\left(\frac{1+\hat{\rho}}{1-\hat{\rho}}\right) - u_{1-\alpha/2}\frac{1}{\sqrt{n-3}}$ et $\hat{\theta}_{max} = \frac{1}{2}\ln\left(\frac{1+\hat{\rho}}{1-\hat{\rho}}\right) + u_{1-\alpha/2}\frac{1}{\sqrt{n-3}}$.

- *Instruction R :* Il est possible d'utiliser la fonction `cor.test()`.

- *Exemple d'application :* On s'intéresse à l'estimation par intervalle de confiance du coefficient de corrélation entre le poids et la taille des personnes âgées vivant à Bordeaux à partir de l'étude alimentaire.

```
> cor.test(poids,taille)$conf
[1] 0.5450122 0.7032775
attr(,"conf.level")
[1] 0.95
```

Attention

Cet intervalle de confiance est uniquement valable sous l'hypothèse de binormalité du couple de variables, ici (poids,taille). Si cette hypothèse n'est pas satisfaite, vous pouvez toujours utiliser la méthode du *bootstrap*.

11.2.6 Tableau récapitulatif des intervalles de confiance

Table 11.3 – Résumé sur les intervalles de confiance.

Type	Condition de validité	Fonction R
proportion	$np \geq 5$ et $n(1-p) \geq 5$	`prop.test(x)$conf`
	aucune	`binom.test(x)$conf`
moyenne	$n > 30$ ou normalité	`t.test(x)$conf`
variance	normalité	`sigma2.test(x)$conf`
médiane	aucune	`wilcox.test(x)$conf`
corrélation	binormale	`cor.test(x)$conf`

SECTION 11.3

Tests d'hypothèses usuels

Nous présentons brièvement la philosophie des **tests d'hypothèses**.

Les tests d'hypothèses consistent à proposer un outil d'aide à la décision pour éventuellement valider une **assertion d'intérêt** $\mathcal{H}_1$ (notée plus communément hypothèse « alternative »). Cette décision statistique sera prise à partir d'une **règle de décision** construite intuitivement à partir d'une **statistique de test** T (dépendant d'un échantillon d'observations). Comme cette décision peut être susceptible d'être erronée, il est important de mesurer le risque d'accepter $\mathcal{H}_1$ pour chacune des situations décrites par non-$\mathcal{H}_1$. Ces risques de décider à tort d'accepter $\mathcal{H}_1$ sont communément appelés les **risques de première espèce** et sont définis sous non-$\mathcal{H}_1$. On cherche alors à contrôler le pire de ces risques (le plus grand) défini pour une situation particulière de non-$\mathcal{H}_1$, communément notée $\mathcal{H}_0$ et appelée « hypothèse nulle ». Donc, à toute règle de décision que l'on se donne est associé un risque maximal. Parmi ces règles de décision potentielles, on choisit celle qui garantit un risque maximal α faible et fixé à l'avance (en général 5 %). Celui-ci est appelé le **seuil de signification** α du test.

Maintenant, lorsque nous appliquons cette règle de décision sur la base d'un jeu de données observées et que nous sommes conduits à accepter $\mathcal{H}_1$, il est intéressant de se demander jusqu'à quel seuil de signification l'acceptation de $\mathcal{H}_1$ aurait été maintenue. De manière alternative, lorsque nous sommes conduits à la non-acceptation de $\mathcal{H}_1$ à un risque α préspécifié, il est aussi intéressant de savoir à partir de quel seuil de signification (le plus petit) on est conduit à accepter $\mathcal{H}_1$. Dans ces deux cas de figure, la solution est donnée par ce que l'on appelle la **valeur-p** (aussi appelée p-valeur), ainsi définie comme le **risque (maximal) à encourir pour accepter** $\mathcal{H}_1$. Cette valeur-p est fournie par tout logiciel de statistique, et l'utilisateur saura alors comparer ce risque à un seuil de signification α fixé (par lui-même ou par certains usages). Son interprétation est extrêmement simple : plus la valeur-p est faible, plus la décision (d'accepter l'assertion d'intérêt $\mathcal{H}_1$) est fiable.

Dans le même contexte, nous pouvons mentionner la notion de **puissance d'un test**, qui est une fonction mesurant la probabilité d'accepter $\mathcal{H}_1$ pour toute situation. Ces notions seront expliquées et détaillées, en utilisant la force du langage R, dans la partie consacrée aux travaux pratiques. La suite de cette section présente un catalogue des procédures de test les plus classiques et des instructions R à utiliser pour les mettre en œuvre. Le seuil de signification des tests présentés ci-après sera fixé à $\alpha = 5$ %, sauf mention explicite du contraire.

Expert

Même si nous préférons notre approche des tests d'hypothèses qui met l'accent sur l'assertion d'intérêt $\mathcal{H}_1$, nous présentons ici une petite originalité mathématique justifiant l'utilisation par certains auteurs (souvent des mathématiciens) de la notation $\mathcal{H}_0$ pour désigner non-$\mathcal{H}_1$. Cela a un sens d'un point de vue mathématique puisque la logique d'un test d'hypothèses est semblable à un raisonnement par contradiction. Considérons ainsi l'exemple volontairement simpliste où l'on cherche à montrer $\mathcal{H}_1 : \mu_X > \mu_0$ sur la base d'un échantillon $\mathbf{X}_n = (X_1, \ldots, X_n)$ de v.a. *i.i.d.* de lois $\mathcal{N}(\mu_X, 1)$.

(1) Raisonnons par contradiction et supposons que notre hypothèse $\mathcal{H}_1$ soit fausse. Nous sommes alors dans le cas contraire de $\mathcal{H}_1$ qui **est noté ici** $\mathcal{H}_0 : \mu_X \leq \mu_0$. On est par exemple dans la situation où $\mu_X = \tilde{\mu}$ pour une certaine valeur $\tilde{\mu}$ inférieure ou égale à μ_0.

(2) On cherche alors à mesurer la plausibilité de l'hypothèse de contradiction $\mathcal{H}_0$, et pour cela on veut évaluer la négativité de l'écart inconnu $d = \mu_X - \mu_0$ qui est estimé par la statistique de test $T = \hat{\mu}_X - \mu_0 = \widetilde{T} + (\tilde{\mu} - \mu_0)$, où $\widetilde{T} = \left(\hat{\mu}_X - \tilde{\mu}\right) \sim \mathcal{N}(0, 1/n)$.

(3) Les données, qui portent l'information sur la valeur de μ_X puisqu'elles ont été générées suivant une loi ayant cette espérance, fournissent une réalisation t_{obs} de cette statistique T. Si $\mathcal{H}_0$ était vraie, alors la variable aléatoire $T = \hat{\mu}_X - \mu_0$, qui mesure l'écart entre μ_X et μ_0, aurait peu de chances de produire de grandes valeurs. Ainsi :

$$\begin{aligned} p(\tilde{\mu}) := P_{\mu_X=\tilde{\mu}}[T \geq t_{obs}] &= P_{\mu_X=\tilde{\mu}}[\widetilde{T} + \tilde{\mu} - \mu_0 \geq t_{obs}] \\ &= P_{\mu_X=\tilde{\mu}}[\widetilde{T} \geq t_{obs} - (\tilde{\mu} - \mu_0)] \\ &\overset{\text{d'après (1)}}{\leq} P_{\mu_X=\tilde{\mu}}[\widetilde{T} \geq t_{obs}] := \tilde{p} \\ &\quad \text{(indépendant de } \tilde{\mu} \text{ ici).} \end{aligned}$$

Notez que $p(\tilde{\mu})$ (dont on ne connaît pas la valeur numérique) peut être vue comme une extension de la valeur-p pour la situation $\tilde{\mu}$ et $\tilde{p}$ n'est qu'un majorant calculable qui sert à contrôler cette valeur-p étendue. Si la valeur $\tilde{p} = P[\mathcal{N}(0, 1/n) > t_{obs}]$ obtenue est très petite (et donc la probabilité $p(\tilde{\mu})$ l'est encore plus), on arrive alors à une (quasi) contradiction puisque l'événement $\{T \geq t_{obs}\}$ de probabilité $p(\tilde{\mu})$ s'est effectivement produit alors que nous ne l'attendions (quasiment) pas sous $\mathcal{H}_0$. Par contradiction, notre hypothèse $\mathcal{H}_1$ est donc très certainement vraie et $\tilde{p}$ (plus petit majorant des $p(\tilde{\mu})$, $\tilde{\mu} \leq \mu_0$, qui est d'ailleurs atteint lorsque $\tilde{\mu} = \mu_0$) exprime ainsi la force de contradiction (ou de conviction) de non-$\mathcal{H}_1$ ou la force de la décision de $\mathcal{H}_1$ (ou risque de se tromper en décidant $\mathcal{H}_1$). La probabilité $\tilde{p}$ n'est rien d'autre que la valeur-p que nous avions définie hors de cet encadré.

Pour finir, notez l'existence d'un livre [38] entièrement dédié aux tests d'hypothèses effectués à l'aide du logiciel R.

11.3.1 Tests paramétriques

11.3.1.1 Tests de moyenne

- **Comparaison de la moyenne théorique à une valeur de référence (cas à un échantillon)**

  ▶ *Descriptif du test :* Soit une variable quantitative X de moyenne théorique μ et de variance σ^2. À partir d'un échantillon de taille n, on veut comparer la moyenne théorique μ à une valeur de référence μ_0. Les hypothèses du test sont $\mathcal{H}_0 : \mu = \mu_0$ et $\mathcal{H}_1 : \mu \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| \mu_0$. Sous $\mathcal{H}_0$, la statistique de test est :

$$T = \sqrt{n}\left(\frac{\bar{X} - \mu_0}{\hat{\sigma}}\right) \sim \mathcal{T}(n-1).$$

  ▶ *Conditions de validité :* Normalité des données ou taille d'échantillon grande ($n > 30$).

  ▶ *Instruction R :* Il est possible d'utiliser la fonction `t.test()`.

  ▶ *Exemple d'application :* Dans le jeu de données INTIMA.MEDIA, on veut savoir si les personnes qui ont un indice de masse corporelle (IMC) supérieur à 30 ont une mesure de l'épaisseur de l'intima-média supérieure en moyenne à la mesure dans la population dont est issu l'échantillon. On suppose que cette moyenne théorique de la mesure intima-média dans cette population est égale à 0.58 mm.

```
> IMC <- poids/((taille/100)^2)
> mesure1 <- mesure[IMC>30]
> t.test(mesure1,mu=0.58,alternative="greater")
        One Sample t-test
data:  mesure1
t = 1.5272, df = 8, p-value = 0.08262
alternative hypothesis: true mean is greater than 0.58
95 percent confidence interval:
 0.5715358       Inf
sample estimates:
mean of x
0.6188889
```

  Il n'est pas possible de répondre positivement à la question au risque $\alpha = 5$ % préspécifié.

- **Comparaison de deux moyennes théoriques (cas à deux échantillons)**

  ▶ *Descriptif du test :* On considère deux variables quantitatives X_1 et X_2 (qui mesurent la même caractéristique, mais dans deux populations différentes). On suppose que X_1 a pour moyenne théorique μ_1 et pour variance σ_1^2 et X_2 a pour moyenne théorique μ_2 et pour variance σ_2^2. À partir des estimations calculées sur deux échantillons de tailles respectives n_1 et n_2 issus des deux populations, on veut comparer μ_1 et μ_2. Les hypothèses du test sont $\mathcal{H}_0 : \mu_1 = \mu_2$ et $\mathcal{H}_1 : \mu_1 \left|\begin{smallmatrix}>\\ \neq \\ <\end{smallmatrix}\right| \mu_2$. Sous $\mathcal{H}_0$, la statistique de test est :

  $$T = \frac{\bar{X}_1 - \bar{X}_2}{\hat{\sigma}\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \sim \mathcal{T}(n_1 + n_2 - 2)$$

  avec ici $\hat{\sigma}^2 = \frac{(n_1-1)\hat{\sigma}_1^2 + (n_2-1)\hat{\sigma}_2^2}{n_1+n_2-2}$, $\hat{\sigma}_1$ et $\hat{\sigma}_2$ les estimateurs des variances des deux populations.

  ▶ *Conditions de validité :* Normalité des variables X_1 et X_2 et variances égales.

  ▶ *Instruction R :* Il est possible d'utiliser la fonction `t.test()`.

  ▶ *Exemple d'application :* On veut savoir s'il y a une différence significative de la mesure de l'intima-média entre les femmes qui ont une activité sportive et celles qui n'en ont pas.

```
> mesure.SPORT.1 <- mesure[SPORT==1&SEXE==2]
> mesure.SPORT.0 <- mesure[SPORT==0&SEXE==2]
> t.test(mesure.SPORT.1,mesure.SPORT.0,var.equal=FALSE)
        Welch Two Sample t-test
data:  mesure.SPORT.1 and mesure.SPORT.0
t = -1.4693, df = 53.179, p-value = 0.1476
alternative hypothesis: true difference in means is not
                        equal to 0
95 percent confidence interval:
 -0.07488130  0.01155649
sample estimates:
mean of x mean of y
0.5130435 0.5447059
```

  Il n'est pas possible de répondre positivement à la question au risque $\alpha = 5$ % préspécifié.

Attention

Cependant, il faut vérifier l'hypothèse d'égalité des variances (voir la section suivante). Ainsi, on spécifiera la valeur du paramètre `var.equal` dans la fonction `t.test()` en fonction du résultat du test d'égalité des variances. La statistique de test est alors :

$$T^* = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{\hat{\sigma}_1^2}{n_1} + \frac{\hat{\sigma}_2^2}{n_2}}}.$$

Cette statistique suit une loi de Student où le degré de liberté est calculé par l'approximation de Satterthwaite. Dans le cas de grands échantillons, la statistique T^* suit une loi $\mathcal{N}(0,1)$.

- **Cas des échantillons appariés**
  - *Descriptif du test :* On veut comparer les moyennes théoriques de deux variables aléatoires X_1 et X_2 sur la base de deux échantillons appariés. Pour cela, on travaille avec la variable aléatoire différence $D = X_1 - X_2$, et l'on compare la moyenne théorique $\delta = \mu_1 - \mu_2$ de D à la valeur de référence 0. On se retrouve donc dans le cas du test de moyenne à un échantillon. Les hypothèses du test sont $\mathcal{H}_0 : \mu_1 - \mu_2 = 0$ et $\mathcal{H}_1 : \mu_1 - \mu_2 \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| 0$. Sous $\mathcal{H}_0$, la statistique de test est :

$$T = \sqrt{n}\frac{\bar{D}}{\hat{\sigma}} \sim \mathcal{T}(n-1).$$

  - *Conditions de validité :* Normalité des données ou taille d'échantillon grande ($n > 30$).

  - *Instruction R :* Il est possible d'utiliser la fonction `t.test()` avec le paramètre `paired=TRUE`.

  - *Exemple d'application :* On souhaite comparer les résultats de deux laboratoires d'analyse pour un examen particulier. Pour une série de quinze patients, on a pu faire effectuer le dosage nécessaire par chacun des deux laboratoires.

```
> dosage.labo1 <- c(22,18,28,26,13,8,21,26,27,29,25,24,
+                   22,28,15)
> dosage.labo2 <- c(25,21,31,27,11,10,25,26,29,28,26,23,
+                   22,25,17)
> t.test(dosage.labo1,dosage.labo2,paired=TRUE)
        Paired t-test
data:  dosage.labo1 and dosage.labo2
t = -1.7618, df = 14, p-value = 0.0999
```

```
alternative hypothesis: true difference in means is not
                        equal to 0
95 percent confidence interval:
 -2.0695338  0.2028671
sample estimates:
mean of the differences
             -0.9333333
```

Il n'est pas possible de décider, au risque $\alpha = 5$ % préspécifié, si les deux laboratoires donnent des résultats différents en moyenne.

Remarque

Ce test est valide quand n est grand ou que l'on peut faire une hypothèse normale des données. Sinon, un test non paramétrique du type test des signes ou test de Wilcoxon est à préconiser (voir la section sur les tests non paramétriques).

11.3.1.2 Tests de variance

- **Comparaison de la variance théorique à une valeur de référence (cas à un échantillon)**
  - ▶ *Descriptif du test :* Soit σ^2 la variance d'un caractère quantitatif X. Les hypothèses du test sont $\mathcal{H}_0 : \sigma^2 = \sigma_0^2$ *versus* $\mathcal{H}_1 : \sigma^2 \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| \sigma_0^2$. La statistique de test sous $\mathcal{H}_0$ est :

$$T = \frac{(n-1)\hat{\sigma}^2}{\sigma_0^2} \sim \chi^2(n-1).$$

  - ▶ *Conditions de validité :* Le caractère X est distribué suivant une loi normale.

  - ▶ *Instruction R :* Il est possible d'utiliser la fonction `sigma2.test()` du *package* associé à ce livre.

  - ▶ *Exemple d'application :* Une usine fabrique des boîtes de conserve de poids μ avec une précision $\sigma^2 = 10$. On veut montrer que la chaîne de production est déréglée (précision différente de $\sigma^2 = 10$). Voici les poids d'une série de vingt boîtes de conserve.

```
> poids <- c(165.1,171.5,168.1,165.6,166.8,170.0,168.8,
+       171.1,168.8,173.6,163.5,169.9,165.4,174.4,171.8,
+       166.0,174.6,174.5,166.4,173.8)
> sigma2.test(poids,var0=10)
        One-sample Chi-squared test for given variance
data:  poids
```

```
X-squared = 24.2045, df = 19, p-value = 0.3768
alternative hypothesis: true variance is not equal to 10
95 percent confidence interval:
  7.367682 27.176225
sample estimates:
var of x
12.73924
```

Il n'est pas possible de répondre positivement à la question au risque $\alpha = 5$ % préspécifié.

Remarque

Sans hypothèse de normalité et pour de grands échantillons, vous pouvez utiliser la fonction `asymp.test(x,parameter="var",reference=10)`. Cette fonction se trouve dans le *package* `asympTest`.

- **Comparaison de deux variances théoriques (cas à deux échantillons)**
  - *Descriptif du test :* Ce test est souvent utile comme préalable à d'autres tests comme celui de la comparaison de deux moyennes dans le cas de faibles effectifs. En effet, dans ce cas, la statistique n'est pas la même suivant que les variances de X_1 (variable concernant le premier échantillon) et de X_2 (variable concernant le second échantillon) peuvent être considérées comme égales ou non. Les hypothèses du test sont $\mathcal{H}_0 : \sigma_1^2 = \sigma_2^2$ *versus* $\mathcal{H}_1 : \sigma_1^2 \left| \begin{matrix} > \\ \neq \\ < \end{matrix} \right| \sigma_2^2$. La statistique de test sous $\mathcal{H}_0$ est :

$$T = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2} \sim \mathcal{F}(n_1 - 1, n_2 - 1).$$

  - *Conditions de validité :* Normalité de X_1 et X_2.

  - *Instruction* R *:* Il est possible d'utiliser la fonction `var.test()`.

  - *Exemple d'application :* Dans le jeu de données INTIMA.MEDIA, on veut savoir si dans la population des femmes il y a une différence significative de la variance de la mesure de l'intima-média entre les femmes qui ont une activité sportive et celles qui n'en ont pas.

```
> mesure.SPORT.1 <- mesure[SPORT==1&SEXE==2]
> mesure.SPORT.0 <- mesure[SPORT==0&SEXE==2]
> var.test(mesure.SPORT.1,mesure.SPORT.0)
        F test to compare two variances
data:  mesure.SPORT.1 and mesure.SPORT.0
F = 0.303, num df = 22, denom df = 33, p-value =
0.00468
```

```
alternative hypothesis: true ratio of variances is not
                        equal to 1
95 percent confidence interval:
 0.1431908 0.6815918
sample estimates:
ratio of variances
         0.3029910
```

On peut conclure qu'il y a une différence significative de la variance de la mesure de l'intima-média entre les femmes qui ont une activité sportive et celles qui n'en ont pas, au risque $\alpha = 5$ % de se tromper.

Astuce

Pensez à utiliser la fonction `asymp.test()` pour de grands échantillons sans hypothèse normale.

Renvoi

Pour la comparaison de plus de deux variances, voir le test de Bartlett qui sera présenté en analyse de la variance, page 548.

11.3.1.3 Tests de proportion

- **Comparaison d'une proportion théorique à une valeur de référence (cas à un échantillon)**
  - ▶ *Descriptif du test :* Soit p la fréquence inconnue d'un caractère dans une population donnée. On observe des données de présence/absence de ce caractère sur les individus d'un échantillon de taille n de cette population. Les hypothèses du test que nous considérons sont $\mathcal{H}_0 : p = p_0$ et $\mathcal{H}_1 : p \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| p_0$. La statistique de test sous $\mathcal{H}_0$ est :

$$U = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} \sim \mathcal{N}(0,1).$$

  - ▶ *Conditions de validité :* L'échantillon doit être suffisamment grand, il faut vérifier que $np_0 \geq 5$ et $n(1-p_0) \geq 5$.
  - ▶ *Instruction R :* Il est possible d'utiliser la fonction `prop.test()`.
  - ▶ *Exemple d'application :* Supposons (et cela est vraiment une supposition) que le recueil des données d'une étude de cas intitulée « Étude chez des femmes enceintes à Abidjan » ait été réalisé après une vaste campagne d'information et de prévention de l'infection par le VIH dont l'objectif était de réduire la proportion de personnes infectées par le

VIH, notamment chez les jeunes de 18 à 25 ans. Supposons encore que l'objectif à atteindre dans un premier temps soit la réduction de la prévalence dans cette population des femmes enceintes de 18 à 25 ans jusqu'à un taux inférieur à 10 %. On veut donc savoir à l'aide du sous-échantillon des femmes de 25 ans ou moins si le taux de prévalence de l'infection par le VIH est inférieur à $p_0 = 0.1$. Le jeu de données se trouve à `http://www.biostatisticien.eu/springeR/prevalsidafric.xls`.

```
> table(VIH[age<=25])
  0   1
137  10
> prop.test(10,147,0.1,alternative="less",correc=FALSE)
        1-sample proportions test without continuity
        correction
data:  10 out of 147, null probability 0.1
X-squared = 1.6697, df = 1, p-value = 0.09815
alternative hypothesis: true p is less than 0.1
95 percent confidence interval:
 0.000000 0.110572
sample estimates:
         p
0.06802721
```

Il n'est pas possible de répondre positivement à la question, avec un risque préspécifié α de se tromper.

- ▶ *Cas des petits échantillons :* Dans ce cas, on peut effectuer un calcul exact fondé sur la loi binomiale grâce à la fonction `binom.test()`.

```
> binom.test(10,147,0.1,alternative="less")
        Exact binomial test
data:  10 and 147
number of successes = 10, number of trials = 147,
p-value = 0.1208
alternative hypothesis: true probability of success is
                        less than 0.1
95 percent confidence interval:
 0.0000000 0.1126571
sample estimates:
probability of success
            0.06802721
```

Là encore, il n'est pas possible de répondre positivement à la question, avec un risque préspécifié α de se tromper.

- **Comparaison de deux proportions théoriques (cas à deux échantillons)**
  - ▶ *Descriptif du test :* Soit p_1 (respectivement p_2) la proportion inconnue d'individus présentant un certain caractère dans une population $\mathcal{P}_1$ (respectivement $\mathcal{P}_2$). On désire comparer p_1 et p_2. Pour cela, on utilise les fréquences (notées $\hat{p}_1$ et $\hat{p}_2$) d'apparition de ce caractère

dans deux échantillons représentatifs respectivement des deux populations de taille n_1 et n_2. Les hypothèses du test sont : $\mathcal{H}_0 : p_1 = p_2$ et $\mathcal{H}_1 : p_1 \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| p_2$. La statistique de test sous $\mathcal{H}_0$ est :

$$U = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{\hat{p}(1-\hat{p})}{n_1} + \frac{\hat{p}(1-\hat{p})}{n_2}}} \sim \mathcal{N}(0,1)$$

avec $\hat{p} = \frac{n_1\hat{p}_1 + n_2\hat{p}_2}{n_1+n_2}$.

- *Conditions de validité :* Grands échantillons, vérifiées si $n_1\hat{p} \geq 5$, $n_1(1-\hat{p}) \geq 5$, $n_2\hat{p} \geq 5$ et $n_2(1-\hat{p}) \geq 5$.

- *Instruction* R *:* La fonction à utiliser est `prop.test()`.

- *Exemple d'application :* Dans l'essai thérapeutique « Ditrame », la question sous-jacente est de savoir si le traitement a un effet sur le statut VIH de l'enfant. Si ce n'est pas le cas, alors le statut VIH de l'enfant est indépendant du traitement suivi par la mère. Pour tenter de répondre à cette question, on utilise le tableau croisé des variables Groupe de traitement de la mère (`TTTGRP`) et Statut VIH de l'enfant (`STATUTVIH`). Le tableau de contingence observé de ces deux variables est fourni. Le jeu de données se trouve à l'adresse `http://www.biostatisticien.eu/springeR/TME_Afrique.xls`.

```
> table(TTTGRP,STATUTVIH)
      STATUTVIH
TTTGRP   0   1   9
     0 139  59   3
     1 152  41   7
> tableau <- as.matrix(table(TTTGRP,STATUTVIH)[,c(2,1)])
> prop.test(tableau,correc=FALSE)
        2-sample test for equality of proportions without
        continuity correction
data:  tableau
X-squared = 3.7574, df = 1, p-value = 0.05257
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.0004122543  0.1715013839
sample estimates:
   prop 1    prop 2
0.2979798 0.2124352
```

On peut conclure, avec un risque de se tromper inférieur à 5 %, que la proportion théorique p_1 d'enfants atteints du VIH dans le groupe non traité est supérieure à la proportion théorique p_2 d'enfants atteints du VIH dans le groupe traité.

11.3.1.4 Tests de coefficient de corrélation

- **Comparaison d'un coefficient de corrélation théorique à une valeur de référence (cas à un échantillon)**
  - *Descriptif du test :* Soit ρ le coefficient de corrélation entre deux variables quantitatives X et Y. On souhaite tester les hypothèses $\mathcal{H}_0 : \rho = \rho_0$ *versus* $\mathcal{H}_1 : \rho \left| \begin{matrix} > \\ \neq \\ < \end{matrix} \right| \rho_0$. La statistique de test sous $\mathcal{H}_0$ est :

$$U = \frac{Z - \mu_Z}{\sigma_Z} \sim \mathcal{N}(0,1)$$

avec $Z = \frac{1}{2}\ln\left(\frac{1+\hat{\rho}}{1-\hat{\rho}}\right)$, $\mu_Z = \frac{1}{2}\ln\left(\frac{1+\rho_0}{1-\rho_0}\right)$, $\sigma_Z^2 = \frac{1}{n-3}$.

Dans le cas où l'on s'intéresse à l'association linéaire entre X et Y (testée en prenant $\rho_0 = 0$), la statistique de test alors utilisée sous $\mathcal{H}_0$ est :

$$T = \frac{\hat{\rho}\sqrt{n-2}}{\sqrt{1-R^2}} \sim \mathcal{T}(n-2).$$

  - *Conditions de validité :* Le couple (X, Y) suit une loi binormale.

  - *Instruction* R *:* Il est possible d'utiliser la fonction `cor.test()` dans le cas où l'on teste l'association linéaire entre X et Y. Sinon, pour une valeur autre que $\rho_0 = 0$, on peut utiliser la fonction `cor0.test()` disponible dans le *package* associé à ce livre.

  - *Exemple d'application :* Dans le jeu de données IMC.ENFANT, on s'intéresse à l'association linéaire entre la taille et le poids.

```
> cor.test(poids,taille)
        Pearson's product-moment correlation
data:  poids and taille
t = 13.4327, df = 150, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.6570527 0.8036174
sample estimates:
      cor
0.7389562
```

On peut donc conclure, avec un risque d'erreur inférieur ou égal à 5 %, qu'il y a une association linéaire entre la taille et le poids.

- **Comparaison de deux coefficients de corrélation théoriques (cas à deux échantillons)**
  - *Descriptif du test :* On dispose de deux populations indépendantes. On note ρ_1 le coefficient de corrélation entre X et Y dans la population 1,

ρ_2, le coefficient de corrélation entre X et Y dans la population 2. On cherche à tester l'égalité de ces deux coefficients de corrélation sur la base de deux échantillons de tailles n_1 et n_2. Les hypothèses du test sont $\mathcal{H}_0 : \rho_1 = \rho_2$ *versus* $\mathcal{H}_1 : \rho_1 \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| \rho_2$. La statistique de test sous $\mathcal{H}_0$ est :

$$U = \frac{Z_1 - Z_2}{\sqrt{1/(n_1 - 3) + 1/(n_2 - 3)}} \sim \mathcal{N}(0, 1)$$

avec $Z_1 = \frac{1}{2} \ln\left(\frac{1+\hat{\rho}_1}{1-\hat{\rho}_1}\right)$ et $Z_2 = \frac{1}{2} \ln\left(\frac{1+\hat{\rho}_2}{1-\hat{\rho}_2}\right)$, où l'on note $\hat{\rho}_1$ et $\hat{\rho}_2$ les estimateurs des coefficients de corrélation.

- *Conditions de validité :* Dans les deux populations, le couple (X, Y) suit une loi binormale.

- *Instruction* R *:* Il est possible d'utiliser la fonction `cor.test.2.sample()` disponible dans le *package* associé à ce livre.

- *Exemple d'application :* Dans le jeu de données IMC.ENFANT, on s'intéresse à comparer l'intensité de la relation taille-poids entre le groupe des filles et le groupe des garçons.

```
> indf <- which(SEXE=="F")  # Permet de récupérer les
                            # indices des filles.
> indg <- which(SEXE=="G")  # Permet de récupérer les
                            # indices des garçons.
> cor.test.2.sample(taille[indf],poids[indf],
+                   taille[indg],poids[indg])
$statistic
[1] -1.67379
$p.value
[1] 0.09417185
```

Il n'est pas possible, au risque d'erreur 5 %, de montrer une différence significative entre les deux coefficients de corrélation linéaire.

11.3.2 Tests d'indépendance

11.3.2.1 Test du χ^2 d'indépendance

- *Descriptif du test :* On a deux variables aléatoires qualitatives X_1 et X_2 (ou rendues qualitatives par regroupement) : X_1 a l modalités et X_2 a c modalités. On veut savoir si ces deux variables sont dépendantes ou encore si la répartition des modalités de la variable X_2 n'est pas la même dans chacune des l sous-populations constituées par les individus qui prennent une des l modalités de la variable X_1. On connaît les valeurs prises par ces deux variables pour n individus, ces données sont en général présentées dans un tableau de contingence appelé aussi tableau des effectifs observés

et l'on compare ce tableau observé au tableau théorique de répartition des n individus calculé sous l'hypothèse d'indépendance des deux variables. Les hypothèses sont $\mathcal{H}_0$: les variables X_1 et X_2 sont indépendantes et $\mathcal{H}_1$: les variables ne sont pas indépendantes. La statistique de test sous $\mathcal{H}_0$ est

$$X^2 = \sum_{i=1}^{l} \sum_{j=1}^{c} \frac{(N_{ij} - t_{ij})^2}{t_{ij}^2} \sim \chi^2((l-1)(c-1))$$

où N_{ij} représentant l'effectif observé et t_{ij} représente l'effectif théorique calculé sous $\mathcal{H}_0$, pour la modalité i de X_1 et j de X_2.

- *Conditions de validité :* Les effectifs théoriques t_{ij} doivent être supérieurs à 5 sinon on peut utiliser le χ^2 de Yates (si les effectifs sont supérieurs à 2.5 et seulement pour des tableaux 2×2) ou encore le test de Fisher exact.

- *Instruction* R *:* La fonction à utiliser est `chisq.test()` avec le paramètre `correct=FALSE`.

- *Exemple d'application :* Reprenons l'exemple de l'essai « Ditrame » où la question sous-jacente est bien de savoir si le traitement a un effet sur le statut VIH de l'enfant.

```
> tableau
      STATUTVIH
TTTGRP   1   0
     0 59 139
     1 41 152
> chisq.test(tableau,correct=FALSE)
        Pearson's Chi-squared test
data:  tableau
X-squared = 3.7574, df = 1, p-value = 0.05257
```

Il n'est pas possible de montrer, au risque 5 %, que le traitement a un effet sur le statut VIH de l'enfant.

Astuce

Il est aussi possible d'utiliser la commande `summary(table(STATUTVIH,TTTGRP))` pour obtenir le résultat du test du χ^2 d'indépendance.

Remarque

Il est possible d'effectuer un test du χ^2 d'indépendance mutuelle pour

$d \geq 2$ variables qualitatives (voir [8]). La statistique du χ^2 s'écrit alors

$$X^2 = \sum_{A \in \mathcal{I}_d} T_A \text{ avec } \begin{cases} T_A = X_A^2 \text{ si } |A| = 2 \ ; \\ T_A = X_A^2 - \sum_{\{B \subset A; 1 < |B| < |A|\}} T_B \text{ si } |A| > 2. \end{cases}$$

Dans l'équation ci-dessus, $\mathcal{I}_d$ désigne la famille de tous les sous-ensembles de $\{1, \ldots, d\}$ de cardinal strictement supérieur à 1. En outre, si l'on dispose de l'*array* de contingence `M` des d variables qualitatives, alors X_A^2 ($|A| \geq 2$) s'obtient au moyen de l'instruction `summary(margin.table(M,A))$statistic` où `A` est le vecteur des indices des éléments dans A. Par ailleurs, l'avantage de cette décomposition (orthogonale) de X^2 obtenue par récurrence est que chaque terme T_A décrit la dépendance mutuelle des variables indicées par l'ensemble A.
La fonction R à utiliser pour réaliser cette opération est `A.dep.tests()`. Elle est contenue dans le *package* `IndependenceTests`.

11.3.2.2 Test du χ^2 de Yates

- *Descriptif du test :* Le test du χ^2 avec correction de Yates (ou χ^2 de Yates) doit être utilisé lorsque l'on veut effectuer un χ^2 d'indépendance à partir d'un tableau 2×2, mais que l'un au moins des effectifs théoriques est inférieur à 5 ; il faut cependant que les effectifs théoriques ne soient pas trop petits (> 2.5). Le cadre général est donc le même que celui du χ^2 d'indépendance. La statistique de test sous $\mathcal{H}_0$ est

$$X^2 = \sum_{i=1}^{l} \sum_{j=1}^{c} \frac{(|N_{ij} - t_{ij}| - 0.5)^2}{t_{ij}^2} \sim \chi^2(1).$$

- *Instruction R :* Il est possible d'utiliser la fonction `chisq.test()`.

- *Exemple d'application :* Dans le jeu de données INTIMA.MEDIA, on sélectionne les personnes âgées de 50 ans ou plus et l'on se demande si leur comportement tabagique est lié au sexe.

```
> table.cont <- as.matrix(table(SEXE[AGE>=50],
+                               tabac[AGE>=50]))
> chisq.test(table.cont)$expected  # Tableau des effectifs
                                   # théoriques.

      0    1    2
1 2.88 0.48 2.64
2 9.12 1.52 8.36
> table.cont1 <- cbind(table.cont[,1],table.cont[,2]+
+                                     table.cont[,3])
```

```
> chisq.test(table.cont1)
        Pearson's Chi-squared test with Yates' continuity
        correction
data:  table.cont1
X-squared = 1.6732, df = 1, p-value = 0.1958
```

Il n'est pas possible de répondre positivement à la question posée, au risque 5 %.

11.3.2.3 Test de Fisher exact

- *Descriptif du test :* On utilise le test de Fisher exact lorsque les conditions pour employer les tests du χ^2 d'indépendance et du χ^2 de Yates ne sont pas satisfaites, c'est-à-dire dans le cas de petits effectifs théoriques. Cependant, il faut savoir que le test de Fisher est applicable pour des tableaux ayant plus de deux lignes ou de deux colonnes.

- *Instruction* R *:* La fonction à utiliser est `fisher.test()`.

- *Exemple d'application :* D'après la définition de l'obésité, un individu dont l'IMC est supérieur à 30 kg/m^2 est un individu dit obèse. On émet l'hypothèse que l'obésité est plus fréquente dans la population des moins de 50 ans. Ce qui veut dire que la variable IMC est liée à l'âge. On désire répondre à cette question à partir de l'étude « Intima-média ». Pour cela, on étudie la distribution croisée des variables dichotomiques suivantes : `-de 50 ans / 50 ans ou +` et `IMC<30 / IMC>30`. On trouve la répartition suivante :

```
> imc <- poids/(taille/100)^2
> obesite <- factor(imc<30)
> levels(obesite) <- c("IMC>30","IMC<30")
> age50 <- factor(AGE>=50)
> levels(age50) <- c("-de 50 ans","50 ans ou +")
> table(age50,obesite)
             obesite
age50          IMC>30 IMC<30
  -de 50 ans        8     77
  50 ans ou +       1     24
> imc.table <- as.matrix(table(obesite,age50))
> fisher.test(imc.table,alternative="greater")
        Fisher's Exact Test for Count Data
data:  imc.table
p-value = 0.3478
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.3830018       Inf
sample estimates:
```

```
odds ratio
  2.477058
```

Il n'est pas possible de montrer significativement que l'obésité est plus fréquente dans la population des moins de 50 ans.

Remarque

Le test de Fisher exact donne la possibilité d'avoir une hypothèse alternative unilatérale.

Astuce

La fonction `CrossTable()` du *package* `gmodels` permet de calculer les tests d'indépendance du Khi-deux, de Fisher exact et de McNemar. Elle produit également des sorties similaires à celles fournies par SAS ou SPSS.

11.3.3 Tests non paramétriques

11.3.3.1 Tests d'adéquation

- **Test de Shapiro-Wilk**

  ▶ *Descriptif du test :* Le test de Shapiro-Wilk est conçu spécialement pour étudier la non-normalité d'une variable continue X. C'est le test le plus puissant pour tester la normalité d'une distribution. Les hypothèses sont : $\mathcal{H}_0$: X suit une loi normale et $\mathcal{H}_1$: X ne suit pas une loi normale. La statistique de test est

$$W = \frac{T^2}{\hat{\sigma}^2}$$

  où $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$ et $T^2 = \frac{1}{n-1}\left[\sum_{i=1}^{[n/2]} a_i (X_{(n-i+1)} - X_{(i)})\right]^2$.
  Les a_i sont des coefficients présents dans la table de Shapiro-Wilk que l'on trouve dans la plupart des recueils de tables statistiques.

  ▶ *Instruction* R *:* L'instruction à utiliser est : `shapiro.test()`.

  ▶ *Exemple d'application :* Reprenons l'exemple du jeu de données INTIMA.MEDIA traité dans le cas de « comparaison de moyenne à un échantillon ». Nous cherchons à invalider l'hypothèse de normalité des mesures de l'épaisseur de l'intima-média pour des personnes ayant un indice de masse corporelle (IMC) supérieur à 30.

```
> mesure1
[1] 0.62 0.52 0.55 0.59 0.59 0.65 0.63 0.79 0.63
> shapiro.test(mesure1)
        Shapiro-Wilk normality test
data:  mesure1
W = 0.8835, p-value = 0.1708
```

Il n'est pas possible de montrer que les données ne sont pas gaussiennes, au risque 5 %.

- **Test du χ^2 d'ajustement**

Le test du χ^2 d'ajustement s'emploie lorsque l'on veut montrer qu'une variable qualitative ne suit pas une loi théorique donnée. Il permet éventuellement de tester si la loi d'une variable quantitative ne suit pas une loi théorique donnée, mais il faut au préalable la regrouper en classes pour la rendre qualitative.

- *Descriptif du test :* Soit X une variable aléatoire qualitative à k modalités. On émet l'hypothèse que X suit une loi donnée, c'est-à-dire que chaque modalité a une probabilité p_i. À partir d'un échantillon de taille n de la population, on teste la répartition de X suivant la loi donnée par les p_i. Les hypothèses du test sont $\mathcal{H}_0$: X suit une loi théorique spécifiée par les p_i et $\mathcal{H}_1$: X ne suit pas la loi théorique. La statistique de test est :
$$\sum_{i=1}^{k} \frac{(\mathrm{N}_i - np_i)^2}{np_i} \sim \chi^2(k-1)$$
où N_i est l'effectif que l'on observera pour la modalité i.

- *Conditions de validité :* Les effectifs théoriques np_i sont supérieurs ou égaux à 5.

- *Instruction* R *:* Il est possible d'utiliser la fonction `chisq.test()`.

- *Exemple d'application :* Une étude sur l'hypertension a été réalisée et on se pose la question suivante : est-ce que l'échantillon de patients retenus n'est pas représentatif de la population générale, d'un point de vue de la répartition ethnique ? On sait que dans la population générale de l'île Maurice, la répartition ethnique est la suivante : Hindous 51 %, Musulmans 17 %, Créoles 27 %, Chinois 3 % et autres 2 %. Les données se trouvent dans le fichier `http://www.biostatisticien.eu/springeR/HTA.xls`.

```
> table(ETHNIE)
ETHNIE
  1   2   3   4
225  77  99   1
```

```
> ni <- cbind(t(as.vector(table(ETHNIE))),0)
> chisq.test(ni,p=c(0.51,0.17,0.27,0.03,0.02))
        Chi-squared test for given probabilities
data:  ni
X-squared = 22.0659, df = 4, p-value = 0.0001945
```

On peut conclure que l'échantillon de patients retenus n'est pas significativement représentatif de la population générale (d'un point de vue de la répartition ethnique), avec un risque de se tromper inférieur ou égal à 5 %.

- **Test de Kolmogorov-Smirnov pour un échantillon**

  ▶ *Descriptif du test :* Le but de ce test est le même que celui du khi-deux d'ajustement. Il s'agit de comparer une distribution empirique à une distribution théorique entièrement spécifiée. On note F_0 la fonction de répartition spécifiée et F la fonction de répartition de X. On s'intéresse au point pour lequel la différence entre les deux fonctions de répartition est la plus grande en valeur absolue et on compare cette valeur à la valeur critique donnée par la table de Kolmogorov-Smirnov pour un échantillon. Les hypothèses du test sont : $\mathcal{H}_0 : F = F_0$ *versus* $\mathcal{H}_1 : F \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| F_0$. La statistique de test est

$$\mathrm{D} = \sup_x |\hat{\mathrm{F}}_{\mathbf{X}_n}(x) - F_0(x)|$$

  où $\hat{\mathrm{F}}_{\mathbf{X}_n}(\cdot)$ est la fonction de répartition empirique de l'échantillon $\mathbf{X}_n$.

  ▶ *Instruction* R *:* La fonction à utiliser est `ks.test()`.

  ▶ *Exemple d'application :* Reprenons l'exemple des boîtes de conserve et tentons d'invalider l'hypothèse de normalité nécessaire au test sur la variance. Les boîtes de conserve sont fabriquées avec un poids moyen $\mu = 170$g et une précision $\sigma^2 = 10$. Testons la normalité de la série des vingt boîtes de conserve fabriquées par l'usine.

```
> poids <- c(165.1,171.5,168.1,165.6,166.8,170.0,168.8,
+            171.1,168.8,173.6,163.5,169.9,165.4,174.4,
+            171.8,166.0,174.6,174.5,166.4,173.8)
> ks.test(poids,"pnorm",170,sqrt(10))
        One-sample Kolmogorov-Smirnov test
data:  poids
D = 0.1942, p-value = 0.4376
alternative hypothesis: two-sided
```

Il n'est pas possible, au risque 5 %, de montrer la non-normalité des données.

- **Test de Kolmogorov-Smirnov pour deux échantillons**

  ▶ *Descriptif du test :* Le but de ce test est de comparer deux distributions notées F_1 et F_2. On s'intéresse au point pour lequel la différence entre les deux fonctions de répartition empiriques (notées $\hat{F}_{\mathbf{X}_{n,1}}$ et $\hat{F}_{\mathbf{X}_{n,2}}$) est la plus grande en valeur absolue et on compare cette valeur à la valeur critique donnée par la table de Kolmogorov-Smirnov pour deux échantillons. Les hypothèses du test sont : $\mathcal{H}_0 : F_1 = F_2$ *versus* $\mathcal{H}_1 : F_1 \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| F_2$. La statistique de test est

$$\mathrm{D} = \sup_x |\hat{F}_{\mathbf{X}_{n,1}}(x) - \hat{F}_{\mathbf{X}_{n,2}}(x)|.$$

  ▶ *Instruction* R *:* Il est possible d'utiliser la fonction `ks.test()`.

  ▶ *Exemple d'application :* On se réfère au jeu de données INTIMA.MEDIA et l'on se demande si les hommes qui ont arrêté de fumer sont plus jeunes que les hommes qui fument. Dans cet échantillon, il y a douze fumeurs et neuf anciens fumeurs. Les effectifs sont donc faibles, on ne connaît pas la distribution de la variable âge dans cette population, on ne peut donc pas comparer les moyennes par un test de Student. Le test ici est unilatéral, car l'hypothèse est que les fumeurs sont plus jeunes que les anciens fumeurs.

```
> table(tabac,SEXE)
      SEXE
tabac  1  2
    0 32 40
    1  9  9
    2 12  8
> ks.test(AGE[SEXE==1&tabac==1],AGE[SEXE==1&tabac==2],
+         alternative="greater")
        Two-sample Kolmogorov-Smirnov test
data:  AGE[SEXE == 1 & tabac == 1] and AGE[SEXE == 1 &
       tabac == 2]
D^+ = 0.6389, p-value = 0.01502
alternative hypothesis: the CDF of x lies above that of y
```

  Dans la population étudiée, on a montré, au risque 5 % de se tromper, que les hommes qui ont arrêté de fumer sont plus jeunes que les hommes qui fument.

Astuce

Le package `PoweR` contient plusieurs autres tests d'ajustement.

11.3.3.2 Tests de position

- **Test du signe ou test de la médiane pour un échantillon**

  ▶ *Descriptif du test :* Il s'agit de comparer la médiane théorique d'une variable quantitative X (notée m_e) à une valeur de référence (notée m_0) sans aucune hypothèse sur les données. Les hypothèses du test sont $\mathcal{H}_0 : m_e = m_0$ ou de façon équivalente $P(X - m_0 > 0) = 0.5$ et $\mathcal{H}_1 : m_e \left|\begin{smallmatrix}>\\ \neq \\ <\end{smallmatrix}\right| m_0$ ou de façon équivalente $P(X - m_0 > 0) \left|\begin{smallmatrix}>\\ \neq \\ <\end{smallmatrix}\right| 0.5$. La statistique de test sous $\mathcal{H}_0$ est :

  $$K \sim \mathcal{B}in(n, 0.5)$$

  où K est le nombre de valeurs strictement supérieures à m_0. Ce test revient donc à faire un test de proportion.

  ▶ *Instruction R :* Il est possible d'utiliser les fonctions `prop.test()` ou `binom.test()`.

  ▶ *Exemple d'application :* La médiane des prix des appartements du type T2 de la région grenobloise en 2008 était de 130 000 euros. On dispose d'un échantillon de taille $n = 32$ des prix des T2 (en milliers d'euros) provenant du magazine mensuel gratuit *L'Offre Immobilière* numéro 91 (janvier 2009). On se demande s'il y a une tendance à l'augmentation des prix.

```
> m0 <- 130
> prix <- c(230.00,148.00,126.00,134.62,155.00,157.70,
+           160.00,225.00,125.00,109.00,157.00,115.00,
+           125.00,225.00,118.00,179.00,176.00,125.00,
+           123.00,180.00,151.00,120.00,143.00,170.00,
+           190.00,233.00,148.72,189.00,121.00,149.00,
+           225.00,240.00)
> sum(prix-m0>0)
[1] 22
> median(prix)
[1] 153
> prop.test(22,32,0.5,"greater")
        1-sample proportions test with continuity
         correction
data:  22 out of 32, null probability 0.5
X-squared = 3.7812, df = 1, p-value = 0.02591
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
 0.5266965 1.0000000
sample estimates:
     p
0.6875
```

On peut donc conclure, avec un risque de se tromper inférieur ou égal à 5 %, qu'il y a une tendance à l'augmentation des prix.

- **Test du signe ou test de la médiane pour deux échantillons indépendants**

  ▶ *Descriptif du test :* Le test de la médiane est un test qui compare les médianes (notées m_{e_1} et m_{e_2}) de deux variables quantitatives X_1 et X_2 à partir des données de deux échantillons indépendants de ces variables. Les hypothèses du test sont $\mathcal{H}_0 : m_{e_1} = m_{e_2}$ et $\mathcal{H}_1 : m_{e_1} \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| m_{e_2}$. Sous $\mathcal{H}_0$, on calcule la médiane commune $\widehat{m_e}$ aux deux échantillons. On construit ensuite un tableau 2×2 de répartition des valeurs qui sont supérieures ou inférieures à la médiane suivant les échantillons. Ce tableau se traite comme un tableau de contingence du χ^2. On est donc amené à faire un test du χ^2 ou encore un test du χ^2 corrigé de Yates ou enfin un test de Fisher exact suivant les effectifs.

  ▶ *Instruction* R *:* Il est possible d'utiliser les fonctions `chisq.test()` ou `fisher.test()`.

  ▶ *Exemple d'application :* On se réfère au jeu de données INTIMA.MEDIA et l'on se demande si les femmes qui ont arrêté de fumer sont plus jeunes que les femmes qui fument. Dans cet exemple, il y a neuf fumeuses et huit anciennes fumeuses. Les effectifs sont donc faibles, on n'a aucune hypothèse de distribution normale de la variable âge dans cette population. On ne peut donc pas comparer les moyennes, il peut alors être intéressant de faire un test de la médiane.

```
> Me <- median(AGE[SEXE==2&tabac>0])
> tab.obs <- table(tabac[tabac>0&SEXE==2&AGE!=Me],
+                  AGE[SEXE==2&tabac>0&AGE!=Me]>Me)
> rownames(tab.obs) <- c("A arrêté de fumer","Fume")
> colnames(tab.obs) <- c("AGE<ME","AGE>ME")
> tab.obs

                  AGE<ME AGE>ME
  A arrêté de fumer      6      2
  Fume                   2      6
> fisher.test(tab.obs,alt="greater")
        Fisher's Exact Test for Count Data
data:  tab.obs
p-value = 0.06597
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.878644      Inf
sample estimates:
```

```
odds ratio
  7.613556
```

On ne peut donc pas conclure, avec un risque de se tromper inférieur ou égal à 5 %, que les femmes qui ont arrêté de fumer sont plus jeunes que les femmes qui fument.

- **Test du signe pour deux échantillons appariés**

  ▶ *Descriptif du test :* On dispose de deux séries quantitatives appariées que l'on désire comparer. On travaille avec la série des différences. On élimine les paires concordantes (c'est-à-dire les paires pour lesquelles les deux valeurs sont égales) et l'on ne garde que les n paires discordantes (différence non nulle entre les deux valeurs). Le test des signes ne tient compte que du signe de la différence des paires discordantes. On appelle N^+ le nombre de paires positives et N^- le nombre de paires négatives. La statistique de test est :

$$\mathrm{S} = \min(\mathrm{N}^+, \mathrm{N}^-) \sim \mathcal{B}in(n, 0.5).$$

  ▶ *Instruction* R *:* Il est possible d'utiliser les fonctions `prop.test()` ou `binom.test()` après avoir calculé la réalisation de S. À noter que dans le cas où $n \geq 20$, on peut approcher la loi de S par une $\mathcal{N}(\frac{n}{2}, \frac{n}{4})$.

  ▶ *Exemple d'application :* On reprend l'exemple des résultats des deux laboratoires, traité avec le test de comparaison de moyenne sous hypothèse gaussienne. Le test des signes ne requiert aucune hypothèse.

```
> dosage.labo1 <- c(22, 18, 28, 26, 13, 8, 21, 26, 27,
+                   29, 25, 24, 22, 28, 15)
> dosage.labo2 <- c(25, 21, 31, 27, 11, 10, 25, 26, 29,
+                   28, 26, 23, 22, 25, 17)
> dif <- (dosage.labo1-dosage.labo2)
> nmoins <- sum(dif<0)
> nplus <- sum(dif>0)
> binom.test(min(nplus,nmoins),nplus+nmoins)
        Exact binomial test
data:  min(nplus, nmoins) and nplus + nmoins
number of successes = 4, number of trials = 13, p-value
= 0.2668
alternative hypothesis: true probability of success is
                        not equal to 0.5
95 percent confidence interval:
 0.0909204 0.6142617
sample estimates:
probability of success
             0.3076923
```

Il n'est pas possible de décider, au risque $\alpha = 5$ % préspécifié, que les deux laboratoires donnent des résultats différents.

- **Test des rangs de Wilcoxon ou Mann-Whitney pour deux échantillons indépendants**

  ▶ *Descriptif du test :* Le test de Wilcoxon ou test des rangs est un test non paramétrique qui teste l'égalité de deux distributions (notées F_1 et F_2). Les hypothèses du test sont : $\mathcal{H}_0 : F_1 = F_2$ *versus* $\mathcal{H}_1 : F_1 \left|\begin{smallmatrix}>\\ \neq \\ <\end{smallmatrix}\right| F_2$. Le principe du test est le suivant : on trie les valeurs des deux séries mises en commun dans le sens croissant, on attribue le rang 1 à la plus petite valeur, le rang 2 à la valeur suivante et ainsi de suite. On calcule ensuite le score de chacune des séries en sommant les rangs obtenus pour chacune d'elles. À l'aide d'une table adéquate, on décide si ces scores sont compatibles avec l'hypothèse $\mathcal{H}_0$ d'égalité des distributions. Par convention, la statistique de test S est le score de l'échantillon qui a le plus petit effectif. On utilise aussi parfois la statistique $\mathrm{W} = \mathrm{S} - \frac{n_0(n_0+1)}{2}$.

  ▶ *Conditions de validité :* Dans le cas où $n_0 = \min(n_1, n_2) \leq 10$, la statistique W ne suit pas une loi usuelle, mais on trouve les probabilités correspondantes dans la table de Mann-Whitney/Wilcoxon (fonction `pwilcox()`).
  Dans le cas où $\min(n_1, n_2) \geq 10$, on peut considérer que $\mathrm{W} \sim \mathcal{N}\left(\mu_W = \frac{n_1 n_2}{2}; \sigma_W^2 = n_1 n_2 \frac{n_1+n_2+1}{12}\right)$. Dans le cas d'*ex æquo* parmi les rangs, on utilisera aussi une approximation normale, mais avec $\sigma_W^2 = \frac{n_1 n_2}{12} \times \left[(n_1 + n_2 + 1) - \sum_{j=1}^{g} \frac{t_j^3 - t_j}{(n_1+n_2)(n_1+n_2-1)}\right]$ où g est le nombre de groupes d'*ex æquo* et t_j le nombre de rangs *ex æquo* dans le groupe j.

  ▶ *Instruction* R *:* Il est possible d'utiliser la fonction `wilcox.test()`. On utilisera le paramètre `exact=FALSE` pour le calcul approché par la loi normale.

  ▶ *Exemple d'application :* On se réfère au jeu de données INTIMA.MEDIA et l'on se demande cette fois-ci si les hommes qui fument sont plus vieux que les hommes qui ont arrêté de fumer. Dans cet échantillon, il y a neuf fumeurs et douze anciens fumeurs.

```
> wilcox.test(AGE[SEXE==1&tabac==2],AGE[SEXE==1&tabac==1],
+     exact=FALSE,alternative="greater")
        Wilcoxon rank sum test with continuity correction
data:  AGE[SEXE == 1 & tabac == 2] and AGE[SEXE == 1 &
 tabac == 1]
W = 88.5, p-value = 0.007756
alternative hypothesis: true location shift is greater
than 0
```

  On peut donc conclure, avec un risque de se tromper de 5 %, que les hommes qui fument sont plus vieux que les hommes qui ont arrêté.

Attention

Pour retrouver la statistique W, il faut mettre l'échantillon le plus petit en premier dans la fonction `wilcox.test()`.

- **Test de Wilcoxon pour deux échantillons appariés**

  ▶ *Descriptif du test :* On dispose de deux séries quantitatives appariées que l'on désire comparer. On travaille avec la série des différences. On élimine les paires concordantes (c'est-à-dire les paires pour lesquelles les deux valeurs sont égales) et l'on ne garde que les paires discordantes (différence non nulle entre les deux valeurs). On note n le nombre de paires discordantes. On ordonne les valeurs en fonction de leurs valeurs absolues puis on leur attribue un rang. On affecte la moyenne des rangs correspondants lorsque les valeurs absolues sont égales. On appelle ensuite S^+ le score de la série des différences qui étaient positives et S^- le score de la série des différences qui étaient négatives. Les scores se calculent avec le même principe que dans le test de Wilcoxon. Les deux hypothèses du test sont : $\mathcal{H}_0 : F_1 = F_2$ *versus* $\mathcal{H}_1 : F_1 \left|\begin{matrix}>\\ \neq \\ <\end{matrix}\right| F_2$. On peut choisir S^+ ou S^- comme statistique de test.

  ▶ *Conditions de validité :* Dans le cas où $n \leq 30$, la statistique de test ne suit pas une loi usuelle, mais on trouve les probabilités correspondantes dans la table de Wilcoxon pour séries appariées.
  Dans le cas où $n \geq 30$, S^+ et S^- suivent une loi $\mathcal{N}\left(\frac{n(n+1)}{4}; \frac{n(n+1)(2n+1)}{24}\right)$.

  ▶ *Instruction R :* R utilise par défaut la statistique $V = S^+$ au travers de la fonction `wilcox.test()` avec le paramètre `paired=TRUE`. On utilisera le paramètre `exact=FALSE` pour le calcul approché par la loi normale.

  ▶ *Exemple d'application :* On reprend l'exemple des résultats des deux laboratoires, traité avec le test des signes. Ce test est plus puissant que le test des signes, car il tient aussi compte de la valeur absolue des écarts.

```
> dosage.labo1 <- c(22, 18, 28, 26, 13, 8, 21, 26, 27,
+                   29, 25, 24, 22, 28, 15)
> dosage.labo2 <- c(25, 21, 31, 27, 11, 10, 25, 26, 29,
+                   28, 26, 23, 22, 25, 17)
> wilcox.test(dosage.labo1,dosage.labo2,paired=T,
>             exact=FALSE)
        Wilcoxon signed rank test with continuity
                    correction
data:  dosage.labo1 and dosage.labo2
V = 22, p-value = 0.1047
```

```
alternative hypothesis: true location shift is not
                        equal to 0
```

Il n'est pas possible de décider, au risque 5 % préspécifié, que les deux laboratoires donnent des résultats différents.

11.3.4 Tableau récapitulatif des tests usuels

Le tableau suivant résume tous les tests qui ont été présentés.

TABLE 11.4 – Les tests usuels.

Nature	Données	Conditions de validité	Fonction R
Tests paramétriques :			
moyenne	1 échantillon	$n > 30$ ou normalité	`t.test(x,...)`
	2 échantillons	normalité et variances égales	`t.test(x,y,...)`
	2 échantillons	normalité	`t.test(x,y,var.equal=F)`
	2 éch. appariés	$n > 30$ ou normalité	`t.test(x,y,paired=T)`
variance	1 échantillon	normalité	`sigma2.test(x,...)`
	2 échantillons	normalité	`var.test(x,y,...)`
	2 échantillons	grand échantillon	`asymp.test(x,y,...)`
corrélation	1 échantillon	normalité, $\mathcal{H}_0 : \rho = \rho_0$	`cor.test(x,y..)`
	2 échantillons	normalité	`cor.test.2.sample(x,y,...)`
proportion	1 échantillon	$np \geq 5$ et $n(1-p) \geq 5$	`prop.test(x,...)`
	1 échantillon		`binom.test(x,...)`
	2 échantillons	grand échantillon	`prop.test(x,y,...)`
Tests d'indépendance :			
χ^2 d'indépendance	tableau de contingence	effectifs théoriques ≥ 5	`chisq.test(.,correct=F)`
χ^2 de Yates	tableau 2×2	effectifs théoriques ≥ 2.5	`chisq.test()`
Fisher exact	tableau de contingence		`fisher.test()`
Tests d'adéquation :			
Shapiro-Wilk	1 échantillon		`shapiro.test(x,...)`
χ^2 d'ajustement	1 échantillon	effectifs théoriques ≥ 5	`chisq.test()`
Kolmogorov-Smirnov	1 échantillon		`ks.test(x,.)`
	2 échantillons		`ks.test(x,y)`
Tests de position :			
médiane	1 échantillon		`binom.test(x,)`
test du signe	2 échantillons		`fisher.test(x,y,)`
	2 éch. appariés		`binom.test(x,y,paired=T)`
Mann-Whitney	2 échantillons	$\min(n_1, n_2) \geq 10$	`wilcox.test(x,y,exact=F)`
Mann-Whitney	2 échantillons	$\min(n_1, n_2) \leq 10$	`wilcox.test(x,y)`
Wilcoxon	2 éch. appariés		`wilcox.test(x,y,paired=T)`

SECTION 11.4

Autres tests d'hypothèses

De nombreux autres tests sont disponibles dans R. Par exemple, vous pouvez obtenir une première liste de tests en tapant l'instruction suivante :

```
> apropos(".test")
 [1] "A.dep.tests"           "ansari.test"
 [3] "as.krandtest"          "as.randtest"
 [5] "as.rtest"              "asymp.test"
 [7] "bartlett.test"         "binom.test"
 [9] "Box.test"              "chisq.test"
[11] "combine.randtest.rlq" "cor0.test"
[13] "cor0.test"             "cor2.test.2.sample"
[15] "cor.test"              "cor.test.2.sample"
[17] "file_test"             "fisher.test"
[19] "fligner.test"          "friedman.test"
[21] "kruskal.test"          "ks.test"
[23] "mantelhaen.test"       "mantel.randtest"
[25] "mantel.rtest"          "mauchly.test"
[27] "mcnemar.test"          "mood.test"
[29] "multispati.randtest"   "multispati.rtest"
[31] "oneway.test"           "ormidp.test"
[33] "pairwise.prop.test"    "pairwise.t.test"
[35] "pairwise.wilcox.test" "poisson.test"
[37] "power.anova.test"      "power.prop.test"
[39] "power.t.test"          "PP.test"
[41] "procuste.randtest"     "procuste.rtest"
[43] "prop.test"             "prop.trend.test"
[45] "quade.test"            "randtest"
[47] "rate2by2.test"         "rtest"
[49] "RVdist.randtest"       "RV.rtest"
[51] "shapiro.test"          "sigma2.test"
[53] "tab2by2.test"          "t.test"
[55] ".valueClassTest"       "var0.test"
[57] "var.test"              "wilcox.test"
[59] "writeSTL"
```

Remarque

D'autres tests sont encore disponibles dans divers *packages*. Par exemple, le *package* `nortest` propose différents tests d'adéquation à la normalité. Les différentes fonctions associées à ces tests sont :

```
> require("nortest")
> ls("package:nortest")
[1] "ad.test"      "cvm.test"     "lillie.test"
[4] "pearson.test" "sf.test"
```

Termes à retenir

`t.test()` : test et intervalle de confiance pour la moyenne
`var.test()` : test pour l'égalité des variances
`prop.test()` : intervalle de confiance approché pour la proportion
`binom.test()` : intervalle de confiance exact pour la proportion
`cor.test()` : test et intervalle de confiance pour la corrélation
`chisq.test()` : test du χ^2
`fisher.test()` : test d'indépendance de Fisher
`ks.test()` : test d'adéquation de Kolmogorov-Smirnov
`shapiro.test()` : test de normalité de Shapiro-Wilk
`med.test()` : test de la médiane
`wilcox.test()` : test de position de Wilcoxon
`boot` : *package* pour le *bootstrap*

Exercices

11.1- Quelle fonction permet d'obtenir les quantiles d'une loi binomiale ?

11.2- À quoi sert la fonction `pnorm()` ?

11.3- Donnez l'instruction R permettant d'obtenir un intervalle de confiance pour la moyenne à partir d'un échantillon de taille 50.

11.4- Expliquez ce qui différencie les fonctions `prop.test()` et `binom.test()`.

11.5- Citez deux fonctions permettant de comparer deux fonctions de répartition à partir de deux échantillons.

11.6- Quelle est la fonction permettant de savoir si un échantillon ne provient pas d'une loi normale ?

11.7- Quelles sont les fonctions permettant de tester la dépendance entre deux caractères qualitatifs ?

11.8- Quel *package* contient plusieurs fonctions permettant d'effectuer des intervalles de confiance par *bootstrap* ?

11.9- Quel est le paramètre formel de la fonction `t.test()` permettant d'effectuer un test apparié ?

11.10- Quelle est la différence entre un χ^2 d'indépendance et un χ^2 d'ajustement ? Comment effectuez-vous ces deux tests sous R ?

Fiche de TP

A- Étude sur les intervalles de confiance

L'objectif de ce TP est de comprendre l'interprétation de l'intervalle de confiance. Pour un intervalle de confiance de niveau $1-\alpha$ d'un certain paramètre inconnu, il n'est en effet pas correct de dire qu'il y a $100\times(1-\alpha)$ % de chances que ce paramètre soit dans l'intervalle réalisé. En effet, le paramètre inconnu est égal à une valeur unique qui ne fluctue pas, et la probabilité qu'il appartienne à l'intervalle réalisé est donc 0 ou 1. En revanche, il est correct de dire qu'il y a un risque de 5 % de se tromper en déclarant que le paramètre appartient à l'intervalle de confiance réalisé.

• Étude de l'intervalle de confiance de la moyenne

11.1- Simulez $M = 50\,000$ échantillons de taille $n = 20$ suivant une loi normale de moyenne $\mu = -1.2$ et de variance $\sigma^2 = 2$.

11.2- Sur chaque échantillon, effectuez une estimation par intervalle de confiance à 90 % de la moyenne μ.

11.3- Calculez la proportion d'intervalles contenant la valeur $\mu = -1.2$. Que remarquez-vous ?

11.4- Refaites cette procédure pour la valeur $\mu = 1$, avec des échantillons de taille $n = 100$ provenant d'une loi du $\chi^2(1)$.

11.5- Même travail que la question précédente avec $n = 10$ et une loi du $\chi^2(1)$. Que remarquez-vous ? Comment expliquez-vous cela ?

11.6- Simulez un échantillon de taille $n = 20$ suivant une loi normale de moyenne $\mu = -1.2$ et de variance $\sigma^2 = 2$. Calculez un intervalle de confiance de niveau de confiance 95 % pour μ.

11.7- Effectuez la même opération pour des échantillons de taille croissante $n = 50, 100, 1\,000, 10\,000, 100\,000$. Que constatez-vous ?

11.8- Pour chacun des six échantillons ci-dessus, calculez la valeur observée de la statistique du test de Student pour l'hypothèse $\mathcal{H}_1 : \mu \neq 0$ ainsi que la valeur-p du test (au niveau de signification 5 %). Que constatez-vous ? Comment l'expliquez-vous ?

11.9- Pour l'échantillon de taille $n = 100\,000$, calculez un intervalle de confiance de niveau 95 % et calculez la valeur-p du test pour l'hypothèse $\mathcal{H}_1 : \mu \neq -1.1$. Comparez cette valeur-$p$ à celle trouvée à la question précédente pour $n = 50$. Qu'en déduisez-vous ?

• Étude de l'intervalle de confiance par la méthode du *bootstrap*

11.1- Simulez $M = 500$ échantillons de taille $n = 20$ suivant une loi exponentielle d'espérance $1/\lambda = 10$.

11.2- Sur chaque échantillon, effectuez une estimation par intervalle de confiance à 90 % de la moyenne $1/\lambda$ par la méthode percentile du *bootstrap*.

11.3- Vérifiez par la méthode décrite précédemment le niveau de l'intervalle de confiance proposé.

11.4- Comparez ce niveau avec celui obtenu par l'intervalle de confiance classique de la moyenne (procédure `t.test()`).

B- Étude des risques dans les tests d'hypothèses

L'objectif de ce TP est d'explorer les risques associés aux tests d'hypothèses, à savoir :

11.1- $P[\text{décider } \mathcal{H}_1 | \mathcal{H}_0 \text{ est vraie}] = \alpha$ le risque de décider $\mathcal{H}_1$ alors que la réalité est $\mathcal{H}_0$;

11.2- $P[\text{ne pas décider } \mathcal{H}_1 | \mathcal{H}_1 \text{ est vraie}] = \beta$ le risque de ne pas décider $\mathcal{H}_1$ alors que la réalité est $\mathcal{H}_1$.

• Étude du risque de première espèce

11.1- Simulez $M = 500$ échantillons de taille $n = 20$ suivant une loi normale de moyenne $\mu = 4$ et de variance $\sigma^2 = 1.2$.

11.2- Sur chaque échantillon, effectuez au seuil $\alpha = 5$ % le test de Student pour les hypothèses $\mathcal{H}_0 : \mu = 4$ et $\mathcal{H}_1 : \mu \neq 4$.

11.3- Comptez le nombre de fois que vous décidez $\mathcal{H}_1$. Quel résultat attendiez-vous ?

11.4- Augmentez la taille M des simulations.

11.5- Refaites cette procédure pour la valeur $\mu = 1$, avec des échantillons de taille $n = 100$ provenant d'une loi du $\chi^2(1)$.

11.6- Même travail que la question précédente avec $n = 10$. Que remarquez-vous ? Comment expliquez-vous cela ?

11.7- En fait, pour chacun des $M = 500$ échantillons, on prend la décision soit d'accepter $\mathcal{H}_1$, soit de ne pas l'accepter. Nous pouvons noter d_j ($1 \leq j \leq M$) la variable aléatoire qui prend la valeur 1 si l'on décide $\mathcal{H}_1$ et 0 sinon (fondé sur le j-ième échantillon). La variable d_j suit une loi de Bernoulli de paramètre $p = P[\mathrm{d}_j = 1] = P[\text{décider } \mathcal{H}_1]$. Les variables d_j étant indépendantes, la variable $\mathrm{d} = \sum_{j=1}^{M} \mathrm{d}_j$ suit une loi binomiale $\mathcal{B}in(M, p)$. Elle compte le nombre de fois que l'on décide $\mathcal{H}_1$. Si $\mathcal{H}_0$ est vraie, et que le test est bien construit (valeur critique choisie correctement), alors on devrait avoir $p = P[\text{décider } \mathcal{H}_1 | \mathcal{H}_0 \text{ est vraie}] = \alpha$.
Calculez un intervalle de confiance de niveau de confiance 95 % pour le

paramètre p puis concluez.

• Étude de la puissance

11.1- Simulez $M = 500$ échantillons de taille $n = 20$ suivant une loi normale de moyenne $\mu = 5$ et de variance $\sigma^2 = 1.2$.

11.2- Sur chaque échantillon, effectuez au seuil $\alpha = 5$ % le test de Student suivant $\mathcal{H}_0 : \mu = 4$ et $\mathcal{H}_1 : \mu \neq 4$.

11.3- Comptez le nombre de fois que vous montrez $\mathcal{H}_1$. Donnez alors une estimation de la puissance de ce test pour la situation de $\mathcal{H}_1$ pour laquelle $\mu = 5$.

11.4- Augmentez la taille de l'échantillon à $n = 100$ et estimez la puissance de ce test. Que remarquez-vous ?

11.5- Refaites cette procédure pour le test $\mathcal{H}_0 : \mu = 1$ et $\mathcal{H}_1 : \mu \neq 1$, avec des échantillons de taille $n = 100$ provenant d'une loi du $\chi^2(2)$.

11.6- Même travail que la question précédente avec $n = 10$. Que remarquez-vous ? Comment expliquez-vous cela ?

C- Quelques exemples pratiques

• Étude vache

La quantité de bactéries par cm^3 de lait provenant de huit vaches différentes est estimée après la traite et 24 heures plus tard. On se demande s'il existe un accroissement significatif du nombre de bactéries par cm^3 au cours du temps.

Vache	Juste après la traite	24 h après la traite
1	12 000	11 000
2	13 000	20 000
3	21 500	31 000
4	17 000	28 000
5	15 000	26 000
6	22 000	30 000
7	11 000	16 000
8	21 000	29 000

11.1- Répondez à la question en supposant une hypothèse normale des données.

11.2- Répondez à la question par un test du signe.

11.3- Répondez à la question par un test de Mann-Whitney.

• Athlètes Est-allemandes

Dans les années 1970, les athlètes féminines de l'Allemagne de l'Est étaient réputées pour leur forte corpulence. Le comité d'éthique olympique de l'époque,

mettant en doute cette étonnante « virilité », avait fait appel au service du Dr Volker Fischbach. Celui-ci sélectionna neuf athlètes féminines présentant des caractéristiques morphologiques identiques, puis effectua des analyses mesurant la quantité de substances hormonales virilisantes (dites androgènes) par litre de sang. Les résultats sont les suivants : 3.22 3.07 3.17 2.91 3.40 3.58 3.23 3.11 3.62.

11.1- En sachant que chez les femmes non dopées la quantité moyenne d'androgène vaut 3.1, proposez un test afin de montrer que les athlètes est-allemandes sont dopées (on suppose la normalité des données).

11.2- Quelle était la conclusion du Dr Fischbach ?

• L'alcool au volant

Pour étudier l'effet de l'alcool sur les réflexes, on fait passer à quatorze sujets un test de dextérité avant et après qu'ils ont consommé 100 ml de vin. Les scores avant et après sont donnés dans le tableau suivant (ce sont les temps de réaction ; donc, un score élevé signifie un ralentissement dans les réflexes) :

Sujet	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Avant	57	54	62	64	71	65	70	75	68	70	77	74	80	83
Après	55	60	68	69	70	73	74	74	75	76	76	78	81	90

Proposez un test afin de montrer que l'alcool a un effet sur les réflexes (on suppose la normalité des données).

• Vitesse de la lumière

En 1879, le physicien américain Michelson a fait plusieurs expériences pour vérifier la valeur de la vitesse de la lumière c proposée par le physicien français Cornu en 1876. La valeur proposée par Cornu était 299 990 km/s. Michelson a obtenu les vingt mesures suivantes pour la vitesse de la lumière (les valeurs données ci-dessous sont les valeurs mesurées par Michelson auxquelles on a soustrait 299 000 afin de ne pas avoir à manipuler des chiffres trop grands) :

850 ; 740 ; 900 ; 1 070 ; 930 ; 850 ; 950 ; 980 ; 980 ; 880 ; 1 000 ; 980 ; 930 ; 1 050 ; 960 ; 810 ; 1 000 ; 1 000 ; 960 ; 960.

Ces vingt observations peuvent être considérées comme les valeurs observées de vingt variables aléatoires ayant une espérance commune mais inconnue μ. Si les conditions expérimentales pour mesurer la vitesse de la lumière sont satisfaisantes, il est alors raisonnable de supposer que μ est la vraie vitesse de la lumière.

11.1- Faites une représentation graphique de ces données. Commentez.

11.2- Testez la normalité de ces données.

11.3- Faites un test de Student pour vérifier si les mesures de Michelson invalident la valeur c de la vitesse de la lumière proposée par Cornu.

11.4- Quelle vitesse de la lumière pourrait proposer Michelson à la suite de ces vingt expériences ?

• Taux de cholestérol

On répartit dix-sept personnes ayant une certaine maladie M par randomisation en deux groupes. Le premier groupe reçoit un placebo A et le second groupe un traitement B contenant une substance vitaminique supposée active. Le critère de jugement est une mesure de la résistance capillaire. On obtient les résultats suivants :
Groupe A : 46.3 ; 42.5 ; 43.0 ; 43.9 ; 42.0 ; 41.5 ; 41.6 ; 44.4 ; 40.7
Groupe B : 47.1 ; 44.5 ; 45.8 ; 49.0 ; 44.6 ; 43.7 ; 44.5 ; 47.4

Que pouvez-vous conclure, en utilisant l'hypothèse que la variable « mesure de la résistance capillaire » suit une loi normale ?

• Indépendance traitement-décès

Deux petits groupes d'animaux sont infectés par un germe virulent. On soumet le premier groupe à une chimiothérapie, l'autre n'est pas traité. On enregistre la mortalité au huitième jour dans les deux groupes.

	Traités	Non traités	Total
Morts	0	9	9
Survivants	8	3	11
Total	8	12	20

La mortalité est-elle indépendante du traitement ?

• Effectifs de cas d'urgence

Afin d'étudier la variation des cas d'urgence dans un hôpital, on a prélevé des dossiers pour les mois de juin, juillet et août et on a compté le nombre de cas d'urgence. Le tableau suivant présente les résultats :

	Juin	Juillet	Août
Nombre de dossiers	1 500	1 600	1 450
Nombre de cas d'urgence	675	720	610

Peut-on conclure que la proportion de cas d'urgence est identique pour chaque mois ?

Chapitre 12

Régression linéaire simple et multiple

Objectif

Ce chapitre a été conçu comme une brève introduction à la méthode de régression linéaire simple et multiple et à son utilisation dans un contexte de données réelles (voir [16] pour une exposition plus complète). Les différentes commandes à taper sous **R** sont présentées et un jeu de données réelles sert de fil conducteur pour l'exposition des concepts clefs de cette méthode. Le cas des variables explicatives qualitatives est traité ainsi que l'interaction entre variables explicatives. Nous discutons de l'étape de validation du modèle par une étude des résidus, et évoquons le problème de colinéarité. Nous présentons également quelques méthodes de sélection de variables.

SECTION 12.1

Introduction

Dans la plupart des situations, nous sommes amenés à étudier la relation entre une variable d'intérêt Y (souvent quantitative) et une ou plusieurs variable(s) X_1, X_2, ..., X_k, avec pour objectif d'expliquer les variations de la variable d'intérêt. La variable Y est appelée variable **« à expliquer »** (ou parfois variable dépendante), et les variables X_1, X_2, ..., X_k sont dites **« explicatives »** et représentent, en épidémiologie, les facteurs de risque ou de confusion. L'utilisation des méthodes d'analyse multivariée, et plus particulièrement des modèles de régression linéaire, permet donc :

- de prendre en compte simultanément plusieurs facteurs pouvant expliquer la variation ou la distribution de la variable Y ;
- d'étudier le rôle de modification d'effet ou de confusion d'un ou de plusieurs facteur(s) ;
- de prédire les valeurs ou la distribution de la variable à expliquer connaissant les valeurs des variables explicatives.

Pour introduire les concepts clefs de la méthode de régression linéaire, nous allons commencer par présenter dans la première partie le modèle de régression linéaire simple en considérant une variable à expliquer quantitative Y et une seule variable explicative X quantitative, même si la variable X peut en principe être aussi qualitative. Dans la deuxième partie, nous présenterons le modèle de régression linéaire multiple permettant d'étudier la relation entre une variable dépendante quantitative Y et plusieurs variables explicatives X_1, X_2, ..., X_k qui peuvent être quantitatives ou qualitatives.

Présentation de l'exemple fil rouge : enquête « poids de naissance » :

Nous allons reprendre le jeu de données POIDS.NAISSANCE déjà présenté au chapitre B. Il s'agit ici d'expliquer la variabilité du poids de naissance de l'enfant en fonction des caractéristiques de la mère, de ses antécédents et de son comportement pendant la grossesse. La variable à expliquer est le poids de naissance de l'enfant (variable quantitative `BWT`, exprimée en grammes) et les facteurs étudiés (variables explicatives) sont décrits dans le chapitre B.

▶ Lecture des données

```
> fichier <-
+  "http://www.biostatisticien.eu/springeR/Poids_naissance.csv"
> mesdonnees <- read.table(fichier,header=TRUE,sep="\t")
```

Le poids de la mère étant exprimé en livres, nous commençons par effectuer une transformation du *data.frame* des données pour recoder cette variable en kilogrammes (1 livre = 0.453 592 37 kg).

```
> mesdonnees <- transform(mesdonnees,LWT=LWT*0.4535923)
> attach(mesdonnees)  # Accès au nom des variables.
```

SECTION 12.2

La régression linéaire simple

12.2.1 Objectif et modèle

▸ Objectif

On cherche à « expliquer » les variations d'une variable quantitative Y (par exemple, le poids de naissance de l'enfant, noté `BWT`) par une variable explicative X également quantitative (par exemple, le poids noté `LWT`).

▸ Le modèle

Il s'écrit sous la forme

$$Y = \beta_0 + \beta_1 X + \epsilon$$

où ϵ représente le bruit du modèle supposé gaussien d'espérance nulle et de variance $\mathbb{V}\text{ar}(\epsilon|X) = \sigma^2$. Les paramètres (inconnus) du modèle de régression sont β_0, β_1 et σ^2.

Attention

L'hypothèse gaussienne du bruit ϵ permet d'obtenir la loi des estimateurs et ainsi d'effectuer des tests d'hypothèses sur les paramètres du modèle. Cependant, cette hypothèse n'est pas très importante puisque l'on peut s'en passer quand le nombre de données est important.

12.2.2 Ajustement sur des données

▸ Inspection graphique

Afin d'étudier la relation entre le poids de naissance de l'enfant et l'âge de la mère, nous pouvons commencer par tracer le nuage des points (poids de l'enfant ; poids de la mère) grâce à l'instruction `plot(BWT~LWT)`.

```
> plot(BWT~LWT,xlab="Poids de la mère",
+      ylab="Poids de naissance de l'enfant")
```

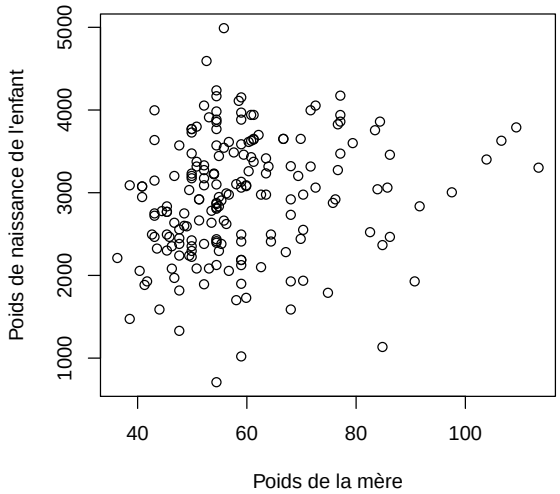


FIGURE 12.1 – Nuage de points du poids de l'enfant (en grammes) *versus* le poids de la mère (en kilogrammes).

Nous observons une légère tendance d'augmentation du poids de l'enfant avec le poids de la mère, même si cette relation n'est pas très nette.

▸ Estimation des paramètres

Nous pouvons maintenant étudier le modèle suivant :

$$\texttt{BWT}_i = \beta_0 + \beta_1 \texttt{LWT}_i + \epsilon_i, \quad i = 1, \dots n$$

où les ϵ_i sont des variables aléatoires indépendantes d'espérance nulle et de **variance constante** σ^2, quel que soit i.

Notons respectivement $\texttt{bwt}_i$ et $\texttt{lwt}_i$ les valeurs observées des variables aléatoires `BWT` et `LWT`. Les estimations $\hat{\beta}_0$ de β_0 et $\hat{\beta}_1$ de β_1 sont obtenues en minimisant le critère des moindres carrés :

$$S(\alpha_0, \alpha_1) = \sum_{i=1}^{n} (\texttt{bwt}_i - \alpha_0 - \alpha_1 \texttt{lwt}_i)^2.$$

La fonction R permettant de réaliser cette opération est la fonction `lm()` (abrégé de *linear models*). Le paramètre principal de cette fonction est une formule, symbolisée par un tilde ~, qui permet de préciser le sens de la relation entre `BWT` et `LWT`.

```
> modele1 <- lm(BWT ~ LWT,data=mesdonnees) # On obtient un ob-
                                           # jet de classe "lm".

> modele1
Call:
lm(formula = BWT ~ LWT, data = mesdonnees)
Coefficients:
(Intercept)          LWT
   2369.672        9.765
```

La sortie **R** ci-dessus fournit les estimations par moindres carrés de β_0 et de β_1. On trouve dans l'exemple ci-dessus $\hat{\beta}_0 = 2\ 369.672$ et $\hat{\beta}_1 = 9.765$.

Nous pouvons maintenant représenter la droite de régression sur le nuage de points au moyen de la fonction `abline()` :

```
> plot(BWT~LWT,xlab="Poids de la mère",ylab="Poids de l'enfant")
> abline(modele1,col="blue")
```

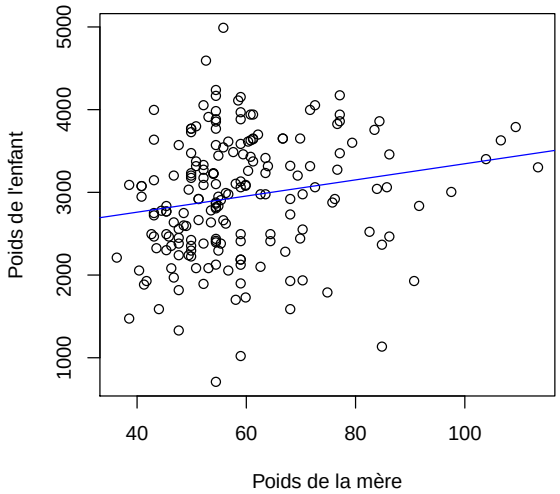


FIGURE 12.2 – Représentation de la droite de régression des moindres carrés sur le nuage de points du poids de l'enfant (en grammes) *versus* le poids de la mère (en kilogrammes).

▸ Tests sur les paramètres

Il est bon de noter que la fonction `lm()` permet une analyse complète du modèle linéaire et que vous pouvez récupérer un résumé des calculs liés au jeu de données en utilisant la fonction `summary()`.

```
> res <- summary(modele1)  # Présentation des résultats.
> res
Call:
lm(formula = BWT ~ LWT, data = mesdonnees)
Residuals:
     Min       1Q   Median       3Q      Max
-2192.18  -503.63    -3.91   508.25  2075.53
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2369.672    228.431  10.374   <2e-16 ***
LWT            9.765      3.777   2.586   0.0105 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 718.2 on 187 degrees of freedom
Multiple R-squared:  0.03452,        Adjusted R-squared:  0.02935
F-statistic: 6.686 on 1 and 187 DF,  p-value: 0.01048
```

Voici la description des différentes informations contenues dans la sortie ci-dessus.

- `Call:` un rappel de la formule utilisée dans le modèle.
- `Residuals:` une analyse descriptive des résidus $\hat{\epsilon}_i = \hat{y}_i - y_i$. Nous verrons par la suite l'intérêt des résidus pour valider les hypothèses du modèle de régression.
- `Coefficients:` ce tableau comprend quatre colonnes :
  - `Estimate` correspond aux estimations des paramètres de la droite de régression ;
  - `Std. Error` correspond à l'estimation de l'écart type des estimateurs de la droite de régression ;
  - `t value` correspond à la réalisation de la statistique du test de Student associé aux hypothèses $\mathcal{H}_0 : \beta_i = 0$ et $\mathcal{H}_1 : \beta_i \neq 0$;
  - `Pr(>|t|)` correspond à la valeur-p du test de Student.
- `Signif. codes:` symboles de niveau de significativité.
- `Residual standard error:` une estimation de l'écart type du bruit σ est fournie ainsi que le degré de liberté associé $n-2$.
- `Multiple R-squared:` valeur du coefficient de détermination r^2 (pourcentage de variance expliqué par la régression).
- `Adjusted R-squared:` r_a^2 ajusté (qui n'a pas grand intérêt en régression linéaire simple).
- `F-statistic:` correspond à la réalisation du test de Fisher associé aux hypothèses $\mathcal{H}_0 : \beta_1 = 0$ et $\mathcal{H}_1 : \beta_1 \neq 0$. Nous y trouvons les degrés de liberté associés (1 et $n-2$) ainsi que la valeur-p.

Remarque

Notez qu'il est possible de récupérer toutes les valeurs contenues dans les différents champs ci-dessus. Par exemple, pour obtenir les quatre colonnes du champs `Coefficients` ci-dessus, on utilise

```
> res$coefficients
               Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 2369.672063 228.430647 10.373705 3.328580e-20
LWT            9.764856   3.776573  2.585639 1.048072e-02
```

Les autres suffixes à utiliser sont donnés par

```
> names(res)
 [1] "call"          "terms"         "residuals"
 [4] "coefficients"  "aliased"       "sigma"
 [7] "df"            "r.squared"     "adj.r.squared"
[10] "fstatistic"    "cov.unscaled"
```

Notez également l'existence de la fonction `coefficients()` qui permet d'obtenir $\hat{\beta}_0$ et $\hat{\beta}_1$ directement à partir de `modele1`.

```
> coefficients(modele1)
(Intercept)         LWT
2369.672063    9.764856
```

▸ Tableau d'analyse de la variance

En régression linéaire simple, le test de Fisher (dont la statistique se lit dans `F-statistic`) est équivalent au test de Student associé à la pente de la régression. Nous avons la relation suivante `F-statistic`$=t^2$ et les valeurs-p des tests sont égales. Le test de Fisher est souvent associé à une table d'analyse de la variance que vous obtenez en utilisant la fonction `anova()`.

```
> anova(modele1)
Analysis of Variance Table
Response: BWT
           Df   Sum Sq Mean Sq F value  Pr(>F)
LWT         1  3448881 3448881  6.6855 0.01048 *
Residuals 187 96468171  515873
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

▸ Interprétation des résultats sur l'étude « Poids de naissance »

- Le test associé à l'*intercept* β_0 du modèle est significatif (valeur-$p < 0.05$), il est donc conseillé de garder l'*intercept* (β_0) dans le modèle. Toutefois, l'*intercept* de cette régression n'a aucun sens. Il pourrait donc être plus

judicieux de considérer une régression sur la variable poids de la mère, préalablement centrée. Dans ce cas-ci, β_0 représenterait le poids moyen des enfants pour des mères ayant un poids égal à la moyenne des poids des mères observées.

Astuce

L'instruction pour effectuer un modèle de régression sans *intercept* est `lm(y~x-1)` ou de façon équivalente `lm(y~0+x)`.

- La relation linéaire entre `BWT` et `LWT` est démontrée par le résultat du test de Student sur le coefficient β_1. La valeur-$p < 0.05$ nous indique une relation linéaire significative entre le poids de l'enfant et le poids de la mère.
- Le pourcentage de variance expliqué par la régression (r^2) vaut 0.035. Ce qui veut dire que seulement 3.5 % de la variabilité du poids de l'enfant est expliquée par le poids de la mère. Il sera donc utile d'introduire dans le modèle d'autres variables explicatives (régression linéaire multiple) afin d'améliorer le pouvoir prédictif du modèle.
- L'estimation de la pente indique que la différence entre les poids moyens de naissance des bébés de mères ayant un écart de poids d'un kilogramme est de 9.765 g.

Astuce

Pour obtenir une estimation par intervalle de confiance des coefficients de la régression, vous pouvez utiliser la fonction `confint()`.

```
> confint(modele1)
                2.5 %     97.5 %
(Intercept) 1919.039836 2820.30429
LWT            2.314692   17.21502
```

L'intervalle de confiance de niveau 95 % pour β_1 est donc $ic_{95\ \%}(\beta_1) = [2.31, 17.22]$.

12.2.3 Intervalle de confiance et de prédiction pour une nouvelle valeur

▸ Définition

Considérons une nouvelle observation x_0 de la variable X pour laquelle nous n'avons pas observé la valeur y_0 correspondante de la variable à expliquer Y. Cette valeur y_0 inconnue, car non observée, est une réalisation de la variable aléatoire $Y_0 = \beta_0 + \beta_1 X_0 + \epsilon_0$. Le **préviseur** (ou prédicteur) de Y_0 pour la

nouvelle valeur x_0 est donné par :

$$\hat{Y}_0^p = \hat{\beta}_0 + \hat{\beta}_1 x_0.$$

Nous pouvons aussi proposer un **intervalle de prévision** (ou de prédiction) de niveau $1-\alpha$ pour Y_0, qui consiste à trouver deux bornes aléatoires qui encadreront la variable aléatoire Y_0 avec une probabilité égale à $1-\alpha$:

$$IP_{1-\alpha}(Y_0|x_0) = \left[\hat{Y}_0^p \pm t_{1-\alpha/2}^{(n-2)} \hat{\sigma} \sqrt{1 + \frac{1}{n} + \frac{(x_0 - \overline{x})^2}{\sum_{i=1}^n (x_i - \overline{x})^2}}\right].$$

Notez que la valeur réalisée $\hat{y}_0^p = \hat{\beta}_0 + \hat{\beta}_1 x_0$ est appelée la **prévision** (ou prédiction) de la valeur non observée $y_0 = \beta_0 + \beta_1 x_0 + \epsilon_0$.

Dans le même ordre d'idées, notez qu'un estimateur de la valeur fixe et inconnue $\mathbb{E}(Y_0|X = x_0) = \beta_0 + \beta_1 x_0$ est donné par :

$$\hat{\mathbb{E}}(Y_0|X = x_0) := \hat{Y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0.$$

Nous pouvons également proposer un intervalle de confiance de niveau $1-\alpha$ pour $\mathbb{E}(Y_0|X = x_0)$:

$$IC_{1-\alpha}(\beta_0 + \beta_1 x_0) = \left[\hat{Y}_0 \pm t_{1-\alpha/2}^{(n-2)} \hat{\sigma} \sqrt{\frac{1}{n} + \frac{(x_0 - \overline{x})^2}{\sum_{i=1}^n (x_i - \overline{x})^2}}\right].$$

Attention

Notez que l'on a $\hat{Y}_0^p = \hat{Y}_0$, mais qu'il ne faut pas confondre l'intervalle de prédiction de Y_0 avec l'intervalle de confiance de la valeur moyenne $\mathbb{E}(Y_0|X = x_0) = \beta_0 + \beta_1 x_0$.

▸ Instruction R

La fonction permettant de définir l'intervalle de prévision et l'intervalle de confiance pour une nouvelle valeur x_0 est `predict()`.

▸ Exemple sur l'étude « Poids de naissance »

Calculons la prédiction pour le poids d'un bébé dont la mère a un poids de `lwt` = 56 kg.

```
> lwt0 <- 56
> predict(modele1,data.frame(LWT=lwt0),interval="prediction")
       fit      lwr      upr
1 2916.504 1495.699 4337.309
```

Pour l'intervalle de confiance de la valeur moyenne du poids des bébés pour un poids de la mère de 56 kg :

```
> predict(modele1,data.frame(LWT=lwt0),interval="confidence")
       fit      lwr      upr
1 2916.504 2811.225 3021.783
```

Représentons maintenant l'intervalle de confiance et l'intervalle de prévision pour une série de nouvelles valeurs de poids de la mère :

```
> x <- seq(min(LWT),max(BWT),length=50)
> intpred <- predict(modele1,data.frame(LWT=x),interval=
+                            "prediction")[,c("lwr","upr")]
> intconf <- predict(modele1,data.frame(LWT=x),interval=
+                            "confidence")[,c("lwr","upr")]
> plot(BWT~LWT,xlab="Poids de la mère",ylab="Poids de l'enfant")
> abline(modele1)
> matlines(x,cbind(intconf,intpred),lty=c(2,2,3,3),
+              col=c("red","red","blue","blue"),lwd=c(2,2,1,1))
> legend("bottomright",lty=c(2,3),lwd=c(2,1),
+              c("confiance","prévision"),col=c("red","blue"))
```

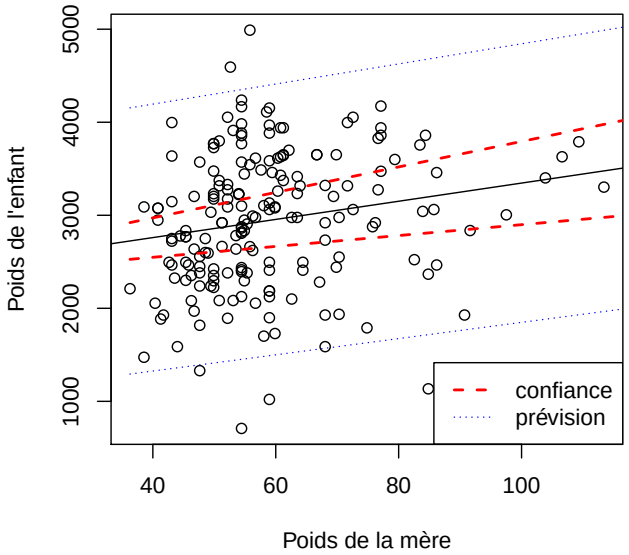


FIGURE 12.3 – Visualisation de l'intervalle de confiance et de l'intervalle de prévision.

12.2.4 Analyse des résidus

▸ Vérification des hypothèses du modèle

L'analyse des résidus consiste à examiner si les hypothèses de base du modèle linéaire sont violées. Différents graphiques tracés à l'aide des résidus permettent de détecter assez facilement si les hypothèses sur les erreurs ϵ_i ne sont pas respectées :

- Tracé de l'histogramme des résidus pour détecter la non-normalité. Le graphique QQ-plot est une autre approche. On peut également appliquer le test de non-normalité de Jarque et Bera sur les résidus (fonction `jarque.bera.test()` disponible dans le *package* `tseries`).

- Tracé des résidus $\hat{\epsilon}_i$ en fonction des valeurs prédites $\hat{y}_i$. En effet, lorsque les hypothèses associées au modèle sont correctes, les résidus et les valeurs prédites sont non corrélées. Par conséquent, le tracé de ces points ne devrait pas avoir de structure particulière. Ce type de tracé donne aussi des indications sur la validité des hypothèses de linéarité, ainsi que sur l'homogénéité de la variance de l'erreur. On devrait observer sur le graphique de $\hat{\epsilon}_i$ *versus* $\hat{y}_i$ une répartition uniforme des résidus suivant une bande horizontale de part et d'autre de l'axe des abscisses.

- Pour l'hypothèse d'indépendance des Y_i, si les valeurs de Y_i sont mesurées chez des individus différents sans aucune relation entre eux, l'hypothèse d'indépendance est en principe valide. En revanche, si les valeurs de Y_i représentent des mesures d'une certaine quantité au cours du temps (par exemple chaque mois), l'hypothèse d'indépendance peut ne pas être valide. On peut alors vérifier l'hypothèse d'indépendance en examinant l'autocorrélation des résidus soit graphiquement en portant sur un graphique les résidus successifs $\hat{\epsilon}_i$, soit en utilisant des tests statistiques comme, par exemple, le test de Durbin-Watson (fonction `dwtest()` dans le *package* `lmtest`).

▸ Exemple d'analyse des résidus dans l'étude « Poids de naissance »

Nous présentons succinctement une analyse des résidus pour le modèle étudié, même si cela n'est pas très pertinent ici compte tenu du faible pouvoir prédictif de ce modèle ($r^2 = 3.5$ %).

Tout d'abord, examinons l'hypothèse de normalité :

```
> par(mfrow=c(1,2))
> hist(residuals(modele1), main="Histogramme")
> qqnorm(resid(modele1),datax=TRUE) # Attention: quantiles
                                    # normalisés en ordonnée.
```

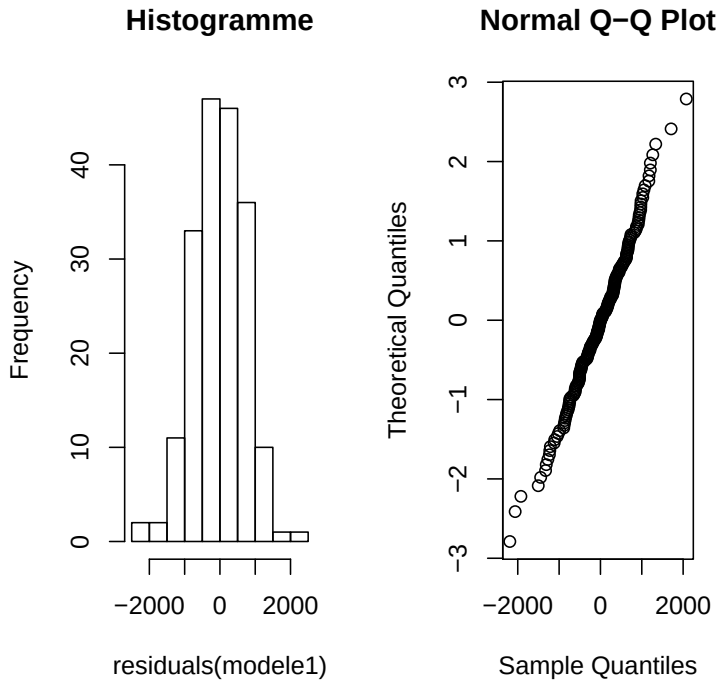


FIGURE 12.4 – Inspection graphique de la normalité des résidus.

```
> require("tseries")
> jarque.bera.test(residuals(modele1)) # Test de non-normalité de
                                       # Jareque et Bera.
        Jarque Bera Test
data:  residuals(modele1)
X-squared = 1.7877, df = 2, p-value = 0.4091
```

Le test ne permet pas de conclure à la non-normalité des erreurs. Par ailleurs, le QQ-plot suggère des erreurs normales puisque les quantiles observés et les quantiles théoriques (obtenus si la distribution est normale) forment une droite. L'hypothèse de normalité ne sera donc pas remise en question.

Examinons maintenant le graphe des résidus en fonction des valeurs prédites. La figure ci-contre montre un nuage de résidus correctement répartis et symétrique autour de l'axe des abscisses, les conditions du modèle ne semblent donc pas invalidées.

```
> plot(residuals(modele1)~fitted(modele1),
+      xlab="Valeurs prédites",ylab="Résidus")
```

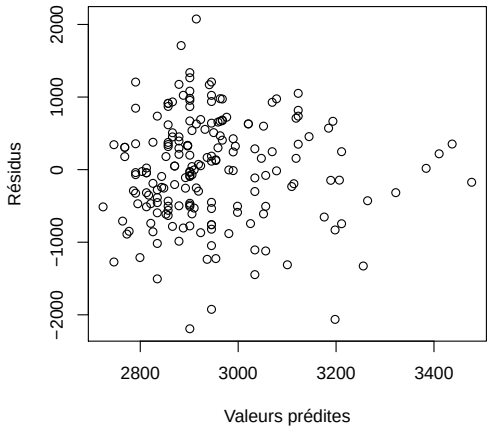
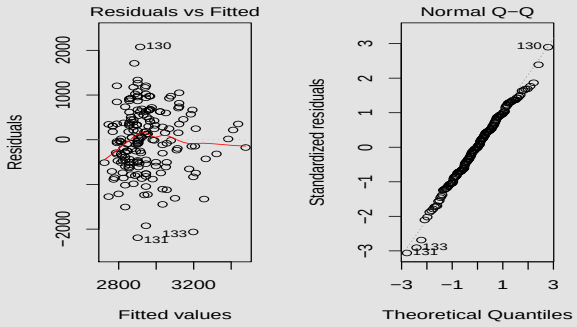


FIGURE 12.5 – Graphe des résidus en fonction des valeurs prédites.

Remarque

Signalons que vous pouvez obtenir certains graphiques diagnostiques du modèle en tapant les instructions suivantes :

```
> par(mfrow=c(1,2))
> plot(modele1,1:2,col.smooth="red")
```



Notons que l'instruction `plot(modele1)` peut tracer six graphiques, dont certains traitent de la détection de valeurs atypiques. Nous y reviendrons en détail dans la section « La régression linéaire multiple ».

12.2.5 Tests de Student pour des moyennes et modèle linéaire

Il est intéressant de remarquer que les tests de Student pour une ou deux moyennes, à un échantillon ou à deux échantillons indépendants ou appariés, sont des cas particuliers du modèle de régression linéaire simple.

- **Test de Student à un échantillon**

  Soit $\mathrm{Y}_1, \ldots, \mathrm{Y}_n$ *i.i.d.* de loi $\mathcal{N}(\mu, \sigma^2)$. On veut tester

  $$\mathcal{H}_0 : \mu = \mu_0 \text{ } \textit{versus} \text{ } \mathcal{H}_1 : \mu \neq \mu_0,$$

  pour une certaine valeur de référence μ_0 donnée. Cela peut se faire par un test de $\beta = 0$ dans le modèle

  $$\mathrm{Y}_i - \mu_0 = \beta + \epsilon_i,$$

  où les ϵ_i sont *i.i.d.* de loi $\mathcal{N}(0, \sigma^2)$, $1 \leq i \leq n$. Voyons ceci sur un exemple.

```
> n <- 100
> y <- rnorm(n,mean=0)
> mu0 <- 0.1
> t.test(y,mu=mu0)[[3]]
[1] 0.4178768
> summary(lm(y-mu0 ~ 1))[[4]][1,4]
[1] 0.4178768
```

- **Test de Student à deux échantillons indépendants**

  Soit $\mathrm{X}_1^{(1)}, \ldots, \mathrm{X}_{n_1}^{(1)}$ *i.i.d.* de loi $\mathcal{N}(\mu_1, \sigma^2)$, et $\mathrm{X}_1^{(2)}, \ldots, \mathrm{X}_{n_2}^{(2)}$ *i.i.d.* de loi $\mathcal{N}(\mu_2, \sigma^2)$, deux échantillons indépendants. On veut tester

  $$\mathcal{H}_0 : \mu_1 = \mu_2 \text{ versus } \mathcal{H}_1 : \mu_1 \neq \mu_2.$$

  Cela peut se faire en testant $\beta_1 = 0$ dans le modèle

  $$\mathrm{Y}_i = \mu_1 + \beta_1 \mathrm{A}_i + \epsilon_i,$$

  où les ϵ_i sont *i.i.d.* de loi $\mathcal{N}(0, \sigma^2)$, et où l'on pose

  $$\mathrm{A}_i = \begin{cases} 0 & \text{si } 1 \leq i \leq n_1, \\ 1 & \text{si } n_1 + 1 \leq i \leq n_1 + n_2, \end{cases}$$

  et

  $$\mathrm{Y}_i = \begin{cases} \mathrm{X}_i^{(1)} & \text{si } 1 \leq i \leq n_1, \\ \mathrm{X}_{i-n_1}^{(2)} & \text{si } n_1 + 1 \leq i \leq n_1 + n_2. \end{cases}$$

  Voyons ceci sur un exemple.

```
> n1 <- 100 ; n2 <- 150
> x1 <- rnorm(n1,mean=0) ; x2 <- rnorm(n2,mean=0.1)
> y <- c(x1,x2) ; a <- c(rep(0,n1),rep(1,n2))
> t.test(x1,x2,var.equal=TRUE)[[3]]
[1] 0.3996454
> summary(lm(y ~ a))[[4]][2,4]
[1] 0.3996454
```

- **Test de Student à deux échantillons appariés**
  Soit $X_1^{(1)}, \ldots, X_n^{(1)}$ *i.i.d.* de loi $\mathcal{N}(\mu_1, \sigma^2)$, et $X_1^{(2)}, \ldots, X_n^{(2)}$ *i.i.d.* de loi $\mathcal{N}(\mu_2, \sigma^2)$, deux échantillons appariés (donc dépendants). On veut tester

$$\mathcal{H}_0 : \mu_1 = \mu_2 \text{ versus } \mathcal{H}_1 : \mu_1 \neq \mu_2.$$

  Cela peut se faire en testant $\beta = 0$ dans le modèle

$$Y_i = \beta + \epsilon_i,$$

  où les ϵ_i sont *i.i.d.* de loi $\mathcal{N}(0, \sigma^2)$, et où $Y_i = X_i^{(1)} - X_i^{(2)}$, $1 \leq i \leq n$. Voyons ceci sur un exemple.

```
> n <- 100
> x1 <- rnorm(n,mean=0) ; x2 <- rnorm(n,mean=0.1)
> t.test(x1,x2,var.equal=TRUE,paired=TRUE)[[3]]
[1] 0.4541976
> summary(lm(x1-x2 ~ 1))[[4]][1,4]
[1] 0.4541976
```

12.2.6 Récapitulatif

Le tableau ci-dessous présente les principales fonctions à utiliser afin d'effectuer une régression linéaire simple entre la variable à expliquer Y et la variable explicative X.

TABLE 12.1 – Liste des principales fonctions R permettant l'analyse d'une régression linéaire simple.

Instruction R	Description
`plot(Y~X)`	graphe du nuage de points
`lm(Y~X)`	estimation du modèle linéaire
`summary(lm(Y~X))`	description des résultats du modèle
`abline(lm(Y~X))`	trace la droite estimée
`confint(lm(Y~X))`	intervalle de confiance des paramètres de régression
`predict()`	fonction permettant d'obtenir des prédictions
`plot(lm(Y~X))`	analyse graphique des résidus

SECTION 12.3

La régression linéaire multiple

12.3.1 Objectif et modèle

▸ Objectif

Il s'agit d'étudier les variations d'une variable quantitative Y (variable dite à expliquer ou dépendante, supposée aléatoire) en fonction de p ($p > 1$) variables explicatives X_1, X_2,...,X_p (variables aussi dites indépendantes). Les variables explicatives peuvent être uniquement quantitatives, uniquement qualitatives (auquel cas on retombe sur l'ANOVA présentée au chapitre 13), ou un mélange de variables quantitatives et qualitatives. Dans ce dernier cas, le modèle de régression linéaire multiple est aussi appelé ANCOVA.

▸ Le modèle

Le modèle de régression linéaire multiple s'écrit

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

où ϵ représente le terme de perturbation aléatoire (bruit) du modèle souvent supposé gaussien d'espérance nulle et de variance σ^2, et indépendant des X_j. Les paramètres (inconnus) du modèle de régression sont $\beta_0, \beta_1 \ \ldots, \ \beta_p$ et σ^2.

12.3.2 Ajustement sur des données

Nous considérons un jeu de données provenant du modèle suivant :

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} + \epsilon_i \quad i = 1, \ldots, n$$

où X_{ij} est la j-ième variable explicative pour l'individu i et les erreurs ϵ_i sont des variables aléatoires indépendantes vérifiant $\mathbb{E}(\epsilon_i) = 0$ et $\mathbb{V}\text{ar}(\epsilon_i | \mathbf{X}_i) = \sigma^2$ avec $\mathbf{X}_i = (X_{i1}, \ldots, X_{ip})$. Les données issues de ce modèle peuvent s'écrire sous forme matricielle :

$$\boldsymbol{y} = \boldsymbol{\mathcal{X}}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

$$\text{avec } \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \boldsymbol{\mathcal{X}} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix} \text{ et } \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}.$$

Les paramètres de régression $\boldsymbol{\beta}$ sont estimés en minimisant (en $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_p)$) les moindres carrés ordinaires :

$$S(\boldsymbol{\alpha}) = \sum_{i=1}^{n} (y_i - \alpha_0 - \alpha_1 x_{i1} - \alpha_2 x_{i2} - \cdots - \alpha_p x_{ip})^2 = \|\boldsymbol{y} - \boldsymbol{\mathcal{X}}\boldsymbol{\alpha}\|^2.$$

Cela donne l'estimateur des moindres carrés $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\mathsf{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathsf{T}\boldsymbol{y}$.

Attention

On suppose que $\boldsymbol{X} : n \times (p+1)$ est une matrice de plein rang ($\text{rang}(\boldsymbol{X}) = p+1 < n$). Cela entraîne alors que $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ est inversible. Cette condition peut être jaugée à l'aide de R avec l'instruction `qr(X)$rank`. Si cette condition n'est pas satisfaite, il est possible d'utiliser d'autres méthodes non présentées dans cet ouvrage comme la régression ridge, la régression Lasso, la régression sur composantes principales ou encore la régression PLS ; [16].

▸ Inspection graphique

Exemple fil rouge : Étude de la régression du poids de naissance de l'enfant en fonction de l'âge de la mère, de son poids et de son statut tabagique durant la grossesse.

Pour répondre à l'objectif de cette étude, on note X_1 l'âge de la mère (variable `AGE`), X_2 le poids de la mère (variable `LWT`), X_3 le statut tabagique de la mère (variable `SMOKE`), et Y le poids de naissance de l'enfant (variable à expliquer `BWT`) ; nous pouvons écrire l'équation de régression suivante :

$$\mathbb{E}(\texttt{BWT}|\texttt{AGE}, \texttt{LWT}, \texttt{SMOKE}) = \beta_0 + \beta_1 \texttt{AGE} + \beta_2 \texttt{LWT} + \beta_3 \texttt{SMOKE}.$$

Avant d'estimer le modèle, nous présentons un diagramme de dispersion de toutes les paires de ces variables.

```
> add.cor <- function(x,y) {
+       usr <- par("usr"); on.exit(par(usr))
+       par(usr = c(0, 1, 0, 1))
+       text(0.5,0.5,round(cor(x,y),2))
+ }
```

```
> newdata <- cbind(BWT,LWT,AGE,SMOKE)
> # Diagramme de dispersion.
> pairs(newdata,lower.panel=panel.smooth,upper.panel=add.cor)
```

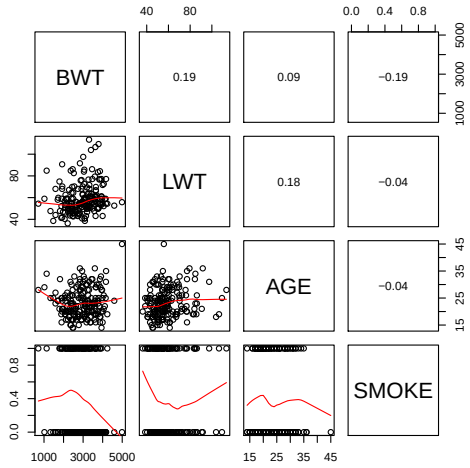


FIGURE 12.6 – Diagramme de dispersion de toutes les paires de variables.

L'objectif de ce diagramme est double. Il permet de visualiser la relation entre la variable à expliquer et chacune des variables explicatives, mais aussi de juger de la corrélation entre les variables explicatives. Le dernier point permet entre autres choses de diagnostiquer un problème potentiel de colinéarité (voir la section spécifique sur ce sujet en 12.3.7).

▸ Estimation des paramètres

Comme en régression linéaire simple, nous estimons le modèle par la fonction `lm()` :

```
> modele2 <- lm(BWT~AGE+LWT+as.factor(SMOKE))
> modele2
Call:
lm(formula = BWT ~ AGE + LWT + as.factor(SMOKE))
Coefficients:
      (Intercept)                AGE                LWT
         2362.720              7.093              8.860
as.factor(SMOKE)1
         -267.213
```

▸ Tests sur les paramètres

Les tests sur les divers paramètres sont obtenus grâce à la fonction `summary()`.

```
> summary(modele2)
Call:
lm(formula = BWT ~ AGE + LWT + as.factor(SMOKE))
Residuals:
     Min       1Q   Median       3Q      Max
-2069.89  -433.18    13.67   516.45  1813.75
Coefficients:
                  Estimate Std. Error t value
(Intercept)       2362.720    300.687   7.858
AGE                  7.093      9.925   0.715
LWT                  8.860      3.791   2.337
as.factor(SMOKE)1 -267.213    105.802  -2.526
                          Pr(>|t|)
(Intercept)       0.000000000000311 ***
AGE                           0.4757
LWT                           0.0205 *
as.factor(SMOKE)1             0.0124 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 708.8 on 185 degrees of freedom
Multiple R-squared:  0.06988,        Adjusted R-squared:  0.05479
F-statistic: 4.633 on 3 and 185 DF,  p-value: 0.003781
```

Les résultats fournis par `summary()` se présentent de façon identique à ceux de la régression linéaire simple. On y retrouve les estimations des paramètres de régression dans la colonne `Estimate`.

Les valeurs réalisées des statistiques des tests de Student associés aux hypothèses $\mathcal{H}_0 : \beta_i = 0$ *versus* $\mathcal{H}_1 : \beta_i \neq 0$ se trouvent dans la colonne `t value`, les valeurs-p associées dans la colonne `Pr(>|t|)`. `Residual standard error` fournit l'estimation de σ ainsi que le nombre de degrés de liberté associés $n - p - 1$. On trouve enfin le coefficient de détermination r^2 (`Multiple R-squared`) ainsi qu'une version ajustée (`Adjusted R-squared`). Enfin, on trouve la réalisation du test de Fisher global (`F-statistic`) ainsi que sa valeur-p (`p-value`) associée.

▸ Tableau d'analyse de la variance

On obtient la table d'analyse de la variance au moyen de la fonction `anova()` :

```
> anova(modele2)
Analysis of Variance Table
Response: BWT
                   Df   Sum Sq Mean Sq F value  Pr(>F)
AGE                 1   806927  806927  1.6063 0.20661
LWT                 1  2970564 2970564  5.9133 0.01598 *
as.factor(SMOKE)    1  3204339 3204339  6.3787 0.01239 *
Residuals         185 92935223  502353
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remarque

Le test F de Fisher global permet de tester l'apport global et conjoint de l'ensemble des variables explicatives présentes dans le modèle pour « expliquer » les variations de Y. L'hypothèse nulle est $\mathcal{H}_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$ (l'ensemble des p variables explicatives n'apporte pas une information utile pour la prédiction de Y, sous le modèle linéaire). L'assertion d'intérêt est $\mathcal{H}_1$: au moins l'un des coefficients β_j $(j = 1, 2, \ldots, p)$ est significativement différent de zéro (au moins une des variables explicatives est associée à Y après ajustement sur les autres variables explicatives).

▸ Interprétation des résultats de l'étude « Poids de naissance »

Au vu du résultat du test de Fisher global (valeur-$p = 0.003\,781$), nous pouvons conclure qu'au moins une des variables explicatives est associée au poids de naissance de l'enfant, ajusté sur les autres variables. Les tests individuels de Student nous indiquent que :

- le poids de la mère est linéairement associé au poids de l'enfant, ajusté sur l'âge et le statut tabagique de la mère, avec un risque d'erreur inférieur à 5 % (valeur-$p = 0.020\,5$). À âge et statut tabagique de la mère identiques, une augmentation du poids de la mère d'un kilogramme correspond à une augmentation de 8.860 g du poids moyen de naissance de l'enfant ;
- l'âge de la mère n'est pas significativement associé linéairement au poids de naissance de l'enfant lorsqu'on a déjà pris en compte le poids et le statut tabagique de la mère (valeur-$p = 0.206\,61$) ;
- le poids moyen de naissance est significativement plus faible pour les enfants nés de mères fumeuses par rapport aux enfants nés de mères non fumeuses de même âge et de même poids, avec un risque d'erreur inférieur ou égal à 5 % (valeur-p=0.012). À âge et poids de la mère identiques, le poids moyen de naissance est plus faible de 267.213 g pour une mère fumeuse par rapport à une mère non fumeuse.

Remarque

On aurait pu conclure de façon équivalente en présentant une estimation par intervalle de confiance et en regardant si la valeur zéro est contenue ou non dans cet intervalle. Si zéro n'appartient pas à l'intervalle de confiance, alors il existe un apport significatif de la variable dans le modèle ajusté sur les autres variables.

```
> confint(modele2)
                    2.5 %     97.5 %
(Intercept)      1769.504181 2955.93509
AGE               -12.486773   26.67319
```

```
LWT                  1.380732   16.34007
as.factor(SMOKE)1 -475.945996  -58.48000
```

12.3.3 Intervalle de confiance et de prédiction pour une nouvelle valeur

Supposons que nous voulions prédire le poids de naissance d'un enfant dont la mère est âgée de 23 ans, pèse 57 kg et fume. La fonction `predict()` permet d'obtenir une prédiction et son intervalle de prévision ainsi que l'intervalle de confiance du poids moyen des enfants dont les mères ont les caractéristiques décrites précédemment.

```
> newdata <- data.frame(AGE=23,LWT=57,SMOKE=1)
> predict(modele2,newdata,interval="pred")
       fit      lwr      upr
1 2763.693 1355.943 4171.444
> predict(modele2,newdata,interval="conf")
       fit      lwr      upr
1 2763.693 2600.914 2926.472
```

12.3.4 Test d'une sous-hypothèse linéaire : test de Fisher partiel

Le test de Fisher partiel permet de tester l'apport d'un sous-ensemble de variables explicatives dans un modèle qui en contient déjà d'autres. Considérons par exemple les deux modèles suivants :

- modèle 1 : $\texttt{BWT} = \beta_0 + \beta_1 \texttt{LWT} + \epsilon$;
- modèle 2 : $\texttt{BWT} = \beta_0 + \beta_1 \texttt{LWT} + \beta_2 \texttt{AGE} + \beta_3 \texttt{SMOKE} + \epsilon$.

Le test de Fisher consiste à tester l'apport simultané des variables `AGE` et `SMOKE` dans le modèle 2. Les hypothèses du test sont : $\mathcal{H}_0 : \beta_2 = \beta_3 = 0$ et $\mathcal{H}_1$: au moins un des deux coefficients β_2 ou β_3 est différent de zéro. Les commandes suivantes permettent d'effectuer ce test :

```
> anova(modele1,modele2)
Analysis of Variance Table
Model 1: BWT ~ LWT
Model 2: BWT ~ AGE + LWT + as.factor(SMOKE)
  Res.Df      RSS Df Sum of Sq      F  Pr(>F)
1    187 96468171
2    185 92935223  2   3532949 3.5164 0.03171 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

La valeur-p du test (`Pr(>F)` $= 0.03171$) nous indique qu'au moins une des deux variables `AGE` ou `SMOKE` apporte de l'information supplémentaire pour la

prédiction du poids de naissance de l'enfant lorsque l'on a déjà pris en compte le poids de la mère.

Remarque

Lorsque l'on compare deux modèles emboîtés qui diffèrent d'une seule variable, alors le test de Fisher partiel est équivalent au test individuel de Student.

```
> modele3 <- lm(BWT~LWT+SMOKE)
> anova(modele3,modele2)
Analysis of Variance Table
Model 1: BWT ~ LWT + SMOKE
Model 2: BWT ~ AGE + LWT + as.factor(SMOKE)
  Res.Df      RSS Df Sum of Sq      F Pr(>F)
1    186 93191828
2    185 92935223  1    256606 0.5108 0.4757
```

On retrouve la même valeur-p que dans le test individuel de Student associé à la variable `AGE` dans le modèle 2.

12.3.5 Cas des variables qualitatives à plus de deux modalités

Le cas des variables explicatives binaires ne pose pas de problème comme nous avons pu le voir dans l'exemple du fil rouge pour la variable `SMOKE`. L'utilisation d'une telle variable dans un modèle de régression linéaire multiple équivaut à comparer les moyennes de la variable à expliquer Y dans les deux groupes définis par la variable qualitative binaire. Cette comparaison est dite ajustée sur les autres variables explicatives, ce qui signifie que nous estimons la moyenne conditionnelle de Y, pour des valeurs fixées de toutes les variables explicatives, sauf pour `SMOKE`.

Cependant, pour une variable qualitative à plus de deux modalités, il est nécessaire d'introduire des **variables indicatrices** (aussi appelées variables fictives ou *dummy variables*) afin de comparer les moyennes de Y dans les différents groupes définis par les modalités de la variable explicative qualitative. Une variable indicatrice d'un groupe ou d'une modalité est une variable binaire qui prend la valeur 1 pour ce groupe et 0 pour les autres groupes.

Exemple fil rouge : Poids de naissance de l'enfant en fonction de la race de la mère et de son poids.

La variable `RACE` est codée avec trois modalités : 1 pour blanche, 2 pour noire et 3 pour autre. Dans cet exemple, on peut donc définir trois variables indicatrices `RACE1`, `RACE2`, `RACE3`.

Catégories	RACE	RACE1	RACE2	RACE3
Blanche	1	1	0	0
Noire	2	0	1	0
Autre	3	0	0	1

```
> RACE1 <- as.integer(RACE==1)
> RACE2 <- as.integer(RACE==2)
> RACE3 <- as.integer(RACE==3)
```

Si on essaye d'ajuster le modèle suivant

$$\texttt{BWT} = \boldsymbol{X\beta} + \boldsymbol{\epsilon} = \beta_0 \mathbf{1} + \beta_1 \texttt{LWT} + \beta_2 \texttt{RACE1} + \beta_3 \texttt{RACE2} + \beta_4 \texttt{RACE3} + \boldsymbol{\epsilon}, \qquad (12.1)$$

un problème va survenir, comme on peut le voir dans la sortie ci-dessous :

```
> summary(lm(BWT~LWT+RACE1+RACE2+RACE3))
Call:
lm(formula = BWT ~ LWT + RACE1 + RACE2 + RACE3)
Residuals:
    Min      1Q  Median      3Q     Max
-2094.9  -420.8    40.1   478.2  1928.4
Coefficients: (1 not defined because of singularities)
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2245.097    226.805   9.899  < 2e-16 ***
LWT           10.267      3.856   2.662  0.00844 **
RACE1        243.667    113.826   2.141  0.03361 *
RACE2       -209.098    168.988  -1.237  0.21752
RACE3             NA         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 702.7 on 185 degrees of freedom
Multiple R-squared:  0.08578,        Adjusted R-squared:  0.07095
F-statistic: 5.786 on 3 and 185 DF,  p-value: 0.0008399
```

Ceci vient du fait que la matrice de *design* $\boldsymbol{X}$ n'est plus de plein rang :

```
> # Premières lignes de la matrice de design.
> head(model.matrix(summary(lm(BWT~LWT+RACE1+RACE2+RACE3))))
  (Intercept)      LWT RACE1 RACE2 RACE3
1           1 82.55380     0     1     0
2           1 70.30681     0     0     1
3           1 47.62719     1     0     0
4           1 48.98797     1     0     0
5           1 48.53438     1     0     0
6           1 56.24545     0     0     1
```

En effet, la somme des 3 dernières colonnes redonne la première colonne : le modèle est dit non identifiable. L'estimateur des moindres carrés de $\boldsymbol{\beta}$ n'est alors plus unique (même si les prédictions basées sur n'importe quel estimateur des moindres carrés de $\boldsymbol{\beta}$ le seront).

Remarque

On pourrait penser à utiliser l'estimateur des moindres carrés $\hat{\boldsymbol{\beta}}^{\dagger} = \boldsymbol{\mathcal{X}}^{+}\texttt{BWT}$, basé sur l'inverse généralisée (de Moore-Penrose) $\boldsymbol{\mathcal{X}}^{+}$ de $\boldsymbol{\mathcal{X}}$, définie à la page 352 du chapitre 8.

```
> mpinv(model.matrix(summary(lm(BWT ~ LWT + RACE1 + RACE2 +
+       RACE3)))) %*% as.matrix(BWT)
             [,1]
[1,] 1692.46453
[2,]   10.26709
[3,]  796.29883
[4,]  343.53360
[5,]  552.63210
```

Mais l'ennui est qu'il est difficile d'interpréter les coefficients de ce vecteur puisque des contraintes linéaires non controlables sur ses composantes ont été utilisées (dans le sens que deux d'entres elles s'expriment en fonction de toutes les autres).

Nous pouvons constater que le modèle (12.1) peut se réécrire ainsi :

$$\begin{aligned}
\texttt{BWT} &= \beta_0 + \beta_1\texttt{LWT} + \beta_2\texttt{RACE1} + \beta_3\texttt{RACE2} + \beta_4\texttt{RACE3} + \epsilon \\
&= \beta_0 + \beta_1\texttt{LWT} + \beta_2(1 - \texttt{RACE2} - \texttt{RACE3}) + \beta_3\texttt{RACE2} + \beta_4\texttt{RACE3} + \epsilon \\
&= \underbrace{(\beta_0 + \beta_2)}_{\gamma_0} + \underbrace{\beta_1}_{\gamma_1}\texttt{LWT} + \underbrace{(\beta_3 - \beta_2)}_{\gamma_2}\texttt{RACE2} + \underbrace{(\beta_4 - \beta_2)}_{\gamma_3}\texttt{RACE3} + \epsilon.
\end{aligned}$$

Sous cette dernière forme, le modèle est maintenant identifiable (les γ_i sont estimables). Ceci se vérifie aisément sur la sortie ci-dessous :

```
> summary(lm(BWT~LWT+RACE2+RACE3))$coeff
             Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 2488.76336 241.863663 10.289943 6.348821e-20
LWT           10.26709   3.856339  2.662393 8.442125e-03
RACE2       -452.76523 157.481809 -2.875032 4.513529e-03
RACE3       -243.66673 113.825910 -2.140697 3.360769e-02
```

Cela consiste à prendre comme groupe de référence `RACE = 1` (blanche), *i.e.* à considérer le modèle de régression linéaire avec uniquement les variables indicatrices des autres groupes (`RACE2` et `RACE3`).

Pour ajuster un modèle contenant une ou des variables qualitatives explicatives, il est d'usage d'utiliser la fonction `factor()` dans l'instruction `lm()`, comme on peut le voir ci-après :

```
> modele4 <- lm(BWT~LWT+factor(RACE))
> summary(modele4)$coeff
               Estimate Std. Error   t value     Pr(>|t|)
```

```
(Intercept)    2488.76336 241.863663 10.289943 6.348821e-20
LWT              10.26709   3.856339  2.662393 8.442125e-03
factor(RACE)2 -452.76523 157.481809 -2.875032 4.513529e-03
factor(RACE)3 -243.66673 113.825910 -2.140697 3.360769e-02
```

Notez que dans cette sortie, **R** a utilisé comme groupe de référence `RACE = 1`.

Dans ce contexte, l'estimation $\hat{\gamma}_2 = \widehat{\beta_3 - \beta_2} = -452.765$ g représente la difference des poids moyens de naissance entre les mères noires (`RACE = 2`) et les mères blanches (groupe de référence), et ce résultat est significativement différent de zéro (valeur-p = 0.00451) dans un modèle ajusté sur le poids de la mère. De même, la différence des poids de naissance moyens entre le groupe `RACE = 3` et le groupe de référence est égale à $\hat{\gamma}_3 = \widehat{\beta_4 - \beta_2} = -243.667$ g et est significativement différente de zéro (valeur-p = 0.03361), après ajustement sur le poids de la mère.

Remarque

Il est possible de changer la classe de référence par le biais de la fonction `relevel()`.

```
> summary(lm(BWT~LWT+relevel(factor(RACE),ref=3)))$coeff
                                  Estimate Std. Error
(Intercept)                     2245.09663 226.805111
LWT                               10.26709   3.856339
relevel(factor(RACE), ref = 3)1  243.66673 113.825910
relevel(factor(RACE), ref = 3)2 -209.09850 168.988169
                                  t value     Pr(>|t|)
(Intercept)                      9.898792 8.296639e-19
LWT                              2.662393 8.442125e-03
relevel(factor(RACE), ref = 3)1  2.140697 3.360769e-02
relevel(factor(RACE), ref = 3)2 -1.237356 2.175232e-01
```

Remarque

Pour tester l'apport global de la variable `RACE`, il faut utiliser un test de Fisher partiel tel que présenté à la section précédente.

```
> anova(modele1,modele4)
Analysis of Variance Table
Model 1: BWT ~ LWT
Model 2: BWT ~ LWT + factor(RACE)
  Res.Df      RSS Df Sum of Sq      F   Pr(>F)
1    187 96468171
2    185 91346474  2   5121697 5.1864 0.006434 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Attention

Il ne faut jamais introduire dans le modèle l'indicatrice du groupe de référence : pour une variable qualitative à p modalités, il ne faut introduire que $p-1$ variables indicatrices.

12.3.6 Interaction entre les variables

On dit donc qu'il y a interaction entre deux variables explicatives X_1 et X_2 si l'association entre l'une des variables et la variable à expliquer Y n'est pas la même selon les valeurs de l'autre variable. Cela correspond à la notion de modification d'effet en épidémiologie.

Supposons qu'on s'intéresse à l'association entre X_1 et Y, et que l'on veuille savoir si l'effet de X_1 sur Y est différent selon les valeurs de X_2. Il suffit pour cela d'introduire dans le modèle les variables X_1 et X_2 ainsi qu'une troisième variable X_3 définie comme le produit de X_1 et X_2 ($X_3 = X_1 \times X_2$, appelée terme d'interaction entre X_1 et X_2). Pour fixer les idées, supposons que l'on veuille déterminer si l'effet d'une variable X_1 quantitative est modifié par une variable X_2 binaire (0/1). On considère alors le modèle :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 \times X_2 + \epsilon.$$

Dans le groupe $X_2 = 0$, le modèle s'écrit : $Y = \beta_0 + \beta_1 X_1 + \epsilon$ et l'effet de X_1 est mesuré par β_1. Dans le groupe $X_2 = 1$, le modèle s'écrit $Y = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) X_1 + \epsilon$ et l'effet de X_1 est mesuré par $\beta_1 + \beta_3$.

Il y a interaction entre X_1 et X_2 (ou modification de l'effet de X_1 par X_2) si l'effet de X_1 est différent dans les groupes $X_2 = 0$ et $X_2 = 1$; autrement dit si $\beta_3 \neq 0$. Il suffit donc d'effectuer un test individuel de Student sur β_3 ($\mathcal{H}_0 : \beta_3 = 0$ *versus* $\mathcal{H}_1 : \beta_3 \neq 0$). Si l'on accepte $\mathcal{H}_1$, on garde le terme d'interaction dans le modèle : il y a modification d'effet.

Exemple fil rouge : Modification de l'effet de l'âge de la mère sur le poids de naissance de l'enfant par le statut tabagique de la mère.

Considérons tout d'abord le modèle suivant :

$$\texttt{BWT} = \beta_0 + \beta_1 \texttt{AGE} + \beta_2 \texttt{SMOKE} + \epsilon.$$

```
> modele5 <- lm(BWT~AGE+SMOKE)
```

Ce modèle suppose un effet de l'`AGE` identique chez les mères fumeuses et chez les non fumeuses. On peut proposer un modèle plus souple, permettant d'avoir des effets différents dans les deux groupes `SMOKE=0` et `SMOKE=1` :

$$\texttt{BWT} = \beta_0 + \beta_1 \texttt{AGE} + \beta_2 \texttt{SMOKE} + \beta_3 \texttt{AGE} \times \texttt{SMOKE} + \epsilon$$

```
> modele6 <- lm(BWT~AGE+SMOKE+AGE:SMOKE)
```

Astuce

Pour introduire un terme d'interaction entre deux variables X_1 et X_2 dans un modèle linéaire, il suffit de taper `X1X2`. Notez que l'écriture `X1*X2` dans une formule correspond à l'écriture `X1+X2+X1X2`. Ainsi, l'appel précédent aurait pu être remplacé par :

```
> modele6 <- lm(BWT~AGE*SMOKE)
```

La différence entre ces deux modèles peut être visualisée sur les deux graphiques qui suivent. Le premier présente deux droites parallèles dont le coefficient directeur représente l'effet de l'`AGE` sur `BWT` qui est donc le même chez les fumeuses et les non fumeuses.

```
> co <- coef(modele5); a0 <- co[1] ; a1 <- co[1]+co[3]
> b <- co[2]
> fSMOKE <- as.factor(SMOKE)
> plot(BWT~AGE, xlab="AGE en années", ylab=
+  "Poids de naissance (g)",main=expression(BWT~"="~
+  beta[0]+beta[1]*AGE+beta[2]*SMOKE+epsilon),
+  col = c('blue','red')[fSMOKE], pch=c(1,18)[fSMOKE])
> abline(a=a0,b,col="blue") ; abline(a=a1,b,col="red")
```

```
> legend("bottomright",c("SMOKE=0","SMOKE=1"),
+        col=c("red","blue"),lty=1)
```

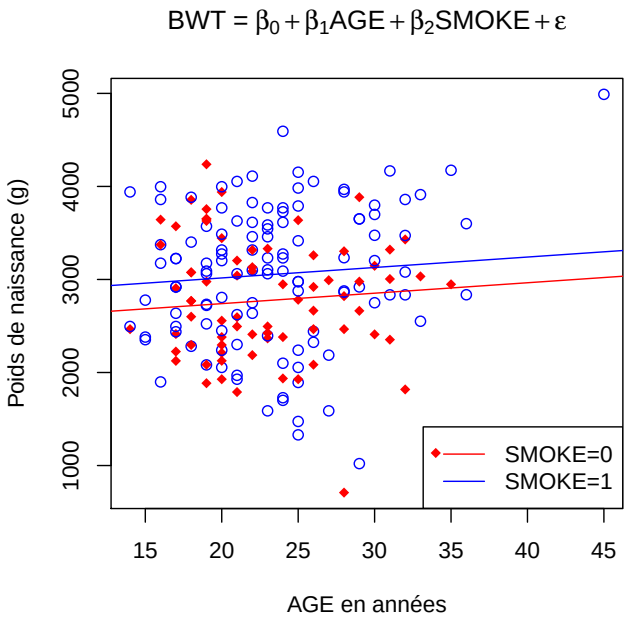


FIGURE 12.7 – Effet de l'âge sur `BWT` dans un modèle sans interaction.

Le graphique ci-contre présente deux pentes différentes illustrant la notion d'interaction.

```
> co <- coef(modele6); a0 <- co[1] ; a1 <- co[1]+co[3]
> b0 <-  co[2]       # Effet de l'âge chez les non-fumeuses.
> b1 <- co[2]+co[4]  # Effet de l'âge chez les fumeuses.
> plot(BWT~AGE,xlab="AGE en années",ylab="Poids de naissance (g)",
+ main=expression(BWT~"="~beta[0]+beta[1]*AGE+beta[2]*SMOKE+
+      beta[3]*AGE~"x"~SMOKE+epsilon))
> points(AGE[SMOKE==1],BWT[SMOKE==1],col="red",pch=18)
> points(AGE[SMOKE==0],BWT[SMOKE==0],col="blue")
> abline(a=a0,b0,col="blue") ; abline(a=a1,b1,col="red")
```

```
> legend("bottomright",c("SMOKE=0","SMOKE=1"),
+        col=c("red","blue"),lty=1)
```

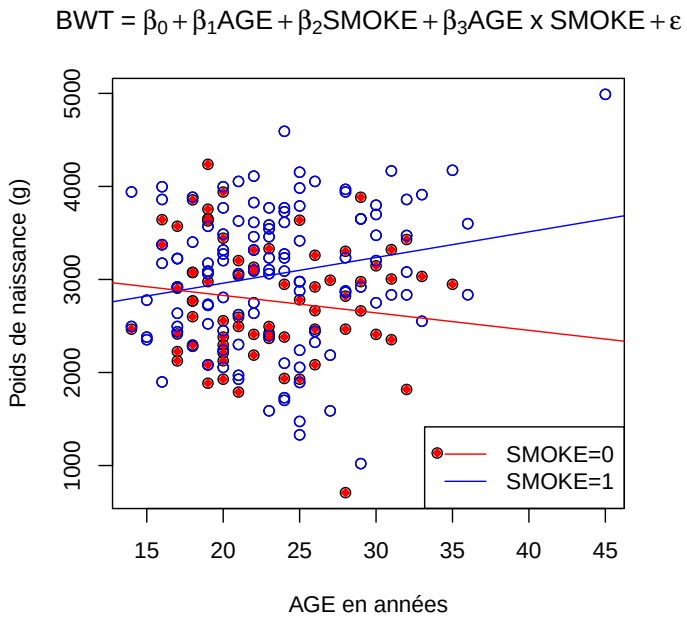


FIGURE 12.8 – Effet de l'âge sur `BWT` dans un modèle avec interaction.

Nous pouvons tester la significativité du terme d'interaction en analysant les résultats du modèle 6.

```
> summary(modele6)
Call:
lm(formula = BWT ~ AGE * SMOKE)
Residuals:
    Min      1Q  Median      3Q     Max
-2187.8  -456.6    52.8   526.6  1522.2
Coefficients:
             Estimate Std. Error t value            Pr(>|t|)
(Intercept)  2408.38     292.24    8.241 0.0000000000000305 ***
AGE            27.60      12.15    2.271              0.0243 *
SMOKE         795.38     484.42    1.642              0.1023
AGE:SMOKE     -46.36      20.45   -2.267              0.0245 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 709.4 on 185 degrees of freedom
Multiple R-squared:  0.0683,        Adjusted R-squared:  0.05319
F-statistic: 4.521 on 3 and 185 DF,  p-value: 0.004378
```

Le coefficient β_3 est significativement différent de zéro (valeur-p = 0.024). On conclut donc que l'effet de l'âge de la mère sur le poids de naissance de l'enfant n'est pas le même selon le statut tabagique de la mère. Par conséquent, on doit présenter les résultats de la mesure d'association entre l'âge de la mère et le poids de naissance de l'enfant séparément dans les groupes de femmes non fumeuses et fumeuses.

Pour être complet, chez les mères non fumeuses, l'effet de l'`AGE` est significatif (valeur-p = 0.024 3). Le poids moyen de naissance de l'enfant augmente avec l'âge de la mère, de 27.60 g ($\hat{\beta}_1$) pour un an de plus de la mère. Un intervalle de confiance est donné par :

```
> confint(modele6)[2,]
    2.5 %    97.5 %
 3.628108 51.573048
```

Chez les mères fumeuses, le poids moyen de naissance de l'enfant diminue avec l'âge de la mère, de 18.762 g ($\hat{\beta}_1 + \hat{\beta}_3$) pour un an de plus de la mère. Pour savoir si ce résultat est significatif, nous proposons de déterminer l'intervalle de confiance de $\beta_1 + \beta_3$. Un moyen astucieux est de créer une nouvelle variable `SMOKE1` qui permet d'inverser le codage de `SMOKE` puis de lancer les instructions suivantes :

```
> SMOKE1 <- 1-SMOKE
> confint(lm(BWT~AGE+SMOKE1+AGE:SMOKE1))[2,]
    2.5 %    97.5 %
-51.21423  13.68941
```

L'intervalle de confiance contient la valeur 0, l'effet de l'`AGE` sur `BWT` n'est pas significatif chez les mères fumeuses.

12.3.7 Problème de la colinéarité

Lorsque plusieurs variables explicatives apportent le même type d'information, plusieurs phénomènes peuvent apparaître :

- qualité des estimations perturbée (variance très grande) ;
- valeurs des coefficients contradictoires (signes opposés) ;
- coefficients devenant non significatifs.

C'est ce que l'on nomme le **problème de colinéarité**.

Le critère utilisé pour juger de la colinéarité entre les variables explicatives est le facteur d'inflation de la variance VIF (*variance inflation factor*) : $\frac{1}{1-r_j^2}$ où r_j^2 ne désigne rien d'autre que le coefficient de corrélation multiple au carré (coefficient de détermination) lorsque l'on régresse la j-ième variable explicative $\boldsymbol{x}_j$ sur l'ensemble des autres régresseurs.

Le VIF joue un rôle fondamental dans la variance des estimateurs puisque $\mathbb{V}\text{ar}\left(\hat{\beta}_j|\mathbf{X}=\boldsymbol{\mathcal{X}}\right) = \frac{\sigma^2}{n\times s_j^2} \times \frac{1}{1-r_j^2}$ où $s_j^2 = \frac{1}{n}\sum_{i=1}^n (x_{ij}-\overline{x}_j)^2$ est la variance de l'échantillon de $\boldsymbol{x}_j$. Plus $\boldsymbol{x}_j$ est colinéaire aux autres régresseurs, plus r_j^2 est proche de 1, donc plus le terme $\frac{1}{1-r_j^2}$ est élevé ; la variance de l'estimateur $\hat{\beta}_j$ est alors très élevée. À l'inverse, plus r_j^2 est proche de 0, plus le VIF associé est proche de 1 (le minimum). Ainsi, plus $\boldsymbol{x}_j$ est « indépendant » des autres régresseurs, et moins les estimations seront détériorées. Ainsi, la colinéarité entre régresseurs se répercute inévitablement dans la précision des estimateurs. On estime qu'il y a une forte colinéarité lorsque $\text{VIF}_j > 10$ (c'est-à-dire $r_j^2 > 0.9$).

L'instruction **R** pour calculer le VIF est `vif()`, disponible dans le *package* `car`.

Nous présentons l'utilisation de la fonction `vif()` dans l'exemple très simple du modèle

$$\mathbb{E}[\texttt{BWT}|\texttt{LWT},\texttt{AGE}] = \beta_0 + \beta_1 \texttt{LWT} + \beta_2 \texttt{AGE}.$$

```
> modele7 <- lm(BWT~LWT+AGE)
> vif(modele7)
     LWT      AGE
1.033513 1.033513
```

Remarque

Dans ce cas, les VIF sont identiques, car il n'y a que deux régresseurs. C'est un exemple très simple, car nous n'avions que deux régresseurs et nous aurions pu analyser cette colinéarité graphiquement, mais l'on comprend aisément son intérêt lorsque l'on a un grand nombre de régresseurs.

12.3.8 Sélection de variables

Parmi le grand nombre de variables explicatives potentielles, il s'agit de sélectionner celles qui sont le plus à même d'expliquer Y. Cela permet d'économiser le nombre de prédicteurs (et ainsi d'obtenir un modèle parcimonieux) et d'obtenir un bon pouvoir prédictif en éliminant les variables redondantes qui augmentent le facteur d'inflation de la variance (VIF).

Plus le nombre de paramètres augmente (nombre important de variables explicatives), plus l'ajustement aux données est bon (r^2 proche de 1). En contrepartie, l'estimation des paramètres est détériorée (la variance des estimateurs augmente) à cause des problèmes de colinéarité.

Nous présentons brièvement dans cette sous-section quelques méthodes de sélection de variables disponibles avec le logiciel R. Celles-ci sont illustrées sur quelques variables du jeu de données POIDS.NAISSANCE.

Les variables explicatives considérées sont : `LWT`, `AGE`, `UI`, `SMOKE`, `HT` et deux variables recodées `FTV1` et `PTL1`. On note `FVT1` $= 1$ s'il y a eu au moins une visite chez le médecin, et `FVT1` $= 0$ sinon. De même, on note `PTL1` $= 1$ s'il y a au moins un antécédent de prématurité, et `PTL1` $= 0$ sinon.

▸ La méthode du meilleur sous-ensemble (*best subset*)

Lorsque le nombre p de variables explicatives n'est pas trop grand, on peut étudier toutes les possibilités. Un algorithme efficace (voir [25] et [26]) permet ainsi d'aller jusqu'à une trentaine de variables ; il s'agit de la procédure *leaps and bounds*. À p fixé, on choisira le modèle de régression qui fournit le r^2 le plus grand. Pour deux modèles de régression ayant un nombre différent de variables explicatives, on peut choisir celle qui fournit le r_a^2 ajusté le plus grand.

La fonction R à utiliser est la fonction `leaps()` disponible dans le *package* `leaps`.

```
> require("leaps")
> FVT1 <- FVT; FVT1 <- as.integer(FVT>=1)
> PTL1 <- PTL; PTL1 <- as.integer(PTL>=1)
> matx <- model.matrix(lm(BWT~-1+LWT+AGE+UI+SMOKE+HT+FVT1+PTL1))
> # Equivalent de: matx <- cbind(LWT,AGE,UI,SMOKE,HT,FVT1,PTL1)
> adjr2.leaps <- leaps(matx,BWT,nbest=1,method="adjr2")
> meilleur.modele.adjr2 <- adjr2.leaps$which[adjr2.leaps$adjr2==
+                     max(adjr2.leaps$adjr2),]
> meilleur.modele.adjr2
    1     2     3     4     5     6     7
 TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
```

Le meilleur modèle au sens du r_a^2 ajusté est donc :

$$\texttt{BWT} = \beta_0 + \beta_1 \texttt{LWT} + \beta_2 \texttt{UI} + \beta_3 \texttt{SMOKE} + \beta_4 \texttt{HT} + \beta_5 \texttt{PTL1} + \epsilon.$$

Remarque

Il est possible d'obtenir d'autres critères de sélection que le r_a^2 ajusté grâce au paramètre `method` de la fonction `leaps()`. Par exemple, `method="Cp"` utilise le critère bien connu du C_p de Mallows [34].

Une autre fonction très intéressante disponible dans le *package* `leaps` est la fonction `regsubsets()`. Elle permet par exemple au moyen de son paramètre `force.in` de spécifier une ou plusieurs variables qui seront incluses dans tous les modèles comparés. Un exemple est donné ici en faisant le choix du meilleur modèle par le critère BIC (*bayesian information criterion*, [43]) :

```
> # On force SMOKE à être présente:
> meilleur.modele.bic <- regsubsets(matx,BWT,nbest=1,force.in=4)
> summary(meilleur.modele.bic)$bic # Valeurs du BIC des meilleurs
                                   # modèles de chaque taille
                                   # (nbest=1).
[1] -6.273748 -7.607040 -9.796126 -7.865648 -3.491572  1.544854
> plot(meilleur.modele.bic)
```

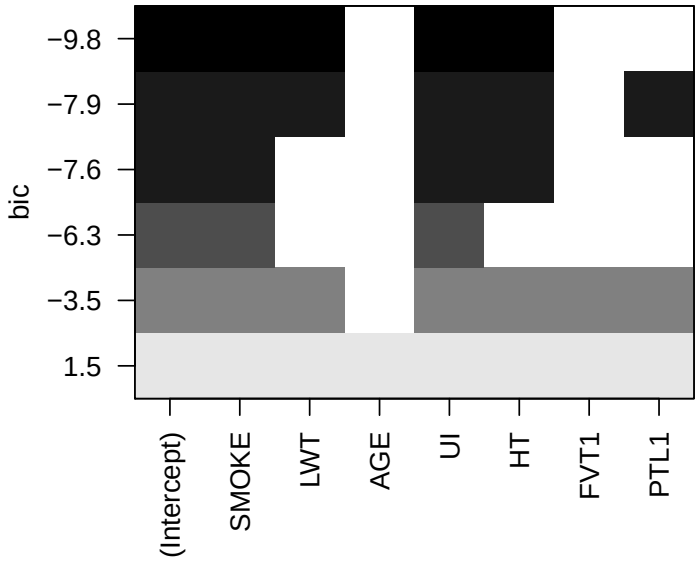


FIGURE 12.9 – Sélection de variables par le critère BIC.

Le meilleur modèle est celui ayant obtenu la valeur du critère BIC la plus petite. Le meilleur modèle au sens du BIC est :

$$\texttt{BWT} = \beta_0 + \beta_1 \texttt{LWT} + \beta_2 \texttt{UI} + \beta_3 \texttt{SMOKE} + \beta_4 \texttt{HT} + \epsilon.$$

Remarque

Il est possible d'obtenir d'autres critères de sélection grâce au paramètre `scale` utilisé dans la fonction `plot()` appliquée à un objet de classe

`regsubsets` ou en utilisant `$rsq`, `$rss`, `$adjr2` et `$cp` (pour *r-squared*, *residual sum of squares*, *adjusted r-squared* et Cp de Mallows respectivement), à la place de `$bic`, avec la fonction `summary()` comme exposé plus haut.

▸ La méthode pas à pas ascendante (*forward selection*)

La régression pas à pas ascendante (ou méthode par additions successives) est une méthode itérative. Elle consiste à sélectionner à chaque étape la variable explicative la plus significative (au seuil α) lorsque l'on régresse Y sur toutes les variables explicatives sélectionnées aux étapes précédentes et la nouvelle variable choisie, tant que l'apport marginal de cette dernière est significatif.

Montrons le fonctionnement de cette procédure à l'aide de la fonction `add1()` pour un seuil $\alpha = 0.05$.

```
> add1(lm(BWT~1),BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,test="F")
Single term additions
Model:
BWT ~ 1
       Df Sum of Sq      RSS    AIC F value     Pr(>F)
<none>              99917053 2492.7
LWT     1   3448881 96468171 2488.0  6.6855   0.010481 *
AGE     1    806927 99110126 2493.1  1.5225   0.218790
UI      1   8028747 91888305 2478.8 16.3391 0.00007732 ***
SMOKE   1   3573406 96343646 2487.8  6.9359   0.009156 **
HT      1   2132014 97785038 2490.6  4.0772   0.044894 *
FVT1    1   1338322 98578731 2492.1  2.5387   0.112772
PTL1    1   4757523 95159530 2485.4  9.3491   0.002558 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

`UI` est la variable la plus significative.

```
> add1(lm(BWT~UI),BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,test="F")
Single term additions
Model:
BWT ~ UI
       Df Sum of Sq      RSS    AIC F value  Pr(>F)
<none>              91888305 2478.8
LWT     1   2076990 89811315 2476.5  4.3015 0.03946 *
AGE     1    472355 91415950 2479.9  0.9611 0.32819
SMOKE   1   2949940 88938365 2474.7  6.1693 0.01388 *
HT      1   3162469 88725836 2474.2  6.6296 0.01081 *
FVT1    1    949028 90939278 2478.9  1.9411 0.16522
PTL1    1   2837049 89051257 2474.9  5.9257 0.01587 *
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

HT est la variable la plus significative.

```
> add1(lm(BWT~UI+HT),BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,test="F")
Single term additions
Model:
BWT ~ UI + HT
       Df Sum of Sq      RSS    AIC F value   Pr(>F)
<none>              88725836 2474.2
LWT     1   3560080 85165756 2468.5  7.7333 0.005982 **
AGE     1    415275 88310561 2475.3  0.8700 0.352184
SMOKE   1   2828310 85897527 2470.1  6.0914 0.014492 *
FVT1    1    698035 88027801 2474.7  1.4670 0.227365
PTL1    1   2682800 86043036 2470.4  5.7683 0.017308 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

LWT est la variable la plus significative.

```
> add1(lm(BWT~UI+HT+LWT),BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,
+         test="F")
Single term additions
Model:
BWT ~ UI + HT + LWT
       Df Sum of Sq      RSS    AIC F value  Pr(>F)
<none>              85165756 2468.5
AGE     1     94703 85071053 2470.3  0.2048 0.65138
SMOKE   1   2579898 82585858 2464.7  5.7480 0.01751 *
FVT1    1    509265 84656491 2469.3  1.1069 0.29414
PTL1    1   2127921 83037835 2465.7  4.7152 0.03118 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

SMOKE est la variable la plus significative.

```
> add1(lm(BWT~UI+HT+LWT+SMOKE),BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,
+         test="F")
Single term additions
Model:
BWT ~ UI + HT + LWT + SMOKE
       Df Sum of Sq      RSS    AIC F value  Pr(>F)
<none>              82585858 2464.7
AGE     1     65305 82520553 2466.5  0.1448 0.70397
FVT1    1    275436 82310423 2466.0  0.6124 0.43491
PTL1    1   1434298 81151560 2463.3  3.2344 0.07375 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Plus aucune variable n'est significative. La méthode s'arrête donc sur le modèle contenant les variables : `UI`, `HT`, `LWT` et `SMOKE`.

▸ La méthode pas à pas descendante (*backward selection*)

Cette méthode est aussi appelée régression par éliminations successives. On part cette fois du modèle complet et on élimine à chaque étape la variable ayant la plus petite valeur pour la statistique du test de Student (valeur-p la plus grande) en valeur absolue, à condition qu'il soit non significatif (au seuil α choisi).

Montrons le fonctionnement de cette procédure à l'aide de la fonction `drop1()` pour un seuil $\alpha = 0.05$.

```
> drop1(lm(BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1),test="F")
Single term deletions
Model:
BWT ~ LWT + AGE + UI + SMOKE + HT + FVT1 + PTL1
       Df Sum of Sq      RSS    AIC F value    Pr(>F)
<none>              80692151 2466.3
LWT     1   2475214 83167365 2470.0  5.5521 0.0195277 *
AGE     1     87708 80779859 2464.5  0.1967 0.6578974
UI      1   5431112 86123263 2476.6 12.1825 0.0006059 ***
SMOKE   1   1622617 82314768 2468.0  3.6397 0.0580009 .
HT      1   3885141 84577292 2473.2  8.7147 0.0035749 **
FVT1    1    272400 80964551 2464.9  0.6110 0.4354262
PTL1    1   1601044 82293195 2468.0  3.5913 0.0596772 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On retire la variable `AGE`.

```
> drop1(lm(BWT~LWT+UI+SMOKE+HT+FVT1+PTL1),test="F")
Single term deletions
Model:
BWT ~ LWT + UI + SMOKE + HT + FVT1 + PTL1
       Df Sum of Sq      RSS    AIC F value    Pr(>F)
<none>              80779859 2464.5
LWT     1   2740905 83520764 2468.8  6.1754 0.0138569 *
UI      1   5536620 86316478 2475.0 12.4742 0.0005228 ***
SMOKE   1   1644322 82424180 2466.3  3.7047 0.0558183 .
HT      1   3954174 84734033 2471.5  8.9089 0.0032279 **
FVT1    1    371701 81151560 2463.3  0.8375 0.3613362
PTL1    1   1530564 82310423 2466.0  3.4484 0.0649284 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On retire la variable `FVT1`.

```
> drop1(lm(BWT~LWT+UI+SMOKE+HT+PTL1),test="F")
Single term deletions
Model:
BWT ~ LWT + UI + SMOKE + HT + PTL1
       Df Sum of Sq      RSS    AIC F value    Pr(>F)
<none>              81151560 2463.3
LWT     1   2891443 84043002 2468.0  6.5203 0.0114803 *
UI      1   5763232 86914792 2474.3 12.9963 0.0004023 ***
SMOKE   1   1886275 83037835 2465.7  4.2536 0.0405794 *
HT      1   4217585 85369145 2470.9  9.5108 0.0023592 **
PTL1    1   1434298 82585858 2464.7  3.2344 0.0737548 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On retire la variable `PLT1`.

```
> drop1(lm(BWT~LWT+UI+SMOKE+HT),test="F")
Single term deletions
Model:
BWT ~ LWT + UI + SMOKE + HT
       Df Sum of Sq      RSS    AIC F value    Pr(>F)
<none>              82585858 2464.7
LWT     1   3311668 85897527 2470.1  7.3783 0.0072310 **
UI      1   6955671 89541530 2477.9 15.4971 0.0001171 ***
SMOKE   1   2579898 85165756 2468.5  5.7480 0.0175082 *
HT      1   4443587 87029445 2472.6  9.9002 0.0019278 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

La méthode s'arrête donc sur le modèle contenant les variables : `UI`, `HT`, `LWT` et `SMOKE`.

▸ La méthode pas à pas (*stepwise*)

Cet algorithme est un perfectionnement de la méthode ascendante. Il consiste à effectuer en plus, à chaque étape, des tests du type Student ou Fisher, ou encore à optimiser un certain critère, pour ne pas introduire une variable non significative et pour éliminer éventuellement des variables déjà introduites qui ne seraient plus informatives compte tenu de la dernière variable sélectionnée. L'algorithme s'arrête quand on ne peut plus ajouter ni retrancher de variables.

Nous présentons la fonction `step()` permettant d'effectuer une méthode du type pas à pas en utilisant à chaque étape de la procédure une sélection faite par le critère bien connu AIC (*An Information Criterion*) proposé par [2]. On peut aussi utiliser le critère bien connu BIC (*Bayesian Information Criterion*) au moyen de l'argument `k=log(n)` de la fonction `step()`.

```
> step(lm(BWT~1),BWT~LWT+AGE+UI+SMOKE+HT+FVT1+PTL1,
+         direction="both")
Start:  AIC=2492.66
BWT ~ 1
        Df Sum of Sq      RSS    AIC
+ UI     1   8028747 91888305 2478.8
+ PTL1   1   4757523 95159530 2485.4
+ SMOKE  1   3573406 96343646 2487.8
+ LWT    1   3448881 96468171 2488.0
+ HT     1   2132014 97785038 2490.6
+ FVT1   1   1338322 98578731 2492.1
<none>               99917053 2492.7
+ AGE    1    806927 99110126 2493.1
Step:  AIC=2478.83
BWT ~ UI
        Df Sum of Sq      RSS    AIC
+ HT     1   3162469 88725836 2474.2
+ SMOKE  1   2949940 88938365 2474.7
+ PTL1   1   2837049 89051257 2474.9
+ LWT    1   2076990 89811315 2476.5
<none>               91888305 2478.8
+ FVT1   1    949028 90939278 2478.9
+ AGE    1    472355 91415950 2479.9
- UI     1   8028747 99917053 2492.7
Step:  AIC=2474.21
BWT ~ UI + HT
        Df Sum of Sq      RSS    AIC
+ LWT    1   3560080 85165756 2468.5
+ SMOKE  1   2828310 85897527 2470.1
+ PTL1   1   2682800 86043036 2470.4
<none>               88725836 2474.2
+ FVT1   1    698035 88027801 2474.7
+ AGE    1    415275 88310561 2475.3
- HT     1   3162469 91888305 2478.8
- UI     1   9059202 97785038 2490.6
Step:  AIC=2468.47
BWT ~ UI + HT + LWT
        Df Sum of Sq      RSS    AIC
+ SMOKE  1   2579898 82585858 2464.7
+ PTL1   1   2127921 83037835 2465.7
<none>               85165756 2468.5
+ FVT1   1    509265 84656491 2469.3
+ AGE    1     94703 85071053 2470.3
- LWT    1   3560080 88725836 2474.2
- HT     1   4645559 89811315 2476.5
- UI     1   7482463 92648219 2482.4
Step:  AIC=2464.66
BWT ~ UI + HT + LWT + SMOKE
        Df Sum of Sq      RSS    AIC
```

```
+ PTL1    1   1434298 81151560 2463.3
<none>                82585858 2464.7
+ FVT1    1    275436 82310423 2466.0
+ AGE     1     65305 82520553 2466.5
- SMOKE   1   2579898 85165756 2468.5
- LWT     1   3311668 85897527 2470.1
- HT      1   4443587 87029445 2472.6
- UI      1   6955671 89541530 2477.9
Step:  AIC=2463.35
BWT ~ UI + HT + LWT + SMOKE + PTL1
       Df Sum of Sq      RSS    AIC
<none>                81151560 2463.3
+ FVT1    1    371701 80779859 2464.5
- PTL1    1   1434298 82585858 2464.7
+ AGE     1    187009 80964551 2464.9
- SMOKE   1   1886275 83037835 2465.7
- LWT     1   2891443 84043002 2468.0
- HT      1   4217585 85369145 2470.9
- UI      1   5763232 86914792 2474.3
Call:
lm(formula = BWT ~ UI + HT + LWT + SMOKE + PTL1)
Coefficients:
(Intercept)           UI           HT          LWT
   2631.449     -506.760     -633.149        9.331
      SMOKE         PTL1
   -208.464     -247.662
```

Attention

Le jeu de données POIDS.NAISSANCE a été adopté ici uniquement pour illustrer l'utilisation des fonctions R propres aux méthodes de sélection automatique alors que la bonne stratégie sur ce jeu de données particulier aurait plutôt consisté à procéder « manuellement ».

En effet, il est important de noter que diverses méthodes de sélection automatique peuvent ne pas conduire aux mêmes choix de variables explicatives à retenir dans le modèle final. Elles ont l'avantage d'être faciles à utiliser, et de traiter le problème de la sélection de variables de façon systématique. En revanche, l'inconvénient majeur est que les variables sont retenues ou éliminées du modèle sur la base de critères uniquement statistiques, sans tenir compte de l'objectif de l'étude. On aboutit généralement à un modèle qui peut être satisfaisant sur le plan purement statistique, alors que les variables retenues ne sont pas les plus pertinentes pour comprendre et interpréter les données de l'enquête.

12.3.9 Analyse des résidus

Nous présentons ici quelques éléments sur l'analyse des résidus permettant de vérifier les hypothèses du modèle et de détecter des valeurs possiblement atypiques ou aberrantes. Pour plus de détails, nous vous conseillons la lecture de [3].

▸ Validation des hypothèses du modèle

En régression linéaire simple, nous avons déjà évoqué l'analyse des résidus pour éprouver les hypothèses du modèle de régression. Deux graphiques ont été présentés permettant d'invalider l'hypothèse de normalité des erreurs et l'hypothèse d'homoscédasticité des erreurs.

Exemple fil rouge : Étude « Poids de naissance de l'enfant ».

Nous étudions la validité des hypothèses pour le modèle suivant :

$$\texttt{BWT} = \beta_0 + \beta_1 \texttt{SMOKE} + \beta_2 \texttt{AGE} + \beta_3 \texttt{LWT} + \beta_4 \texttt{RACE2} + \beta_5 \texttt{RACE3} + \beta_6 \texttt{UI} + \beta_7 \texttt{HT} + \beta_8 \texttt{SMOKE} \times \texttt{AGE} + \epsilon.$$

```
> modelefinal<-lm(BWT~SMOKE+AGE+LWT+factor(RACE)+UI+HT+SMOKE:AGE)
> par(mfrow=c(1:2))
> plot(modelefinal,1:2,col.smooth="red")
```

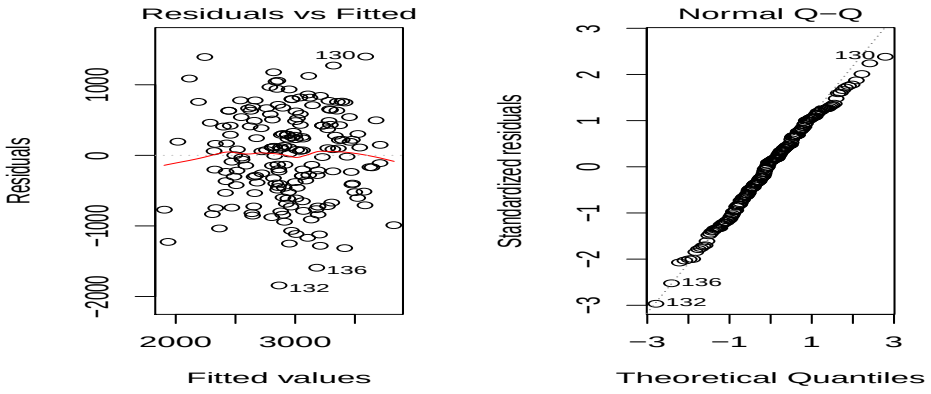


FIGURE 12.10 – Inspection de l'hypothèse d'homoscédasticité (à gauche) et de normalité (à droite).

```
> res <- residuals(modelefinal)
> par(mfrow=c(2,3))
> plot(res~SMOKE);plot(res~AGE);plot(res~LWT)
> plot(res~RACE);plot(res~UI);plot(res~HT)
```

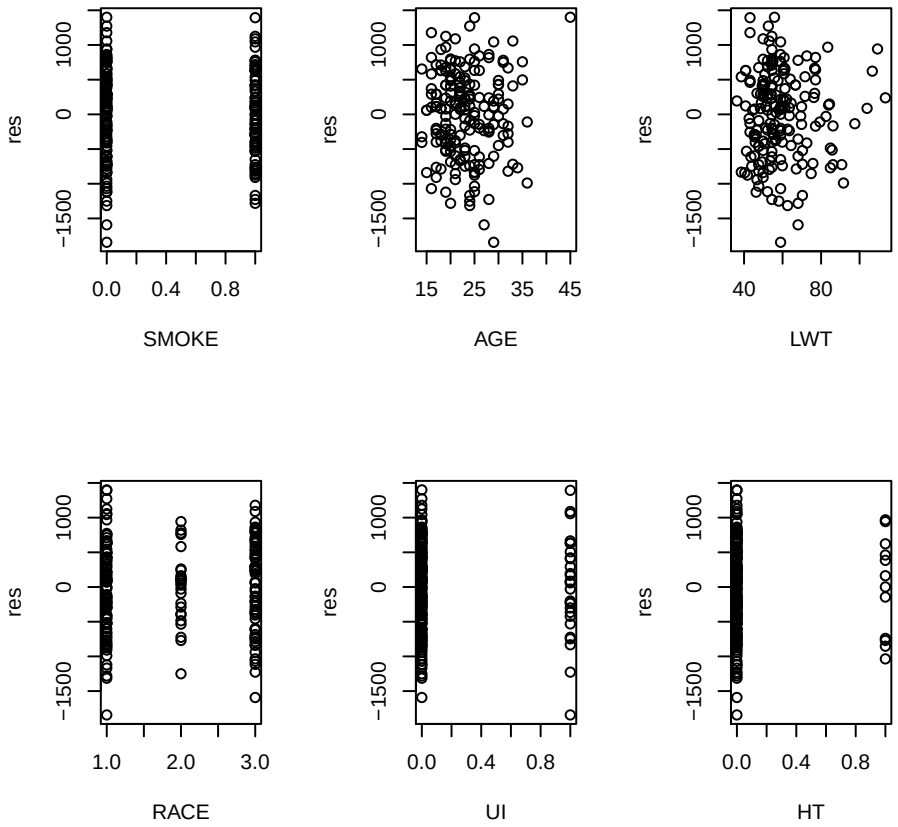


FIGURE 12.11 – Résidus en fonction des variables explicatives.

Il peut aussi être utile de représenter les résidus en fonction de chaque variable explicative comme cela est illustré sur le graphique ci-dessus. Ce type de graphique permet de détecter s'il existe une relation entre le terme d'erreur et les variables explicatives et ainsi de vérifier l'hypothèse d'indépendance entre les erreurs et les variables explicatives. Outre le fait que ce genre de graphique permet de vérifier l'hypothèse d'indépendance entre les erreurs et les variables explicatives, il est aussi utile afin de visualiser des points potentiellement atypiques.

▸ Points atypiques et/ou influents

Un point atypique ou aberrant est un point ayant un fort résidu, s'écartant ainsi des autres. Il peut être visualisé sur le graphique des résidus *versus* les valeurs prédites (ou *versus* l'une des variables explicatives), comme un point très éloigné. Plusieurs types de résidus sont alors définis :

- **les résidus standardisés** $t_i = \dfrac{\hat{\epsilon}_i}{\hat{\sigma}\sqrt{1-h_{ii}}}$ avec h_{ii} le « levier » (défini ultérieurement). Ces résidus sont obtenus par la fonction `rstandard()`. Les résidus standardisés sont « principalement » compris entre -2 et 2, mais ils sont dépendants ;
- **les résidus studentisés** $t_i^* = \dfrac{\hat{\epsilon}_i}{\hat{\sigma}_{(-i)}\sqrt{1-h_{ii}}} = t_i\sqrt{\dfrac{n-p-2}{n-p-1-t_i^2}}$ où $\hat{\sigma}_{(-i)}$ est l'estimation résiduelle, obtenue sans l'utilisation de l'observation i. Les résidus studentisés sont obtenus via la fonction `rstudent()`. Une observation sera considérée comme aberrante lorsque $|t_i^*| > t_{0.975}^{(n-p-2)}$ ($t_{0.975}^{(n-p-2)}$ étant le quantile d'ordre 0.975 d'une loi de Student à $n-p-2$ degrés de liberté).

```
> res.stud <- rstudent(modelefinal)     # Calcul des résidus
                                        # studentisés.
> seuil.stud <- qt(0.975,189-8-2)       # Calcul du seuil par la
                                        # loi de Student.
> cond <- res.stud<(-seuil.stud) | res.stud > seuil.stud
> # Liste des individus susceptibles d'être considérés comme
  # aberrants.
> id.student <- ID[cond] # ID (numéro d'identification) est la
                         # première colonne du jeu de données.
> val.ajust <- fitted(modelefinal)
> plot(res.stud~val.ajust,xlab="Valeurs ajustées",
+      ylab="Résidus studentisés",pch=20)
> abline(h=c(-seuil.stud,seuil.stud))
```

```
> text(val.ajust[cond],res.stud[cond],id.student,col="red",pos=1)
```

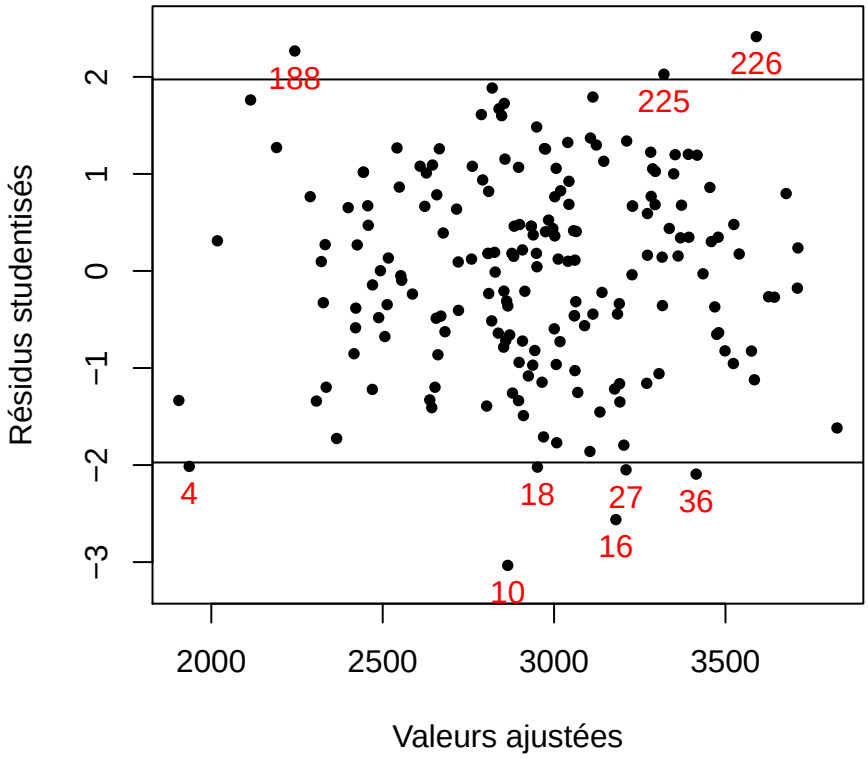


FIGURE 12.12 – Visualisation des points atypiques : résidus studentisés *versus* valeurs ajustées.

Un autre moyen d'étudier les points atypiques est la notion de « points leviers ». Le levier pour l'observation i (noté h_{ii}) est la valeur lue sur la diagonale de la matrice $\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{X}}(\boldsymbol{\mathcal{X}}^\mathsf{T}\boldsymbol{\mathcal{X}})^{-1}\boldsymbol{\mathcal{X}}^\mathsf{T}$ (dite *hat matrix*). Cette mesure intervient principalement dans la variance des résidus : $\mathbb{V}\text{ar}(\hat{\epsilon}_i) = \sigma^2(1 - h_{ii})$. Un levier dépassant $2(p+1)/n$ peut être considéré comme important. Un h_{ii} élevé indique que la i-ième observation est éloignée du centre de gravité.

Les h_{ii}, stockés dans le vecteur `levier`, sont obtenus par :

```
> # Équivalent à hat(model.matrix(modelefinal)):
> levier <- hatvalues(modelefinal)
> # On aurait pu également utiliser les deux instructions
  # suivantes:
> atyp <- influence.measures(modelefinal)
> levier <- atyp$infmat[,"hat"]
```

Pour détecter les valeurs atypiques au sens du levier, on peut taper :

```
> seuil.levier <- 2*(8+1)/189
> atyp.levier <- ID[levier>seuil.levier]
> # Liste des individus qui ont un levier important:
> atyp.levier
 [1]  85  98 119 126 138 159 168 187 197 202 226  11  13  19
[15]  20  28  75  83  84
```

D'autres mesures diagnostiques sont aussi utilisées pour inspecter les valeurs atypiques et déterminer l'influence de ces valeurs sur le modèle de régression réalisé :

– **La distance de Cook**. Elle permet de mesurer l'influence de l'observation i sur l'estimation de l'ensemble des paramètres de régression. Elle est définie par :

$$\begin{aligned} C_i &= \frac{\sum_{j=1}^{n}\left(\hat{y}_j - \hat{y}_j^{(-i)}\right)^2}{\hat{\sigma}^2(p+1)} = \left(\frac{1}{p+1}\right)\left(\frac{h_{ii}}{1-h_{ii}}\right)t_i^2 \\ &= \left(\frac{1}{p+1}\right)\left(\frac{h_{ii}}{(1-h_{ii})^2}\right)\frac{\hat{\epsilon}_i^2}{\hat{\sigma}^2} \\ &= \left(\frac{1}{p+1}\right)\left(\frac{h_{ii}}{1-h_{ii}}\right)\frac{\hat{\sigma}^2_{(-i)}}{\hat{\sigma}^2}t_i^{*2}, \end{aligned}$$

où $\hat{y}_j^{(-i)}$ est la prédiction au point $\boldsymbol{x}_j = (1, x_{j1}, \ldots, x_{jp})^\mathsf{T}$ obtenue en utilisant le modèle estimé sans la i-ième observation.

Une forte valeur de C_i indique que la i-ième observation est influente (1 est parfois considéré comme limite). La suppression de cette observation peut entraîner de grosses modifications sur l'équation de la régression. La fonction R permettant de récupérer les distances de Cook est `cooks.distance()`.

Voici une représentation graphique :

```
> plot(cooks.distance(modelefinal),type="h")
> # Ou encore: plot(modelefinal,4)
```

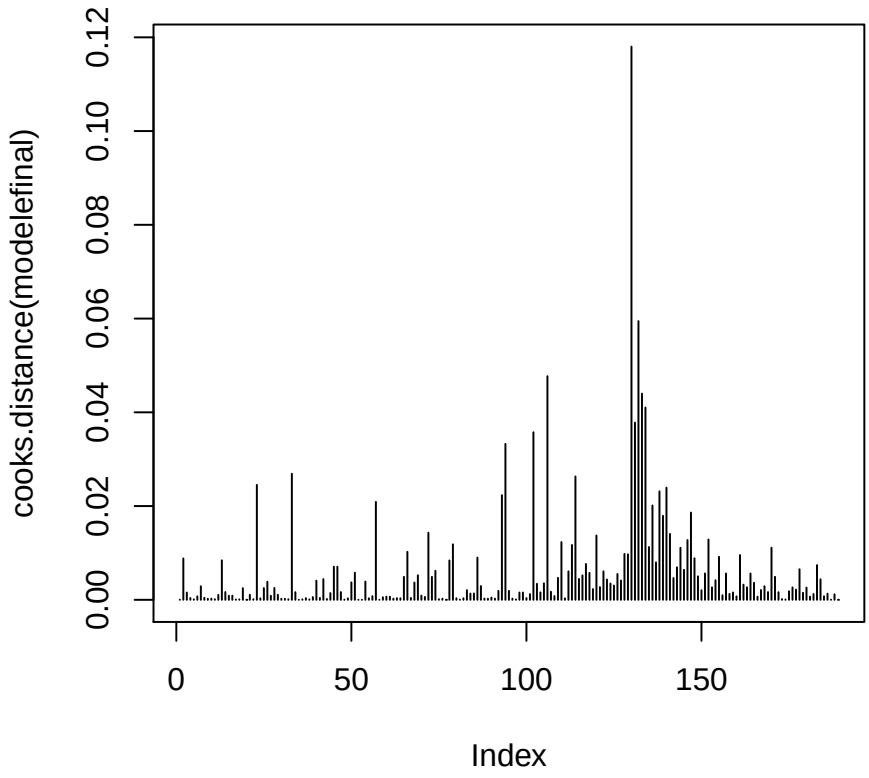


FIGURE 12.13 – Visualisation d'observations influentes : distance de Cook.

Du point de vue de la distance de Cook, aucune valeur ne semble être considérée comme influente globalement.

– **La distance de Welsh-Kuh ou Dffits**. Elle est définie par :

$$\text{Dffits}_i = \frac{\hat{y}_i - \hat{y}_i^{(-i)}}{\hat{\sigma}_{(-i)}\sqrt{h_{ii}}} = t_i^* \sqrt{\frac{h_{ii}}{1-h_{ii}}}.$$

Une forte valeur de $|\text{Dffits}_i|$ indique une influence de l'observation i sur l'estimation $\hat{y}_i$, et permet donc d'affirmer que cette observation est in-

fluente sur les résultats de la régression. Dans la pratique, on considère qu'une observation peut être influente dès que $|\text{Dffits}_i| \geq 2\sqrt{\frac{p+1}{n}}$.

La fonction R permettant de récupérer les Dffits est `dffits()`.

```
> seuil.dffit <- 2*sqrt((8+1)/189)
> ID[abs(dffits(modelefinal))>=seuil.dffit]
 [1] 108 119 187 188 197 202 210 226   4  10  11  13  18  20
```

– **La mesure Dfbetas**. Elle est définie par :

$$\text{Dfbetas}_{j,i} = \frac{\hat{\beta}_j - \hat{\beta}_j^{(-i)}}{\hat{\sigma}_{(-i)}\sqrt{(\boldsymbol{X}^\intercal\boldsymbol{X})^{-1}_{j+1;j+1}}}$$

où $\hat{\beta}_j^{(-i)}$ est l'estimation de β_j obtenue sans utiliser la i-ième observation. Cette quantité mesure l'influence de l'observation i sur l'estimation du j-ième coefficient. Pour des jeux de données de petite taille ou de taille modérée, une valeur supérieure à 1 semble suspecte. Pour de gros jeux de données, on considérera que l'observation i est suspecte si $|\text{Dfbetas}_{j,i}| > 2/\sqrt{n}$ pour au moins un j.

La fonction R permettant de récupérer les Dfbetas est `dfbetas()`.

```
> seuil.dfbetas <- 1
> ID[apply(abs(dfbetas(modelefinal))>seuil.dfbetas,
+     FUN=any,MARGIN=1)]
integer(0)
```

Ici, aucune valeur ne semble suspecte.

– **Le rapport de covariance**. Il est défini, pour la i-ème observation, comme le rapport du déterminant de la matrice estimée des variances-covariances de $\widehat{\boldsymbol{\beta}}_{(-i)}$ (estimateur de $\boldsymbol{\beta}$ n'utilisant pas la i-ème observation) par le déterminant de la matrice estimée des variances-covariances de $\widehat{\boldsymbol{\beta}}$ (estimateur de $\boldsymbol{\beta}$) :

$$\frac{\det\left[\widehat{\mathbb{V}\text{ar}(\boldsymbol{\beta}_{(-i)})}\right]}{\det\left[\widehat{\mathbb{V}\text{ar}(\boldsymbol{\beta})}\right]}.$$

Si le *ratio* est près de 1, l'observation i n'influence pas significativement la matrice de covariance.

```
> max(abs(covratio(modelefinal)-1))
[1] 0.2897528
```

Remarque

Pour plus de détails sur ces mesures diagnostiques, nous conseillons la lecture de [3], [7] ou [14].

12.3.10 Cas de la régression polynomiale

Le modèle polynomial consiste à représenter la relation entre la variable explicative Y et une variable explicative X sous une forme non linéaire du type :

$$\mathrm{Y} = \beta_0 + \beta_1 \mathrm{X} + \beta_2 \mathrm{X}^2 + \cdots + \beta_p \mathrm{X}^p + \epsilon.$$

Ce modèle est un modèle de régression multiple avec p régresseurs qui sont les puissances successives de la variable explicative.

Ainsi, pour effectuer une régression avec modèle polynomial, il suffit de spécifier correctement la formule associée au modèle dans la fonction `lm()`. Deux fonctions R sont alors utiles : la fonction `I()` et la fonction `poly()`. Le tableau ci-dessous présente quelques exemples de syntaxe de formules pour effectuer des modèles polynomiaux.

Modèle envisagé	**Formule R**
$M_1 : Y = \beta_0 + \beta_1 X + \beta_2 X^2$	`Y~poly(X,2,raw=TRUE)`
$M_2 : Y = \beta_1 X + \beta_2 X^2 + \beta_3 X^3$	`Y~-1+poly(X,3,raw=TRUE)`
$M_3 : Y = \beta_0 + \beta_1 X + \beta_2 X^3$	`Y~X+I(X^3)`
$M_4 : Y = \beta_1 X + \beta_2 X^3 + \beta_3 X^4$	`Y~-1+X+I(X^3)+I(X^4)`

12.3.11 Récapitulatif

Le tableau ci-dessous présente les principales fonctions à utiliser afin d'effectuer une régression linéaire multiple.

Table 12.2 – Liste des principales fonctions R permettant l'analyse d'une régression linéaire multiple.

Instruction R	**Description**
`pairs()`	inspection graphique
`lm(Y~X1+X2+...+X3)`	estimation du modèle linéaire multiple
`summary(lm())`	description des résultats du modèle
`confint(lm())`	intervalle de confiance des paramètres de régression
`predict()`	fonction permettant d'obtenir des prédictions
`plot(lm())`	analyse graphique des résidus
`anova(mod1,mod2)`	test de Fisher partiel
`X1:X2`	interaction entre X_1 et X_2
`vif()`	calcul du VIF

Termes à retenir

`lm()` : permet d'effectuer un modèle de régression linéaire
`summary(lm())` : résultats du modèle linéaire
`confint()` : intervalle de confiance des paramètres de régression
`predict()` : prédiction de nouvelles valeurs
`residuals()` : récupère les résidus d'un modèle linéaire
`plot(lm(())` : graphiques pour la validation du modèle
`pairs()` : diagramme de dispersion
`anova(lm())` : table d'analyse de la variance d'un modèle linéaire
`X1X2` : terme d'interaction
`rstandard()` : résidus standardisés
`rstudent()` : résidus studentisés
`vif()` : calcul du VIF
`cooks.distance()` : calcul des distances de Cook
`dffits()` : calcul des distances de Welsh-Kuh
`dfbetas()` : calcul de la mesure Dfbetas
`step()` : procédure pas à pas avec le critère AIC
`regsubsets()` : procédure de sélection exhaustive

Exercices

12.1- Donnez l'instruction permettant d'ajuster le modèle suivant $Y = \beta_0 + \beta_1 X_1 + \epsilon$.

12.2- Donnez l'instruction permettant d'ajuster le modèle suivant $Y = \beta_1 X_1 + \epsilon$.

12.3- Donnez l'instruction permettant d'ajuster le modèle suivant $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$.

12.4- Donnez l'instruction permettant d'ajuster le modèle suivant $Y = \beta_0 + \beta_1 X_1 \times \beta_2 X_2 + \beta_3 X_1 + \beta 4 X_2 + \epsilon$.

12.5- Donnez l'instruction permettant d'ajuster le modèle suivant $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_1^4 + \epsilon$.

12.6- Quelle est l'instruction pour faire un test de Fisher partiel ?

12.7- Quelle fonction permet de récupérer les résidus d'un modèle ?

12.8- Quelle fonction permet de récupérer les estimations d'un modèle de régression ?

12.9- Soit Z une variable qualitative, quelle fonction doit-on utiliser pour ajuster un modèle de régression avec Z comme variable explicative ?

12.10- Quelle est l'instruction permettant d'ajuster le modèle polynomial suivant : $Y = \beta_0 + \beta_1 X^1 + \beta_2 X^2 + \beta_3 X^3 + \epsilon$?

12.11- Quelle fonction permet de faire de la sélection du type pas à pas ascendant ?

12.12- Quelle fonction permet de faire de la sélection du type pas à pas descendant ?

Fiche de TP

A- Étude sur la régression linéaire simple

• Étude sur données simulées

12.1- Simulez un jeu de données (x_i, y_i), $i = 1, \ldots, n$ provenant d'un modèle de régression linéaire simple. Pour cela :
- choisissez les paramètres réels β_0 et β_1, ainsi que $\sigma > 0$;
- simulez le vecteur des erreurs $\boldsymbol{e}$ de taille n selon une loi normale $\mathcal{N}(0, \sigma^2)$;
- simulez le vecteur des valeurs de la variable explicative x de taille n selon une loi uniforme sur $[0, t]$ où t est un réel positif à choisir ;
- construisez enfin le vecteur des valeurs de la variable à expliquer y de taille n selon le modèle de régression linéaire.

12.2- Visualisez le nuage des n points (x_i, y_i).

12.3- Donnez une estimation des paramètres de régression et de la variance de l'erreur.

12.4- Faites une étude sur les résidus afin de valider le modèle :
- graphique des résidus en fonction des valeurs ajustées ;
- graphique des valeurs ajustées en fonction des valeurs observées ;
- graphique permettant de juger de la normalité des résidus.

12.5- Faites varier les valeurs de n et de σ pour appréhender les conséquences que cela peut avoir sur la précision des estimations des paramètres de régression (en termes de variance).

• Étude sur l'intima-média

Dans l'étude « Intima-média », on cherche à étudier la relation entre la mesure de l'épaisseur de l'intima-média et l'âge.

12.1- Récupérez le fichier de données sur l'intima-média.

12.2- Tracez le nuage de points de la variable `mesure` en fonction de la variable `AGE`. Décrivez-le.

12.3- Existe-t-il une liaison entre ces deux variables ? Précisez l'indicateur de liaison qui permet de mesurer l'intensité entre ces deux variables.

12.4- On cherche maintenant à ajuster une droite de régression sur ce nuage de points :
- proposez un modèle de régression et estimez les paramètres du modèle ;
- tracez la droite obtenue sur le nuage de points.

12.5- Faites une étude sur les résidus afin de valider le modèle.

12.6- Donnez un intervalle de prévision de la mesure de l'intima-média pour une personne de 33 ans.

12.7- Donnez l'intervalle de confiance de la mesure de l'intima-média moyen pour une personne de 33 ans.

12.8- Proposez un modèle permettant d'améliorer la prédiction de la mesure de l'intima-média avec pour seule variable explicative `AGE`.

B- Étude sur la régression linéaire multiple

• Étude sur l'intima-média

Au TP précédent, nous nous sommes intéressés à la relation entre la mesure de l'épaisseur de l'intima-média et l'âge. L'objectif ici est d'ajuster un modèle de régression sur l'ensemble des variables pouvant expliquer les variations de la mesure de l'épaisseur de l'intima-média. L'étude portera sur les variables suivantes : `AGE`, `SPORT`, `alcool`, `paqan` et la variable `IMC` que vous devez créer à partir des variables `taille`, `poids`.

On s'intéresse plus particulièrement au tabac au travers de la variable `paqan` comme facteur d'exposition principal. On décide donc de garder cette variable dans le modèle même si elle n'est pas significative.

12.1- Présentez un diagramme de dispersion de toutes les paires de ces variables (explicatives et variable d'intérêt). Pouvez-vous suspecter un possible problème de colinéarité ?

12.2- Effectuez une analyse univariée de la mesure de l'intima-média sur chacune des variables explicatives.

12.3- On ne retient que les variables explicatives associées avec un degré de signification $p < 0.25$ en analyse univariée. Testez maintenant, une à une, les interactions possibles entre les variables explicatives sélectionnées et la variable d'exposition principale `paqan`.

12.4- Estimez et analysez le modèle contenant toutes les variables déclarées significatives lors des analyses univariées (α = 25 %) et les termes d'interaction significatifs à 10 %.

12.5- Les termes d'interaction sont-ils encore significatifs ? Enlevez les termes d'interaction s'ils ne sont plus significatifs au seuil de 10 %.

12.6- À partir du modèle trouvé à la question précédente, enlevez une à une les variables non significatives au seuil de 5 % en vous assurant que l'élimination des variables ne change pas considérablement l'estimation du coefficient associé au statut tabagique.

12.7- Interprétez le modèle final.

• Étude sur le taux de chômage

Ce TP a pour thème l'analyse du taux de chômage de 1960 à 1993. Le jeu de données nommé `chomage` est constitué de $n = 34$ observations annuelles (de 1960 à 1993). Voici une description des variables fournies :

- `an` : année ;
- `chom` : taux de chômage ;
- `txpib` : taux de variation du produit intérieur brut (PIB) représentant le taux de croissance de l'économie ;
- `deppub` : part des dépenses publiques par rapport au PIB, qui peut ainsi représenter le degré d'intervention de l'État dans l'économie ;
- `pfisc` : pressions fiscales, pour voir si une imposition trop importante des entreprises nuit à leur embauche et donc au niveau du chômage ;
- `salva` : la part des salaires par rapport à la valeur ajoutée permettant de connaître l'influence du coût sur l'embauche ;
- `infl` : taux d'inflation afin de vérifier la relation inverse entre le chômage et l'inflation définie par la courbe de Phillips.

12.1- On envisage un modèle linéaire expliquant la variable `chom` en fonction de la seule variable `txpib`. Récupérez le fichier de données `http://www.biostatisticien.eu/springeR/chomage.RData`. Faites une analyse complète du modèle de régression linéaire simple sous-jacent.

12.2- On envisage un modèle linéaire multiple expliquant la variable `chom` en fonction de toutes les variables explicatives du jeu de données (excepté la variable `an`). Donnez la matrice de corrélation des variables mises en jeu.

12.3- Présentez un diagramme de dispersion de toutes les paires de ces variables.

12.4- Quelles sont les variables explicatives qui vous semblent être les plus explicatives ? Peut-on suspecter de la colinéarité entre les régresseurs ?

12.5- Présentez les résultats du modèle de régression linéaire multiple avec toutes ces variables explicatives.

12.6- Calculez le `VIF` associé à chaque variable explicative pour le modèle étudié.

12.7- Effectuez une régression pas à pas descendante avec un seuil $\alpha = 0.2$.

12.8- Présentez le modèle final.

12.9- Supposons que l'on ne connaisse pas la valeur de `chom` en 1993. Pourriez-vous prévoir sa valeur et calculer un intervalle de prédiction au niveau 95 % ?

12.10- Quelle était la valeur observée de `chom` en 1993, et est-ce surprenant ?

C- Étude sur la régression polynomiale

• Étude sur données simulées

12.1- Simulez un échantillon de taille $n = 100$ selon le modèle ci-après :

$$Y = X + 2X^2 + 3.5X^3 - 2.3X^4 + \epsilon$$

où X suit une loi uniforme sur $[-2, +2]$ et ϵ suit une loi $\mathcal{N}(0, 1)$.

12.2- Tracez le nuage des points ainsi que le polynôme simulé.

12.3- Ajustez un modèle de régression linéaire simple. Pensez à inspecter les résidus.

12.4- Ajustez une régression polynomiale en prenant un polynôme de degré 4. Représentez le modèle estimé sur le nuage de points.

• Ajustement d'un nuage de points par un polynôme

On considère le cas où l'on vous demande en tant que statisticien de proposer un modèle permettant de prédire une variable Y à partir de la connaissance d'une variable explicative X. Vous disposez pour cela d'un échantillon de taille n.

12.1- Récupérez le fichier de données nommé `http://www.biostatisticien.eu/springeR/ajustpoly.RData`.

12.2- Tracez le nuage de points de la variable Y en fonction de la variable X.

12.3- Existe-t-il une liaison linéaire entre ces deux variables ? Ajoutez la droite de régression sur le graphique précédent.

12.4- Effectuez une analyse de régression polynomiale pour ajuster au mieux les données.

12.5- Représentez sur le nuage de points le polynôme estimé ainsi que la courbe de confiance de la moyenne de Y pour $X \in [-3.5, 3.5]$. Rajoutez enfin l'intervalle de prédiction du modèle pour $X \in [-3.5, 3.5]$.

Chapitre 13

Analyse de variance élémentaire

Pré-requis et objectif

- Lecture du chapitre 12.
- Ce chapitre décrit les différentes commandes à taper sous R pour effectuer une analyse de la variance. Nous présentons les cas classiques de l'analyse de la variance à 1 facteur et à 2 facteurs avec ou sans interaction. Nous exposons également brièvement l'analyse de variance à mesures répétées.

SECTION 13.1

Analyse de la variance à un facteur

13.1.1 Les objectifs, les données et le modèle

▶ *Objectif :* L'analyse de la variance (ANOVA) est une méthode qui permet d'étudier la modification de la moyenne μ du phénomène étudié Y (variable quantitative) selon l'influence d'un ou de plusieurs facteurs d'expérience qualitatifs (traitements). Dans le cas où la moyenne n'est influencée que par un seul facteur (noté facteur X), il s'agit d'**une analyse de la variance à un facteur** (*one way ANOVA*). Un facteur est souvent une variable qualitative présentant un nombre restreint de modalités. Le nombre de modalités (c'est-à-dire de niveaux) du facteur X sera noté I. On suppose que Y suit une loi normale $\mathcal{N}(\mu_i, \sigma^2)$ sur chaque sous-population i définie par les modalités de X. L'objectif est de tester l'égalité des moyennes de ces I populations, à savoir de tester l'hypothèse nulle

$$\mathcal{H}_0 : \mu_1 = \mu_2 = \cdots = \mu_I$$

contre l'assertion d'intérêt

$\mathcal{H}_1 : \exists\, i \neq i' \;/\; \mu_i \neq \mu_{i'}$ (« il existe au moins deux moyennes différentes »).

- *Les données :* Pour chaque sous-population i (ou modalité i de X ou groupe i), on dispose d'un échantillon de n_i observations de la variable quantitative Y :

$$y_{i,1},\ y_{i,2}, \ldots,\ y_{i,n_i}.$$

- *Le modèle :* Il s'écrit

$$\mathrm{Y}_{ik} = \mu_i + \epsilon_{ik}, \quad \text{pour} \quad k = 1, \ldots, n_i \quad \text{et} \quad i = 1, \ldots, I$$

où les erreurs ϵ_{ik} sont des variables aléatoires indépendantes de loi $\mathcal{N}(0, \sigma^2)$. On peut aussi écrire μ_i sous la forme suivante : $\mu_i = \mu + \alpha_i$ pour $i = 1, \ldots, I$. Dans ce cadre-là, μ est appelé l'*effet moyen* du facteur et $\alpha_i = \mu_i - \mu$ est appelé l'*effet différentiel du niveau* i du facteur. Le modèle ci-dessus peut donc aussi s'écrire sous la forme :

$$\mathrm{Y}_{ik} = \mu + \alpha_i + \epsilon_{ik}, \quad \text{pour} \quad k = 1, \ldots, n_i \quad \text{et} \quad i = 1, \ldots, I.$$

Ce modèle n'est cependant pas identifiable (c'est-à-dire que certains paramètres ne peuvent pas être estimés). Il est donc nécessaire d'appliquer une contrainte (linéaire) pour le rendre identifiable, par exemple $\sum_{i=1}^{I} \alpha_i = 0$ qui correspond à prendre l'effet moyen μ comme référence.

Attention

Le logiciel R propose par défaut d'imposer la contrainte $\alpha_1 = 0$. Les comparaisons sont alors faites par rapport à μ_1. La classe de référence est le niveau 1 du facteur.

Renvoi

La notion de groupe de référence a déjà été expliquée à la page 512 du chapitre 12.

13.1.2 Exemple et inspection graphique

- *Exemple d'application :* Boutons de fièvre.

Cinq traitements $(T_1, \ldots, T_5)$ contre les boutons de fièvre, dont un placebo, ont été administrés par tirage au sort à trente patients (six patients par groupe de traitement). Le délai (en jours) entre l'apparition des boutons et la cicatrisation complète a été recueilli chez chaque patient.

Traitements				
T_1 (placebo)	T_2	T_3	T_4	T_5
5	4	6	7	9
8	6	4	4	3
7	6	4	6	5
7	3	5	6	7
10	5	4	3	7
8	6	3	5	6

L'objectif ici est de comparer les moyennes théoriques des délais de cicatrisation, délais observés sur cinq échantillons indépendants (groupes de traitement).

► *Inspection graphique :* Tout d'abord, nous allons effectuer une brève analyse descriptive de ces données pour voir si certaines tendances probables se dégagent.

```
> X<-data.frame(Placebo=c(5,8,7,7,10,8),T2=c(4,6,6,3,5,6),
+   T3=c(6,4,4,5,4,3),T4=c(7,4,6,6,3,5),T5=c(9,3,5,7,7,6))
> delai <- stack(X)$values # stack() permet d'opérer un
                           # empilement.
> traitement <- stack(X)$ind
> tapply(delai,traitement,summary)
$Placebo
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    5.0     7.0     7.5     7.5     8.0    10.0
$T2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00    4.25    5.50    5.00    6.00    6.00
$T3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   4.000   4.333   4.750   6.000
$T4
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.250   5.500   5.167   6.000   7.000
$T5
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.250   6.500   6.167   7.000   9.000
```

```
> plot(delai~traitement)
```

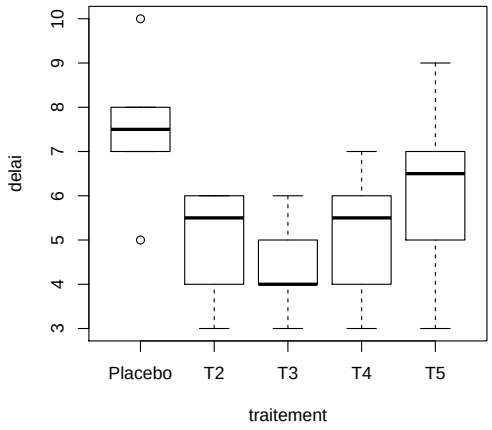


FIGURE 13.1 – Boîtes à moustaches des délais de cicatrisation pour chaque traitement.

13.1.3 Table d'ANOVA et estimations des paramètres

- *Instruction* R *pour la table d'ANOVA :* La fonction à utiliser est `aov()`. Comme pour le modèle de régression, l'ANOVA fonctionne avec des formules R. Il faut donc spécifier le modèle à utiliser.

```
> mon.aov <- aov(delai~traitement)
> summary(mon.aov)
            Df Sum Sq Mean Sq F value Pr(>F)
traitement   4  36.47   9.117   3.896 0.0136 *
Residuals   25  58.50   2.340
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Attention

Si cela n'est pas le cas, pensez à déclarer votre variable facteur (ici la variable `traitement`) comme un objet R du type `factor` au moyen de la fonction `factor()`.

```
> class(traitement)
[1] "factor"
```

Comme l'ANOVA est en fait un modèle linéaire, notons qu'il est aussi possible d'effectuer l'analyse de la variance du modèle linéaire sous-jacent :

```
> modele <- lm(delai~traitement)
> anova(modele)
Analysis of Variance Table
Response: delai
           Df Sum Sq Mean Sq F value  Pr(>F)
traitement  4 36.467  9.1167   3.896 0.01359 *
Residuals  25 58.500  2.3400
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Astuce

Notez qu'il est également possible d'utiliser la fonction `Anova()` disponible dans le *package* `car`. Cette dernière fonction est plus complète et permet de traiter des données plus complexes.

Le tableau d'analyse de la variance renvoie le résultat du test de Fisher associé aux hypothèses : $\mathcal{H}_0 : \mu_1 = \mu_2 = \cdots = \mu_I$ et $\mathcal{H}_1 : \exists\ i \neq i' \ / \ \mu_i \neq \mu_{i'}$ (« il existe au moins deux moyennes différentes »). La valeur-$p = 0.013$ nous permet de conclure que les effets d'au moins deux traitements diffèrent, sans toutefois savoir exactement lesquels.

- *Estimation des paramètres du modèle :* Ces estimations sont obtenues avec la fonction `summary()` pour le modèle `lm(delai~traitement)`. Nous rappelons la contrainte imposée par R qui est $\alpha_1 = 0$.

```
> summary(modele)
Call:
lm(formula = delai ~ traitement)
Residuals:
    Min      1Q  Median      3Q     Max
-3.1667 -0.8750 -0.0833  0.8333  2.8333
Coefficients:
             Estimate Std. Error t value          Pr(>|t|)
(Intercept)    7.5000     0.6245  12.010 0.00000000000706 ***
traitementT2  -2.5000     0.8832  -2.831           0.00903 **
traitementT3  -3.1667     0.8832  -3.586           0.00142 **
traitementT4  -2.3333     0.8832  -2.642           0.01401 *
traitementT5  -1.3333     0.8832  -1.510           0.14366
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 1.53 on 25 degrees of freedom
Multiple R-squared:  0.384,        Adjusted R-squared:  0.2854
F-statistic: 3.896 on 4 and 25 DF,  p-value: 0.01359
```

Ainsi l'*intercept* correspond à l'estimation du délai moyen du placebo (traitement 1 pris comme référence). L'estimation associée à la variable `T2` correspond à l'effet différentiel entre le traitement `T2` et le placebo. Il en va de même pour les autres variables. Les assertions d'intérêt des tests

bilatéraux de Student effectués dans ce modèle (avec la contrainte $\alpha_1 = 0$) sont résumées dans le tableau suivant :

	$\mathcal{H}_1$
Intercept	$\mu_1 \neq 0$
traitement `T2`	$\alpha_2 \neq 0 \Leftrightarrow \mu_1 \neq \mu_2$
traitement `T3`	$\alpha_3 \neq 0 \Leftrightarrow \mu_1 \neq \mu_3$
traitement `T4`	$\alpha_4 \neq 0 \Leftrightarrow \mu_1 \neq \mu_4$
traitement `T5`	$\alpha_5 \neq 0 \Leftrightarrow \mu_1 \neq \mu_5$

Les résultats fournis par R nous indiquent qu'il existe une différence significative entre le placebo et les traitements 2, 3 et 4. Dans ce contexte, il était naturel de prendre comme référence le placebo. Il est toutefois possible de choisir une autre référence ou encore une autre contrainte linéaire au moyen de l'instruction `C()` comme le montre l'exemple suivant :

```
> summary(lm(delai~C(traitement,base=2)))
Call:
lm(formula = delai ~ C(traitement, base = 2))
Residuals:
    Min      1Q  Median      3Q     Max
-3.1667 -0.8750 -0.0833  0.8333  2.8333
Coefficients:
                          Estimate Std. Error t value
(Intercept)                 5.0000     0.6245   8.006
C(traitement, base = 2)1    2.5000     0.8832   2.831
C(traitement, base = 2)3   -0.6667     0.8832  -0.755
C(traitement, base = 2)4    0.1667     0.8832   0.189
C(traitement, base = 2)5    1.1667     0.8832   1.321
                             Pr(>|t|)
(Intercept)                0.0000000232 ***
C(traitement, base = 2)1       0.00903 **
C(traitement, base = 2)3       0.45739
C(traitement, base = 2)4       0.85184
C(traitement, base = 2)5       0.19847
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 1.53 on 25 degrees of freedom
Multiple R-squared:  0.384,         Adjusted R-squared:  0.2854
F-statistic: 3.896 on 4 and 25 DF,  p-value: 0.01359
```

La statistique de test de Fisher ne change pas puisqu'il ne dépend pas de la contrainte linéaire. En revanche, les estimations et les tests individuels de Student changent. Au vu de ces résultats, on ne montre pas que le traitement 2 diffère des traitements 3, 4 et 5, mais on retrouve le test de Student significatif pour la comparaison du placebo *versus* le traitement 2.

Remarque

Pour obtenir la contrainte $\sum_{i=1}^{I} \alpha_i = 0$, il faut utiliser `C(traitement, sum)`.

Notez que vous pouvez utiliser la fonction `model.matrix()` sur le modèle linéaire ajusté pour obtenir la matrice des variables explicatives. Dans le modèle d'ANOVA avec une contrainte du type $\alpha_i = 0$, la matrice contient un *intercept* (une colonne de 1) et les $I - 1$ variables indicatrices.

13.1.4 Validation des hypothèses

- *Validation des hypothèses :* Le modèle d'ANOVA correspond à un modèle linéaire avec variable explicative qualitative. Les hypothèses du modèle peuvent être validées par la méthode de l'analyse des résidus présentée dans le modèle de régression. Nous rappelons que le graphique des résidus s'obtient au moyen des instructions :

```
> par(mfrow=c(2,2))
> plot(modele,col.smooth="red")
```

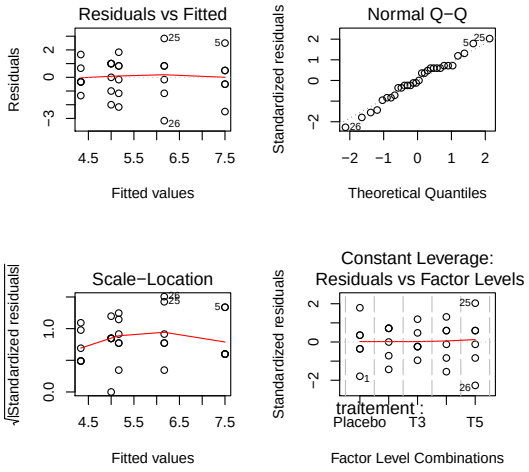


FIGURE 13.2 – Analyser les résidus dans une ANOVA à un facteur.

En outre, dans une ANOVA, il est aussi possible d'utiliser un test d'égalité des variances pour explorer si l'hypothèse d'homoscédasticité est non admissible. Le test de Bartlett (sous condition de normalité dans les sous-populations) est obtenu par

```
> bartlett.test(delai~traitement)
```

```
        Bartlett test of homogeneity of variances
data:  delai by traitement
Bartlett's K-squared = 2.4197, df = 4, p-value = 0.6591
```

Cependant, ce test n'est pas robuste à la non-normalité, un cas où l'on préférera utiliser le test de Levene [32] :

```
> levene.test(delai,traitement)  # Disponible dans le
                                 # package car.
Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group  4  0.5851 0.6763
      25
```

13.1.5 Comparaisons multiples et contrastes

- *Comparaisons multiples :* Si, après avoir effectué l'analyse de variance, on rejette l'hypothèse d'égalité des moyennes relatives à un facteur à I niveaux, une question intéressante est de savoir quelles sont les moyennes qui diffèrent significativement des autres. Dans l'exemple des boutons de fièvre, on aimerait sélectionner le traitement le plus efficace, à savoir celui qui permet d'obtenir une cicatrisation la plus rapide.
  Le test individuel de Student dans le modèle linéaire est parfaitement valide pour comparer deux traitements choisis ***a priori***. En revanche, il n'est plus du tout utilisable pour comparer par exemple le traitement qui donne en apparence les résultats les meilleurs avec celui qui donne en apparence les résultats les plus mauvais. En effet, cela revient à comparer tous les traitements entre eux. Chaque test a alors une probabilité α (le niveau du test) de déclarer présente une différence qui n'existe pas. Au total, sur les $I(I-1)/2$ comparaisons possibles, la probabilité d'en déclarer une significative « par hasard » devient importante. Pour contrôler un risque global sur les $I(I-1)/2$ comparaisons deux à deux, il existe diverses méthodes.
  La fonction `pairwise.t.test()` permet d'effectuer toutes les comparaisons deux à deux en proposant plusieurs méthodes de correction du risque α afin de tenir compte du problème de la multiplicité des tests.

```
> pairwise.t.test(delai,traitement,p.adjust="bonf")
        Pairwise comparisons using t tests with pooled SD
data:  delai and traitement
   Placebo T2    T3    T4
T2 0.090   -     -     -
T3 0.014   1.000 -     -
T4 0.140   1.000 1.000 -
T5 1.000   1.000 0.483 1.000
P value adjustment method: bonferroni
```

Le logiciel R donne les valeurs-p ajustées suivant la correction de Bonferroni, c'est-à-dire que les valeurs-p corrigées sont obtenues en multipliant les valeurs-p des tests de Student par le nombre de tests effectués. Au vu de ces résultats (valeur-$p = 0.014$ entre le traitement 1 et le traitement 3), il existe une différence significative entre le traitement 1 (placebo) et le traitement 3 au risque 5 %.

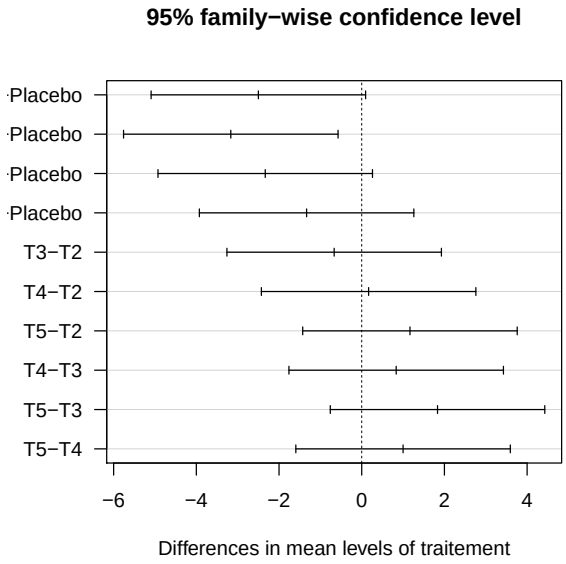
Remarque

La comparaison entre le traitement 1 et le traitement 3 avait été effectuée en analysant le modèle 1. La valeur-p du test individuel de Student était de 0.001 4. Comme dix comparaisons ont été réalisées, cette valeur-p est multipliée par 10 par la méthode de Bonferroni.

Beaucoup d'autres méthodes de correction sont possibles. Cependant, dans le cas d'une analyse de la variance à 1 facteur avec le même nombre d'observations par groupe, la méthode de Tukey [37] est la plus précise. Elle fournit des intervalles de confiance simultanés pour les différences entre paramètres $\mu_i - \mu_j$ où $1 \leq i < j \leq I$.

```
> mon.aov <- aov(delai~traitement)
> TukeyHSD(mon.aov)
  Tukey multiple comparisons of means
    95% family-wise confidence level
Fit: aov(formula = delai ~ traitement)
$traitement
                  diff        lwr         upr     p adj
T2-Placebo -2.5000000 -5.0937744  0.09377442 0.0627671
T3-Placebo -3.1666667 -5.7604411 -0.57289224 0.0113209
T4-Placebo -2.3333333 -4.9271078  0.26044109 0.0927171
T5-Placebo -1.3333333 -3.9271078  1.26044109 0.5660002
T3-T2      -0.6666667 -3.2604411  1.92710776 0.9410027
T4-T2       0.1666667 -2.4271078  2.76044109 0.9996956
T5-T2       1.1666667 -1.4271078  3.76044109 0.6811222
T4-T3       0.8333333 -1.7604411  3.42710776 0.8770466
T5-T3       1.8333333 -0.7604411  4.42710776 0.2614661
T5-T4       1.0000000 -1.5937744  3.59377442 0.7881333
```

```
> par(las=1) # Écriture horizontale des étiquettes.
> plot(TukeyHSD(mon.aov))
```



La méthode de Tukey va dans le même sens que les résultats obtenus avec la méthode de Bonferroni. En effet, le seul intervalle de confiance ne contenant pas la valeur 0 est celui concernant la différence entre le traitement 3 et le traitement 1. Il existe donc une différence significative entre le traitement 1 et le traitement 3. Le délai de cicatrisation étant plus court avec le traitement 3, nous proposons d'utiliser le traitement 3.

- *Méthodes des contrastes :* En ANOVA, un contraste (noté par exemple L) est défini comme une combinaison linéaire des moyennes théoriques dont la somme des coefficients est égale à zéro :

$$L = \sum_{i=1}^{I} \lambda_i \mu_i = \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{\mu} \qquad \text{avec} \qquad \sum_{i}^{I} \lambda_i = 0$$

où $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_I)^{\mathsf{T}}$ et $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_I)^{\mathsf{T}}$.
Les contrastes permettent d'effectuer des comparaisons entre les moyennes de certains groupes de niveaux. Par exemple, pour comparer dans l'exemple des boutons de fièvre les traitements 2 et 3, il convient d'utiliser le contraste $L_1 = \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{\mu}$ avec $\boldsymbol{\lambda} = (0, 1, -1, 0, 0)^{\mathsf{T}}$ et d'effectuer le test $\mathcal{H}_0 : L_1 = 0$ *versus* $\mathcal{H}_1 : L_1 \neq 0$.
Vous avez la possibilité d'effectuer des tests sur les contrastes en utilisant la fonction `fit.contrast()` disponible dans le *package* `gmodels`.

```
> require("gmodels")
> cmat <- rbind(" : 2 versus 3"=c(0,1,-1,0,0))
> fit.contrast(mon.aov,traitement,cmat)
                          Estimate Std. Error   t value
traitement : 2 versus 3 0.6666667  0.8831761 0.7548514
                          Pr(>|t|)
traitement : 2 versus 3 0.4573908
```

Remarque

Ce résultat avait été donné par le test individuel de Student dans le modèle 2.

Supposons maintenant que les traitements 2 et 3 sont à base de pommade alors que les traitements 4 et 5 sont des timbres anti-tabac. Afin de comparer ces deux types de remèdes, on peut utiliser le contraste suivant : $L_2 = \boldsymbol{\lambda}^\mathsf{T}\boldsymbol{\mu}$ avec $\boldsymbol{\lambda} = (0, -1, -1, 1, 1)^\mathsf{T}$ et effectuer le test $\mathcal{H}_0 : L_2 = 0$ *versus* $\mathcal{H}_1 : L_2 \neq 0$.

```
> cmat <- rbind(" : 2 versus 3"=c(0,1,-1,0,0),
+               " : 2 et 3 versus 4 et 5"=c(0,-1,-1,1,1))
> fit.contrast(mon.aov,traitement,cmat)
                                    Estimate Std. Error
traitement : 2 versus 3           0.6666667  0.8831761
traitement : 2 et 3 versus 4 et 5 2.0000000  1.2489996
                                    t value  Pr(>|t|)
traitement : 2 versus 3           0.7548514 0.4573908
traitement : 2 et 3 versus 4 et 5 1.6012815 0.1218767
```

13.1.6 Récapitulatif

Le tableau ci-dessous présente les principales fonctions à utiliser afin d'effectuer une analyse de variance à un facteur.

Table 13.1 – Principales fonctions à utiliser en ANOVA à un facteur.

Instruction R	Description
`plot(Y~factor(X))`	inspection graphique
`aov(Y~factor(X))`	analyse de la variance
`summary(aov(Y~factor(X)))`	tableau d'analyse de la variance
`anova(lm(Y~factor(X)))`	tableau d'analyse de la variance
`pairwise.t.test()`	comparaisons deux à deux
`fit.contrast()`	test sur les contrastes (*package* `gmodels`)
`barlett.test()`	test d'hypothèse d'homoscédasticité
`levene.test()`	test d'hypothèse d'homoscédasticité
`plot(aov(Y~factor(X)))`	analyse graphique des résidus

SECTION 13.2

Analyse de la variance à deux facteurs

13.2.1 Objectifs, données et modèle

► *Objectif :* L'ANOVA à deux facteurs (*two way ANOVA*) est une méthode visant à expliquer une variable quantitative par deux variables explicatives qualitatives (appelées facteurs) « croisées ».

Remarque

En ANOVA, les variables explicatives sont souvent appelées (notamment dans les recherches en psychologie) les variables indépendantes (*independent variables*, noté *IV*).

► *Les données :* Soit un facteur A ayant I modalités et un facteur B ayant J modalités. Pour tout couple (i, j), $i = 1, \ldots, I$ et $j = 1, \ldots, J$, on observe une variable quantitative Y_{ij}, n_{ij} fois. On suppose que Y_{ij} suit une loi normale $\mathcal{N}(\mu_{ij}, \sigma^2)$ sur chaque population formée par le croisement ij des deux facteurs.

► *Le modèle :* Il s'écrit

$$Y_{ijk} = \mu_{ij} + \epsilon_{ijk}, \quad \text{pour} \quad k = 1, \ldots, n_{ij}, \quad i = 1, \ldots, I, \quad j = 1, \ldots, J,$$

et où les erreurs ϵ_{ijk} sont variables aléatoires indépendantes de loi $\mathcal{N}(0, \sigma^2)$. Dans ce modèle, les paramètres réels $\mu_{11}, \ldots, \mu_{I1}, \ldots, \mu_{1J}, \ldots, \mu_{IJ}$ sont inconnus ainsi que la variance σ^2.
On décompose μ_{ij} de façon à faire « apparaître » les effets des facteurs A et B et de leur éventuelle interaction :

$$\mu_{ij} = \mu_{\bullet\bullet} + \alpha_i^{\mathrm{A}} + \alpha_j^{\mathrm{B}} + \beta_{ij}$$

où
- $\mu_{\bullet\bullet} = \frac{1}{IJ} \sum_{i=1}^{I} \sum_{j=1}^{J} \mu_{ij}$: effet moyen général ;
- $\mu_{i\bullet} = \frac{1}{J} \sum_{j=1}^{J} \mu_{ij}$: effet du niveau i du facteur A ;
- $\alpha_i^A = \mu_{i\bullet} - \mu_{\bullet\bullet}$: effet différentiel du niveau i du facteur A ;
- $\mu_{\bullet j} = \frac{1}{I} \sum_{i=1}^{I} \mu_{ij}$: effet du niveau j du facteur B ;
- $\alpha_j^B = \mu_{\bullet j} - \mu_{\bullet\bullet}$: effet différentiel du niveau j du facteur B ;
- $\beta_{ij} = \mu_{ij} - \mu_{\bullet\bullet} - \alpha_i^{\mathrm{A}} - \alpha_j^{\mathrm{B}}$: effet d'interaction du niveau i du facteur A et du niveau j du facteur B.

Remarque

Par construction, $\sum_{i=1}^{I} \alpha_i^{\mathrm{A}} = 0$, $\sum_{j=1}^{J} \alpha_j^{\mathrm{B}} = 0$ et $\forall i$, $\sum_{j=1}^{J} \beta_{ij} = 0$, $\forall j$, $\sum_{i=1}^{I} \beta_{ij} = 0$.

L'objectif va être ici de détecter :
- si le facteur A a un effet sur la variable quantitative Y ;
- si le facteur B a un effet sur la variable quantitative Y ;
- et s'il y a un effet d'interaction entre les facteurs A et B sur la variable quantitative Y.

13.2.2 Exemple et inspection graphique

▶ *Exemple d'application :* Le tableau ci-dessous donne le rendement de petit épeautre provenant de parcelles cultivées dans quatre régions différentes avec trois types d'engrais.

	Région I	Région II	Région III	Région IV
Engrais E_1	15 14 17	21 20 21	14 15 14	16 17 17
Engrais E_2	16 19 20	23 24 25	15 14 14	12 11 12
Engrais E_3	18 17 17	20 21 21	17 19 17	12 13 13

Ces données sont entrées dans R au moyen des instructions suivantes :

```
> rdt <- c(15,14,17,21,20,21,14,15,14,16,17,17,16,19,20,23,
+          24,25,15,14,14,12,11,12,18,17,17,20,21,21,17,19,
+          17,12,13,13)
> engrais <- gl(3,12,36,labels=paste("Engrais",1:3))
> region <- gl(4,3,36,labels=paste("Region",1:4))
> epeautre <- data.frame(rdt,engrais,region)
```

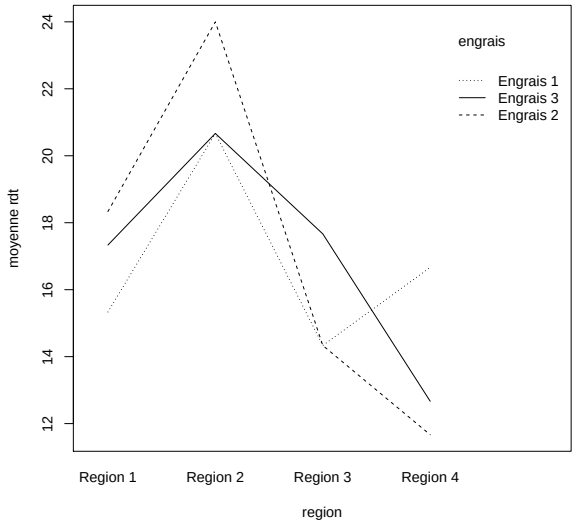
On désire étudier les effets du type d'engrais (E_1, E_2 ou E_3) sur le rendement à l'hectare du petit épeautre et savoir s'il existe une différence significative de rendement suivant ces quatre régions.

Remarque

Cet exemple présente un cas d'analyse de variance à deux facteurs équirépétés (même nombre d'observations par croisement des modalités des deux facteurs). Pour des analyses sur des données non équirépétées, le tableau d'analyse de la variance n'est plus unique et il est alors conseillé d'utiliser une décomposition de la variance de type III (voir [5]).

- *Inspection graphique :* Dans un modèle d'ANOVA à deux facteurs avec interaction, l'effet d'un facteur sur la variable explicative peut être différent suivant les modalités de l'autre facteur mis en jeu. Cette souplesse dans le modèle peut être appréciée visuellement. La commande permettant d'explorer cette interaction est `interaction.plot()`. La fonction `plotMeans()`, disponible dans le *package* `Rcmdr`, permet aussi une inspection graphique de l'interaction.

```
> interaction.plot(region,engrais,rdt,ylab="moyenne rdt")
```




```
> interaction.plot(engrais,region,rdt,ylab="moyenne rdt")
```

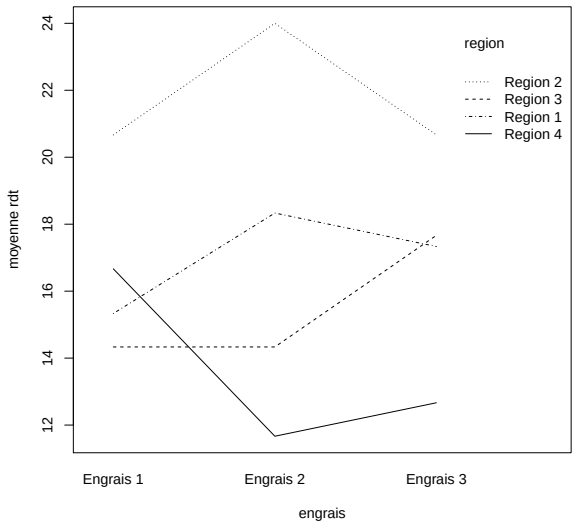


FIGURE 13.3 – Exploration de l'interaction dans une ANOVA à deux facteurs.

Au vu de ces graphiques, il semble y avoir un effet de l'interaction (courbes qui se croisent).

13.2.3 Table d'ANOVA, tests et estimation des paramètres

- *Instruction* R *pour la table d'ANOVA :* Plusieurs fonctions offrent la possibilité d'effectuer une ANOVA à deux facteurs avec interaction : `aov()`, `anova(lm())` et `Anova()` (présent dans le *package* `car`).

Attention

Lorsque vous disposez d'une seule observation par croisement des facteurs A et B (c'est-à-dire $n_{ij} = 1\ \forall i, j$), vous pouvez estimer seulement un modèle d'ANOVA à deux facteurs sans interaction.
`aov(rdt~region+engrais)`.

```
> modele2 <- summary(aov(rdt~region*engrais,data=epeautre))
> modele2
              Df Sum Sq Mean Sq F value             Pr(>F)
region         3  327.2  109.06 112.181 0.0000000000000295
engrais        2    0.9    0.44   0.457              0.638
region:engrais 6   99.6   16.59  17.067 0.0000001359421306
Residuals     24   23.3    0.97

region         ***
engrais
region:engrais ***
Residuals
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> anova(lm(rdt~region*engrais,data=epeautre))
Analysis of Variance Table
Response: rdt
              Df Sum Sq Mean Sq  F value              Pr(>F)
region         3 327.19 109.065 112.1810 0.00000000000002955
engrais        2   0.89   0.444   0.4571              0.6385
region:engrais 6  99.56  16.593  17.0667 0.00000013594213061
Residuals     24  23.33   0.972

region         ***
engrais
region:engrais ***
Residuals
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> Anova(lm(rdt~region*engrais,data=epeautre))
```

```
Anova Table (Type II tests)
Response: rdt
               Sum Sq Df  F value              Pr(>F)
region         327.19  3 112.1810 0.00000000000002955 ***
engrais          0.89  2   0.4571              0.6385
region:engrais  99.56  6  17.0667 0.00000013594213061 ***
Residuals       23.33 24
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remarque

La formule `region*engrais`, utilisée dans `aov()` et `lm()`, correspond en fait à la formule `region+engrais+region:engrais`, c'est-à-dire le facteur région, le facteur engrais et l'interaction entre ces deux facteurs.

La valeur-p associée au test de la présence d'interaction est significative. Cela entraîne par exemple que l'effet de l'engrais sur le rendement peut être différent selon les régions.

Attention

Nous considérons qu'il n'y a aucun effet d'interaction si la valeur-p associée est supérieure à 5 %. Dans ce cas, nous effectuons une analyse de la variance sans terme d'interaction, ce qui rend l'effet principal plus facile à interpréter. En présence d'interaction, il ne faut pas interpréter les tests des effets principaux qui sont présentés dans les sorties de la table d'analyse de la variance.

- *Test des effets conditionnels en présence d'interaction :* On souhaite par exemple savoir s'il existe un effet engrais dans la région 1. Pour cela, on peut utiliser la fonction `subset()` qui permet de n'utiliser que les données relatives à une région particulière.

```
> engrais.region1 <- summary(aov(rdt~engrais,subset=
+                                region=="Region 1"))
> engrais.region1
            Df Sum Sq Mean Sq F value Pr(>F)
engrais      2     14   7.000       3  0.125
Residuals    6     14   2.333
```

Attention

Le test effectué dans ce tableau d'analyse de la variance correspond à une ANOVA à 1 facteur (engrais) sur le rendement d'épeautre de la région 1. Cependant, il ne prend pas en compte l'information des données pour les autres régions qui permettent une meilleure estimation de la variance résiduelle. Pour tester l'effet de l'engrais dans la

région 1, il faut diviser les carrés moyens du facteur engrais trouvé dans l'ANOVA restreint à la région 1 par les carrés moyens résiduels de l'ANOVA du modèle avec interaction :

```
> F.engrais.region1 <- engrais.region1[[1]]$Mean[1]/
+                      modele2[[1]]$Mean[4]
> valeurp <- 1-pf(F.engrais.region1,df1=2,df2=24)
> valeurp
[1] 0.003552714
```

La valeur-p étant inférieure à 5 %, on peut conclure qu'il existe un effet de l'engrais dans la région 1.

- *Estimation des paramètres du modèle :* Ces estimations sont obtenues avec la fonction `summary()` pour le modèle `lm(rdt~region*engrais)`. Nous rappelons les contraintes imposées par **R** qui sont $\alpha_1^{\mathrm{A}} = 0$, $\alpha_1^{\mathrm{B}} = 0$, $\beta_{1j} = 0\ \forall j = 1, \ldots, J$ et $\beta_{i1} = 0\ \forall i = 1, \ldots, I$.

```
> summary(lm(rdt~region*engrais))
Call:
lm(formula = rdt ~ region * engrais)
Residuals:
    Min      1Q  Median      3Q     Max
-2.3333 -0.6667  0.1667  0.3333  1.6667
Coefficients:
                                  Estimate Std. Error t value
(Intercept)                        15.3333     0.5693  26.935
regionRegion 2                      5.3333     0.8051   6.625
regionRegion 3                     -1.0000     0.8051  -1.242
regionRegion 4                      1.3333     0.8051   1.656
engraisEngrais 2                    3.0000     0.8051   3.726
engraisEngrais 3                    2.0000     0.8051   2.484
regionRegion 2:engraisEngrais 2     0.3333     1.1386   0.293
regionRegion 3:engraisEngrais 2    -3.0000     1.1386  -2.635
regionRegion 4:engraisEngrais 2    -8.0000     1.1386  -7.026
regionRegion 2:engraisEngrais 3    -2.0000     1.1386  -1.757
regionRegion 3:engraisEngrais 3     1.3333     1.1386   1.171
regionRegion 4:engraisEngrais 3    -6.0000     1.1386  -5.270
                                    Pr(>|t|)
(Intercept)                          < 2e-16 ***
regionRegion 2                   0.000000749 ***
regionRegion 3                       0.22619
regionRegion 4                       0.11071
engraisEngrais 2                     0.00105 **
engraisEngrais 3                     0.02036 *
regionRegion 2:engraisEngrais 2      0.77221
regionRegion 3:engraisEngrais 2      0.01451 *
regionRegion 4:engraisEngrais 2  0.000000290 ***
```

```
regionRegion 2:engraisEngrais 3     0.09174 .
regionRegion 3:engraisEngrais 3     0.25306
regionRegion 4:engraisEngrais 3 0.000021016 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.986 on 24 degrees of freedom
Multiple R-squared:  0.9483,        Adjusted R-squared:  0.9245
F-statistic: 39.99 on 11 and 24 DF,  p-value: 0.000000000001009
```

Ainsi l'*intercept* correspond à l'estimation du rendement moyen pour l'engrais 1 dans la région 1. Par exemple, le coefficient associé à la région 2 (6.625) correspond à l'estimation de la différence du rendement moyen avec l'engrais 1 de la région 2 et du rendement moyen de l'engrais 1 de la région 1. Les tests de Student associés aux facteurs sont donc interprétables. En revanche, ceux associés aux coefficients estimés des facteurs croisés n'ont aucune pertinence.

Remarque

Dans cet exemple, il n'y a *a priori* pas de raison de choisir cette contrainte. Pour la changer, il suffit d'utiliser la fonction `C()`. Par exemple :
`summary(lm(rdt~C(region,sum)*C(engrais,sum)))`
correspond aux contraintes $\sum_{i=1}^{I} \alpha_i^{\mathrm{A}} = 0$, $\sum_{j=1}^{J} \alpha_j^{\mathrm{B}} = 0$ et $\forall i,\ \sum_{j=1}^{J} \beta_{ij} = 0$, $\forall j,\ \sum_{i=1}^{I} \beta_{ij} = 0$.

13.2.4 Validation des hypothèses

- *Validation des hypothèses :* Comme dans l'analyse de la variance à un facteur, nous validons le modèle par une étude des résidus du modèle linéaire sous-jacent.

```
> par(mfrow=c(2,2))
> plot(modele,col.smooth="red")
```

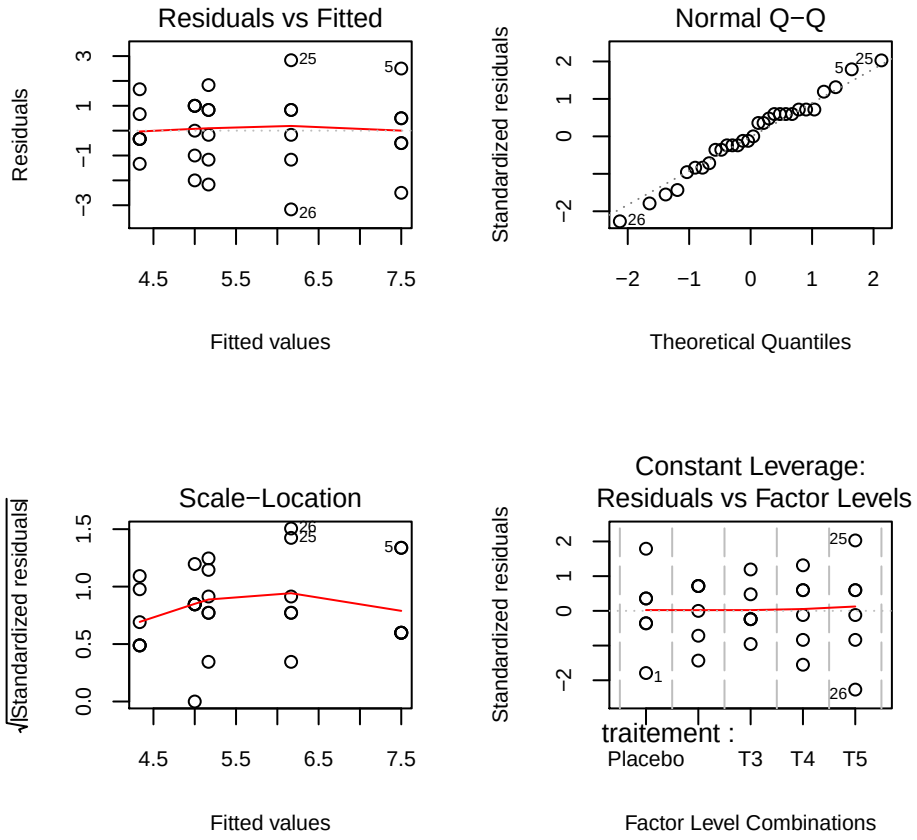


FIGURE 13.4 – Analyser les résidus dans une ANOVA à deux facteurs.

Cependant, si le nombre de données par croisement des deux facteurs est assez important, il sera préférable d'inspecter la normalité dans chaque sous-population ainsi que l'hypothèse d'homoscédasticité.

13.2.5 Contrastes

- *Méthode des contrastes :* Nous renvoyons le lecteur à la définition des contrastes présentée à l'analyse de la variance à un facteur.
  Par exemple, supposons que l'on souhaite savoir s'il existe une différence

significative des rendements dans la région 1 entre l'engrais 1 et 2. Il suffit alors d'effectuer un test fondé sur les contrastes grâce à la fonction `estimable()` disponible dans le *package* `gmodels`.

```
> mod.inter <- lm(rdt ~ engrais:region-1)
> cm <- rbind("E1 vs E2 dans R1"=
+                   c(1,-1,0,0,0,0,0,0,0,0,0,0))
> estimable(mod.inter,cm)
                Estimate Std. Error   t value DF    Pr(>|t|)
E1 vs E2 dans R1       -3  0.8050765 -3.726354 24 0.001048837
```

On retrouve la valeur-p du test effectué lors de *l'estimation des paramètres du modèle*.
Un autre exemple d'utilisation de la méthode des contrastes est la comparaison du rendement par l'engrais 1 pour des régions du Sud (régions 1 et 2) par rapport aux régions du Nord (régions 3 et 4) :

```
> cm <- rbind("E1 vs E2 dans R1"=
+             c(1,-1,0,0,0,0,0,0,0,0,0,0),
+     "R1 & R2 vs R3 & R4 pour E1" =
+      c(1,0,0,1,0,0,-1,0,0,-1,0,0))
> estimable(mod.inter,cm)
                           Estimate Std. Error   t value DF
E1 vs E2 dans R1                 -3  0.8050765 -3.726354 24
R1 & R2 vs R3 & R4 pour E1        5  1.1385501  4.391550 24
                              Pr(>|t|)
E1 vs E2 dans R1            0.0010488374
R1 & R2 vs R3 & R4 pour E1  0.0001951599
```

13.2.6 Récapitulatif

Le tableau ci-dessous présente les principales fonctions à utiliser afin d'effectuer une analyse de variance à deux facteurs.

TABLE 13.2 – Principales fonctions pour une ANOVA à deux facteurs.

Instruction R	Description
`interaction.plot(Y,factor(X),factor(Z))`	inspection graphique
`aov(Y~factor(X)*factor(Z))`	analyse de la variance à deux facteurs avec interaction
`summary(aov(Y~factor(X)*factor(Z)))`	tableau d'analyse de la variance
`anova(lm(Y~factor(X)*factor(Z)))`	tableau d'analyse de la variance
`Anova(lm(Y~factor(X)*factor(Z)))`	tableau d'analyse de la variance (*package* `car`)

Attention

Pour une analyse de la variance à deux facteurs non équirépétée, il est conseillé d'utiliser une décomposition des sommes de carrés du type III (voir [5]).

```
> model.lm <- lm(rdt~region*engrais,contrasts=list(region=
+                  contr.sum,engrais=contr.sum))
> Anova(model.lm,type="III")
Anova Table (Type III tests)
Response: rdt
                Sum Sq Df    F value              Pr(>F)
(Intercept)    10370.0  1 10666.3143           < 2.2e-16 ***
region           327.2  3   112.1810 0.00000000000002955 ***
engrais            0.9  2     0.4571               0.6385
region:engrais    99.6  6    17.0667 0.00000013594213061 ***
Residuals         23.3 24
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Il faut veiller à bien préciser l'option `contrasts` dans la fonction `lm()`.

SECTION 13.3

Analyses de variance à mesures répétées

Cette section est une brève introduction aux modèles d'ANOVA à mesures répétées. Nous commençons par rappeler quelques éléments de terminologie permettant de mieux comprendre les trois modèles présentés ci-après.

On parle de modèle à effets fixes lorsque les variables explicatives (facteurs) sont toutes traitées comme étant non aléatoires (contrôlées par exemple). On parle de modèle à effets aléatoires ou de modèle mixte lorsque certaines, voire toutes les variables explicatives sont supposées aléatoires.

En ANOVA, une unité statistique est appelée un « sujet ». Quand une variable dépendante est mesurée sur des groupes de sujets indépendants, où chaque groupe est exposé à une condition différente, l'ensemble des conditions est appelé un facteur inter-sujets (*between-subjects factor*). Les modèles présentés aux sections 13.1 et 13.2 impliquent uniquement des facteurs de ce type.

Dans le cas où chacun des « sujets » possède une mesure de la variable dépendante pour toutes les modalités du facteur, alors ce facteur est appelé facteur intra-sujets (*within-subjects factor*). On parlera ainsi de plan d'expérience intra-sujets (*within-subjects design*) lorsqu'au moins l'un des facteurs est un facteur intra-sujets. Ceux-ci sont aussi appelés des plans d'expérience à mesures répétées (*repeated-measures designs*) puisque les facteurs intra-sujets impliquent toujours des mesures répétées sur chaque sujet. Par construction, cela revient à considérer un facteur aléatoire supplémentaire qui est le facteur sujet.

Quand une analyse implique à la fois des facteurs intra-sujets et des facteurs inter-sujets, on parlera d'ANOVA à mesures répétées avec facteurs inter-sujets (*repeated measures ANOVA with between-subjects factors*) ou encore d'ANOVA à plan d'expérience mixte (*mixed-design ANOVA* ou *split-plot ANOVA*).

13.3.1 Modèle à un facteur à mesures répétées

- *Objectif :* On considère le cas où l'on mesure pour chacun des n sujets s la variable réponse (dépendante) Y pour chacune des I modalités du facteur à effet fixe X.

- *Exemple support :* Pendant 1/4 d'heure on compte le nombre d'actions exercées par chacun des sept rats sur un levier, et cela dans trois conditions de renforcement. Dans la première condition, on présente au rat des aliments très appréciés, dans la deuxième on présente au rat des aliments moyennement appréciés, et dans la troisième on présente au rat des aliments peu appréciés. On obtient les résultats donnés dans le tableau ci-dessous :

	Facteur : condition		
sujet	cond_1	cond_2	cond_3
s_1	8	6	2
s_2	6	5	1
s_3	7	5	0
s_4	9	3	3
s_5	5	4	1
s_6	7	5	2
s_7	6	2	0

- *Le modèle :* Le modèle sous-jacent est un modèle mixte :

$$Y_{si} = \mu_i + \pi_s + \epsilon_{si}, \quad s = 1, \ldots, n,\ i = 1, \ldots, I$$

où μ_i mesure l'effet fixe de la modalité i du facteur X, les π_s sont des variables aléatoires indépendantes de loi $\mathcal{N}(0, \sigma_\pi^2)$ permettant de tenir compte de la dépendance entre les mesures faites sur l'individu s, les ϵ_{si} sont des variables aléatoires indépendantes de loi $\mathcal{N}(0, \sigma^2)$. On suppose en outre que les π_s et les ϵ_{si} sont indépendantes. Par construction, ce modèle à un facteur à mesures répétées est en fait un modèle à deux facteurs : l'un fixe intra-sujets (X) et l'autre aléatoire qui est le facteur sujet.

- *Instructions* R :

```
summary(aov(Y ~ X + Error=sujet/X,data=mon.data.frame))
```

où `mon.data.frame` est un *data.frame* comprenant les variables `Y` et `X`, ainsi qu'une variable `sujet` indiquant le numéro d'identification du sujet. Les variables `X` et `sujet` doivent **impérativement** être définies comme des facteurs.

Remarque

Il est aussi possible d'utiliser la fonction `lme()` du *package* `nlme` :
`anova(lme(Y ~ X,random=~1|sujet,data=mon.data.frame))`

- *Retour à l'exemple support :*

```
> rat <- data.frame(levier=c(8,6,2,6,5,1,7,5,0,9,3,3,5,4,1,
+            7,5,2,6,2,0),sujet=gl(7,3,21),cond=gl(3,1,21))
> summary(aov(levier~cond+Error(sujet/cond), data=rat))
Error: sujet
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  6   15.9   2.651
Error: sujet:cond
          Df Sum Sq Mean Sq F value     Pr(>F)
cond       2 108.86   54.43    47.3 0.00000204 ***
Residuals 12  13.81    1.15
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

13.3.2 Modèle à deux facteurs à mesures répétées sur les deux facteurs

- *Objectif :* On considère le cas où l'on mesure pour chacun des n sujets s la variable réponse Y pour chacun des $I \times J$ croisements des modalités de deux facteurs à effets fixes A et B.

- *Exemple support :* Un expérimentateur veut étudier l'effet de la consommation de lécithine sur les troubles de mémoire. Il choisit quatre sujets auxquels il administre un traitement quotidien. Au bout d'un mois, de deux mois et de six mois de traitement, il fait passer à chaque sujet deux tests (test 1 et test 2). On obtient les résultats présentés dans le tableau ci-dessous :

	Test 1			Test 2		
sujet	M_1	M_2	M_6	M_1	M_2	M_6
s_1	10	11	9	3	6	3
s_2	18	20	17	16	20	14
s_3	6	8	8	5	6	3
s_4	4	9	9	10	10	6

► *Le modèle :* Le modèle sous-jacent est un modèle mixte :

$$Y_{sij} = \mu_{\bullet\bullet} + \alpha_i^A + \alpha_j^B + \beta_{ij} + \pi_s + \pi_{si}^A + \pi_{sj}^B + \epsilon_{sij}$$

avec $s = 1, \ldots, n$, $i = 1, \ldots, I$, $j = 1, \ldots, J$ et où les termes $\mu_{\bullet\bullet}$, α_i^A, α_j^B et β_{ij} ont été définis à la section 13.2. L'effet aléatoire sujet est représenté par les variables aléatoires *i.i.d.* π_s de loi $\mathcal{N}(0, \sigma_\pi^2)$. Les variables aléatoires *i.i.d.* $\pi_{si}^A \sim \mathcal{N}(0, \sigma_{\pi^A}^2)$ mesurent les effets aléatoires d'interaction entre le facteur sujet et le facteur fixe A. Les variables aléatoires *i.i.d.* $\pi_{sj}^B \sim \mathcal{N}(0, \sigma_{\pi^B}^2)$ mesurent les effets aléatoires d'interaction entre le facteur sujet et le facteur fixe B. Les erreurs ϵ_{sij} sont des variables *i.i.d.* de loi $\mathcal{N}(0, \sigma^2)$. En outre, on suppose que les erreurs sont indépendantes des π_s, π_{si}^A et π_{sj}^B.

► *Instructions* R *:*

```
summary(aov(Y ~ A*B + Error(sujet/(A\verb*B)),
                                data = mon.data.frame))
```

► *Retour à l'exemple support*

```
> lecithine <- data.frame(memoire=c(10,11,9,3,6,3,18,20,17,
+                         16,20,14,6,8,8,5,6,3,4,9,9,10,10,6),
+            sujet=gl(4,6,24),test=gl(2,3,24),mois=gl(3,1,24))
> summary(aov(memoire~mois*test+Error(sujet/(test*mois)),
+              data=lecithine))
Error: sujet
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  3  508.1   169.4
Error: sujet:test
          Df Sum Sq Mean Sq F value Pr(>F)
test       1  30.37   30.38   2.216  0.233
Residuals  3  41.12   13.71
Error: sujet:mois
          Df Sum Sq Mean Sq F value  Pr(>F)
mois       2  32.25  16.125   16.83 0.00346 **
Residuals  6   5.75   0.958
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Error: sujet:test:mois
          Df Sum Sq Mean Sq F value Pr(>F)
mois:test  2  12.25   6.125   2.333  0.178
Residuals  6  15.75   2.625
```

13.3.3 Modèle à deux facteurs à mesures répétées sur un seul facteur

- *Objectif :* On considère le cas où les sujets sont répartis dans des groupes définis par les I modalités du facteur fixe A. Pour chaque sujet, on mesure la variable réponse à toutes les J modalités du facteur fixe B.

- *Exemple support :* Dans une expérience, les sujets doivent estimer la longueur d'une barre métallique. Les barres présentées ont trois longueurs différentes. On forme deux groupes de quatre sujets distincts. Dans chaque groupe, on présente à chaque sujet trois barres de longueurs différentes.

	G1		
Sujet	L1	L2	L3
1	10	11	9
2	18	20	17
3	6	8	8
4	4	9	9

	G2		
Sujet	L1	L2	L3
1	3	6	3
2	16	20	14
3	5	6	3
4	10	10	6

- *Le modèle :*

$$Y_{s(i)j} = \mu_{\bullet\bullet} + \alpha_i^A + \alpha_j^B + \beta_{ij} + \pi_{s(i)}^A + \epsilon_{s(i)j}, \quad s = 1, \ldots, n,\ i = 1, \ldots, I,\ j = 1, \ldots, J$$

où les $\pi_{s(i)}$ sont des variables aléatoires *i.i.d.* de loi $\mathcal{N}(0, \sigma_\pi^2)$ qui mesurent les effets aléatoires des modalités s du facteur sujet. Les erreurs $\epsilon_{s(i)j}$ sont des variables aléatoires *i.i.d.* de loi $\mathcal{N}(0, \sigma^2)$. En outre, on suppose que les $\epsilon_{s(i)j}$ sont indépendantes des $\pi_{s(i)}$. Les termes $\mu_{\bullet\bullet}$, α_i^A, α_j^B , β_{ij} ont été définis à la section 13.2. Afin que le modèle soit identifiable, il faut imposer les contraintes $\sum_{i=1}^I \alpha_i^A = 0$, $\sum_{j=1}^J \alpha_j^B = 0$, $\forall j, \sum_{i=1}^I \beta_{ij} = 0$, $\forall i, \sum_{j=1}^J \beta_{ij} = 0$. Par ailleurs, $y_{s(i)j}$, réalisation de la variable aléatoire $Y_{s(i)j}$, représente l'observation du s-ième sujet du i-ième groupe du facteur A, pour le niveau j du facteur B. La notation $s(i)$ permet de souligner le fait que le facteur sujet est « emboîté » (*nested*) dans le facteur A.

- *Instructions* R *:*

```
summary(aov(Y~A*B + Error(sujet %in% A),data=mon.data.frame))
```

ou de façon équivalente :

```
summary(aov(Y ~ A*B + Error(sujet:A),data=mon.data.frame))
```

- *Retour à l'exemple support :*

```
> barre.estim <-  data.frame(barre=c(10,11,9,3,6,3,18,20,17,
+                16,20,14,6,8,8,5,6,3,4,9,9,10,10,6),
+                 sujet=gl(4,6,24),
```

```
+                 groupe=gl(2,3,24), # Facteur A.
+                 long=gl(3,1,24)    # Facteur B.
+ )
> summary(aov(barre ~ groupe*long +
+             Error(sujet %in% groupe),data=barre.estim))
Error: sujet:groupe
          Df Sum Sq Mean Sq F value Pr(>F)
groupe     1   30.4   30.37   0.332  0.586
Residuals  6  549.3   91.54
Error: Within
            Df Sum Sq Mean Sq F value Pr(>F)
long         2  32.25  16.125   9.000 0.0041 **
groupe:long  2  12.25   6.125   3.419 0.0668 .
Residuals   12  21.50   1.792
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Termes à retenir

`aov()` : permet d'effectuer une ANOVA
`anova(lm())`, `Anova(lm())` : table d'analyse de la variance
`factor()`, `as.factor()` : déclaration d'une variable en facteur
`C()` : permet de spécifier la contrainte dans une ANOVA
`barlett.test()`, `levene.test()` : tests d'égalité des variances
`pairwise.t.test()` : comparaisons deux à deux
`fit.contrast()` : test sur les contrastes (*package* `gmodels`)
`estimable()` : test sur les contrastes (*package* `gmodels`)
`interaction.plot()` : inspection graphique pour l'interaction
`Error()` : constituant d'une formule permettant de spécifier l'emboîtement du facteur sujet

Exercices

13.1- Donnez l'instruction permettant de réaliser le modèle d'ANOVA à un facteur (noté `A`).

13.2- Donnez l'instruction permettant de réaliser le modèle d'ANOVA à deux facteurs (notés `A` et `B`) sans interaction.

13.3- Donnez l'instruction permettant de réaliser le modèle d'ANOVA à deux facteurs (notés `A` et `B`) avec interaction.

13.4- Quels sont les tests permettant de vérifier la non-homoscédasticité dans un modèle d'ANOVA ?

13.5- Quelle est l'instruction pour faire des tests deux à deux à la suite d'une analyse de la variance à un facteur ?

13.6- Quelle fonction permet de récupérer les estimations du modèle d'ANOVA à un facteur ?

13.7- Quelle fonction permet de choisir le type de contrainte dans une ANOVA ?

Fiche de TP

A- Étude sur l'ANOVA à un facteur

• Étude sur le niveau sonore

Pour étudier l'influence du facteur « intensité du bruit environnant » sur la capacité d'un sujet à résoudre un problème, l'expérimentateur construit l'expérience suivante : vingt-quatre écoliers sont répartis de façon aléatoire dans

quatre pièces. Des bruits de la rue ont été enregistrés et sont diffusés dans chaque pièce avec un niveau sonore particulier. Les enfants doivent résoudre une série de problèmes. La variable réponse est la note finale obtenue à la série d'épreuves. Les résultats obtenus sont présentés dans le tableau suivant :

Niveau sonore			
1	2	3	4
62	56	63	68
60	62	67	66
63	60	71	71
59	61	64	67
63	63	65	68
59	64	66	68

On souhaite savoir s'il existe un effet du facteur « intensité du bruit environnant » sur la capacité d'un sujet à résoudre un problème.

13.1- Saisissez ce jeu de données dans une structure adéquate en vue d'une analyse de la variance.

13.2- Écrivez le modèle d'analyse de la variance permettant de répondre à la question posée.

13.3- Effectuez l'analyse correspondant à votre modèle.

13.4- Effectuez toutes les comparaisons deux à deux des niveaux sonores en tenant compte du problème posé par la multiplicité des tests.

• Étude sur l'intima-média

Dans l'étude « Intima-média », on s'intéresse à la relation entre la mesure de l'épaisseur de l'intima-média et la consommation d'alcool.

13.1- Récupérez le fichier de données sur l'intima-média.

13.2- Proposez un graphique permettant de visualiser des différences de mesure de l'épaisseur de l'intima-média suivant la consommation d'alcool.

13.3- Existe-t-il une différence de mesure moyenne de l'épaisseur de l'intima-média suivant la consommation d'alcool ?

13.4- Faites une étude sur les résidus afin de valider les hypothèses de votre étude statistique.

• Étude sur les sportifs

Dans une étude du trouble de la conduite à risque chez les jeunes sportifs, un chercheur a observé les comportements délictueux (vol, racket, bagarres, etc.) de jeunes âgés de 14 à 25 ans selon la durée hebdomadaire de pratique sportive. Ce comportement délictueux est mesuré sur une échelle de gravité de 0 à 100. Voici un extrait du jeu de données que vous devez récupérer à l'adresse `http://www.biostatisticien.eu/springeR/sportif.RData` :

```
> print(sportif[sample(1:105,10),],row.names=FALSE)
 score temps
     9 [2;3[
    75 [0;1[
     0 [3;4[
    21 [2;3[
    67 [4;5[
    69 [1;2[
    36 [2;3[
     0 [3;4[
    87 [0;1[
    16 [2;3[
```

13.1- Décrivez les facteurs mis en jeu et écrivez le modèle (et les hypothèses sous-jacentes).

13.2- Effectuez un test au seuil de 5 % pour décider s'il existe un effet significatif de la durée de la pratique du sport sur les troubles de conduite à risque.

On désigne par « peu sportifs » l'ensemble des jeunes pratiquant moins de 2 heures de sport par semaine, par « moyennement sportifs » l'ensemble des jeunes dont la pratique sportive hebdomadaire est comprise dans l'intervalle [2,4[et par « très sportifs » l'ensemble des jeunes pratiquant au moins 4 heures de sport par semaine.
Le chercheur émet les hypothèses de recherche suivantes :

- hypothèse de recherche 1 : les « peu sportifs » ont une conduite à risque plus importante que les « moyennement sportifs » ;
- hypothèse de recherche 2 : la conduite à risque des « très sportifs » est sensiblement différente de celle des « peu sportifs ».

13.3- Traduisez les hypothèses de recherche en contrastes.

13.4- Testez, au seuil de 5 %, ces deux hypothèses de recherche.

B- Étude sur l'ANOVA à deux facteurs

• Étude sur les piles

Dans le cadre d'une expérience sur la durée de vie des piles, le but est de déterminer la durée de vie en fonction du type de pile. Comme on sait que les piles ont une durée de vie qui dépend de la température d'utilisation, un plan à deux facteurs (type de pile et température d'utilisation) est créé. Les durées de vie en minutes suivant le croisement de ces facteurs sont présentées ci-après :

	15 °C		70 °C		125 °C	
Type I	130	155	34	40	20	70
	74	180	80	75	82	58
Type II	150	188	136	122	25	70
	159	126	106	115	58	45
Type III	138	110	174	120	96	104
	168	160	150	139	82	60

13.1- Quels sont les facteurs mis en jeu dans cette expérience ? Quelles sont leurs modalités ? Quelle est la variable à expliquer ?

13.2- Proposez et définissez un modèle d'analyse de variance pour ce jeu de données.

13.3- Effectuez une représentation graphique permettant de mettre en évidence une éventuelle interaction dans le modèle.

13.4- Estimez les différents paramètres du modèle.

13.5- Dressez le tableau d'analyse de variance pour le modèle proposé.

13.6- Effectuez les tests adéquats afin de finaliser cette analyse.

• Rendement laitier

On s'intéresse à l'influence du type et de la quantité d'alimentation sur le rendement laitier. On a relevé les quarante observations suivantes :

	Paille					Foin					Herbe					Aliments ensilés				
Dose faible	8	11	11	10	7	12	13	14	11	10	10	12	12	13	14	17	13	17	14	13
Dose forte	8	9	8	10	9	10	7	10	12	11	11	9	11	11	12	13	12	11	15	14

13.1- Proposez et définissez un modèle d'analyse de variance pour étudier l'influence sur le rendement laitier du type d'alimentation et de la dose.

13.2- Effectuez une représentation graphique permettant de mettre en évidence une éventuelle interaction dans le modèle.

13.3- Estimez les différents paramètres du modèle.

13.4- Dressez le tableau d'analyse de variance pour le modèle proposé.

13.5- Effectuez les tests adéquats afin de finaliser cette analyse.

• Étude sur l'intima-média

Dans l'étude « Intima-média », on s'est intéressé (au TP A) à la relation pouvant exister entre la mesure de l'épaisseur de l'intima-média et la consommation d'alcool. On se demande maintenant si la mesure de l'épaisseur de l'intima-média est liée à la consommation d'alcool ainsi qu'à la consommation de tabac.

13.1- Récupérez le fichier de données sur l'intima-média.

13.2- Proposez et définissez un modèle d'analyse de variance pour étudier l'influence de la consommation de tabac et d'alcool sur la mesure de l'épaisseur de l'intima-média.

13.3- Effectuez une représentation graphique permettant de mettre en évidence une éventuelle interaction dans le modèle.

13.4- Existe-t-il une différence de mesure moyenne de l'épaisseur de l'intima-média suivant la consommation d'alcool ? suivant la consommation de tabac ?

Annexes : Installation du logiciel R et des *packages* R

Pré-requis et objectif

- Aucun pré-requis n'est nécessaire. La lecture du chapitre 1 pourrait toutefois se révéler intéressante.
- Ce chapitre décrit comment installer le logiciel R, dans sa version x (remplacer partout dans le reste de ce document x par le numéro de la dernière version disponible), sous le système d'exploitation Microsoft Windows et aussi comment ajouter des *packages* supplémentaires sous Windows ou sous Linux.

SECTION C.1

Installation de R sous Microsoft Windows

Commencez par télécharger le logiciel R (fichier R-x-win.exe où x est le numéro de la dernière version disponible) à l'aide de votre navigateur web usuel à l'adresse suivante : `http://cran.r-project.org/bin/windows/base/`
Enregistrez ensuite ce fichier exécutable sur le bureau de Windows puis double-cliquez sur le fichier R-x-win.exe dont voici l'icône :  .

Le logiciel s'installe alors et vous n'avez plus qu'à suivre les instructions qui s'affichent et à conserver les options proposées par défaut.

Lorsque l'icône  est ajoutée sur le bureau, l'installation peut être considérée comme terminée.

SECTION C.2

Installation de *packages* supplémentaires

De nombreux modules (*packages* ou librairies) supplémentaires sont disponibles sur le site internet :
http://cran.r-project.org/web/packages/available_packages_by_name.html, ou bien encore ici :
http://cran.r-project.org/bin/windows/contrib/, dans le dossier correspondant au numéro x de votre version de R.

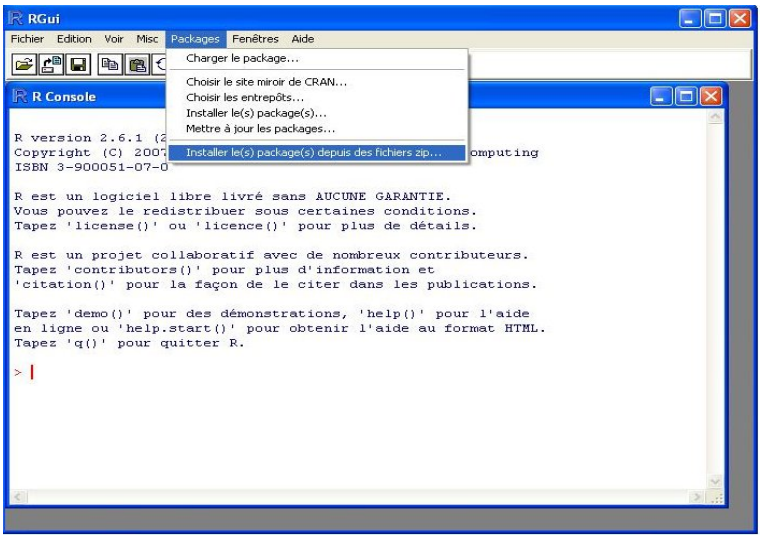
Ils étendent les fonctionnalités de R. Il existe plusieurs moyens pour installer un nouveau *package*, que nous présentons ci-dessous.

C.2.1 Installation à partir d'un fichier situé sur le disque

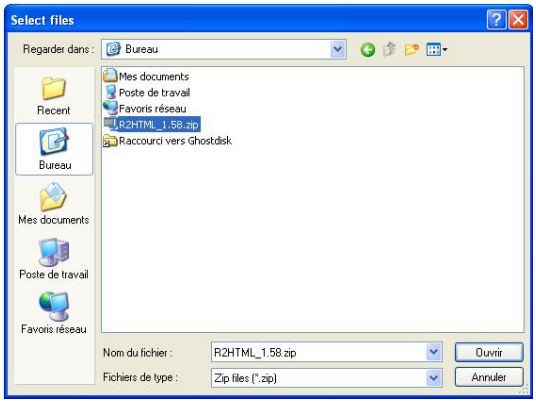
Vous pouvez par exemple télécharger depuis le site mentionné ci-dessus le fichier : R2HTML_*numero*.zip et l'enregistrer sur le bureau de Windows.

Pour installer ce *package*, commencez par lancer le logiciel R en double-cliquant sur son icône  .

Ensuite, allez dans le menu `Packages`, puis dans le sous-menu `Installer le(s) package(s) depuis des fichiers zip...`



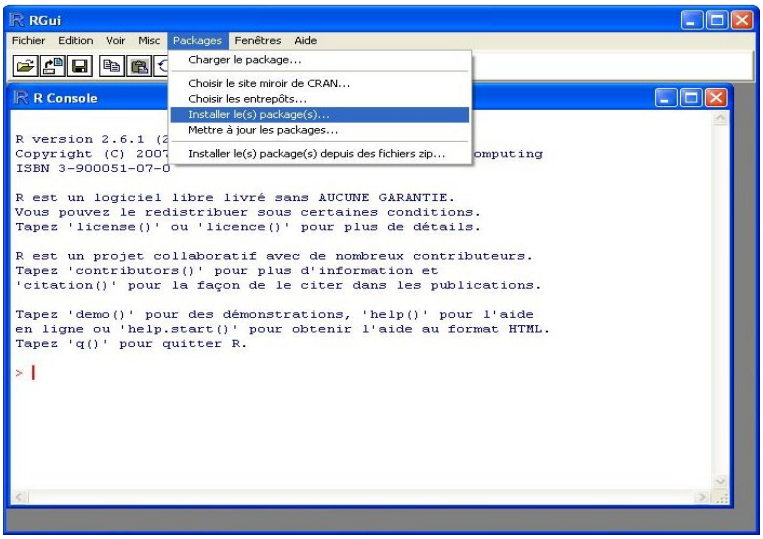
Sélectionnez alors le fichier R2HTML_*numero*.zip situé sur le bureau de Windows, puis cliquez sur « Ouvrir ».



C.2.2 Installation directement depuis l'Internet

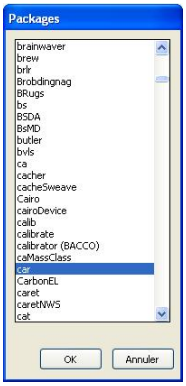
Pour installer, par exemple, les *packages* `car` et `Rcmdr`, commencez par lancer le logiciel **R** en double-cliquant sur son icône située sur le bureau.

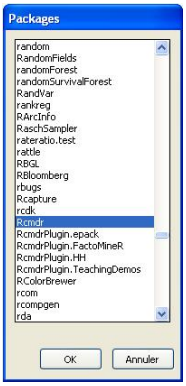
Ensuite, allez dans le menu `Packages`, puis dans le sous-menu `Installer le(s) package(s)...`



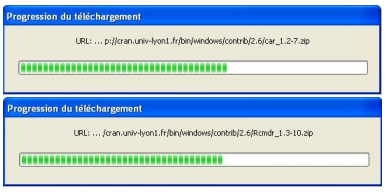
Sélectionnez un miroir (*CRAN mirror*) proche de votre situation géographique et cliquez sur `OK`.

Puis, à l'étape suivante, sélectionnez les entrées « car » et « Rcmdr ». Pour cela, cliquez d'abord sur « car » puis glissez l'ascenseur vers le bas et cliquez sur « Rcmdr » tout en maintenant la touche `CTRL` enfoncée. Vous devez vérifier que les deux entrées sont bien sélectionnées (surlignées en bleu).





Puis cliquez sur « OK ». Vous devriez alors voir apparaître successivement les deux écrans suivants, indiquant que les *packages* que vous avez sélectionnés sont en cours d'installation.



Attention

Il se peut que cette procédure échoue, par exemple si votre administrateur a bloqué l'accès à certains sites internet par un pare-feu, a imposé l'utilisation d'un *proxy* pour accéder à l'Internet ou encore s'il a interdit l'écriture dans certains dossiers de Windows. Nous vous conseillons de le contacter en cas de problème. Notez que vous pouvez forcer R à utiliser un *proxy* et que vous pouvez aussi installer des *packages* localement dans votre propre compte utilisateur. N'hésitez pas à consulter à ce sujet les sections 2.15 (*How do I set environment variables ?*) ou 2.19 (*The Internet download functions fail*) de la FAQ de Windows disponible ici : `http://cran.r-project.org/bin/windows/base/rw-FAQ.html`

C.2.3 Installation depuis la ligne de commande

On peut se passer des menus de l'interface graphique de R. C'est par exemple utile sous Unix/Linux où le logiciel R ne possède pas d'interface graphique. Pour cela, tapez directement dans la console de R les commandes suivantes :

– pour des *packages* dont les fichiers `*.zip` sont situés sur votre disque dur :

```
install.packages(choose.files(),repos = NULL)
```

– pour un *package* (par exemple `Rcmdr`) dont le fichier est sur le site internet CRAN :

```
install.packages("Rcmdr")
```

Une autre possibilité est d'installer un *package* sans passer par R, c'est-à-dire directement depuis une fenêtre de commandes MS-DOS sous Microsoft, ou une fenêtre de terminal sous Linux ou MacOS. Mais dans ce dernier cas, vous aurez besoin de nombreux outils de compilation. Si ces outils ne sont pas installés sur votre ordinateur, reportez-vous au paragraphe sur la création de *packages* dans le chapitre 7.

Si ces outils sont installés sur votre ordinateur, vous pouvez essayer ceci :

Téléchargez par exemple le fichier (*package source*) `Rcmdr_`*numero*`.tar.gz` à partir de cette adresse internet : `http://cran.r-project.org/web/packages/Rcmdr`.
Enregistrez-le (sur le bureau) et lancez une fenêtre de commande MS-DOS (Menu Démarrer/Exécuter/`cmd`), puis tapez :
`cd Bureau`
`R CMD INSTALL Rcmdr_`*numero*`.tar.gz` (remplacez bien entendu *numero* par ce qu'il faut).

C.2.4 Installation de *packages* sous Linux

Notez que les commandes de la section précédente fonctionnent également sous Linux. Mais leur utilisation nécessite parfois d'être connecté en tant que super-utilisateur *root* (commande `su -` à taper dans une fenêtre de terminal).

Si vous ne disposez pas des droits d'accès *root*, vous pouvez choisir d'installer des *packages* localement, c'est-à-dire dans votre répertoire personnel (*home directory*). Par exemple, pour le *package* `Rcmdr`, tapez dans un terminal :

```
R CMD INSTALL --library=/home/user/Rlibs Rcmdr_numero.tar.gz
```

(où, après que vous avez créé le dossier Rlibs au moyen de la commande `mkdir Rlibs`, le chemin `/home/user/Rlibs` devra être remplacé par le chemin approprié et *numero* devra également être remplacé par ce qu'il faut).

Ensuite, afin que R puisse savoir où chercher les *packages* que vous aurez installés, il vous faut créer un fichier nommé ~/.Renviron qui devra contenir la ligne

```
R_LIBS=/home/user/Rlibs
```

Astuce

Si votre ordinateur est situé derrière un pare-feu et que vous devez utiliser un *proxy* pour accéder à l'Internet, vous pouvez utiliser la commande suivante pour installer un *package* directement depuis R :

```
Sys.setenv("http_proxy"="http://user:pass@url_vers_
                                le_proxy:num_port")
install.packages("Rcmdr",method="wget")
```

N'hésitez pas à consulter l'aide en ligne de la fonction `download.file()` pour plus de détails.

SECTION C.3

Chargement des *packages* installés

Attention

Pour bien comprendre cette section, vous devez avoir une idée grossière de la différence entre la mémoire volatile (RAM) de votre ordinateur et la mémoire physique d'un disque dur.

Installer un *package* signifie que les fichiers qu'il contient sont « écrits » physiquement sur le disque dur. Lorsque l'on éteint puis rallume l'ordinateur, ces fichiers seront toujours présents à l'endroit où ils auront été copiés. Vous n'aurez donc plus besoin de réinstaller ce *package*, sauf pour en avoir une version plus récente.
Au contraire, charger un *package* (en mémoire) signifie qu'il est temporairement mis à disposition de l'utilisateur dans R. Mais si l'on ferme puis rouvre R, ce *package* ne sera plus disponible depuis R. Il faudra donc le charger de nouveau.

Pour résumer, une fois que les *packages* souhaités ont été installés sur le disque dur de votre ordinateur, il faut les charger dans la mémoire de R pour pouvoir les utiliser.

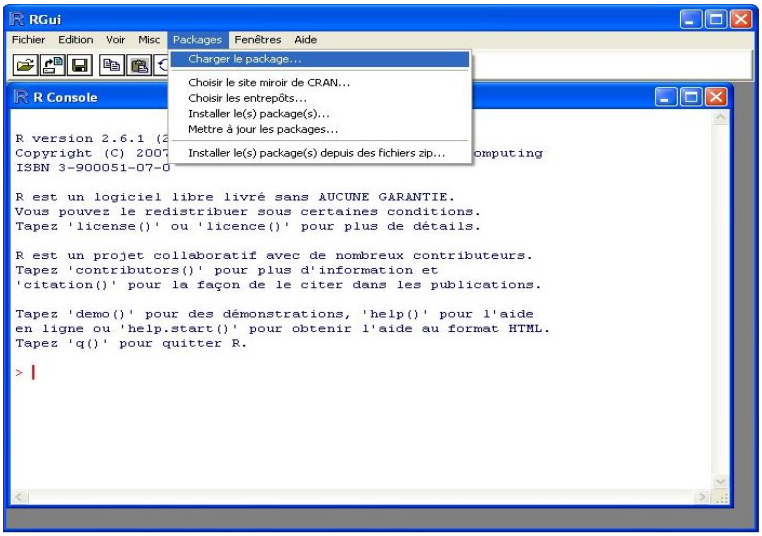
Par exemple, si vous tapez dans la console de R :

```
Commander()
```

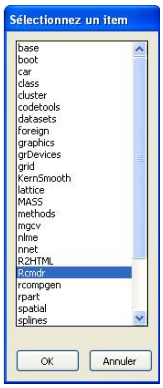
vous devriez voir apparaître le message d'erreur suivant indiquant que le *package* dont cette fonction est issue n'est pas accessible depuis R :

```
Erreur: impossible de trouver la fonction "Commander"
```

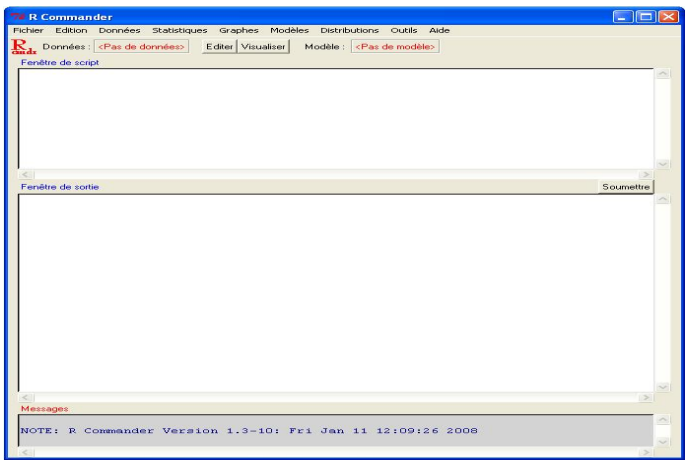
Il faut donc d'abord charger `Rcmdr` en mémoire. Pour cela, on peut soit taper `require("Rcmdr")` dans la console, soit aller dans le menu `Packages/Charger le package...`,



et charger le *package* `Rcmdr` à l'aide de la souris.



La fenêtre suivante apparaît alors.



Fermez-la et tapez de nouveau dans la console de R :

```
Commander()
```

Notez maintenant que le *package* `Rcmdr` a bien été chargé et donc que la commande précédente ne renvoie plus de message d'erreur.

Astuce

Notez que le logiciel R vous offre la possibilité de charger automatiquement certains *packages* au démarrage de la façon suivante :

```
.First <- function() {
  require("pkg1") # Remplacez pkg1 par le
                  # nom du package souhaité
  require("pkg2")
  # etc.
}
```

En fait, les fonctions `.First()` et `.Last()` permettent respectivement d'exécuter au démarrage et à l'arrêt de R les instructions que l'on aura pris le soin de spécifier dans le corps de ces deux fonctions. Ces fonctions peuvent être placées dans un fichier nommé .Renviron situé dans le répertoire courant ou bien dans le répertoire de l'utilisateur donné par l'instruction R : `Sys.getenv("R_USER")`.

Références

[1] J. Adler : **R** *in a Nutshell*. O'Reilly, 2010.

[2] H. Akaike : Information Theory and an Extension of the Maximum Likelihood Principle. *In* Petrov B. N., Csaki F., éditeurs : *Proc. of the 2nd Int. Symp. on Information Theory*, pages 267-81, 1973.

[3] A. Antoniadis, J. Berruyer, R. Carmona : *Régression non linéaire et applications*. Economica, 1992.

[4] Y. Aragon : *Séries temporelles avec* **R**. *Méthodes et cas*. Springer, Collection Pratique **R**, 1re édition, 2011.

[5] J. M. Bardet, J. M. Azais : *Le modèle linéaire par l'exemple : Régression, Analyse de la variance et Plans d'expérience illustrés avec* **R**, *SAS et Splus*. Dunod, Sciences Sup, 2006.

[6] R. A. Becker, J. M. Chambers, A. R. Wilks : *The New S Language: A Programming Environment for Data Analysis and Graphics*. Chapman & Hall, 1988.

[7] D. A. Belsley, E. Kuh, R. E. Welsch : *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. John Wiley & Sons, New York-Chichester-Brisbane, 1980. Wiley Series in Probability and Mathematical Statistics.

[8] M. Bilodeau, P. Lafaye de Micheaux : A-dependence Statistics for Mutual and Serial Independence of Categorical Variables. *Journal of Statistical Planning and Inference*, 139:2407-19, 2009.

[9] J. W. Braun, D. J. Murdoch : *A First Course in Statistical Programming with* **R**. Cambridge University Press, 1st edition, January 2008.

[10] P. Burns : The **R** inferno. Unpublished manual, 2011.

[11] J. M. Chambers : *Software for Data Analysis : Programming with* **R**. Statistics and Computing. Springer, June 2008.

[12] I. Chivers : *Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77*. Springer, 2nd edition, 2012.

[13] Y. Cohen, J. Cohen : *Statistics and Data with* **R***: An Applied Approach Through Examples*. Wiley, 2008.

[14] R. D. Cook, S. Weisberg : *Residuals and Influence in Regression*. Monographs on Statistics and Applied Probability. Chapman & Hall, London, 1982.

[15] P. A. Cornillon, A. Guyader, N. Husson, N. Jégou, J. Josse, M. Kloareg, E. Matzner-Løber, L. Rouvière : *Statistiques avec* R. Presses Universitaire de Rennes, 2008.

[16] P. A. Cornillon, E. Matzner-Lober : *Régression. Théorie et applications*. Springer-Verlag France, 2007.

[17] M. J. Crawley : *The* R *Book*. Wiley, Chichester, June 2007.

[18] P. Dalgaard : *Introductory Statistics with* R *(Statistics and Computing)*. Springer, 2nd edition, August 2008.

[19] A. C. Davison, D. V. Hinkley : *Bootstrap Methods and Their Application*, volume 1 de *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge, 1997.

[20] Y. Dodge, G. Melfi : *Premiers pas en simulation*. Springer-Verlag, 2008.

[21] D. Eddelbuettel : *Seamless* R *and C++ Integration with Rcpp*. Use R ! Springer, 2013.

[22] B. S. Everitt, T. Hothorn : *A Handbook of Statistical Analyses Using* R. Chapman & Hall/CRC, 1st edition, February 2006.

[23] R. A. Fisher : *Statistical Methods for Research Workers, 4th edition § 21.1*. Oliver & Boyd, Edinburgh, 1932.

[24] J. Fox : Extending the R Commander by "plug in" Packages. R *News*, 7(3):46-52, 2007.

[25] G. M. Furnival, R. W. Jr. Wilson : Regression by Leaps and Bounds. *Technometrics*, 16:499-511, 1974.

[26] D. J. Hand : Branch and Bounds in Statistical Data Analysis. *The Statistician*, 30:1-13, 1981.

[27] R. M. Heiberger, E. Neuwirth : R *Through Excel*. Use R ! Springer, 2009.

[28] R. Ihaka, R. Gentleman : R : A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3):299-314, 1996.

[29] B. W. Kernighan, D. M. Ritchie : *Le langage C Norme ANSI*. Dunod, 2^e édition, 2004.

[30] P. Lafaye de Micheaux, B. Liquet : ConvergenceConcepts : an R Package to Investigate Various Modes of Convergence. R *Journal*, 1(2):18-26, 2009.

[31] P. Lafaye de Micheaux, B. Liquet : Understanding Convergence Concepts : A Visual-Minded and Graphical Simulation-Based Approach. *The American Statistician*, 63(2):173-8, 2009.

[32] H. Levene : Robust Tests for Equality of Variances. *In Contributions to probability and statistics*, pages 278-92. Stanford Univ. Press, Stanford, Calif., 1960.

[33] J. Maindonald, J. W. Braun : *Data Analysis and Graphics Using* **R***: An Example-based Approach (Cambridge Series in Statistical and Probabilistic Mathematics)*. Cambridge University Press, December 2006.

[34] C. L. Mallows : Some Comments on c_p. *Technometrics*, 15:661-75, 1973.

[35] N. Matloff : *The Art of* **R** *Programming*. No Starch Press, 2011.

[36] C. Meyer : *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.).

[37] Jr. Miller, G. Rupert : *Simultaneous Statistical Inference*. Springer-Verlag, 2nd edition, New York, 1981. Springer Series in Statistics.

[38] G. Millot : *Comprendre et réaliser les tests statistiques à l'aide de* **R**. De Boeck, 2009.

[39] R. A. Muenchen : **R** *for SAS and SPSS Users*. Springer Series in Statistics and Computing. Springer, 2009.

[40] R. A. Muenchen, J. M. Hilbe : **R** *for Stata Users*. Statistics and Computing. Springer, 2010.

[41] S. K. Park, K. W. Miller : Random Number Generators: Good Ones are Hard to Find. *Commun. ACM*, 31(10):1192-1201, 1988.

[42] D. Sarkar : *Lattice : Multivariate Data Visualization with* **R**. Use **R**! Springer, March 2008.

[43] G. Schwarz : Estimating the Dimension of a Model. *The Annals of Statistics*, 6:461-64, 1978.

[44] B. Stroustrup : *The C++ Programming Language*. Addison-Wesley Professional, 4th Edition, 2013.

[45] P. Teetor : **R** *Cookbook*. O'Reilly, 2011.

[46] A. F. Zuur, E. N. Ieno, E. Meesters : *A Beginner's Guide to* **R**. Springer, 2009.

Index général

H

I

W

Index des commandes et symboles R

Symboles

A

B

C

D

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Index des auteurs

Liste des *packages* R mentionnés dans le livre

Solutions des exercices

Solutions des exercices du chapitre 1

1.1- `[1] 1 2 3 4 5 6 7 8 9`

1.2- `[1] 2 4 6 8 10`

1.3- L'instruction `var<-3` ne renvoie rien. L'instruction `Var*2` renvoie un message d'erreur car `Var` n'a pas encore été définie.

1.4- L'instruction `x<-2` ne renvoie rien. L'instruction `2x<-2*x` renvoie un message d'erreur car un nom de variable ne devrait pas commencer par un chiffre.

1.5- L'instruction `racine.de.quatre<-sqrt(4)` ne renvoie rien. L'instruction `racine.de.quatre` renvoie `[1] 2`

1.6- L'instruction `x<-1` ne renvoie rien. L'instruction `x< -1` renvoie `[1] FALSE`

1.7- L'instruction `Un chiffre pair <- 16` renvoie un message d'erreur car un nom de variable ne devrait pas contenir d'espaces.

1.8- L'instruction `"Un chiffre pair" <- 16` ne renvoie rien.

1.9- L'instruction `"2x" <- 14` ne renvoie rien.

1.10- L'instruction `Un chiffre pair` renvoie un message d'erreur.

1.11-
```
>2 +
+ 4
[1] 6
```

1.12- L'instruction `TRUE + T + FALSE*F + T*FALSE + F` renvoie `[1] 2`.

1.13- Les cinq types de données sous **R** sont : `numeric`, `complex`, `logical`, `character`, `raw`.

1.14- `X <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)`

1.15- Les structures de données sous **R** sont : `c()`, `matrix()`, `array()`, `list()`, `data.frame()`, `factor()`, `ordered()`.

Solutions des exercices du chapitre 2

2.1- Les trois principales fonctions R à utiliser pour importer des données depuis un fichier texte au format ASCII sont : `read.table()`, `scan()` et `read.ftable()`.

2.2- `header` : permet d'indiquer si le fichier contient le nom des variables (ex. : `header=TRUE`).
`sep` : permet d'indiquer le caractère séparant les valeurs sur chaque ligne (ex. : `sep=" "` ou `sep="\t"`).
`dec` : séparateur décimal (ex. : `dec="."` ou `dec=","`).
`row.names` : permet de spécifier les noms des individus (ex. : `row.names=2` ; la deuxième colonne contient le nom des individus).
`skip` : indique le nombre de lignes du fichier de données à omettre avant de commencer à lire les données (ex : `skip=4` pour exclure la lecture des 4 premières lignes).
`nrows` : permet de spécifier le nombre maximum de lignes à lire (ex. : `row.names=19`).

2.3- La fonction `readLines()` permet de visualiser le début d'un fichier de données.

2.4- La fonction `fix()` permet de modifier un `data.frame` ou une matrice au moyen d'un mini-tableur.

2.5- `read.csv()` : permet de lire des données séparées par des virgules et avec le « . » pour séparateur décimal.
`read.csv2()` : permet de lire des données séparées par des points-virgules et avec la « , » pour séparateur décimal.
`read.delim()` : permet de lire des données séparées par des tabulations et avec le « . » pour séparateur décimal.
`read.delim2()` : permet de lire des données séparées par des tabulations et avec le « , » pour séparateur décimal.

2.6- La fonction `read.ftable()` permet de lire des tableaux de contingence.

2.7- La fonction `scan()` est à privilégier lorsque les données ne sont pas organisées sous la forme d'un tableau rectangulaire. La fonction `read.table()` est utilisée pour des jeux de données organisés sous la forme de tableaux.

2.8- Importation des données se trouvant dans une feuille de classeur Excel :
• Utilisation du copier-coller : sélectionner sous Excel les données à incorporer, copier ces données dans le presse-papiers, utiliser l'instruction :

```
x <- read.table(file("clipboard"),sep="\t",header=TRUE,dec=",")
```

• Passer par un fichier ASCII : enregistrer la feuille Excel en `.txt` (séparateur : tabulation), puis utiliser la fonction `read.table()`.
• Utilisation du *package* `gdata` et de la fonction `read.xls()`.

2.9- Le *package* `foreign`.

2.10- L'argument `colClasses` de la fonction `read.table()` permet d'indiquer le type de chacune des colonnes et ainsi permet d'augmenter considérablement la vitesse de lecture des gros fichiers.

2.11- La fonction `write.table()` permet d'écrire dans un fichier le jeu de données contenu dans un `data.frame`. Une autre fonction possible est la fonction `write()` qui est à utiliser pour des objets de type vecteur ou matrice.

2.12- Voici quatre fonctions de base permettant de fabriquer des vecteurs :
- la fonction `c()`
- la fonction `seq()`
- la fonction `rep()`
- la fonction `":"()` (exemple `1:10`)

2.13- L'instruction `seq(1,2,by=0.1)` permet d'obtenir le vecteur suivant :
```
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

2.14- L'instruction `rep(1:3,each=2)` permet d'obtenir le vecteur suivant :

```
1 1 2 2 3 3
```

2.15- L'instruction `rep(1:3,2)` permet d'obtenir le vecteur suivant :

```
1 2 3 1 2 3
```

2.16- Les fonctions permettant d'entrer les données à la main dans le minitableur sont : `data.entry()` et `de()`.

Solutions des exercices du chapitre 3

3.1-
```
[1] 2 12
```

3.2-
```
     [,1] [,2]
[1,]    1    1
[2,]    2    2
```

3.3- Avec les fonctions `rownames()` et `colnames()`.

3.4- `cbind(X,Y)`

3.5- Pour le produit de tous les éléments d'une matrice `X` : `prod(X)`.
Pour le produit des éléments de chacune des colonnes de la matrice `X` : `apply(X,FUN=prod,MARGIN=2)`.

3.6-
```
[1] 4
[1] 2 6 8 3
```

3.7- `poids[taille>180]`

3.8-
```
[1] 7 8 9
[1] 5 6
```

3.9- `L[[4]] <- 1:10`

3.10- `[1] 67`

3.11-
```
attach(X)
poids[sexe=="F"]
taille[sexe=="F"]
# ou bien:
X[sexe=="F",-3]
```

3.12- `[1] 1 2 3` et `integer(0)`

3.13- `[1] TRUE TRUE FALSE` et `[1] TRUE`

3.14- `[1] 4 4`

3.15- `[1] "acbd"`

3.16-
```
[[1]]
[1] "ab" "cd"
```

3.17- `[1] "" "cd"`

3.18- `tolower(c("Pierre","Paul","Pascal"))`

3.19- La fonction `strptime()`.

3.20- Car la fonction `logp()` n'est pas vectorielle. Elle ne renvoie qu'un seul nombre.

Solutions des exercices du chapitre 4

4.1- `help(mean)`.

4.2- Cette instruction renvoie une liste de toutes les fonctions dont le nom contient le mot requêté.

4.3- Cette instruction donne des exemples sur l'utilisation de la fonction requêtée.

4.4- La commande `RSiteSearch()` permet de faire une requête sur le site `http://search.r-project.org/nmz.html` directement depuis R. Les informations extraites sont alors affichées dans votre navigateur.

4.5-
1. L'entête du fichier contenant :
   - le nom de la fonction pour laquelle on recherche de l'aide ;
   - le nom du *package* qui contient cette fonction (ex. : `base`) ;
   - l'origine du fichier d'aide : `R Documentation` ;
2. Un titre explicite pour la fonction (ex. : `Arithmetic Mean`).
3. Une brève description de ce que fait la fonction : `Description`.
4. La façon d'utiliser la fonction avec notamment ses arguments obligatoires et optionnels : `Usage`.
5. Une description des différents arguments d'entrée de la fonction : `Arguments`.

6. Des explications sur ce que renvoie la fonction lorsqu'elle est exécutée : `Value`.
7. Des références d'articles ou d'ouvrages ayant rapport avec le domaine statistique d'application de la fonction : `References`.
8. La rubrique `See Also` qui donne le nom de fonctions similaires ou en rapport avec la fonction en cours d'investigation.
9. Quelques exemples montrant comment utiliser cette fonction : `Examples`

4.6- `help(package="stats")` ou `library(help="stats")`.

4.7- Tapez `data()` pour voir la liste des jeux de données et tapez ensuite le nom du jeu de données choisi.

Solutions des exercices du chapitre 5

5.1- La commande `windows()` sert à ouvrir une fenêtre graphique. La commande `dev.off()` ferme la fenêtre spécifiée par *numero-device* (si aucun numéro de *device* n'est fourni, c'est la fenêtre active qui est fermée).

5.2- `savePlot(filename="monplot",type="pdf",device=dev.cur())`

5.3- L'instruction `par(mfrow=c(3,2))` permet d'ouvrir une fenêtre graphique où les figures seront successivement tracées dans une « matrice » ayant 3 lignes et 2 colonnes (traçage par lignes).

5.4- La fonction `layout()` permet d'obtenir un découpage plus évolué de la fenêtre graphique que l'utilisation de la fonction `par()`.

5.5- `points()`

5.6- `type="l"`

5.7- `abline()`

5.8- La fonction `curve()` permet de tracer n'importe quelle fonction de x.

5.9- Le paramètre `col`.

5.10- La fonction `image()`. L'instruction

```
image(as.matrix(rev(as.data.frame(t(X)))))
```

permet d'afficher de façon cohérente l'image dont les valeurs sont données dans la matrice `X`.

5.11- La fonction `text()`.

5.12- La fonction `identify()` ou `locator()`.

5.13- L'instruction `par(ask=TRUE)` renvoie un message demandant à l'utilisateur d'appuyer sur la touche `ENTREE` avant que chaque nouveau graphique soit tracé.

5.14- `lty`

5.15- `pch`

5.16-
```
curve(cos(x),xlim=c(-10,10),xlab="Axe des X",col="blue",
main="Courbes de sinus et cosinus",ylim=c(-2,2),ylab="sin(x)")
curve(sin(x),add=TRUE)
abline(h=0,col="red")
abline(v=0,col="red")
arrows(3*pi/2,1,pi/2,1)
text((3*pi)/2,1,expression(hat(beta)[1]))
```

Solutions des exercices du chapitre 6

6.1-
- `function(nom) {nom}` : l'objet R retourné est de classe `function`, l'affichage produit est : `function(nom) {nom}`.
- `(function(nom) {nom})("Ben")` : l'objet R retourné est de classe `character`, l'affichage produit est : `"Ben"`
- `(function(nom) {cat(nom,"\n")})("Ben")` : l'objet R retourné n'a pas de classe, l'affichage produit est : `Ben`
- `(function(nom) {invisible(nom)})("Ben")` : l'objet R retourné est de classe `character`, aucun affichage.

6.2-
- Aucune différence.
- Aucune différence.
- Aucune différence.

6.3- Aucune différence lors de l'exécution de `nom()` et `nom("Peter")` lorsque la fonction `nom()` est définie par `nom <- function(nom="Peter") nom` et aussi par `nom <- function(nom="Peter") nom2 <- nom`. Pour ces deux fonctions la nature de `res`, obtenu par `res <- nom("Ben")`, est de type `character`.

6.4- `"Peter"`.

6.5-
- `"Ben L"`
- `"Ben L"`
- `"R D"`

6.6-
```
nom <- function(nom="Peter") {cat(nom,"\n")}
```

6.7- Pour l'exécution de `noms("peteR","Ben","R")`
- `[1] "peteR" "Ben"   "R"`
-
```
[[1]]
[1] "peteR"

[[2]]
[1] "Ben"

[[3]]
[1] "R"
```

- `[1] "peteR"`
  `[1] "Ben"`
  `[1] "R"`
- `[1] "peteR"`
  `[1] "Ben"`
  `[1] "R"`

Pour l'exécution de `noms(c("peteR","L"),c("Ben","L"),c("R","D"))`

- `[1] "peteR" "L"  "Ben"  "L"  "R"  "D"`
- `[[1]]`
  `[1] "peteR" "L"`

  `[[2]]`
  `[1] "Ben" "L"`

  `[[3]]`
  `[1] "R" "D"`
- `[1] "peteR"`
  `[1] "L"`
  `[1] "Ben"`
  `[1] "L"`
  `[1] "R"`
  `[1] "D"`
- `[1] "peteR" "L"`
  `[1] "Ben" "L"`
  `[1] "R" "D"`

6.8- Pour la fonction `noms <- function(noms=c("Ben","R"),...) c(noms,...)`

- `[1] "PeteR"`
- `[1] "PeteR"`
- `[1] "PeteR"`

Pour la fonction `noms<-function(...,noms=c("Ben","R")) c(noms,...)`

- `[1] "Ben" "R" "PeteR"`
- ```
                  nom
    "Ben"     "R" "PeteR"
  ```
- `[1] "PeteR"`

6.9-
```
- Homme <- function(prénom,nom) {
    obj <- list(prénom=prénom,nom=nom)
    class(obj) <- "Homme"
    obj
  }
- bonjour.Homme <- function(obj)
    cat("Bonjour Monsieur",obj$prénom,obj$nom,"!\n")
- Bonjour Monsieur Ben L !
- Erreur : impossible de trouver la fonction "bonjour"
- bonjour <- function(obj) UseMethod("bonjour")
```
```

6.10-
```
– Femme <- function(prénom,nom) {
    obj <- list(prénom=prénom,nom=nom)
    class(obj) <- "Femme"
    obj
  }
– bonjour.Femme <- function(obj)
   cat("Bonjour Madame",obj$prénom,obj$nom,"!\n")
– Bonjour Monsieur Dominique L !
– Bonjour Madame Dominique L !
– Bonjour Madame Dominique L !
```

6.11-
```
– Bonjour Monsieur Ben L !
  Bonjour Madame Dominique L !
– Erreur dans UseMethod("bonjour") :
    pas de méthode pour 'bonjour' applicable pour
    un objet de classe "character"
– Bonjour Ben !
  Bonjour L !
  Bonjour Dominique !
  Bonjour L !
```

Solutions des exercices du chapitre 7

7.1- `ls()` ou `objects()`.

7.2- `rm(toto)`.

7.3- `getwd()`.

7.4- `setwd(choose.dir())`.

7.5- Enregistre dans un fichier (`nom.RData`) les objets créés.

7.6- L'historique des commandes, l'environnement de travail, le contenu affiché dans la console, et les graphiques.

7.7- Cela permet de rappeler et réexécuter les commandes précédemment tapées (au moyen des flèches haut et bas du clavier).

7.8- La fonction `history()` affiche dans une nouvelle fenêtre la liste de l'historique de toutes les commandes passées de la session en cours.

7.9-
```
png(file="myplot.png")
curve(x**2)
dev.off()
```

7.10- Elle permet d'avoir accès aux variables du *data.frame* directement par leur nom.

7.11- `require()` (ou `library()`)

7.12- Importe une suite d'instructions **R** présentes dans un fichier vers la console, en en vérifiant la syntaxe.

Solutions des exercices du chapitre 8

8.1- La commande `choose()`.

8.2- L'instruction `sum(1:n)`.

8.3- La commande `range()`.

8.4- Le produit termes à termes des deux matrices suivantes :

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

et

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

qui donne

```
     [,1] [,2]
[1,]    1    0
[2,]    0    4
```

8.5- La commande `%*%`.

8.6- La fonction `solve()` pour l'inverse et la fonction `t()` pour la transposée.

8.7- L'instruction `diag(5)`.

8.8- La commande `det()` pour le déterminant et `sum(diag())` pour la trace.

8.9- `scale(A)`.

8.10- La fonction `eigen()`.

8.11-
```
maf <- function(x) {3*x^2+2}
integrate(maf,lower=-1,upper=1)
```

8.12-
```
optimize(f=function(x)(sin(x))**2,lower=0,upper=2,maximum=TRUE)
```

8.13- La commande `uniroot()` pour une fonction et `polyroot()` pour un polynôme.

Solutions des exercices du chapitre 9

9.1- L'instruction `table(x)/length(x)`.

9.2- L'instruction `table(x,y)`.

9.3- La commande `margin.table()`.

9.4- La commande `prop.table()`.

9.5- L'instruction `names(which.max(table(matable)))`.

9.6- L'instruction `diff(range(x))`.

9.7- L'instruction `IQR(x)`.

9.8- L'instruction `var(x)*(length(x)-1)/length(x)`.

9.9- L'instruction `sqrt(var(x)*(length(x)-1)/length(x))/mean(x)`.

9.10- L'instruction `mean(abs(x-mean(x)))`.

9.11- Le *package* `moments`.

9.12- Il faut tout d'abord calculer le χ^2 avec l'instruction suivante :

```
khi2 <- summary(table(matable))$statistic
```

Le Φ^2 de Cramér s'obtient alors par l'instruction `khi2/N`.

9.13- Voici le code permettant de calculer le rapport de corrélation $\eta^2_{Y|X}$:

```
eta2 <- function(x,gpe) {
 moyennes <- tapply(x,gpe,mean)
 effectifs <- tapply(x,gpe,length)
 varinter <- (sum(effectifs * (moyennes - mean(x))^2))
 vartot <- (var(x) * (length(x) - 1))
 res <- varinter/vartot
 return(res)
}
```

9.14- La fonction `barplot()` permet d'obtenir un diagramme de Pareto.

9.15- Un diagramme empilé s'obtient au moyen de la fonction `barplot()` en fournissant un objet de type `matrix` comme premier argument.

9.16- La fonction `pie()` permet d'obtenir un diagramme circulaire.

9.17- La fonction `boxplot()` permet d'obtenir un diagramme en boîte à moustaches.

9.18- La fonction `hist()` permet d'obtenir un histogramme.

Solutions des exercices du chapitre 10

10.1- La fonction `rnorm()` permet de générer des nombres suivant une loi $\mathcal{N}(0,1)$.

10.2- L'instruction `rnorm(n,mean=2,sd=sqrt(10))` permet de générer n nombres issus d'une variable aléatoire de loi $\mathcal{N}(2,10)$.

10.3- La fonction `qchisq()` permet de calculer les quantiles d'une loi du χ^2.

10.4- La fonction `df()` permet de calculer des valeurs de densité d'une loi de Fisher.

10.5- La fonction `qt()` permet de calculer les quantiles d'une loi de Student.

10.6- `pnorm(5,mean=4,sd=sqrt(2))-pnorm(3,mean=4,sd=sqrt(2))`

10.7- `qnorm(0.95)`

Solutions des exercices du chapitre 11

11.1- La fonction `qbinom()` permet d'obtenir les quantiles d'une loi binomiale.

11.2- La fonction `pnorm()` permet de calculer la fonction de répartition d'une loi normale.

11.3- `t.test(x,conf.level=0.9)$conf.int`

11.4- La fonction `prop.test()` permet d'obtenir un intervalle de confiance sur une proportion ou un test statistique en approchant la loi binomiale par la loi normale, alors que la fonction `binom.test()` effectue des calculs exacts.

11.5- `ks.test()` et `wilcox.test()`.

11.6- `shapiro.test()`.

11.7- Les fonctions permettant de tester la dépendance entre deux caractères qualitatifs sont `chisq.test()` et `fisher.test()`.

11.8- Le *package* `boot`.

11.9- L'argument `paired=TRUE`.

11.10- Le test du χ^2 d'indépendance permet de tester la dépendance entre deux caractères qualitatifs alors que le test du χ^2 d'ajustement permet de tester si une variable qualitative suit une distribution donnée. Pour effectuer ces deux tests, vous devez utiliser la fonction `chisq.test()`. Le test du χ^2 d'indépendance s'effectue sur une `table`. Pour le test du χ^2 d'ajustement, il est nécessaire de spécifier l'argument `p` représentant les probabilités théoriques des modalités de la variable qualitative testée.

Solutions des exercices du chapitre 12

12.1- `lm(Y~X1)`.

12.2- `lm(Y~X1-1)` ou `lm(Y~0+X1)`.

12.3- `lm(Y~X1+X2)`.

12.4- `lm(Y~X1+X2+X1:X2)` ou `lm(Y~X1*X2)`.

12.5- `lm(Y~X1+I(X1^2)+I(X1^4))`.

12.6- `anova(lm(Y~X1+X2),lm(Y~X1+X2+X3+X4))`.

12.7- `residuals()`.

12.8- `coefficients()`.

12.9- Il faut introduire la variable Z comme un facteur en utilisant la commande `factor()`.

12.10- `lm(Y~poly(X,3))`.

12.11- `add1()`.

12.12- `drop1()`.

Solutions des exercices du chapitre 13

13.1- `aov(Y~factor(A))`.

13.2- `aov(Y~factor(A)+factor(B))`.

13.3- `aov(Y~factor(A)*factor(B))`.

13.4- `bartlett.test()` et `levene.test()`.

13.5- `pairwise.t.test()`.

13.6- `summary(lm(Y~factor(X)))`.

13.7- La fonction `C()`.

Solutions des TPs

Solutions des TPs du chapitre 1

Étude sur l'indice de masse corporelle

1.1-
```
Individus <- c("Érika","Célia","Éric","Ève","Paul",
               "Jean","Adan","Louis","Jules","Léo")
Poids <- c(16,14,13.5,15.4,16.5,16,17,14.8,17,16.7)
Taille <- c(100,97,95.5,101,100,98.5,103,98,101.5,100)
Sexe <- c("F","F","G","F","G","G","G","G","G","G")
```

1.2-
```
mean(Poids)  # résultat :  [1] 15.69
mean(Taille) # résultat :  [1] 99.45
```

1.3-
```
IMC <- Poids/(Taille/100)^2
IMC   # Affichage des résultats.
```

1.4-
```
maTable <- data.frame(Individus,Poids,Taille,Sexe,IMC)
maTable   # Affichage des résultats.
```

1.5-
```
help(plot)
```

1.6-
```
plot(Taille,Poids,main="Nuage de points du Poids en
                        fonction de la Taille")
```

Solutions des TPs du chapitre 2

Lecture de différents jeux de données

A- Rentrer des données issues d'un format papier

- Boutons de fièvre

2.1- `bout <- as.data.frame(de(""))`

Commencez d'abord par changer le type de chacune des colonnes en `Real`, ainsi que le nom en `trt`i, $1 \leq i \leq 5$ (en cliquant sur la première case de chacune des colonnes), puis entrez vos données. Cliquez sur `Quit` à la fin. On peut ensuite taper la commande suivante pour voir s'afficher les données dans **R** :

```
bout # Affichage des données.
```

2.2-
```
attach(bout)
mean(trt1)  # Résultat: [1] 7.5
mean(trt2)  # Résultat: [1] 5
mean(trt3)  # Résultat: [1] 4.333333
mean(trt4)  # Résultat: [1] 5.166667
mean(trt5)  # Résultat: [1] 6.166667
```

2.3- On obtient directement les mêmes résultats en tapant :

```
colMeans(bout)
```

2.4-
```
write.table(file="boutons.txt",bout,row.names=FALSE)
```

2.5- Vous pouvez utiliser votre éditeur de texte favori pour visualiser le contenu du fichier boutons.txt et ainsi vérifier qu'il a bien été créé (l'instruction `getwd()` vous dira où ce fichier a été enregistré).

2.6-
```
ls()
rm(bout)
ls()
bout # On constate que l'objet bout a disparu.
```

2.7-
```
bout <- read.table("boutons.txt",header=TRUE,sep="")
bout # Le revoilà!
```

- Facteurs de risque de l'athérosclérose

2.1- Entrez les données telles qu'organisées dans le tableau de contingence, en pressant la touche `ENTRÉE` après chaque fin de ligne. Après la dernière ligne, pressez une autre fois `ENTRÉE`.

```
X <- scan() # Les données sont alors regroupées dans un vecteur.
X <- matrix(X,ncol=3,nrow=6,byrow=TRUE)
```

2.2-
```
class(X) <- "ftable"
```

2.3-
```
attributes(X)$col.vars<-list(alcool=c("ne boit pas",
          "boit occasionnellement","boit régulièrement"))
attributes(X)$row.vars<-list(SEXE=c("H","F"),tabac=
             c("ne fume pas","a arrêté de fumer","fume"))
```

2.4-
```
X
```

2.5- `write.ftable(X,file="athero.txt")`

2.6- Vous pouvez utiliser votre éditeur de texte favori pour visualiser le contenu du fichier athero.txt et ainsi vérifier qu'il a bien été créé (l'instruction `getwd()` vous dira où ce fichier a été enregistré).

2.7-
```
rm(X)
X # L'objet a été supprimé.
```

2.8-
```
X <- read.ftable(file="athero.txt")
X # Le revoilà!
```

B- Importer depuis un logiciel externe

2.1- Vous pouvez commencer par télécharger le fichier http://biostatisticien.eu/springeR/imcenfant.xls à l'aide de votre navigateur. Si vous avez Excel, vous pouvez alors le transformer en fichier imcenfant.txt, avec des tabulations pour les séparateurs, puis utiliser la commande R suivante :

```
imc.XLS <- read.table(file.choose(),header=TRUE,sep="\t",dec=",")
```

Si vous n'avez pas Excel, vous pouvez utiliser le *package* `xlsReadWrite`, après l'avoir installé :

```
require("xlsReadWrite")
imc.XLS <- read.xls("imcenfant.xls")
```

Enfin, sous Linux, vous pouvez utiliser les commandes :

```
require("gdata")
imc.XLS <- read.xls("http://biostatisticien.eu/springeR/
                     imcenfant.xls")
```

2.2- Installez le *package* `foreign`. Téléchargez le fichier http://www.biostatisticien.eu/springeR/imcenfant.xpt à l'aide de votre navigateur.

```
require("foreign")
imc.SAS <- read.xport(file.choose()) # Sélectionnez imcenfant.xpt.
```

2.3-
```
imc.SPSS <- read.spss("http://www.biostatisticien.eu/
                       springeR/imcenfant.sav")
imc.SPSS <- as.data.frame(imc.SPSS)
```

2.4- Installez le *package* `R.matlab`.

```
require("R.matlab")
x <- readMat("http://www.biostatisticien.eu/
              springeR/imcenfant.mat")
class(x) # x est une liste.
x        # On voit que les données sont dans $imc[,,1].
```

```
x <- x$imc[,,1]
# Notez que les éléments de SEXE et zep sont
# enregistrés dans une liste.
x$SEXE
class(x$SEXE) <- "character"
x$SEXE
class(x$zep) <- "character"
imc.MAT <- as.data.frame(x)
```

2.5-
```
summary(imc.XLS)
summary(imc.SAS)
summary(imc.SPSS)
summary(imc.MAT)
```

2.6-
```
write.table(imc.SPSS,"imcenfant.txt",row.names=FALSE)
```

C- Importer des fichiers de données plus compliqués

2.1-
```
readLines("http://biostatisticien.eu/springeR/formatgeoide.txt")
X <- scan("http://biostatisticien.eu/springeR/raf98.gra",skip=3)
X <- matrix(X,ncol=421,nrow=381,byrow=TRUE)
dim(X)
```

2.2- Enregistrez le fichier http ://biostatisticien.eu/springeR/Infarct.xls à l'aide de votre navigateur, puis transformez-le en un fichier Infarct.txt (tabulation comme séparateur). Ensuite, utilisez la commande :

```
infarct <- read.table("Infarct.txt",header=TRUE,sep="\t",
                      na.strings = ".",dec=",")
```

Sous Linux, utilisez plutôt :

```
require("gdata")
infarct <- read.xls("http://biostatisticien.eu/springeR/
   Infarct.xls",header=TRUE,sep=",",na.strings=".",dec=".")
```

2.3- Téléchargez le fichier http://biostatisticien.eu/springeR/nutriage.txt dans le dossier donné par la commande `getwd()`, puis entrez les commandes suivantes :

```
X <- read.table("nutriage.txt",row.names=1)
X <- t((X))
X <- as.data.frame(X)
```

2.4-
```
url <- "http://www.biostatisticien.eu/springeR/
        Poids_naissance.txt"
readLines(url) # Pour se faire une idée
               # de l'organisation du fichier.
# Constatez les trois lignes manquantes [33] à [35],
```

```
# ainsi que la présence de tirets à la ligne [194].
X <- read.table(url,row.names=1,skip=1,header=FALSE,sep=";",
                nrows=189,blank.lines.skip=TRUE)
Y <- read.table(url,nrows=1,row.names=1)
colnames(X) <- as.matrix(Y)
head(X) # Affichage des premières lignes de X.
```

Solutions des TPs du chapitre 3

Manipulation de différents jeux de données

A- Quelques manipulations sur les jeux de données présentés en début d'ouvrage

• Fichier nutriage.xls

3.1- Installez le *package* `gdata`. Attention l'utilisation de ce *package* repose sur le logiciel PERL qui n'est pas présent par défaut sur Windows. Installez donc aussi aussi PERL (`http://www.biostatisticien.eu/springeR/Rtools.exe`).

```
require("gdata")
nutri1 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri1.xls")
nutri2 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri2.xls")
colnames(nutri1) <- tolower(colnames(nutri1))
resultat <- rbind(nutri1,nutri2)
```

3.2-
```
nutri3 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri3.xls")
nutri4 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri4.xls")
resultat <- merge(nutri3,nutri4,all=TRUE)
```

3.3-
```
nutri5 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri5.xls")
nutri6 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri6.xls")
```

Le codage des différentes variables est donné dans le tableau de la section B.4. On va tâcher d'identifier les valeurs anormales au moyen de l'instruction suivante :

```
apply(nutri5,FUN=table,MARGIN=2)
```

On peut identifier six valeurs anormales dans `nutri5` : `sexe=12`, `sexe=21`, `taille=1.67`, `poids=200`, `age=8` et `chocol=7`. Les individus ayant ces valeurs anormales sont repérés ainsi :

```
ind5sexe <- nutri5$Sujet[which(nutri5$sexe %in% c(12,21))]
ind5taille <- nutri5$Sujet[which(nutri5$taille==1.67)]
ind5poids <- nutri5$Sujet[which(nutri5$poids==200)]
ind5age  <- nutri5$Sujet[which(nutri5$age==8)]
ind5chocol  <- nutri5$Sujet[which(nutri5$chocol==7)]
ind5anorm <- c(ind5sexe,ind5taille,ind5poids,ind5age,ind5chocol)
```

Faisons maintenant de même avec `nutri6`.

```
apply(nutri6,FUN=table,MARGIN=2)
```

On détecte sept valeurs anormales dans `nutri6` : `sexe=12`, `sexe=21`, `poids=8`, `age=7`, `viande=6`, `chocol=6`, et `matgras=9`. Les individus ayant ces valeurs anormales sont repérés ainsi :

```
ind6sexe <- nutri6$Sujet[which(nutri6$sexe %in% c(12,21))]
ind6poids <- nutri6$Sujet[which(nutri6$poids==8)]
ind6age  <- nutri6$Sujet[which(nutri6$age==7)]
ind6viande  <- nutri6$Sujet[which(nutri6$viande==6)]
ind6chocol  <- nutri6$Sujet[which(nutri6$chocol==6)]
ind6matgras  <- nutri6$Sujet[which(nutri6$matgras==9)]
ind6anorm <- c(ind6sexe,ind6poids,ind6age,ind6viande,ind6chocol,
               ind6matgras)
```

Nous allons maintenant regarder s'il est possible de corriger certaines de ces valeurs anormales. Pour cela, nous allons chercher si un individu présentant une valeur anormale dans l'une des tables pourrait être présent dans l'autre table avec une valeur normale. Voici déjà la liste des sujets communs aux deux fichiers :

```
ind.comun <- intersect(nutri6$Sujet,nutri5$Sujet)
```

Liste des sujets anormaux dans `nutri5` qui sont aussi présents dans `nutri6` :

```
intersect(ind.comun,ind5anorm)
# Résultat : aucun sujet.
```

Liste des sujets anormaux dans `nutri6` qui sont aussi présents dans `nutri5` :

```
intersect(ind.comun,ind6anorm)
# Résultat: le sujet 30
nutri5[nutri5$Sujet==30,]
nutri6[nutri6$Sujet==30,]
# On a sexe=21 dans nutri6 et sexe=2 dans nutri5
```

On remplace donc cette valeur dans `nutri6` :

```
nutri6$sexe[which(nutri6$Sujet==30)] <- nutri5$sexe[which(
                                          nutri5$Sujet==30)]
```

On peut maintenant regrouper les deux tables :

```
resultat <- merge(nutri5,nutri6,all=TRUE)
```

Sans autre information sur les données anormales, nous décidons de ne pas les changer à ce stade.

3.4-
```
nutri7 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri7.xls")
nutri8 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri8.xls")
resultat <- cbind(nutri7,nutri8)
```

3.5-
```
nutri9 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri9.xls")
nutri10 <- read.xls("http://www.biostatisticien.eu/
                    springeR/nutri10.xls")
apply(nutri9,FUN=table,MARGIN=2,useNA="ifany")
apply(nutri10,FUN=table,MARGIN=2,useNA="ifany")
```

La variable `chocol` contient 29 données manquantes. Nous supprimons cette variable avant de combiner les deux tables :

```
nutri9bis <- nutri9[,-which(colnames(nutri9)=="chocol")]
resultat <- cbind(nutri9bis,nutri10)
```

3.6-
```
nutri11 <- read.xls("http://www.biostatisticien.eu/
                     springeR/nutri11.xls")
nutri12 <- read.xls("http://www.biostatisticien.eu/
                     springeR/nutri12.xls")
# Nombre de valeurs manquantes pour tous les individus
# de chacune des tables:
ind11NA <- apply(is.na(nutri11),FUN=sum,MARGIN=1)
ind12NA <- apply(is.na(nutri12),FUN=sum,MARGIN=1)
# Détermination de l'individu ayant le plus de valeurs manquantes:
ind11max <- which.max(ind11NA) # On trouve l'individu 86.
ind12max <- which.max(ind12NA) # On trouve l'individu 86.
ind11NA[ind11max] # 3 valeurs manquantes.
ind12NA[ind12max] # 4 valeurs manquantes.
```

L'individu 86 possède 7 valeurs manquantes sur 13 variables. Nous le supprimons de la table fusionnée :

```
resultat <- cbind(nutri11,nutri12)[-86,]
```

3.7-
```
nutriage <- read.xls("http://www.biostatisticien.eu/
                      springeR/nutriage.xls")
nrow(nutriage[nutriage$poisson==0 & nutriage$viande==0,])
```

Il n'y a aucun végétarien parmi les individus étudiés.

• Fichier Intima_Media.xls

3.1-
```
Intima <- read.xls("http://www.biostatisticien.eu/
                    springeR/Intima_Media.xls")
Intima <- transform(Intima,IMC=poids/(taille/100)^2)
# ou de façon équivalente:
# IMC <- Intima$poids/((Intima$taille/100)^2)
# Intima <- cbind(Intima,IMC)
```

3.2-
```
Intima$mesure[Intima$IMC>30]
```

3.3-
```
Intima[Intima$SPORT==1,]
```

3.4-
```
Intima[Intima$IMC<=30 & Intima$AGE>=50,]
```

• Fichier imcenfant.xls

3.1-
```
imcenfant <- read.xls(
   "http://www.biostatisticien.eu/springeR/imcenfant.xls")
imcenfant <- transform(imcenfant,IMC=poids/(taille/100)^2)
# ou de façon équivalente:
# IMC <- imcenfant$poids/((imcenfant$taille/100)^2)
# imcenfant <- cbind(imcenfant,IMC)
```

3.2-
```
subset(imcenfant,IMC<15 & an<=3.5 & mois <=5)
# ou de façon équivalente:
# imcenfant[imcenfant$IMC<15 & imcenfant$an <=3 &
#           imcenfant$mois<=5,]
```

3.3-
```
sum(imcenfant$IMC<15 & imcenfant$an<=3 & imcenfant$mois<=5)
# ou de façon équivalente:
# nrow(imcenfant[imcenfant$IMC<15 & imcenfant$an<=3 &
                 imcenfant$mois<=5,])
```

On trouve 7 enfants.

• Fichier Poids_naissance.xls

3.1-
```
url <- "http://www.biostatisticien.eu/
        springeR/Poids_naissance.xls"
poid.nais <- read.xls(url)
PTL1 <- poid.nais$PTL
PTL1[poid.nais$PTL>=2] <- 2
poid.nais <- cbind(poid.nais,PTL1)
```

3.2-
```
FVT1 <- poid.nais$FVT
FVT1[poid.nais$FVT>=2] <- 2
poid.nais <- cbind(poid.nais,FVT1)
```

3.3-
```
poid.nais[order(poid.nais$BWT),]
```

3.4-
```
poid.nais[poid.nais$RACE<=2 & poid.nais$SMOKE==1,]
```

B- Gestion des valeurs manquantes

Installez le *package* `gdata`. Et puisque ce *package* fonctionne avec PERL, installez aussi PERL (`http://www.biostatisticien.eu/springeR/Rtools.exe`). Maintenant, on lit les données :

```
url <- "http://www.biostatisticien.eu/springeR/Infarct.xls"
read.xls(url)
```

On constate que les valeurs manquantes sont codées avec des `"."`. On utilise donc l'instruction suivante :

```
infarct <- read.xls(url,na.strings=".")
```

3.1-
```
indNA <- which(apply(is.na(infarct),FUN=any,MARGIN=1)==TRUE)
```

3.2-
```
sup1 <- function(x) {sum(x)>1}
indNA <- which(apply(is.na(infarct),FUN=sup1,MARGIN=1)==TRUE)
infarct$NUMERO[indNA]
```

3.3-
```
colnames(infarct)[which(apply(is.na(infarct),FUN=any,
                              MARGIN=2) == TRUE)]
```

3.4- a)
```
infarct[as.logical(apply(!is.na(infarct),1,prod)),]
```
b)
```
infarct[!apply(apply(infarct,1,is.na),2,any),]
```
c)
```
infarct[apply(!apply(infarct,1,is.na),2,all),]
```
d)
```
infarct[complete.cases(infarct),]
```
e)
```
na.omit(infarct)
```

C- Gestion des chaînes de caractères

3.1-
```
url <- "http://www.biostatisticien.eu/springeR/dept-pop.csv"
dept <- read.csv(url,dec=",")
```

3.2-
```
numdep <- substring(dept$Departement,1,3)
Dept <- substring(dept$Departement,4)
dept <- cbind(numdep,Dept,dept[,-1])
```

D- Épidémies de grippe en France depuis 1984

3.1-
```
url <- "http://www.biostatisticien.eu/springeR/grippe.csv"
head(read.csv(url))
```

On constate que les valeurs manquantes sont codées par des tirets (-). On utilise donc l'instruction suivante :

```
grippe <- read.csv(url,na.strings="-")
```

3.2-
```
names(grippe)
grippe$Date
```

3.3-
```
unique(sort(substring(grippe$Date,5,6)))
```

On trouve les numéros `"01"` à `"53"`.

3.4-
```
strptime("198444",format="%Y%W")
```

On constate que l'instruction ci-dessus donne la bonne année, mais renvoie en fait le jour et le mois de la date actuelle.

3.5- On lit sur le calendrier que le premier lundi de la 44^e semaine correspond au 29 octobre 1984.

3.6-
```
strptime("1984441",format="%Y%W%w")
```

3.7-
```
grippe$Date[1:10]
strptime(paste(as.character(grippe$Date[1:10]),"1",sep=""),
         format="%Y%W%w")
```

On constate sur le calendrier du site web que cela ne fonctionne pas pour le 31 décembre 1984. Il faudrait plutôt utiliser :

```
strptime("1984531",format="%Y%W%w")
```

alors que l'on a utilisé :

```
strptime("1985011",format="%Y%W%w")
```

Il y a donc un problème avec le format des semaines.

3.8-
```
date1 <- as.POSIXlt("29,10,1984",format="%d,%m,%Y")
```

3.9-
```
date1
date1+7
```

L'opération ci-dessus a ajouté 7 secondes à `date1`.

3.10-
```
date1+7*24*60*60
```

3.11-
```
dates <- date1 + 7*24*60*60*(0:(nrow(grippe)-1))
```

3.12-
```
grippe$Date <- substring(dates,1,10)
```

3.13-
```
masque1 <- (as.POSIXlt(dates) >= as.POSIXlt("1992-09-15"))
masque2 <- (as.POSIXlt(dates) <= as.POSIXlt("1993-11-03"))
portion <- grippe[masque1 & masque2,]
```

3.14-
```
casgrippe <- colSums(portion[,-1],na.rm=TRUE)
casgrippe
```

E- Combinaison de tables ou de listes, autres manipulations

3.1-
```
a <- matrix(1:6,3,2)
rownames(a) <- c(1,2,6)
b <- matrix(1:8,4,2)
rownames(b) <- c(3,4,5,7)
```

3.2-
```
ab <- rbind(a,b)
ab <- ab[order(rownames(ab)),]
ab
```

3.3-
```
list1 <- list()
list1[[1]] <-  matrix(runif(25),nr=5)
list1[[2]] <-  matrix(runif(30),nr=5)
list1[[3]] <-  matrix(runif(15),nr=5)

matrix(unlist(list1),nrow=5)
# ou bien:
do.call(cbind,list1)
```

3.4-
```
list2 <- list()
list2[[1]] <-  matrix(runif(25),nc=5)
list2[[2]] <-  matrix(runif(35),nc=5)
list2[[3]] <-  matrix(runif(15),nc=5)

tmp <- lapply(list2,FUN=t)
t(matrix(unlist(tmp),nrow=5))
# ou bien:
do.call(rbind,list2)
```

3.5-
```
tmp <- data.frame(Maladie = c("Infarctus", "Hépatite",
"Cancer du poumon"),FR = c("tabac, alcool", "alcool", "tabac"))
tmp
tmp$Maladie[grep("tabac",tmp$FR)]
```

F- Le chevalier de Méré

```
– # Fonction auxiliaire
  quatrejets <- function() {
     unsixouplus <- 0
     jeu <- sample(1:6,4,replace=TRUE)
     nbsix = sum(jeu == 6)
     if(nbsix >= 1) unsixouplus <- 1
     return(unsixouplus)
  }
– # Fonction auxiliaire
  vingtquatrejets <- function() {
     undoublesixouplus <- 0
     de1 <- sample(1:6,24,replace=TRUE)
     de2 <- sample(1:6,24,replace=TRUE)
     nbdoublesix <- sum(de1[which(de1 == de2)] == 6)
     if(nbdoublesix >= 1) undoublesixouplus <- 1
     return(undoublesixouplus)
  }
– # Estimation de la probabilité d'un événement
  meresix <- function(nsim = 2000) {
     aumoinsunsix <- 0
     aumoinsundoublesix <- 0
     for(j in 1:nsim) {
     aumoinsunsix <- aumoinsunsix + quatrejets()
     aumoinsundoublesix <- aumoinsundoublesix + vingtquatrejets()
                        }
  #************* Affichage des résultats ******************
     cat("Fréquence des six =",aumoinsunsix/nsim,"\n")
     cat("Fréquence des doubles six =",aumoinsundoublesix/nsim,"\n")
  \}
```

Solutions des TPs du chapitre 4

Où trouver de l'information

4.1- On tape

```
help.search("combinaison")
```

puis on repère la bonne fonction. On choisit celle du *package* `utils` qui est présent dans R par défaut.

```
help(combn)
```

4.2-
```
combn(c(5,8,2,9),3)
```

4.3-
```
help.search("crime")
help(USArrests)
```

4.4-
```
dim(USArrests)      # Dimension du jeu de données.
names(USArrests)    # Noms des variables.
rownames(USArrests) # Noms des individus (états).
```

4.5- Abonnez vous à : https://stat.ethz.ch/mailman/listinfo/r-help

4.6- Lisez les règles à suivre pour poser une question sur http://www.r-project.org/posting-guide.html

4.7- Pour se désabonner, il suffit de suivre les instructions données en bas de la page https://stat.ethz.ch/mailman/listinfo/r-help

4.8- Connectez vous sur IRC, par exemple via le site http://webchat.freenode.net

4.9- Abonnez vous à http://stackoverflow.com/questions/tagged/r

4.10- Parcourir la FAQ pour Windows depuis l'adresse http://cran.r-project.org/faqs.html. Sa section 5.5 en explique le mécanisme.

4.11- Entrez un guillemet (") puis pesez deux fois successivement sur la touche TAB.

Solutions des TPs du chapitre 5

Création de graphiques divers et variés

A- Nombres complexes

5.1-
```
z <- 1+2i
plot(1,3,xlab="Re(z)",ylab="Im(z)",xlim=c(0,2.5),ylim=c(0,2.5),
     main="Nombres complexes")
segments(0,0,Re(z),Im(z))
points(Re(z),Im(z),pch=19)
abline(h=0,v=0)
segments(Re(z),0,Re(z),Im(z),lty=3)
segments(0,Im(z),Re(z),Im(z),lty=3)
text(Re(z),Im(z),"z",pos=4)
text(0.4,1.1,"Mod(z)",srt=Arg(z)*180/pi)
r<-0.5
x<-seq(from=Re(r*exp(1i*Arg(z))),to=r,length=100)
y<-sqrt(0.5^2-x^2)
points(x,y,type="l")
text(0.55,0.35,"Arg(z)",srt=-45,cex=0.8)
```

B- Dessiner le drapeau du Canada

5.1-
```
require("caTools")
```

5.2- Enregistrez le fichier http://www.biostatisticien.eu/springeR/canada.gif dans votre repertoire de travail. Puis tapez l'instruction :

```
X <- read.gif("canada.gif")
```

5.3-
```
image(as.matrix(rev(as.data.frame(t(X$image)))),col=X$col)
```

5.4-
```
coord <- locator(25) # Recupérer les coordonnées des points
                     # constitutifs du contour de la feuille
                     # d'érable.
rec1 <- locator(2)   # Récupérer les coordonnées des sommets
                     # bas-gauche
                     # et haut-droite du rectangle de gauche.
rec2 <- locator(2)   # Récupérer les coordonnées des sommets
                     # bas-gauche
                     # et haut-droite du rectangle de droite.
windows() # ou X11() sous Linux.
```

```
plot(0,xlim=c(0,1),ylim=c(0,1),type="n",ann=FALSE,axes=FALSE)
rect(rec1$x[1],rec1$y[1],rec1$x[2],rec1$y[2],col="red")
rect(rec2$x[1],rec2$y[1],rec2$x[2],rec2$y[2],col="red")
polygon(coord,col="red")
```

C- Graphiques de tables de fréquences

5.1- Rentrez le jeu de données :

```
X <-  matrix(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,1,1,1,1,3,2,1,1,1,1,1,1,2,
1,4,1,1,1,1,1,2,2,1,1,1,1,2,3,3,1,2,1,1,1,1,1,3,4,1,1,1,2,3,4,3,
1,2,1,1,4,1,1,4,4,1,1),ncol=8)
colnames(X) <- c("Nr","W1","W2","W3","W4","W5","W6","W7")
fi <- tabulate(X[,"W7"],4)/16
as.vector(t(cbind(fi,1-fi)))
```

5.2- Créez pour cela une function :

```
f <- function(x){
fi <- tabulate(x,4)/16
as.vector(t(cbind(fi,1-fi)))
}
freq <- apply(X[,-1],FUN=f,MARGIN=2)
rownames(freq) <- c("f1","1-f1","f2","1-f2","f3","1-f3","f4","1-f4")
```

5.3-
```
barplot(freq,col=c("black","white"))
```

5.4-
```
windows() # ou X11() sous Linux.
barplot(freq,col=c("red","white"),width=1,space=0.1,axes=FALSE,
        border="black",names.arg=rep("",7))
axis(2,labels=1:4,at=1:4-0.5,lty=0,las=1,col.axis="blue")
par(cex=.8)
axis(3,labels=c("W1","W2","W3","W4","W5","W6","W7"),
     at=0:6+0:6/10+0.5,lty=0,col.axis="blue")
par(cex=1)
title(main="Scores de brulure",col="black")
```

D- Affichage d'images anatomiques du cerveau

5.1- Lecture des données :

```
dim <- 256
url <- "http://www.biostatisticien.eu/springeR/anat.img"
octets <- readBin(url,what="raw",n=dim*dim*2)
```

5.2- Permutation des paires d'octets :

```
nbval <- dim^2
indices <- rbind((1:nbval)\fois2,(1:nbval)\fois2-1)
indices <- as.numeric(indices)
x <- octets[indices]
```

5.3- Transformation vers les décimaux :

```
x <- matrix(x,ncol=2,byrow=TRUE)
test <- function(x) {
  as.numeric(paste("0x",paste(x,collapse=""),sep=""))
}
valeurs <- apply(x,MARGIN=1,FUN=test)
```

5.4- `X <- matrix(valeurs,nrow=dim,ncol=dim)`

5.5- `image(X,col=gray(0:100 / 100))`

5.6- Installez le *package* `AnalyzeFMRI`. Téléchargez les fichiers anat.img et anat.hdr sur votre espace de travail, puis tapez les instructions suivantes :

```
require("AnalyzeFMRI")
Y <- f.read.volume("anat.img")
image(Y[,,1],col=gray(0:1000 / 1000))
```

E- Dessiner la carte d'une région française

5.1- Installez les *packages* `maps` et `mapdata`, puis chargez les en mémoire :

```
require("maps")
require("mapdata")
```

5.2- `map("france")`

5.3- `france <- map("france",plot=FALSE)`

5.4- Les données de latitude/longitude sont organisées dans `france$x/france$y` pour chaque région dans `france$names` (jusqu'au prochain `NA`).

5.5- Vecteur des indices des `NA` :

```
indNA <- which(is.na(france$x))
```

5.6- `nomdept <- "Gard"`

5.7- `inddept <- which(france$names==nomdept)`

5.8-
```
plot(france$x[indNA[inddept-1]:indNA[inddept]],
     france$y[indNA[inddept-1]:indNA[inddept]],
     type="l",main=nomdept,xlab="Longitude",ylab="Latitude")
```

5.9- Allez récupérer les cooordonnées de la ville d'Alès sur http://www.gpsvisualizer.com/geocode.

```
ales <- c(4.08268,44.121288)
points(ales[1],ales[2],pch=16,col="red")
text(ales[1],ales[2],"Alès",pos=1)
```

F- Représentation du géoïde en France

5.1- Lecture des données :

```
readLines("http://www.biostatisticien.eu/
           springeR/formatgeoide.txt")
data <- scan("http://www.biostatisticien.eu/
              springeR/raf98.gra", skip = 3)
```

5.2-
```
Z <- matrix(data,nrow=421,ncol=381,byrow=FALSE)
Z <- as.matrix(rev(as.data.frame(Z)))
layout(mat=matrix(1:2,ncol=2),widths=c(5,1))
par(mar=c(4,4,3,0),mai=c(1, 1, 1, 0),las=1,tck=-0.01)
x<-seq(from=-5.5,to=8.5,length=421)
y<-seq(from=42,to=51.5,length=381)
image(x,y,Z,col=rainbow(17),xlab="Longitude",ylab="Latitude",
      axes=FALSE)
par(ps=18)
title("Le modèle de quasigéoide QGF98",family="HersheyScript")
par(ps=11)
axis(1,at=(-6):8,labels=paste((-6):8,"o",sep=""))
axis(2,at=42:51,labels=paste(42:51,"o",sep=""))
contour(x,y,Z,add=TRUE)
par(mai=c(0, 0, 0, 0),bty="n")
plot(0:1,0:1,axes=FALSE,xlab="",ylab="",type="n")
legend("center",legend=42:55,fill=rainbow(17),bty="n",
       title="Mètres")
```

Solutions des TPs du chapitre 6

Programmation de fonctions et programmation R orientée objet

A- Gestion d'un compte bancaire

6.1-
```
.repertoires.comptes <- "./Comptes"
chemin.compte <- function(nom) {
```

```
 file.path(.repertoires.comptes,paste(nom,".RData",sep=""))
}
```

6.2-
```
compte <- function(nom) {
 chemin <- chemin.compte(nom)
 compte <- data.frame(somme=numeric(0),mode =
           factor(levels=c("Débit","Crédit")),
           date=character(0),remarque=character(0))
 save(compte,file=chemin)
 cat("Compte",nom,"créé !\n")
}
```

6.3-
```
débite <- function(nom,somme,remarque="",
                   date=format(Sys.time(),"%d/%m/%Y")) {
  chemin <- chemin.compte(nom)
  load(chemin)
  compte <- rbind(compte,data.frame(somme=somme,mode="Débit",
                 date=date,remarque=remarque))
  save(compte,file=chemin)
}

crédite <- function(nom,somme,remarque="",
                    date=format(Sys.time(),"%d/%m/%Y")) {
  chemin <- chemin.compte(nom)
  load(chemin)
  compte <- rbind(compte,data.frame(somme=somme,mode="Crédit",
                 date=date,remarque=remarque))
  save(compte,file=chemin)
}
```

6.4- L'instruction

```
sum(compte[compte$mode=="Crédit","somme"])
```

renvoie la somme créditée sur le compte. Nous modifions la fonction `compte()` ainsi :

```
compte <- function(nom) {
  chemin <- chemin.compte(nom)
  if(!file.exists(chemin)) {
    compte <- data.frame(somme=numeric(0),mode=
         factor(levels=c("Débit","Crédit")),date=character(0),
         remarque=character(0))
    save(compte,file=chemin)
    cat("Compte",nom,"créé !\n")
  } else {
    load(chemin)
    cat("Etat compte",nom,"=",sum(compte[compte$mode=="Crédit",
```

```
                                    "somme"])
          - sum(compte[compte$mode=="Débit","somme"]),"euros.\n")
  }
}
```

B- Organisation d'objets graphiques

6.1-
```
Fenêtre <- function(x=0,y=0,largeur=2,hauteur=2,log="") {
  obj <- list(x=x,y=y,largeur=largeur,hauteur=hauteur,log=log)
  class(obj) <- "Fenêtre"
  obj
}
```

6.2-
```
Cercle <- function(x=0,y=0,rayon=0.5) {
  cercle <- list(x=x,y=y,rayon=rayon)
  class(cercle) <- "Cercle"
  cercle
}

Rectangle <- function(x=0,y=0,largeur=1,hauteur=largeur) {
  rectangle <- list(x=x,y=y,largeur=largeur,hauteur=hauteur)
  class(rectangle) <- "Rectangle"
  rectangle
}
```

6.3-
```
plot.Fenêtre <- function(obj) {
  plot.new()
  plot.window(xlim=obj$x+c(-1,1)*obj$largeur/2,
    ylim=obj$y+c(-1,1)*obj$hauteur/2,obj$log,asp=1)
}

plot.Rectangle <- function(rectangle) {
  rect(rectangle$x-rectangle$largeur/2,
       rectangle$y-rectangle$hauteur/2,
       rectangle$x+rectangle$largeur/2,
       rectangle$y+rectangle$hauteur/2)
}

plot.Cercle <- function(cercle) {
 symbols(cercle$x,cercle$y,circle=cercle$rayon,
         inches=FALSE,add=TRUE)
}
```

6.4-
```
fenêtre <- Fenêtre(0,0,2,2)
cercle <- Cercle(0,0,.5)
```

```
rectangle <- Rectangle(0,0,1,1)
plot(fenêtre);plot(cercle);plot(rectangle)
```

6.5-
```
Graphe <-function(x=0,y=0,largeur=2,hauteur=2,log="") {
  graphe <- list(objets=list())
  class(graphe) <- "Graphe"
  graphe
}
```

6.6-
```
ajout.Graphe <- function(graphe,...) {
  graphe$objets <- c(graphe$objets,list(...))
  graphe
}

ajout <- function(obj,...) UseMethod("ajout")

plot.Graphe <- function(graphe) {
  for(objet in graphe$objets) {
    plot(objet)
  }
}

graphe <- Graphe()
graphe <- ajout(graphe,Fenêtre(0,0,2,2),Cercle(0,0,.5),
                Rectangle(0,0,1,1))
plot(graphe)
```

6.7-
```
Graphe <- function(...,x=0,y=0,largeur=2,hauteur=2,log="") {
  graphe <- list(objets=list())
  class(graphe) <- "Graphe"
  graphe <- ajout(graphe,Fenêtre(x,y,largeur,hauteur,log),...)
  graphe
}

graphe <- Graphe(Cercle(),Rectangle())
plot(graphe)
```

6.8-
```
affiche <- function(obj,...) UseMethod("plot")
graphe <- Graphe(Cercle(),Rectangle())
affiche(graphe)
```

6.9- À vous de jouer !

C- Création d'une classe `lm2` pour la régression linéaire avec deux régresseurs

```
lm2 <- function(...) {
  obj <- lm(...)
  if(ncol(model.frame(obj))!=3)
    stop("two independent variables are required!")
  class(obj) <- c("lm2",class(obj)) # ou c("lm2","lm")
  obj
}

n <- 20
x1 <- runif(n,-5,5)
x2 <- runif(n,-50,50)
y <- .3+2*x1+2*x2+rnorm(n,0,20)
reg2 <- lm2(y~x1+x2)
summary(reg2)

plot3d.lm2 <- function(obj,radius=1,lines=TRUE,windowRect,...) {
  matreg <- model.frame(obj)
  colnames(matreg) <- c("y","x1","x2")
  predlim <- cbind(c(range(matreg[,2]),rev(range(matreg[,2]))),
  rep(range(matreg[,3]),c(2,2)))
  predlim <- cbind(predlim,apply(predlim,1,
    function(l) sum(c(1,l)*coef(obj))
  ))
  if(missing(windowRect)) windowRect=c(2,2,500,500)
  open3d(windowRect=windowRect,...)
  bg3d(color = "gray")
  plot3d(formula(obj),type="n")
  spheres3d(formula(obj),radius=radius,specular="green")
  quads3d(predlim,color="blue",alpha=0.7,shininess=128)
  quads3d(predlim,color="cyan",size=5,front="lines",back="lines",
          lit=FALSE)
  if(lines) {
    matpred <- cbind(matreg[2:3],model.matrix(obj)%*%coef(obj))
    points3d(matpred)
    colnames(matpred) <- c("x1","x2","y")
    matlines <- rbind(matreg[,c(2:3,1)],matpred)
    nr <- nrow(matreg)
    matlines <- matlines[rep(1:nr,rep(2,nr))+c(0,nr),]
    segments3d(matlines)
  }
}
```

```
require("rgl")
plot3d(reg2)

rgl.snapshot("lm2vue1.png")
par3d(userMatrix=rotate3d(par3d("userMatrix"),-pi*.1, 0, 0, 1))
rgl.close()
```

Solutions des TPs du chapitre 7

Maintenance et création de *packages*

A- Utilisation des fonctions `attach()` et `detach()`

7.1- Téléchargez le fichier `http://www.biostatisticien.eu/springeR/imcenfant.xls`, au moyen de votre navigateur favori, dans votre répertoire de travail.

7.2- Installez le *package* `gdata`. Puisque ce *package* fonctionne avec PERL, installez aussi PERL (`http://www.biostatisticien.eu/springeR/Rtools.exe`).

```
require("gdata")
imcenfant <- read.xls("imcenfant.xls")
names(imcenfant)
```

7.3- En tapant `SEXE`, R nous répond `Erreur : objet 'SEXE' introuvable`.

7.4- En tapant `ls()`, on ne voit pas la variable `SEXE`.

7.5-
```
attach(imcenfant)
SEXE
```

Le contenu de l'objet `SEXE` apparaît.

7.6- La variable `SEXE` n'est toujours pas visible au moyen de l'instruction `ls()`.

7.7-
```
search()
```

On constate que l'objet `imcenfant` apparaît en position 2.

7.8-
```
ls(pos=2)
```

7.9-
```
detach(imcenfant)
search()
SEXE
```

7.10-
```
SEXE <- "Homme"
SEXE
```

7.11-
```
attach(imcenfant)
```

Un message d'avertissement s'affiche.

```
SEXE
```

C'est `"Homme"` qui s'affiche mais pas le contenu de l'objet `SEXE` du *data.frame*.

7.12-
```
SEXE # n'affiche pas le contenu de l'objet SEXE du data.frame.
poids
```

7.13-
```
ls() # imcenfant et SEXE sont présents.
search() # imcenfant est présent.
```

7.14-
```
ls(pos=2)
```

7.15-
```
get("SEXE",pos=2)
imcenfant[,"SEXE"]
# ou
rm(SEXE)
SEXE
# Notez l'existence de la fonction suivante:
browseEnv()
```

B- création d'un mini-*package*

7.1-
```
setwd(choose.dir())
# ou bien sous Linux:
library("tcltk")
setwd(tk_choose.dir())
```

7.2- Création des deux fonctions et des deux jeux de données :

```
f <- function(x,y) x+y
g <- function(x,y) x-y
d <- data.frame(a=1,b=2)
e <- rnorm(1000)
```

7.3-
```
package.skeleton(name="PetitPkgR",list=c("f","g","d","e"))
```

7.4- Création effective du fichier du *package*. Modifiez impérativement les fichiers d'aides `.Rd`.

```
file.show(file.path(R.home("doc"),"KEYWORDS"))
```

7.5- Modifiez le fichier DESCRIPTION.

7.6- Lire puis effacer le fichier Read-and-delete-me.

7.7- Vérification (et éventuellement modification) de la variable d'environnement `PATH`.

7.8- Dans une invite de commande DOS, tapez les instructions suivantes :

```
# Placez vous dans le dossier contenant votre package:
cd C:\Documents and Settings\dupont\Bureau
R CMD check PetitPkgR
R CMD INSTALL --build PetitPkgR
```

7.9- Installez votre *package* PetitPkgR.zip. Pour cela, tapez par exemple :

```
help(package="PetitPkgR")
help(d)
```

Solutions des TPs du chapitre 8

Calcul matriciel, optimisation, intégration

A- Un premier problème d'optimisation

8.1-
```
A <- matrix(c(2,3,5,4),nrow=2,ncol=2)
myf <- function(x) {
    res <- det(A-x*diag(2))
    return(res)
}
```

8.2-
```
myf <- function(x) {
    n <- length(x)
    res <- rep(NA,n)
    for (i in 1:n) res[i] <- det(A-x[i]*diag(2))
    return(res)
}
```

8.3-
```
curve(myf,xlim=c(-10,10))
abline(h=0,v=0)
```

Notez que l'on peut utiliser l'instruction `locator(2)$x` pour « trouver » les deux racines.

8.4-
```
uniroot(myf,lower=-5,upper=0)$root
uniroot(myf,lower=0,upper=10)$root
```

8.5- Le polynome s'écrit $P(x) = -7 - 6x + x^2$. Ses racines sont obtenues ainsi :

```
polyroot(c(-7,-6,1))
```

8.6-
```
eigen(A)$values
```

B- Un second problème d'optimisation

8.1- Pour simplifier les choses, nous ne reproduisons pas la figure tout à fait à l'échelle.

```
plot.new()
par(xpd=NA)
rect(0,0,1,1)
points(0,1,pch=16)
text(0,1,"A",adj=c(2,-0.1),col="blue")
points(1,1,pch=16)
text(1,1,"B",adj=c(-1,0),col="blue")
points(1,0,pch=16)
text(1,0,"C",adj=c(-0.5,1),col="blue")
points(0,0,pch=16)
text(0,0,"D",adj=c(1.7,1),col="blue")
points(0.5,0,pch=16)
text(0.5,0,"H",adj=c(1.5,-0.5),col="blue")
points(1,0.4,pch=16)
text(1,0.4,"Q",pos=4,col="blue")
points(0.5,0.4,pch=16)
text(0.5,0.4,"M",adj=c(0.5,-1),col="blue")
segments(0.5,0,0.5,0.4)
segments(0.5,0.4,1,0.4)
polygon(x=c(0.5,0,1),y=c(0.4,1,1),col=rgb(t(col2rgb("brown"))/256,
        alpha=20/100),border="brown")
x <- seq(from=0.6,to=0.565,length=100)
points(x,sqrt(0.1^2-(x-0.5)^2)+0.4,type="l")
text(0.61,0.45,expression(alpha))
```

8.2- On a $\cos(\alpha) = MQ/MB$ donc $MB = 5/\cos(\alpha)$ et $MA = MB$. $MH = BC - BQ$, $\tan(\alpha) = BQ/MQ = BQ/(AB/2)$ donc $BQ = 5\tan(\alpha)$ et $MH = 6 - 5\tan(\alpha)$. D'où $g(\alpha) = 10/\cos(\alpha) + 6 - 5\tan(\alpha) = (10 - 5\sin(\alpha))/\cos(\alpha) + 6 = 5(2 - \sin(\alpha))/\cos(\alpha) + 6$.

8.3-
```
g <- function(alpha) {5*(2-sin(alpha))/cos(alpha)+6}
```

8.4-
```
optimize(g,lower=0,upper=pi/2,tol=0.000001)
```

8.5- $g'(\alpha) = 5(-\cos^2(\alpha) - (2 - \sin(\alpha))(-\sin(\alpha)))/\cos^2(\alpha) = 5(-\cos^2(\alpha) + 2\sin(\alpha) - \sin^2(\alpha))/\cos^2(\alpha) = 5(2\sin(\alpha) - 1))/\cos^2(\alpha)$ car $\cos^2(\alpha) + \sin^2(\alpha) = 1$.

8.6-
```
D(expression(5*(2-sin(alpha))/cos(alpha)+6),"alpha")
```

8.7-
```
gprime <- function(alpha) {5*(2*sin(alpha)-1)/(cos(alpha))^2}
```

8.8-
```
uniroot(gprime,lower=0,upper=pi/2,tol=0.000001)$root
```

C- Table de la loi normale centrée-réduite

8.1-
```
phi <- function(x) {exp(-x^2/2)/sqrt(2*pi)}
```

8.2-
```
quantiles <- seq(0,5.5,by=0.01)
n <- length(quantiles)
probs <- vector(mode="numeric",length=n)
for (i in 1:n) probs[i] <- integrate(phi,lower=-Inf,
     upper=quantiles[i],rel.tol=0.00001)$value
```

8.3-
```
all.equal(probs,pnorm(quantiles))
```

8.4-
```
plot(c(rev(-quantiles),quantiles),c(rev(1-probs),probs),type="l")
```

8.5-
```
curve(pnorm(x),add=TRUE,col="blue")
```

D- Une analyse en composantes principales

8.1-
```
url <- "http://www.biostatisticien.eu/springeR/climatvin.csv"
climatvin <- read.table(url,sep="\t",header=TRUE)
attach(climatvin)
names(climatvin)
```

8.2-
```
X <- as.matrix(climatvin[,-c(1,6)])
```

8.3-
```
g <- colMeans(X)
round(g,2)
```

8.4-
```
Xpoint <- scale(X,scale=FALSE)
```

8.5-
```
n <- nrow(X)
inertie <- sum(Xpoint^2)/n
inertie
```

8.6-
```
contrinertie <- rowSums(Xpoint^2)/n/inertie
contrinertie
```

8.7-
```
unn <- as.matrix(rep(1,n))
```

8.8-
```
g
(g <- t(X)%*%unn/n)
```

8.9-
```
Xpoint
(Xpoint <- X-unn%*%t(g))
```

8.10-
```
(S <- t(Xpoint)%*%Xpoint/n)
cov(X)*(n-1)/n
```

8.11-
```
Dunsurs <- diag(1/sqrt(diag(S)))
```

8.12-
```
Z <- Xpoint%*%Dunsurs
```

8.13-
```
t(Z)%*%Z/n
(R <- cor(X))
```

8.14-
```
Lambda <- diag(eigen(R)$values)
W <- eigen(R,symmetric=TRUE)$vectors
```

8.15-
```
theta <- seq(0,2*pi,.05)
x <- cos(theta)
y <- sin(theta)
plot(x,y,type="l")
abline(h=0,v=0)
A <- W%*%Lambda^(1/2)
text(A[,1:2],colnames(X))
arrows(rep(0,4),rep(0,4),A[,1],A[,2],length=0.1)
```

8.16-
```
p <- ncol(X)
cumsum(diag(Lambda))/p*100
```

Les deux premiers axes expliquent 87.6 % de l'inertie totale.

8.17-
```
CW <- Z%*%W
```

8.18-
```
dev.new()
plot(CW[,c(1,2)],type="p",xlab="Axe 1",ylab="Axe 2")
text(CW[,c(1,2)],labels=ANNEE)
abline(h=0,v=0)
```

8.19-
```
plot(CW[,c(1,2)],type="n",xlab="Axe 1",ylab="Axe 2")
bon <- CW[QUALITE==1,c(1,2)]
moyen <- CW[QUALITE==2,c(1,2)]
mediocre <- CW[QUALITE==3,c(1,2)]
text(bon,labels=ANNEE[QUALITE==1],col="red")
text(moyen,labels=ANNEE[QUALITE==2],col="blue")
text(mediocre,labels=ANNEE[QUALITE==3],col="green")
abline(h=0,v=0)
legend("topleft",c("bon","moyen","mediocre"),fill =
                                c("red","blue","green"))
```

8.20-
```
QLT <- apply(sweep(CW^2,1,apply(CW^2,FUN=sum,1),FUN="/")[,1:2],
             FUN=sum,1)
round(QLT,2)
```

8.21-
```
require("ade4")
```

8.22-
```
rownames(X) <- climatvin[,1]
res <- dudi.pca(X) # Entrez 2 pour le nombre d'axes.
scatter(res)
s.class(res$li,as.factor(climatvin[,6]))
```

Solutions des TPs du chapitre 9

Études descriptives de données

A- Réflexion sur l'indépendance en statistique descriptive

9.1-
```
url <- "http://www.biostatisticien.eu/springeR/snee74.txt"
snee <-read.table(url,header=TRUE)
```

9.2-
```
head(snee)
tail(snee)
dim(snee)
str(snee)
```

Il y a 592 individus et 3 variables qualitatives.

9.3-
```
attach(snee)
names(snee)
class(cheveux)
levels(cheveux) # Modalités de cheveux.
class(yeux)
levels(yeux)    # Modalités de yeux.
class(sexe)
levels(sexe)    # Modalités du sexe.
```

9.4- Étude de la variable `cheveux` :

```
table(cheveux) # Effectifs.
# Fréquences en pourcentages:
round(table(cheveux)/length(cheveux)*100,2)
names(which.max(table(cheveux))) # Mode de la variable.
barplot(sort(table(cheveux),TRUE),
        col=c("yellow2","tan4","black","tan1"))
```

Étude de la variable `yeux` :

```
table(yeux) # Effectifs.
# Fréquences en pourcentages:
round(table(yeux)/length(yeux)*100,2)
names(which.max(table(yeux))) # Mode de la variable.
barplot(sort(table(yeux),TRUE),
        col=c("blue","brown","tan3","green"))
```

Étude de la variable `sexe` :

```
table(sexe) # Effectifs.
# Fréquences en pourcentages:
round(table(sexe)/length(sexe)*100,2)
names(which.max(table(sexe))) # Mode de la variable.
barplot(sort(table(sexe),TRUE),col=c("pink","blue"),
        pareto=TRUE)
```

9.5-
```
yeuxcheveux <- table(yeux,cheveux) # Table de contingence.
```

9.6-
```
fchev <- margin.table(yeuxcheveux,2)/592 # Profils colonnes.
fchev
```

9.7-
```
# Nombre d'individus ayant les yeux bleux:
nbleus <- margin.table(yeuxcheveux,1)[1]
nbleus
```

9.8-
```
round(fchev*nbleus)
```

9.9-
```
margeX <- margin.table(yeuxcheveux,1)
tab.ind1 <- margeX%*%t(fchev)
round(tab.ind1)
```

9.10-
```
fyeux <- margin.table(yeuxcheveux,1)/592 # Profils lignes.
fyeux
```

9.11-
```
# Nombre d'individus ayant les cheveux blonds:
nblonds <- margin.table(yeuxcheveux,2)[1]
nblonds
```

9.12-
```
round(fyeux*nblonds)
```

9.13-
```
margeY <- margin.table(yeuxcheveux,2)
tab.ind2 <- fyeux%*%t(margeY)
round(tab.ind2)
```

9.14-
```
all.equal(tab.ind1,tab.ind2)
```

Les deux tableaux sont identiques, ce qui est rassurant puisque s'intéresser à l'indépendance des yeux et des cheveux est équivalent à s'intéresser à l'indépendance des cheveux et des yeux.

9.15-
```
(yeuxcheveux-tab.ind1)^2
```

9.16-
```
tab.contr <- (yeuxcheveux-tab.ind1)^2/tab.ind1
tab.contr
```

9.17-
```
khi2 <- sum(tab.contr)
khi2

Phi2 <- khi2/sum(yeuxcheveux)
Phi2
```

```
C <- sqrt(khi2/(sum(yeuxcheveux)+khi2))
C

V2 <- Phi2/(min(dim(yeuxcheveux))-1)
V2
```

Tous ces indicateurs sont non nuls ; il y a donc une certaine forme de dépendance dans cette population (étudiée au sens de la statistique descriptive).

9.18-
```
# Distributions conditionnelles de la variable cheveux
# sachant que yeux = (bleu, ou marron ou ...):
prop.table(yeuxcheveux,1)
```

Les lignes ne sont pas égales. Ceci confirme la dépendance (d'un point de vue statistique descriptive) entre les deux caractères.

```
# Distributions conditionnelles de la variable yeux
# sachant que cheveux = (blond ou ...):
prop.table(yeuxcheveux,2)
```

Les colonnes ne sont pas égales. Ceci confirme aussi la dépendance (d'un point de vue statistique descriptive) entre les deux caractères.

9.19-
```
table(cheveux,sexe)
plot(cheveux~sexe)
plot(table(cheveux,sexe))
```

9.20-
```
table(yeux,sexe)
plot(yeux~sexe)
plot(table(yeux,sexe))
```

9.21- Voir le TP dans le chapitre 2 pour savoir comment importer le tableau, puis suivre les étapes ci-dessus.

B- Analyse descriptive du jeu de données NUTRIAGE

9.1-
```
require("gdata")
url <- "http://www.biostatisticien.eu/springeR/nutriage.xls"
nutriage <- read.xls(url,header=TRUE)
attach(nutriage)
```

9.2-
```
names(which.max(table(situation)))
names(which.max(table(chocol)))
names(which.max(table(taille)))
```

9.3-
```
res <- hist(taille,breaks=seq(140,190,by=5),right=TRUE,plot=FALSE)
ind <- which.max(res$count)
classe.modale <- paste(res$breaks[ind],res$breaks[ind+1],sep="-")
```

La classe modale est la classe]155; 160].

9.4-
```
median(chocol)
```

Notez que si la variable étudiée est considérée comme une variable ordinale, alors la fonction `median()` de R ne fonctionne plus :

```
median(as.ordered(chocol))
```

Mais on peut proposer une fonction maison pour réaliser cette opération :

```
ma.mediane <- function(x) {
 if (is.numeric(x)) return(median(x))
 if (is.ordered(x)) {
  N <- length(x)
  if (N%%2) return(sort(x)[(N+1)/2]) else {
  inf <- sort(x)[N/2]
  sup <- sort(x)[N/2+1]
  if (inf==sup) return(inf) else return(list(inf,sup))
 }}
stop("Calcul de médiane impossible pour ce type.")
}
ma.mediane(as.ordered(chocol))
```

9.5-
```
table(chocol)
table(fruit_crus)
```

9.6- On peut utiliser la fonction maison suivante :

```
med <- function(tabx) names(which.max(cumsum(tabx)/sum(tabx)>0.5))
med(table(chocol))
med(table(fruit_crus))
```

9.7-
```
quartile.sur.freq <- function(tabx,quart) {
 # tabx est le tableau des effectifs.
 tab.freq.cum <- cumsum(tabx)/sum(tabx)
 index <- order(tab.freq.cum < quart)[1]
 f1 <- tab.freq.cum[index]
 f2 <- tab.freq.cum[index-1]
 x1 <- as.numeric(names(f1))
 x2 <- as.numeric(names(f2))
 quartile <- as.numeric(x1 + (x2-x1)*(quart-f1)/(f2-f1))
return(quartile)
}

tab <- res$counts
names(tab) <- res$breaks[-1]

quartile.sur.freq(tab,0.25)
quartile.sur.freq(tab,0.5)
quartile.sur.freq(tab,0.75)
```

9.8-
```
bornes <- res$breaks
plot(bornes,ecdf(taille)(bornes),type="l",main=
     paste("Polygone des fréquences cumulées",
     "de la variable taille",sep="\n"),ylab="Fréquences",
     col="darkolivegreen",lwd=3)
abline(h=c(0.25,0.5,0.75))
locator(1)$x # Cliquez sur l'intersection recherchée.
```

9.9-
```
mean(taille)
mean(poids)
mean(age)
```

9.10-
```
table(the)
sum(c(0:6,9,10)*as.numeric(table(the)))/sum(table(the))
```

9.11-
```
sum(res$mids*res$counts)/sum(res$counts)
```

9.12-
```
diff(range(poids))
```

9.13-
```
boxplot(poids)
```

9.14-
```
var.pop <- function(x) var(x)*(length(x)-1)/length(x)
sd.pop <- function(x) sqrt(var.pop(x))
sd.pop(taille)
```

9.15-
```
coeffvar.tab <- function(tabx) {
 val <- as.numeric(names(tabx))
 freq <- as.numeric(tabx)/sum(tabx)
 moy <- sum(val*freq)
 var <- sum(val^2*freq) - moy^2
 res <- sqrt(var)/moy
return(res)
}
coeffvar.tab(table(the))
```

9.16-
```
eta2 <- function(x, gpe) {
moyennes <- tapply(x, gpe, mean)
effectifs <- tapply(x, gpe, length)
varinter <- (sum(effectifs * (moyennes - mean(x))^2))
vartot <- (var(x) * (length(x) - 1))
res <- varinter/vartot
list(var.tot=vartot,var.inter=varinter,
     var.intra=vartot-varinter,eta2=res)
}

res <- eta2(the,sexe)
```

Solutions des TPs du chapitre 10

Simulations

A- Étude de la loi $f(x) = \frac{3}{2}\sqrt{x}$ sur $[0,1]$

10.1-
```
f <- function(x) (3/2)*sqrt(x)
integrate(f,lower=0,upper=1)
```

10.2- Méthode de la transformation inverse :

```
Finv <- function(x) x^(2/3)
u <- runif(1000)
x <- Finv(u)
```

10.3-
```
mean(x)
var(x)
```

10.4- Le calcul analytique donne $\mathbb{E}(X) = 3/5$ et $\mathbb{V}\text{ar}(X) = 12/175$.

10.5- La fonction de répartition vaut $F(x) = x^{2/3}$. Les probabilités théoriques des différentes classes valent : $0.3^{3/2} = 0.1643168$, $0.5^{3/2} - 0.3^{3/2} = 0.1892366$, $0.7^{3/2} - 0.5^{3/2} = 0.2321086$, $0.85^{3/2} - 0.7^{3/2} = 0.1979993$ et $1 - 0.85^{3/2} = 0.2163387$. On peut les évaluer numériquement ainsi :

```
classe <- c(0.3,0.5,0.7,0.85)
prob <- function(x) {integrate(f,lower=0,upper=x)$value}
repartition <- c(0,apply(as.matrix(classe),MARGIN=1,FUN=prob),1)
(prob.theorique <- diff(repartition))
```

Les probabilités empiriques peuvent être calculées en utilisant le code suivant :

```
compte <- function(x,vect.y) sum(vect.y<x)/length(vect.y)
repartition.emp <-  c(0,apply(as.matrix(classe),MARGIN=1,
                      FUN=compte,x),1)
(prob.emp <- diff(repartition.emp))
```

ou le code suivant :

```
c(mean(0<=x & x<=0.3),mean(0.3<x & x<=0.5),mean(0.5<x & x<=0.7),
  mean(0.7<x & x<=0.85),mean(0.85<x & x<=1))
```

B- Étude sur la loi *generalized Pareto*

10.1-
```
rGP <- function(n,mu,sigma,xi) mu+sigma*((runif(n))^(-xi)-1)/xi
```

10.2-
```
n <- 1000
simu <- rGP(n,0,1,0.25)
```

10.3-
```
mean(simu)
var(simu)
```

10.4-
```
mu <- 0
sigma <- 1
xi <- 0.25
(mu.theo <- mu + sigma/(1-xi))
(var.theo <- (sigma^2)/(((1-xi)^2)*(1-2*xi)))
```

10.5-
```
n <- 10000
simu.new <- rGP(n,0,1,0.25)
mean(simu.new)
var(simu.new)
```

10.6-
```
hist(simu.new,breaks=500,xlim=c(0,10),prob=TRUE,col="red")
```

10.7-
```
dens.pareto <- function(x,mu,sigma,xi) {
 (1/sigma)*((1+(xi*(x-mu))/sigma)^(-1/xi-1))
}
curve(dens.pareto(x,mu,sigma,xi),add=TRUE,col="blue")
```

C- Uniforme sur un carré

10.1-
```
n <- 1000
simu <- data.frame(X1=runif(n),X2=runif(n))
```

10.2- La distance de (X_1, X_2) au côté le plus proche est donnée par $D = \min(X_1, X_2, 1 - X_1, 1 - X_2)$ (faites un dessin pour vous en convaincre).

```
simu.d <- cbind(simu,1-simu)
D <- apply(simu.d,MARGIN=1,FUN=min)
mean(D<0.25)
```

10.3-
```
dist.som <- function(coord){
 d <- min(coord[1]^2+coord[2]^2,coord[1]^2+coord[4]^2,
          coord[3]^2+coord[2]^2,coord[3]^2+coord[4]^2)
 d <- sqrt(d)
}

D.som <- apply(simu.d,MARGIN=1,FUN=dist.som)
mean(D.som<0.25)
```

10.4-
```
mean(D)
var(D)
hist(D,prob=TRUE)
```

```
curve(-8*x+4,add=TRUE)
# Densité sur [0,0.5] : f(x)=-8x+4
```

D- Vers la notion de modélisation

10.1-
```
X <- function() ifelse(rbinom(1,1,0.5),"PILE","FACE")
X()
```

10.2-
```
lancer.le.dé <- function() sample(1:6,1)
lancer.le.dé()
```

10.3-
```
yams <- function() sample(1:6,size=5,replace=TRUE)
```

10.4-
```
n <- 1000000
result <- replicate(n,yams(),TRUE)
test <- function(res) ifelse(length(unique(res))> 1,0,1)
prop <- mean(apply(result,MARGIN=2,FUN=test))
```

E- Théorème de Box et Muller

10.1-
```
n <- 1000
u1 <- runif(n)
u2 <- runif(n)
z1 <- sqrt(-2*log(u1))*cos(2*pi*u2)
z2 <- sqrt(-2*log(u1))*sin(2*pi*u2)
```

10.2- Estimation non paramétrique de la densité :

```
require("MASS")
ngrid <- 40
denobj<-kde2d(z1,z2,n=ngrid)
den.z <-denobj$z
```

10.3- Surface de densité de la normale bivariée :

```
require("rgl")
xgrid <- denobj$x
ygrid <- denobj$y
bi.z <- dnorm(xgrid)%*%t(dnorm(ygrid))
```

Nouvelle fenêtre :

```
open3d()
bg3d(color="#887777")
light3d()
```

Représentation des données simulées :

```
spheres3d(z1,z2,rep(0,n),radius=0.1,color="#CCCCFF")
```

Représentation de la densité estimée de façon non paramétrique :

```
surface3d(xgrid,ygrid,den.z*20,color="#FF2222",alpha=0.5)
```

Représentation de la loi bivariée normale centrée et réduite :

```
surface3d(xgrid,ygrid,bi.z*20,color="#CCCCFF",front="lines")
```

Solutions des TPs du chapitre 11

A- Étude sur les intervalles de confiance

• Étude de l'intervalle de confiance de la moyenne

11.1-
```
n <- 20
M <- 50000
ech <- rnorm(n,mean=-1.2,sd=sqrt(2)) # Seulement un échantillon.
simu <- replicate(M,rnorm(n,mean=-1.2,sd=sqrt(2)),simplify=TRUE)
```

11.2-
```
inter.mu <- function(x) t.test(x,conf.level=0.9)$conf.int
res <- t(apply(simu,MARGIN=2,FUN=inter.mu))
```

11.3-
```
prop <- mean((res[,1] < -1.2) & (-1.2 < res[,2]))
```

Environ 90 % des intervalles contiennent la valeur -1.2.

11.4-
```
n <- 100
M <- 50000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
res.chi2 <- t(apply(simu,MARGIN=2,FUN=inter.mu))
prop <- mean((res.chi2[,1] < 1) & (1 < res.chi2[,2]))
```

11.5-
```
n <- 10
M <- 50000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
res.chi2 <- t(apply(simu,MARGIN=2,FUN=inter.mu))
prop <- mean((res.chi2[,1] < 1) & (1 < res.chi2[,2]))
```

Le niveau de confiance (empirique) de l'intervalle n'est pas égal à 90 %. Cela est dû à la faible taille de l'échantillon ($n = 10$), qui est insuffisante pour utiliser le théorème central limite.

11.6-
```
n <- 20
ech.20 <- rnorm(n,mean=-1.2,sd=sqrt(2))
inter.mu <- function(x) t.test(x,conf.level=0.95)$conf.int
inter.mu(ech.20)
```

11.7-
```
ech.50 <- rnorm(50,mean=-1.2,sd=sqrt(2))
inter.mu(ech.50)
ech.100 <- rnorm(100,mean=-1.2,sd=sqrt(2))
inter.mu(ech.100)
ech.1000 <- rnorm(1000,mean=-1.2,sd=sqrt(2))
inter.mu(ech.1000)
ech.10000 <- rnorm(10000,mean=-1.2,sd=sqrt(2))
inter.mu(ech.10000)
ech.100000 <- rnorm(100000,mean=-1.2,sd=sqrt(2))
inter.mu(ech.100000)
```

L'intervalle de confiance se resserre autour de la vraie valeur de μ.

11.8-
```
tt.20 <- c(t.test(ech.20)$st,t.test(ech.20)$p.val)
tt.50 <- c(t.test(ech.50)$st,t.test(ech.50)$p.val)
tt.100 <- c(t.test(ech.100)$st,t.test(ech.100)$p.val)
tt.1000 <- c(t.test(ech.1000)$st,t.test(ech.1000)$p.val)
tt.10000 <- c(t.test(ech.10000)$st,t.test(ech.10000)$p.val)
tt.100000 <- c(t.test(ech.100000)$st,t.test(ech.100000)$p.val)
sortie <- rbind(tt.20,tt.50,tt.100,tt.1000,tt.10000,tt.100000)
colnames(sortie) <- c("tobs","pvaleur")
sortie
```

Plus n augmente, plus t_{obs} devient grand et la valeur-p petite. Il devient de plus en plus facile de « démontrer » que μ est différent de 0.

11.9-
```
inter.mu(ech.100000)
t.test(ech.100000,mu=-1.1)$p.val
tt.50
```

La valeur-p est plus petite ici alors que -1.1 est plus proche du vrai μ ($= -1.2$) que la valeur de référence 0 utilisée à la question précédente. Par contre, l'IC ne trompe pas. On voit bien l'intérêt des IC par rapport à juste donner la valeur-p !

• Étude de l'intervalle de confiance par la méthode du *bootstrap*

11.1-
```
n <- 20
M <- 500
ech <- rexp(n,rate=1/10) # seulement un échantillon
simu.exp <- replicate(M,rexp(n,rate=1/10),simplify=TRUE)
```

11.2- Pour 1 échantillon :

```
require("boot")
moyenne <- function(x,indices) mean(x[indices])
moy.boot <- boot(simu.exp[,3],moyenne,R=999,stype="i",
                 sim="ordinary")
boot.ci(moy.boot,conf=0.9, type="perc")$percent[4:5]
```

Pour les M échantillons :

```
inter.boot.moy <- function(y) {
 moy <- boot(y,moyenne,R=999,stype="i",sim="ordinary")
 boot.ci(moy,conf=0.9,type="perc")$percent[4:5]
}

res <- t(apply(simu.exp,MARGIN=2,FUN=inter.boot.moy))
```

11.3-
```
prop.boot <- mean((res[,1] < 10) & (res [,2] > 10))
```

11.4-
```
res.ttest <- t(apply(simu.exp,MARGIN=2,FUN=inter.mu))
prop.ttest <- mean((res.ttest[,1] < 10) & (res.ttest [,2] > 10))
```

B- Étude des risques dans les tests d'hypothèses

• Étude du risque de première espèce

11.1-
```
n <- 20
M <- 500
ech <- rnorm(n,mean=4,sd=sqrt(1.2)) # Seulement un échantillon.
simu <- replicate(M,rnorm(n,mean=4,sd=sqrt(1.2)),simplify=TRUE)
```

11.2-
```
ttest <- function(x) t.test(x,mu=4)$p.value
res <- apply(simu,MARGIN=2,FUN=ttest)
```

11.3-
```
(compt <- sum(res < 0.05))
```

`compt` hypothèses sont rejetées ; on s'attend à rejeter 25 hypothèses nulles.

11.4-
```
M <- 5000
simu <- replicate(M,rnorm(n,mean=4,sd=sqrt(1.2)),simplify=TRUE)
res <- apply(simu,MARGIN=2,FUN=ttest)
proportion <- mean(res < 0.05)
proportion
```

11.5-
```
n <- 100
M <- 5000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
ttest <- function(x) t.test(x,mu=1)$p.value
res.chi2 <- t(apply(simu,MARGIN=2,FUN=ttest))
proportion <- mean(res.chi2 < 0.05)
proportion
```

11.6-
```
n <- 10
M <- 5000
simu <- replicate(M,rchisq(n,df=1),simplify=TRUE)
ttest <- function(x) t.test(x,mu=1)$p.value
```

```
res.chi2 <- t(apply(simu,MARGIN=2,FUN=ttest))
proportion <- mean(res.chi2 < 0.05)
proportion
```

Le risque de première espèce du test est supérieur à 5 % car le test de Student n'est ici pas valide du fait de la faible taille de l'échantillon.

11.7-
```
IC.p <- t.test((res.chi2<0.05)*1)$conf
IC.p
# ou bien:
binom.test(sum(res.chi2<0.05),length(res.chi2))$conf
```

• Étude de la puissance

11.1-
```
n <- 20
M <- 500
simu <- replicate(M,rnorm(n,mean=5,sd=sqrt(1.2)),simplify=TRUE)
```

11.2-
```
ttest <- function(x) t.test(x,mu=4)$p.value
res <- apply(simu,MARGIN=2,FUN=ttest)
```

11.3-
```
compt <- sum(res < 0.05)
puissance <- mean(res < 0.05)
```

11.4-
```
n <- 100
M <- 500
simu <- replicate(M,rnorm(n,mean=5,sd=sqrt(1.2)),simplify=TRUE)
res <- apply(simu,MARGIN=2,FUN=ttest)
(puissance <- mean(res < 0.05))
```

La puissance est de 100 %. Elle a augmenté avec la taille échantillonnale.

11.5-
```
n <- 100
M <- 500
simu <- replicate(M,rchisq(n,df=2),simplify=TRUE)
ttest <- function(x) t.test(x,mu=1)$p.value
res.chi2 <- apply(simu,MARGIN=2,FUN=ttest)
proportion <- mean(res.chi2 < 0.05)
proportion
```

11.6-
```
n <- 10
M <- 500
simu <- replicate(M,rchisq(n,df=2),simplify=TRUE)
ttest <- function(x) t.test(x,mu=1)$p.value
res.chi2 <- t(apply(simu,MARGIN=2,FUN=ttest))
proportion <- mean(res.chi2 < 0.05)
proportion
```

La puissance du test est faible pour une taille d'échantillon $n = 10$. Nous ne disposons pas d'assez d'information pour montrer que la moyenne est différente de 1.

C- Quelques exemples pratiques

• Étude vache

11.1-
```
juste.apres.traite <- c(12000,13000,21500,17000,15000,22000,
                        11000,21000)
apres.traite.24h <- c(11000,20000,31000,28000,26000,30000,
                      16000,29000)
t.test(juste.apres.traite,apres.traite.24h,
       paired=TRUE,alt="less")
```

11.2-
```
# Test du signe:
dif <- juste.apres.traite - apres.traite.24h
smoins <- sum(dif<0)
splus <- sum(dif>0)
binom.test(min(splus,smoins),splus+smoins,alt="less")
```

11.3-
```
# Test de Mann-withney:
wilcox.test(juste.apres.traite,apres.traite.24h,paired=TRUE,
            exact=FALSE,alt="less")
```

• Athlètes Est-allemandes

11.1-
```
hormonale <- c(3.22,3.07,3.17,2.91,3.4,3.58,3.23,3.11,3.62)
t.test(hormonale,mu=3.1,alt="greater")
```

11.2- La quantité d'androgène moyenne des athlètes de l'Allemagne de l'Est est significativement supérieure à la quantité moyenne de référence 3.1. Elles sont dopées.

• L'alcool au volant

```
avant <- c(57,54,62,64,71,65,70,75,68,70,77,74,80,83)
apres <- c(55,60,68,69,70,73,74,74,75,76,76,78,81,90)
t.test(avant,apres,paired=TRUE)
```

• Vitesse de la lumière

11.1-
```
vitesse <- c(850,740,900,1070,930,850,950,980,980,880,1000,
             980,930,1050,960,810,1000,1000,960,960)
vitesse <- vitesse + 299000
hist(vitesse)
```

Il y a un mode absolu très visible.

11.2-
```
shapiro.test(vitesse)
```

11.3-
```
t.test(vitesse,mu=299990)
```

11.4-
```
mean(vitesse)
```

• Taux de cholestérol

```
groupeA <- c(46.3,42.5,43,43.9,42,41.5,41.6,44.4,40.7)
groupeB <- c(47.1,44.5,45.8,49,44.6,43.7,44.5,47.4)
t.test(groupeA,groupeB,alt="less")
```

On peut conclure que le traitement est efficace au seuil de significativité de $\alpha = 5$ %.

• Indépendance traitement-décès

```
x <- matrix(c(0,9,8,3),ncol=2,byrow=TRUE)
# Tableau des effectifs théoriques:
chisq.test(x)$ex # Certaines valeurs sont < 5.

chisq.test(x)  # Khi-2 de Yates.
fisher.test(x) # Test exact de Fischer.
```

La mortalité dépend donc du traitement.

• Effectifs de cas d'urgence

```
dossier <- c(1500,1600,1450)
cas <- c(675,720,610)
chisq.test(cas/dossier)
```

À la vue des résultats, on ne peut pas conclure que la proportion de cas d'urgence est différente suivant le mois.

Solutions des TPs du chapitre 12

A- Étude sur la régression linéaire simple

• Étude sur données simulées

12.1-
```
n <- 50
b0 <- 1
b1 <- 2
sigma <- 0.1
t <- 5
erreur <- rnorm(n,sd=sigma)
x <- runif(n,min=0,max=t)
y <- b0+b1*x+erreur
```

12.2-
```
plot(y~x)
```

12.3-
```
modele <- lm(y~x)
coefficients(modele)
sd(residuals(modele))*sqrt((n-1)/(n-2))
# ou bien:
summary(modele)[6]
```

12.4-
```
plot(modele,1:2)
plot(modele$fitted~y)
```

12.5-
```
precision.parametre.b1 <- function(n,sigma) {
 erreur <- rnorm(n,sd=sigma)
 x <- runif(n,min=0,max=t)
 y <- b0 + b1*x + erreur
 modele <- lm(y~x)
 summary(modele)$coef[2,2]
}

taille <- seq(50,1000,by=50)
sd.b1 <- vector("numeric",20)
for (i in 1:20) {
 sd.b1[i] <- precision.parametre.b1(taille[i],0.1)
}
```

Nous pouvons observer, sur le graphique suivant, la précision de l'estimation de `b1` en fonction de `n`, pour un écart-type de bruit égal à 0.1 :

```
plot(sd.b1~taille,xlab="n",ylab="Estimation de l'écart-type de
     l'estimateur de b1",
     main="Précision de l'estimation de b1 en fonction de n")
```

Sur le graphique ci-dessous, nous pouvons observer la précision de l'estimation de `b1` en fonction de `sigma`, pour `n` égal à 100 :

```
sd <- seq(0.1,10,by=0.1)
sd.b1 <- vector("numeric",100)
for (i in 1:100) {
 sd.b1[i] <- precision.parametre.b1(n=100,sd[i])
}

titre <- "Précision de l'estimation de b1 en fonction \n de
          l'écart-type du bruit"
plot(sd.b1~sd,xlab=expression(sigma),ylab=
"estimation de l'écart-type de l'estimateur de b1",
 main=titre)
```

• Étude sur l'intima-média

12.1-
```
url <- "http://biostatisticien.eu/springeR/Intima_Media.txt"
intima.media <- read.table(url,header=TRUE,dec=",",sep=" ")
attach(intima.media)
```

12.2-
```
plot(mesure~AGE)
```

Évolution croissante de la `mesure` de l'épaisseur de l'intima-média en fonction de l'`AGE`.

12.3-
```
# Calcul du coefficient de corrélation entre les deux variables:
cor(mesure,AGE)
# Test de la significativité du coefficient de corrélation:
cor.test(mesure,AGE)
```

12.4-
```
modele <- lm(mesure~AGE,data=data.frame(mesure,AGE))
summary(modele)
abline(modele,col="blue")
```

12.5-
```
hist(residuals(modele),main="Histogramme")
plot(modele,1:6)
```

12.6-
```
age0 <- 33
predict(modele,data.frame(AGE=age0),interval="prediction")
```

12.7-
```
predict(modele,data.frame(AGE=age0),interval="confidence")
```

12.8-
```
modele2 <- lm(mesure~I(AGE^2))
summary(modele2)
```

Le R^2 a (légèrement) augmenté.

B- Étude sur la régression linéaire multiple

• Étude sur l'intima-média

On constate (`tabac[is.na(paqan)]` et `paqan[tabac==0]`) que la variable `paqan` (donnant le nombre de paquets de cigarettes par année) présente des valeurs manquantes `NA` à chaque fois que la variable `tabac` prend la valeur 0 (qui signifie ne fume pas). Il est donc naturel de remplacer les données manquantes de la variable `paquan` par des 0 :

```
intima.media$paqan[is.na(paqan)] <- 0
```

Création de la variable `IMC` :

```
IMC <- intima.media$poids/((intima.media$taille/100)^2)
attach(intima.media)
mon.data <- data.frame(AGE,SPORT,alcool,paqan,IMC,mesure)
```

12.1-
```
pairs(mon.data)
```

Certains nuages de points ont une allure linéaire, pouvant entraîner de la colinéarité.

12.2-
```
modele.age <- lm(mesure~AGE,data=mon.data)
summary(modele.age)
modele.sport <- lm(mesure~SPORT,data=mon.data)
summary(modele.sport) ### valeur-p > 0.25
modele.alcool <- lm(mesure~factor(alcool),data=mon.data)
summary(modele.alcool)
modele.paqan <- lm(mesure~paqan,data=mon.data)
summary(modele.paqan)
modele.IMC <- lm(mesure~IMC,data=mon.data)
summary(modele.IMC)
```

12.3-
```
modele.age.inter <- lm(mesure~AGE*paqan)
# Interaction non-significative au seuil de 0.25:
summary(modele.age.inter)

modele.alcool.sans.inter <- lm(mesure~factor(alcool)+paqan,
                               data=mon.data)
modele.alcool.inter <- lm(mesure~factor(alcool)*paqan,
                               data=mon.data)
summary(modele.alcool.inter)
anova(modele.alcool.sans.inter,modele.alcool.inter)
```

```
modele.IMC.inter <- lm(mesure~IMC*paqan,data=mon.data)
# Interaction non-significative au seuil de 0.25:
summary(modele.IMC.inter)
```

12.4-
```
modele.inter <- lm(mesure~AGE+factor(alcool)*paqan+IMC,
                   data=mon.data)
summary(modele.inter)
```

12.5-
```
modele.sans.inter <- lm(mesure~AGE+factor(alcool)+paqan+IMC,
                        data=mon.data)
# Le terme d'interaction n'est plus significatif:
anova(modele.inter,modele.sans.inter)
summary(modele.sans.inter)
```

12.6-
```
modele.sans.alcool <- lm(mesure~AGE+paqan+IMC,data=mon.data)
anova(modele.sans.alcool,modele.sans.inter)
```

La variable `alcool` n'apporte pas d'information.

```
summary(modele.sans.alcool)
```

12.7- Dans le modèle final, la mesure d'intima-média est expliquée par les variables `AGE` et `IMC`, avec un ajustement sur la variable `paqan`. Quel que soit le nombre de paquets de cigarettes fumées chaque année, la mesure de l'intima-média augmente avec l'âge et avec l'IMC.

• Étude sur le taux de chômage

12.1- Téléchargez le fichier http://biostatisticien.eu/springeR/chomage.RData dans votre répertoire de travail.

```
load("chomage.RData")
names(chomage)
attach(chomage)

modele.txpib <- lm(chom~txpib,data=chomage)
summary(modele.txpib)
plot(chom~txpib)
abline(modele.txpib)

hist(residuals(modele.txpib))
plot(modele.txpib,1:2)
```

12.2-
```
cor(chomage[,2:7])
```

12.3-
```
pairs(chomage[,2:7])
```

12.4- Variables les plus explicatives : `pfisc`, `deppub`, `txpib`. Variables colinéaires : `deppub` et `pfisc`, ainsi que `deppub` et `txpib`, et enfin `pfisc` et `txpib`.

12.5-
```
modele.complet <- lm(chom~txpib+deppub+pfisc+salva+infl,
                     data=chomage)
summary(modele.complet)
```

12.6-
```
require("car")
vif(modele.complet)
```

12.7-
```
drop1(lm(chom~txpib+deppub+pfisc+salva+infl,data=chomage),
         test="F")
# On enlève la variable infl.
drop1(lm(chom~txpib+deppub+pfisc+salva,data=chomage),test="F")
# On enlève la variable txpib.
drop1(lm(chom~deppub+pfisc+salva,data=chomage),test="F")
# On enlève la variable pfisc.
drop1(lm(chom~deppub+salva,data=chomage),test="F")
```

12.8-
```
# Modèle final:
modele.final <- lm(chom~deppub+salva,data=chomage)
summary(modele.final)
```

12.9-
```
deppub93 <- chomage$deppub[chomage$an==1993]
salva93 <- chomage$salva[chomage$an==1993]
predict(modele.final,data.frame(deppub=deppub93,salva=salva93),
        interval="prediction")
```

12.10- La valeur observée de `chom` en 1993 était de 11.2 et cette valeur est contenue dans l'intervalle de prévision.

C- Étude sur la régression polynomiale

• Étude sur données simulées

12.1-
```
n <- 100
x <- runif(n,min=-2,max=2)
eps <- rnorm(n)
y <- x+2*(x^2)+3.5*(x^3)-2.3*(x^4)+eps
```

12.2-
```
plot(y~x)
curve(x+2*x*x+3.5*x^3-2.3*x^4,add=TRUE)
```

12.3-
```
modele.simple <- lm(y~x)
summary(modele.simple)
hist(residuals(modele.simple))
plot(modele.simple,1:6)
```

12.4-
```
modele.poly <- lm(y~-1+poly(x,4,raw=TRUE))
summary(modele.poly)
coef <- coef(modele.poly)
plot(y~x)
curve(coef[1]*x+coef[2]*x^2+coef[3]*x^3+coef[4]*x^4,add=TRUE)
```

• Ajustement d'un nuage de points par un polynôme

12.1- Téléchargez le fichier http://biostatisticien.eu/springeR/ajustpoly.RData dans votre répertoire courant.

```
load("ajustpoly.RData")
attach(ajustpoly)
```

12.2-
```
plot(Y~X)
```

12.3-
```
modele.lin <- lm(Y~X)
summary(modele.lin)
abline(modele.lin)
```

12.4-
```
modele.poly <- lm(Y~poly(X,3,raw=TRUE),data=ajustpoly)
summary(modele.poly)
```

12.5-
```
coef <- coef(modele.poly)
curve(coef[1]+coef[2]*x+coef[3]*x^2+coef[4]*x^3,add=TRUE)
x <- seq(-3.5,3.5,length=50)
intpred <- predict(modele.poly,data.frame(X=x),
                   interval="prediction")[,c("lwr","upr")]
intconf <- predict(modele.poly,data.frame(X=x),
                   interval="confidence")[,c("lwr","upr")]
matlines(x,cbind(intconf,intpred),lty=c(2,2,3,3),
         col=c("red","red","blue","blue"),lwd=c(2,2,1,1))
legend("bottomright",lty=c(2,3),lwd=c(2,1),
       c("confiance","prévision"),col=c("red","blue"))
```

Solutions des TPs du chapitre 13

A- Étude sur l'ANOVA à un facteur

• Étude sur le niveau sonore

13.1-
```
sonore <- gl(4,6,24)
note <- c(62,60,63,59,63,59,56,62,60,61,63,64,63,67,
          71,64,65,66,68,66,71,67,68,68)
```

13.2- $\text{note}_{ik} = \mu + \alpha_i + \epsilon_{ik}$ où les ϵ_{ik} sont des variables aléatoires *i.i.d.* suivant une loi $\mathcal{N}(0, \sigma^2)$.

13.3-
```
modele <- aov(note~sonore)
summary(modele)
```

13.4-
```
pairwise.t.test(note,sonore,p.adjust="bonf")
TukeyHSD(modele)
```

• Étude sur l'intima-média

13.1-
```
url <- "http://biostatisticien.eu/springeR/Intima_Media.txt"
intima.media <- read.table(url,header=TRUE,dec=",",sep=" ")
attach(intima.media)
```

13.2-
```
plot(mesure~factor(alcool))
```

13.3-
```
modele.intima <- aov(mesure~factor(alcool))
summary(modele.intima)
```

Il existe une différence de mesure de l'épaisseur de l'intima-média suivant la consommation d'alcool (valeur-$p \leq 0.05$).

13.4-
```
plot(modele.intima)
# Homoscédasticité :
require("car")
leveneTest(mesure,factor(alcool))
```

• Étude chez les sportifs

13.1- Téléchargez le fichier http://biostatisticien.eu/springeR/sportif.RData dans votre répertoire courant.

```
load("sportif.RData")
attach(sportif)
```

Facteur temps de pratique sportive (7 modalités). $\text{score}_{ik} = \mu + \alpha_i + \epsilon_{ik}$ où les ϵ_{ik} sont des variables aléatoires *i.i.d.* suivant une loi $\mathcal{N}(0, \sigma^2)$.

13.2-
```
modele.sport <- aov(score~temps)
summary(modele.sport)
```

13.3-
```
cmat <- rbind(" : peu sportif versus moyennement sportif"=
              c(1,1,-1,-1,0,0,0),
" : très sportif versus peu sportif"=c(3,3,0,0,-2,-2,-2))
```

13.4-
```
require("gmodels")
fit.contrast(modele.sport,temps,cmat)
```

B- Étude sur l'ANOVA à deux facteurs

• Étude sur les piles

13.1- Facteur température à 3 modalités : 15, 70 et 125 degrés. Facteur type à 3 modalités : I, II et III. Variable dépendante : durée de vie.

13.2- $Y_{ijk} = \mu + \mu_{ij} + \epsilon_{ijk}$ où les ϵ_{ijk} sont des variables aléatoires *i.i.d.* suivant une loi $\mathcal{N}(0, \sigma^2)$.

13.3-
```
duree <- c(130,155,74,180,34,40,80,75,20,70,82,58,150,188,159,
           126,136,122,106,115,25,70,58,45,138,110,168,160,174,
           120,150,139,96,104,82,60)
temperature <- as.factor(rep(rep(1:3,each=4,3)))
levels(temperature) <- c("15°C", "70°C","125°C")
type.pile <- as.factor(rep(1:3,each=12))
levels(type.pile) <- c("type I","type II","type III")
pile <- data.frame(duree=duree,temperature=temperature,
                   type.pile=type.pile)
interaction.plot(temperature,type.pile,duree)
interaction.plot(type.pile,temperature,duree)
```

13.4-
```
summary(lm(duree~temperature*type.pile))
```

13.5-
```
anova(lm(duree~temperature*type.pile))
```

Interaction significative : attention à l'interprétation des paramètres et aux tests sur les effets fixes ; il faut regarder les effets conditionnels.

13.6- Par exemple, test de l'effet température pour le type de pile I

```
modele <- summary(aov(duree~temperature*type.pile))
duree.pile1 <- summary(aov(duree~temperature,
                           subset=type.pile=="type I"))
duree.pile1
F.duree.pile1 <- duree.pile1[[1]]$Mean[1]/modele[[1]]$Mean[4]
pvaleur <- 1-pf(F.duree.pile1,df1=2,df2=27)
```

• Rendement laitier

13.1- Facteur aliment à 4 modalités : Paille, Foin, Herbe, Aliments ensilés. Facteur dose à 2 modalités : Faible et Forte. Variable dépendante : rendement. $Y_{ijk} = \mu + \mu_{ij} + \epsilon_{ijk}$ où les ϵ_{ijk} sont des variables aléatoires *i.i.d.* suivant une loi $\mathcal{N}(0, \sigma^2)$.

13.2-
```
rendement <- c(8,11,11,10,7,12,13,14,11,10,10,12,12,13,14,17,
               13,17,14,13,8,9,8,10,9,10,7,10,12,11,11,9,11,
               11,12,13,12,11,15,14)
aliment <- as.factor(rep(rep(1:4,each=5,2)))
levels(aliment) <- c("Paille","Foin","Herbe","Aliments ensilés")
dose <- as.factor(rep(1:2,each=20))
levels(dose) <- c("faible", "forte")
laitier <- data.frame(rendement=rendement, aliment= aliment,
                      dose=dose)
interaction.plot(aliment,dose,rendement)
interaction.plot(dose,aliment,rendement)
```

13.3-
```
summary(lm(rendement~dose*aliment))
```

13.4-
```
anova(lm(rendement~dose*aliment))
```

Interaction non significative : on préfèrera un modèle plus simple.

13.5-
```
anova(lm(rendement~dose+aliment))
```

• Étude sur l'intima-média

13.1-
```
url <- "http://biostatisticien.eu/springeR/Intima_Media.txt"
intima.media <- read.table(url,header=TRUE,dec=",",sep=" ")
attach(intima.media)
```

13.2- Modèle ANOVA à deux facteurs avec un terme d'interaction.

13.3-
```
tabac <- factor(tabac)
alcool <- factor(alcool)
interaction.plot(tabac,alcool,mesure)
interaction.plot(alcool,tabac,mesure)
```

13.4-
```
require("car")
modele.intima <- Anova(lm(mesure~alcool*tabac),type="III")
modele.intima
# interaction non-significative:
modele.intima.additif <- Anova(lm(mesure~alcool+tabac),type="III")
# alcool: pvaleur=0.09  Tabac: p-valeur=0,16
```