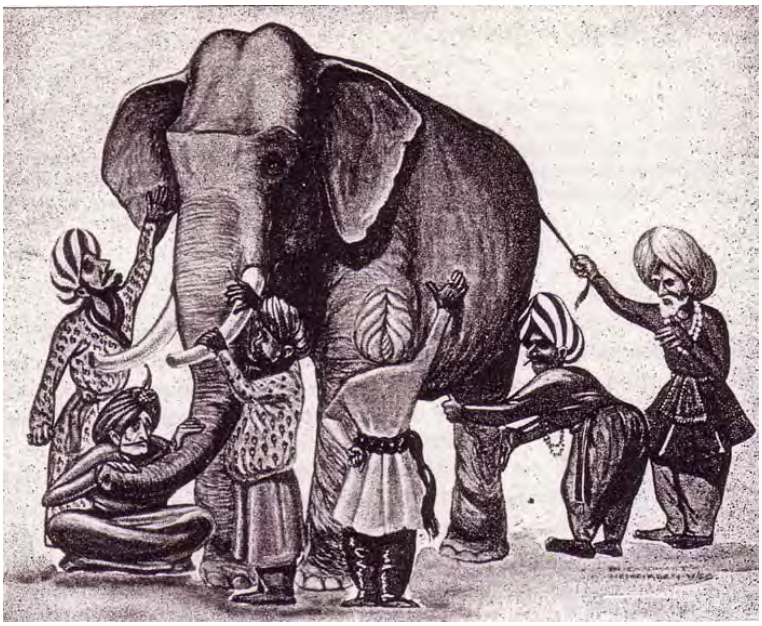
Modèles Probabilistes pour l'Apprentissage

Olivier François

Ensimag – Grenoble INP

olivier.francois@grenoble-inp.fr



The blind men and the elephant

It was six men of Indostan

To learning much inclined

Who went to see the Elephant

(Though all of them were blind)

That each by observation

Might satisfy his mind

...

And so these men of Indostan

Disputed loud and long,

Each in his own opinion

Exceeding stiff and strong,

Though each was partly in the right,

And all were in the wrong!

J.G. Saxe (1816 – 1887)

Brève présentation du cours de MPA

Créer des machines capables d'autonomie, capables d'apprendre à réaliser des tâches par elles-mêmes sans le contrôle de l'homme, est un objectif de longue haleine des sciences numériques. Descartes qui voyait les animaux comme des assemblages de mécanismes, prétendait que l'on pourra un jour créer une machine indistinguable d'un animal, capable de remplacer l'homme pour certaines de ses activités.

La vision de Descartes a probablement conduit au développement des sciences de l'automatisme. Les automates, tels que le fameux *canard de Vaucanson* (figure 1), ne sont toutefois pas dotés de mécanismes adaptatifs. Une fois programmé pour une tâche donnée, ils sont limités à répéter cette tâche. Ce que l'on entend dans ce cours par le mot *apprentissage*, représente un ensemble de méthodes et d'algorithmes qui permettent à une machine d'évoluer dans un monde incertain grâce à un processus adaptatif.

Le principe fondateur de ce cours est que la notion d'apprentissage peut être conceptualisée à l'aide d'un modèle probabiliste. Nous cherchons à modéliser la réponse d'un système à partir de variables d'entrées et des données empiriques aléatoires recueillies pour ce système (nous considérons des versions souvent très simplifiées de tels systèmes dans le cours). Les connaissances accumulées grâce à l'expérimentation permettent de réviser et de mettre à jour un modèle probabiliste conçu *a priori* afin d'en réduire l'incertitude initiale et d'en augmenter le pouvoir prédictif. Les algorithmes permettant la mise à jour du modèle sont adaptatifs et flexibles. Ils sont sensés prendre en compte l'évolution de la base d'information des comportements enregistrés. En pratique, la phase de révision

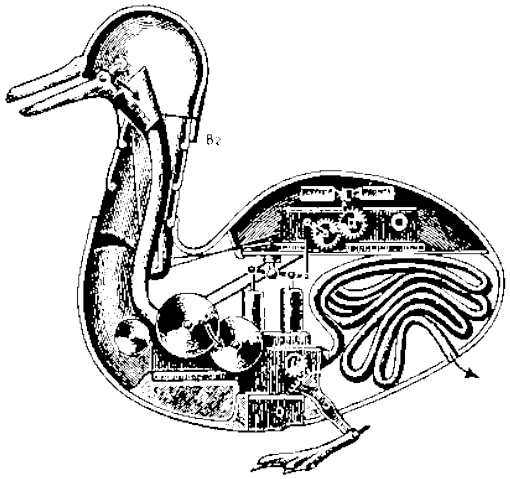


FIGURE 1 – *Le canard de Vaucanson, célèbre automate dont une copie est visible au musée dauphinois à Grenoble.*

du modèle utilise la célèbre formule d'inversion des causes et des effets – la **formule de Bayes** – un outil central de ce cours.

Les applications concrètes du formalisme présenté dans ce cours sont très nombreuses, en reconnaissance des formes, en reconnaissance vocale, en vision par ordinateur, pour l'aide au diagnostic, pour la détection de fraude, d'anomalies, de spams, pour l' analyse financière, pour la bio-informatique. On retrouve ces concepts au cœur de la conception de sites web adaptatifs qui pratiquent la recommandation individualisée de produits, ou encore dans les moteurs de recherche.

Quelques mots sont aussi nécessaires pour expliquer l'allégorie présentée dans l'image de couverture de ce cours. La figure de couverture illustre une légende hindoue, traduite en anglais par le poète John Saxe sous le titre *The blind men and the elephant*. Dans cette légende, six sages, tous aveugles, décident

d'apprendre ce qu'est un éléphant – un gros animal, servant par ailleurs de mascotte à l'Ensimag. Chacun des sages ne peut accéder qu'à une seule partie de l'animal et l'évalue avec son propre système de représentation (un modèle donc). Ainsi, le sage qui tient la queue de l'éléphant prétend qu'il tient une *corde*, celui qui tient la jambe de l'éléphant dit qu'il tient un *tronc*, etc. Les six hommes débattent très fort sur la nature de l'éléphant, mais ne peuvent pas se mettre d'accord. La morale de cette fable dit que chacun des six sages détient une part de vérité, mais tous sont dans l'erreur. Aucun des six sages n'a pu construire une bonne représentation de ce qu'est un éléphant à partir de ses connaissances a priori.

Les élèves intéressés par *aller plus loin* pourront approfondir le cours de MPA – et peut être la philosophie sous-jacente – en étudiant l'ouvrage de référence écrit par Gelman et ses collègues (2004). Chacun aura grand bénéfice à consulter le guide d'initiation au langage `R` écrit par E. Paradis (2005). Pour suivre ce cours, il est en effet nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`. Les scripts `R` illustrant le cours de MPA sont disponibles sur la page personnelle de Olivier François (pages destinées à l'enseignement).

Références citées dans l'introduction :

Gelman A, Carlin JB, Stern HS, Rubin DB (2004) Bayesian Data Analysis 2nd ed. Chapman & Hall, New-York.

E. Paradis (2005) R pour les débutants. Univ. Montpellier II.

1 Séance 1 : Rappels de probabilités – Formule de Bayes

1.1 Rappels et notations

Dans ce cours, toutes les variables, **paramètres ou données** sont considérées comme étant des variables **aléatoires**. Pour alléger les notations, nous ne faisons pas de distinction entre une variable aléatoire et sa réalisation. Ainsi, lorsque x est une variable de Bernoulli susceptible de prendre la valeur 1 avec la probabilité π ou la valeur 0 avec la probabilité $(1-\pi)$, nous écrivons

$$p(x) = \pi^x (1-\pi)^{(1-x)}\,, \quad x = 0, 1\,.$$

Nous venons de décrire une loi de probabilité discrète prenant deux valeurs. La notation se généralise à une loi discrète prenant plus de 2 valeurs sans difficulté. La grandeur $p(x)$ est alors comprise entre 0 et 1 et la somme totale des valeurs est égale à 1. Pour mémoire, la notation traditionnelle distinguant la variable aléatoire X de sa réalisation x consiste à écrire

$$\mathbf{P}(X = x) = \pi^x (1-\pi)^{(1-x)}\,.$$

On choisit donc de ne pas faire mention explicite à X. Lorsque x est une variable continue, prenant par exemple ses valeurs dans $\mathbb{R}$, on parle alors de densité de probabilité. Une densité de probabilité est une fonction positive dont l'intégrale est égale à 1. Par exemple, si x est une variable réelle de loi normale, on écrit alors

$$p(x) = \exp(-x^2/2)/\sqrt{2\pi}\,, \quad x \in \mathbb{R}.$$

Supposons que y est une seconde variable aléatoire, et qu'elle est positive de loi exponentielle de paramètre 1. Nous notons alors sa loi de probabilité de la

manière suivante

$$p(y) = \exp(-y)\,, \quad y > 0.$$

Pour mémoire, la notation traditionnelle distinguant la variable aléatoire Y de sa réalisation y consiste à écrire

$$f_Y(y) = \exp(-y)\,, \quad y > 0.$$

Il est important de remarquer que la notation simplifiée, supprimant la référence la variable Y, peut être ambigüe. C'est le contexte et la notation en lettre minuscule associée à une variable qui permet de comprendre de quelle loi il s'agit. Il est clair que $p(x)$ (égale à $f_X(x)$) et $p(y)$ (égale à $f_Y(y)$) ne désignent pas la même densité, bien que la notation mathématique, $p(.)$, semble l'indiquer.

La loi exponentielle de paramètre $\theta > 0$ se note de la manière suivante

$$p(y|\theta) = \theta \exp(-\theta y)\,, \quad y > 0.$$

La barre | introduite dans la notation précédente n'est pas anodine. Puisque θ est un paramètre, nous le considérons comme étant une variable aléatoire. Ainsi, y peut être vue comme une réalisation d'une loi conditionnelle sachant le paramètre θ. Nous y reviendrons dans la section suivante.

En définitive, nous considérons de manière unifiée des variables aléatoires continues ou discrètes en notant les lois de la même manière. Cela signifie que l'on utilise la même notation pour désigner des intégrales et des sommes (c'est tout à fait rigoureux !). Par exemple, pour une variable aléatoire continue, nous avons

$$\int p(y)dy = 1\,.$$

Pour une variable y discrète, prenant un nombre fini de valeurs, il faut comprendre (ou réécrire) cette intégrale comme étant égale à la somme

$$\sum p(y) = 1\,.$$

Dans le cas discret ou dans le cas continu, nous appelons **espérance** de la variable y, la grandeur

$$\mathrm{E}[y] = \int y p(y) dy\,.$$

Lorsque cette intégrale est définie, on dit que la variable aléatoire y est intégrable. Cela correspond à la situation où l'intégrale de la valeur absolue de la variable y converge, c'est à dire, $\mathrm{E}[|y|] < \infty$. Lorsque y est de carré intégrable, nous appelons **variance** la grandeur définie par

$$\mathrm{Var}[y] = \mathrm{E}[(y - \mathrm{E}[y])^2] = \mathrm{E}[y^2] - \mathrm{E}[y]^2\,.$$

1.2 Loi de couple et probabilité conditionnelle

Nous considérons maintenant deux variables aléatoires, y et θ, discrètes ou continues. Groupées, ces deux variables forment le couple (y, θ). Il est commode d'imaginer que y représente une donnée issue d'un tirage aléatoire et que θ représente un paramètre déterminant ce tirage. Dans nos notations, la loi jointe du couple (y, θ) s'écrit $p(y, \theta)$. Il s'agit en général d'une fonction positive et nous avons

$$\int p(y, \theta) dy d\theta = 1.$$

On parle alors de modèle probabiliste pour le couple (y, θ). Dans ce cours, la loi de la donnée y s'appelle la **loi marginale**. Elle est définie à l'aide de l'intégrale suivante

$$p(y) = \int p(y, \theta) d\theta\,,$$

dont la valeur exacte est souvent difficile à expliciter. La loi $p(\theta)$ est théoriquement aussi une loi marginale. Nous l'appelons plus communément la **loi a priori**. Elle est en général donnée directement lors de la description du modèle.

La loi conditionnelle de y sachant θ est notée $p(y|\theta)$

$$p(y|\theta) = \frac{p(y,\theta)}{p(\theta)}\,.$$

Cette loi est appelée la **loi d'échantillonnage** ou la **loi générative**. Elle décrit la manière avec laquelle, sachant la valeur du paramètre θ, on peut échantillonner une variable y (on dit aussi générer y).

Nous voyons avec la loi conditionnelle, un intérêt évident à utiliser les notations allégées. En effet, une notation rigoureuse nous demanderait d'écrire

$$\mathbf{P}(Y = y|\Theta = \theta) = p(y|\theta)$$

dans le cas ou y est une variable discrète et

$$f_Y^{\Theta=\theta}(y) = p(y|\theta)$$

dans le cas d'une variable continue. L'utilisation d'indices et d'exposants peut être source de confusion et nous l'abandonnons donc par la suite pour utiliser la notation allégée.

Important : la donnée de la loi a priori, $p(\theta)$ et de la loi générative $p(y|\theta)$ permet de définir un modèle probabiliste. En effet, nous avons

$$p(y,\theta) = p(y|\theta)p(\theta)\,.$$

Par exemple, considérons une probabilité inconnue, θ, répartie de manière uniforme sur l'intervalle $(0,1)$, et supposons que l'on tire un nombre au hasard

selon une loi binomiale de paramètre $n = 20$ et de probabilité θ. Dans ce cas, nous avons

$$p(y, \theta) = \frac{n!}{y!(n-y)!}\theta^y(1-\theta)^{n-y}\,.$$

pour tout $0 \leq y \leq n$ et $\theta \in (0, 1)$.

1.3 Formule de Bayes

La **formule de Bayes** est une **formule essentielle** du cours de MPA. Elle permet de calculer la loi de probabilité conditionnelle de θ sachant y à partir de la donnée de la loi générative. Pour la distinguer de la loi a priori, cette loi s'appelle la **loi a posteriori**

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}\,.$$

Dans cette formule, la loi marginale $p(y)$ peut être difficile à exprimer de manière explicite. Nous notons qu'elle correspond en fait à la constante de normalisation de l'expression du numérateur

$$p(y) = \int p(y|\theta)p(\theta)d\theta\,.$$

On se contentera donc, sauf mention contraire, du terme général de la loi a posteriori

$$p(\theta|y) \propto p(y|\theta)p(\theta)\,,$$

où le symbole $\propto$ signifie *proportionnel à*. La formule ci-dessus fait clairement apparaître le lien existant entre la loi a priori et la loi a posteriori. La loi a posteriori peut être vue comme une **mise à jour** de la loi a priori, une fois que la variable y est générée et observée.

Par exemple, considérons une proportion inconnue, θ, répartie de manière uniforme sur l'intervalle $(0, 1)$, et supposons que l'on tire une valeur au hasard, 0 ou 1, telle que la probabilité d'observer 1 est égale à θ. A l'issue de cette expérience, supposons que le résultat soit égal à 1. Nous avons

$$p(\theta|y = 1) \propto p(y = 1|\theta)p(\theta) = \theta p(\theta) = \theta$$

Dans ce cas, la constante de normalisation est en fait très facile à calculer. Nous obtenons

$$p(\theta|y = 1) = 2\theta, \quad \theta \in (0, 1)\,.$$

La loi **a priori** décrit l'incertitude sur le paramètre θ avant l'expérience, et la loi **a posteriori** décrit l'incertitude sur ce paramètre à l'issue de l'expérience. La formule de Bayes formalise le processus d'**apprentissage** correspondant à la mise à jour de l'information en fonction du résultat de l'expérience.

1.4 Principes de base de simulation

Une difficulté rencontrée en calcul des probabilités est que de nombreuses lois ne sont pas calculables explicitement. En revanche, il est souvent possible de les simuler à l'aide d'un générateur aléatoire afin de calculer les probabilités ou les espérances (valeurs moyennes de certaines fonctions) numériquement. Cette méthode est communément appelée la **méthode de Monte Carlo**, en référence aux tirages aléatoires qu'elle effectue.

Simulation par inversion. Supposons que l'on cherche à simuler une variable aléatoire θ, de loi donnée $p(\theta)$ (non uniforme). Lorsque cela est possible,

on calcule l'inverse, G, de la fonction de répartition, F, définie par

$$F(t) = p(\theta \leq t), \quad t \in \mathbb{R}\,.$$

La méthode de simulation par inversion consiste à tirer un nombre au hasard

$$\theta = G(u)\,,$$

où u est un tirage de loi uniforme sur (0,1). Formellement séduisante, cette méthode reste toutefois difficilement applicable en pratique, sauf pour des cas simples. Pour utiliser la méthode d'inversion, on peut chercher à décomposer la loi cible comme un mélange de lois très simples, afin d'appliquer la méthode à chaque composante du mélange.

Simulation par rejet. La méthode de simulation par rejet ne demande que très peu de calcul analytique. Il en existe de nombreuses variantes fort utiles dans ce cours. Dans ce paragraphe, nous rappelons l'algorithme vu en première année de l'Ensimag lors du cours de probabilités appliquées. Soit $p(\theta)$ la loi de probabilité d'une variable définie sur l'intervalle $(0, 1)$ supposée continue sur cet intervalle, et c une constante supérieure au maximum de $p(\theta)$. L'algorithme de rejet peut s'écrire de la manière suivante.

```
Repeat
theta <- unif(0,1)
x <- unif(0,1)
Until (c * x < p(theta) )
return(theta)
```

Simuler un couple de variables aléatoires. Pour simuler un couple de variables aléatoires (y, θ) de loi $p(y, \theta)$, une méthode élémentaire consiste à simuler la loi a priori, $p(\theta)$, puis à simuler la loi générative $p(y|\theta)$. Dans ce cas, on se ramène donc à la simulation de variables uni-dimensionnelles. Nous verrons dans la suite de ce cours de nouvelles méthodes de simulation de lois multivariées.

1.5 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

1.6 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. On considère la loi de densité

$$p(\theta) \propto \theta + 2\theta^4\,, \quad \theta \in (0,1).$$

1. Calculer la constante de proportionnalité.
2. Proposer un algorithme de simulation par rejet par rapport à la loi uniforme pour la loi $p(\theta)$.
3. Evaluer le nombre moyen de rejets nécessaires pour obtenir un seul tirage.
4. Proposer un algorithme de simulation de mélange pour la loi $p(\theta)$.
5. Ecrire les algorithmes précédents dans le langage de programmation `R`.
6. Simuler un échantillon de taille 10000 à l'aide de chacun des algorithmes.
7. Vérifier que les histogrammes estiment bien la densité $p(\theta)$, et pour l' algorithme de rejet, estimer la probabilité de rejet numériquement.

Exercice 2. Couples de variables aléatoires. On considère la loi de densité

$$\forall x, y \in \mathbb{R}, \quad p(x,y) = \frac{1}{2\sqrt{xy}}\, \mathbb{1}_D(x,y)\,,$$

où $D = \{(x,y) \in \mathbb{R}^2,\, 0 < x < y < 1\}$.

1. Rappeler le principe de simulation d'une loi de densité $p(x,y)$ définie sur $\mathbb{R}^2$.
2. Calculer les lois marginales $p(x)$, $p(y)$ du couple de densité $p(x,y)$.
3. Lorsqu'elles sont définies, calculer les lois conditionnelles $p(x|y)$, $p(y|x)$

du couple de densité $p(x, y)$.

4. Proposer un algorithme de simulation pour la loi de densité $p(x, y)$ et prouver sa validité en appliquant le théorème de changement de variables.
5. Programmer cet algorithme dans le langage `R`.
6. Effectuer 10000 tirages selon cet algorithme. Afficher les résultats.
7. Proposer des approximations de Monte Carlo des grandeurs $\mathrm{E}[x]$ et ρ (coefficient de corrélation). Calculer les valeurs théoriques de ces grandeurs.

2 Séance 2 : Quantifier l'incertitude et effectuer des prédictions à partir de résultats d'expériences

2.1 Introduction

Dans ce cours, nous appelons **processus d'apprentissage** le processus consistant à mettre à jour l'information dont on dispose sur un phénomène à partir de l'observation répétée ou non de ce phénomène. Mathématiquement, l'information disponible sur un phénomène est généralement résumée par un ensemble de paramètres permettant d'expliquer le phénomène.

Afin de quantifier et mettre à jour l'incertitude sur les paramètres des modèles, les modèles considérés dans ce cours sont probabilistes (au sens large, car les modèles déterministes peuvent être un cas particulier intéressant). L'approche probabiliste permet aussi de prédire les nouvelles données, d'évaluer l'incertitude des prédictions et d'évaluer la pertinence des modèles.

L'idée d'apprentissage probabiliste se traduit par l'utilisation systématique de la formule de Bayes. Cette formule traduit comment l'incertitude a priori sur les paramètres d'un modèle se réduit lorsque des données supplémentaires alimentent le modèle. L'utilisation de la formule de Bayes pour le calcul de la loi a posteriori justifie que l'on parle traditionnellement d'**inférence bayésienne**. Le terme *inférence* est utilisé ici comme un terme générique mettant en exergue la nature probabiliste des résultats obtenus et la possibilité d'utiliser les lois a posteriori dans un but prédictif.

2.2 Définitions

Nous reprenons et généralisons les notations introduites dans la séance précédente. Dans un modèle, il y a des **variables observées** et des **variables non-observées**. Les variables observées sont les **données** disponibles sur le phénomène à expliquer ou à prédire

$$y = (y_1, \ldots, y_n), \quad n \geq 1.$$

Les données forment un vecteur de longueur n, que l'on appelle l'**échantillon**. Pour cette raison la loi générative des données s'appelle la loi d'échantillonnage. Plus généralement, échantillonner, c'est tirer ou recueillir des données au hasard.

Les variables non-observées sont appelées **variables cachées**, **variables latentes** ou **paramètres**

$$\theta = (\theta_1, \ldots, \theta_J), \quad J \geq 1.$$

L'ensemble des variables non-observées constitue un vecteur de dimension J (J peut être plus grand que n). Les composantes de ce vecteur peuvent être des paramètres probabilistes tels que la moyenne ou la variance, mais aussi des données manquantes dont l'introduction "augmente" le modèle.

Un modèle est donc décrit par une loi de probabilité multivariée, autrement dit, multi-dimensionnelle :

$$p(y, \theta) = p(y|\theta)p(\theta)\,,$$

Nous avons déjà parlé de la loi du paramètre, notée $p(\theta)$. Cette loi s'appelle la loi *a priori*. Elle peut être choisie afin de modéliser l'incertitude initiale sur un phénomène, ou bien les connaissances d'un expert n'ayant pas accès aux données y.

Important : La loi a priori peut parfois être **dégénérée**. Dans ce cas la fonction $p(\theta)$ n'est pas intégrable

$$\int p(\theta) d\theta = \infty \,.$$

En pratique, cela peut être acceptable si la loi *a posteriori*

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

est bien définie et vérifie

$$\int p(\theta|y) d\theta = 1 \,.$$

2.3 Apprentissage d'une fréquence d'émission

Nous considérons un exemple très élémentaire d'apprentissage probabiliste : l'apprentissage de la fréquence d'émission d'une source à partir d'observations émises par la source. Une source émet donc des signaux binaires, $x_i = 0$ ou 1, et on cherche à estimer la probabilité pour cette source d'émettre un 1. On note cette probabilité θ. Sans information préalable sur la source, nous supposons que la loi a priori est uniforme

$$p(\theta) = 1 \,.$$

La théorie de Shannon nous indique que la loi uniforme maximise l'incertitude sur le paramètre θ. Le modèle est entièrement spécifié par la loi générative

$$p(x_i = 1|\theta) = \theta \,, \quad p(x_i = 0|\theta) = 1 - \theta \,,$$

Il s'agit d'une loi de Bernoulli de paramètre θ. Nous supposons que l'on observe une suite de signaux indépendants émis par la source. Par exemple, cette suite est donnée par $x_1 = 1, x_2 = 0, x_3 = 1, \ldots$

Afin de bien comprendre comment l'incertitude sur le paramètre θ se modifie au fur et à mesure que les données sont acquises, nous cherchons à calculer la loi conditionnelle de θ sachant les données. Suite à la première observation, nous avons

$$p(\theta|x_1 = 1) \propto p(x_1 = 1|\theta)p(\theta) = \theta.$$

Suite à la seconde observation, nous avons

$$p(\theta|x_2 = 0, x_1 = 1) \propto p(x_2 = 0|\theta)p(\theta|x_1 = 1) = (1 - \theta)\theta\,.$$

On remarque que l'on peut calculer la loi conditionnelle $p(\theta|x_2 = 0, x_1 = 1)$ directement ou bien, comme ci-dessus, en utilisant la loi $p(\theta|x_1 = 1)$. De la même manière, les lois suivantes peuvent être calculées en utilisant un argument de récurrence simple. Suite à la troisième observation, nous avons

$$p(\theta|x_3 = 1, x_2 = 0, x_1 = 1) \propto p(x_3 = 1|\theta)(1 - \theta)\theta = (1 - \theta)\theta^2\,.$$

Plus généralement, après n observations, nous obtenons

$$p(\theta|x_n, \ldots, x_1) \propto p(x_n|\theta)p(\theta|x_{n-1}, \ldots, x_1) = (1 - \theta)^{n-y}\theta^y$$

où l'on note $y = \sum_{i=1}^{n} x_i$. La figure 2 illustre graphiquement l'évolution de la connaissance acquise ou l'évolution de l'incertitude résiduelle sur la probabilité d'émission θ en fonction de l'observation successive des données suivantes : $x_1 = 1, x_2 = 0$ et $x_3 = 1$.

2.4 Un exemple de sondage : modèle bêta-binomial

L'exemple précédent met en évidence une loi particulière jouant un rôle important lorsque l'on cherche à modéliser une fréquence. Il s'agit de la **loi**

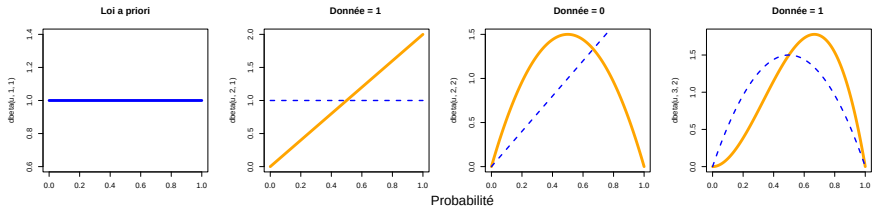


FIGURE 2 – *Mise à jour de l'information a priori sur la fréquence d'émission d'une source binaire en fonction de l'observation successive des données* $x_1 = 1, x_2 = 0$ *et* $x_3 = 1$.

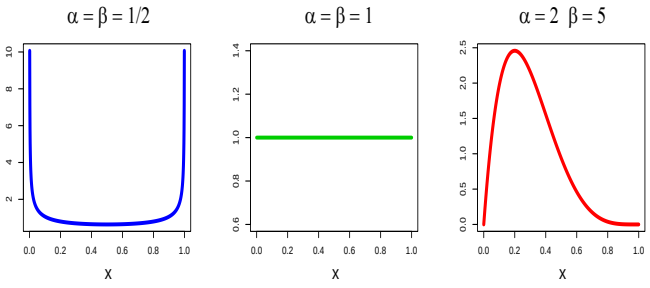


FIGURE 3 – *Densité de probabilité de la loi bêta pour différentes valeurs de ses paramètres.*

bêta. La loi bêta modélise une variable aléatoire comprise entre 0 et 1. Cette loi comporte 2 paramètres positifs, notés α et β. Elle est définie de la manière suivante

$$p(\theta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}\,.$$

La figure 3 représente les formes typiques que peut prendre la courbe de la densité de la loi bêta. Pour des valeurs de α et β inférieures à 1, nous obtenons une forme en U. Pour α et β supérieurs à 1, nous obtenons une forme en cloche. Notons que pour $\alpha = \beta = 1$, la loi correspond à la loi uniforme sur $(0, 1)$. La valeur moyenne de la loi bêta se calcule grâce aux propriétés mathématiques de

la fonction $\Gamma(\alpha)$. Nous avons

$$\mathrm{E}[\theta] = \int_0^1 \theta p(\theta) d\theta = \frac{\alpha}{\alpha + \beta} \, .$$

Voyons maintenant comment la loi bêta intervient lorsque l'on cherche à modéliser une fréquence. Pour cela, considérons le problème de sondage – tout à fait fictif – suivant.

Avant le second tour d'une élection présidentielle, il reste deux candidats que nous appelons A et B. Nous interrogeons au hasard $n = 20$ *électeurs supposés représentatifs et indépendants les uns des autres. Le résultat est* $y = 9$ *voix pour A et* $n - y = 11$ *voix pour B. 1) Comment évaluer la probabilité pour A soit élu ? 2) Si on interroge un* $(n+1)^{\mathrm{e}}$ *électeur, quelle est la probabilité que ce nouvel électeur vote pour A ?*

Commençons par éliminer les raisonnements faux consistant à dire que la probabilité cherchée dans la question 1) est égale à la fréquence empirique, soit 45% (= 9/20), ou nulle car $9/20 < 1/2$. Pour résoudre ce problème et répondre aux deux questions posées, nous introduisons un modèle probabiliste. Nous considérons la proportion totale inconnue, θ, d'électeurs qui voteront pour A lors du second tour de l'élection. La réponse à la première question demande de calculer la probabilité $p(\theta > .5|y)$. A priori, nous supposons pour cela que nous n'avons pas d'information disponible sur cette proportion. Cette hypothèse n'est certainement pas tout à fait vérifiée, et elle peut conduire à des simplifications du modèle qui pourraient être discutées si nécessaire (six men required). Nous supposons donc que la loi *a priori* est uniforme sur $(0, 1)$. En considérant les n voix de manière simultanée, la loi générative est la loi binomiale de paramètres

n et θ

$$p(y|\theta) = \text{binom}(n, \theta)(y) \propto \theta^y (1-\theta)^{n-y}\,.$$

Le calcul de la loi *a posteriori* est immédiat. Par identification avec la formule donnée pour la loi bêta, nous avons

$$p(\theta|y) = \text{beta}(y+1, n+1-y)(\theta)\,, \quad \theta \in (0,1).$$

Notons que les constantes de normalisation ont été inutiles pour trouver ce résultat. Les termes généraux de la loi générative et de la loi a posteriori permettent seuls l'identification de la loi a posteriori. À l'aide de la fonction de répartition de la loi bêta, calculable en `R` grâce à la commande `pbeta`, nous pouvons donner une valeur numérique précise de la probabilité que le candidat A soit élu au second tour

$$p(\theta > .5|y) \approx 0.331811\,.$$

Cela répond donc à la première question. Suivant les mêmes calculs, nous pouvons donner un intervalle de crédibilité, I, tel que $p(\theta \in I|y) = .95$. Dans ce cas précis, cet intervalle est égal à $I = (0.25, 0.65)$.

Afin de répondre à la seconde question, nous devons introduire un concept appelé loi prédictive a posteriori, que nous décrirons avec plus de détails dans la séance suivante. La loi prédictive correspond à la loi d'une nouvelle donnée connaissant les n données précédemment recueillies. Pour la calculer, on calcule la probabilité pour qu'un $(n+1)^{\text{e}}$ électeur vote pour A sachant y en intégrant sur toutes les valeurs possibles de θ (sachant y)

$$p(\text{vote pour A}|y) = \int_0^1 p(\text{vote pour A}|\theta)p(\theta|y)d\theta = \int_0^1 \theta p(\theta|y)d\theta$$

On reconnaît l'espérance de la loi conditionnelle de θ sachant y

$$p(\text{vote pour A}|y) = \frac{y+1}{n+2} \approx 0.4545\,.$$

Sans surprise, ce résultat est très proche de la fréquence empirique égale à 45%.

2.5 Approche par simulation numérique

Les calculs présentés dans le paragraphe précédent sont séduisants car ils donnent une valeur exacte des probabilités cherchées. En réalité, ils cachent des calculs d'intégrale qui ne sont pas toujours possibles dans les modèles probabilistes. La philosophie de ce cours est de se passer le plus possible du calcul intégral en le remplaçant par des estimations obtenues par simulation numérique. Les méthodes considérées sont appelées des méthodes de Monte Carlo.

Revenons un instant sur le modèle considéré dans le paragraphe précédent. Ce modèle consiste à créer une variable aléatoire θ uniformément répartie sur l'intervalle $(0, 1)$, puis, sachant θ, à tirer une variable aléatoire y selon la loi binomiale de paramètre $n = 20$ et θ. Ces deux opérations sont très faciles à réaliser à l'aide des générateurs aléatoires `runif` et `rbinom` du langage `R`. Les commandes ci-dessous génèrent 10000 simulations de ce modèle.

```
theta <- runif(10000)
y.s <- rbinom(10000, 20, theta)
```

Un résultat correspondant à cette simulation est représenté dans la figure 4. La colonne en vert correspond aux valeurs de la proportion θ qui ont permis de générer exactement $y.s = 9$ voix dans un échantillon de taille $n = 20$. Les valeurs en ordonnée représentent les points simulés selon la loi conditionnelle de

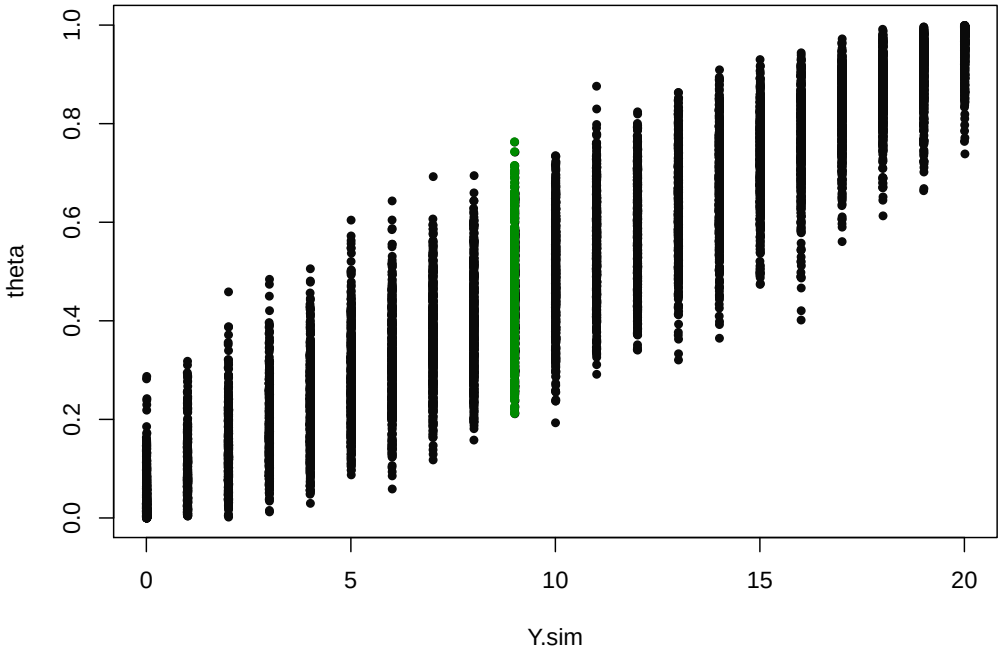


FIGURE 4 – *Simulation de la loi jointe $p(y, \theta)$ pour le modèle binomial ($n = 20$). La colonne en vert représente les points simulés selon la loi conditionnelle de θ sachant $y = 9$. En conservant les valeurs de θ correspondant à ces points, nous obtenons des tirages effectués selon la loi a posteriori.*

θ sachant $y = 9$. En conservant les valeurs de θ correspondant à la colonne de points verts, nous obtenons des tirages de θ effectués selon la loi a posteriori. A partir de ces tirages nous pouvons facilement donner des réponses numériques aux questions posées précédemment. La précision de ces réponses dépend alors du nombres de points présents dans la colonne $y = 9$.

Avec ce procédé de simulation très simple, nous venons en fait de décrire un algorithme de rejet pour simuler la loi a posteriori. En effet, l'algorithme

consiste à rejeter les valeurs du couple (y, θ) telles que la valeur y_s simulée est différente de la valeur observée $y = 9$. En langage `R`, nous pouvons écrire la ligne suivante

```
theta.post <- theta[ y.s == y ]
```

De manière générale, nous pouvons prouver que cet algorithme produit bien des simulations selon la loi *a posteriori*. En effet, notons $p_y(\theta)$ la loi de θ à l'issue de la procédure de rejet, nous avons

$$p_y(\theta) = \sum_{s=1}^{\infty} (1 - p(y))^{s-1} p(y, \theta) = p(\theta|y).$$

Cette formule exprime le fait que le temps d'attente, s, nécessaire à l'observation d'une donnée simulée égale à y est de loi géométrique de paramètre $p(y)$. Le résultat découle des propriétés de la série géométrique.

De retour au problème initial, la probabilité pour que le candidat A soit élu au second tour se calcule numériquement à l'aide de la commande `R` suivante :

```
mean(theta.post > .5)
```

Dans notre simulation, la valeur obtenue est autour de 31%. Cette valeur peut être rendue plus précise en augmentant le nombre de simulations. La figure 5 montre l'histogramme de la loi a posteriori obtenu par l'algorithme de rejet. La courbe superposée à cet histogramme correspond à la courbe de la densité théorique, dans ce cas égale à la loi bêta(10,12). Nous voyons que l'approximation numérique est tout à fait satisfaisante.

2.6 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

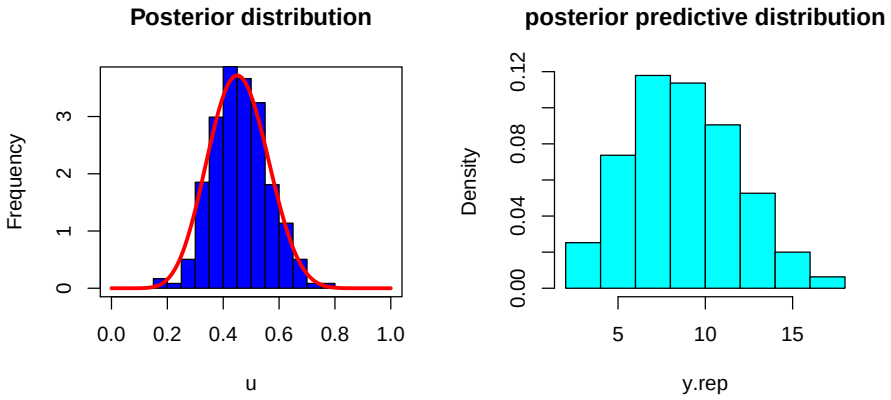


FIGURE 5 – *A gauche : Loi a posteriori obtenue par simulation (en rouge la loi théorique). A droite : Loi prédictive a posteriori pour de nouveaux échantillons.*

2.7 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. Chez les mammifères, les males possèdent un chromosome X et un chromosome Y, tandis que les femelles possèdent deux copies du chromosome X. Chez un individu, chacun des 2 chromosomes est hérité d'un seul des 2 parents.

L'hémophilie est une maladie génétique humaine liée à un allèle récessif localisé sur le chromosome X. Cela signifie qu'une femme portant l'allèle responsable de la maladie sur l'un de ses deux chromosomes n'est pas affectée par la maladie. La maladie est, en revanche, généralement fatale pour une femme qui porterait les deux copies mutées (un événement très rare de toutes façons).

1. Une femme sait que son frère est affecté, mais que son père ne l'est pas.

A priori, quelle est la loi de son propre état, θ, que l'on considère comme un paramètre binaire (porteuse ou non de l'allèle récessif) ?

2. On prend maintenant en compte les données suivantes. Cette femme a deux fils (non-jumeaux) et aucun des deux n'est affecté. Quelle est la loi a posteriori du paramètre θ ?

Exercice 2. La loi de l'ouest. A Las Vegas, on rencontre une proportion p de tricheurs, $0 < p < 1$. Lorsque l'on joue contre un tricheur, la probabilité de gagner une partie est nulle, tandis que, lorsque l'on joue contre une personne honnête, cette probabilité est $1/2$. On joue et on perd une partie à Las Vegas. Quelle est la probabilité d'avoir joué contre un tricheur ?

Exercice 3. Tarzan est-il un bon biologiste de terrain ? La loi du célèbre naturaliste Ramon Homstein-Grunberger indique que la taille des girafes pigmées est uniformément répartie entre 0.5 et 1.5m. Adepte du scepticisme, Tarzan fait une sortie dans la jungle avec son mètre mesureur. Au retour il prétend que les girafes pigmées ne dépassent pas 1.25m. Le problème consiste à estimer le nombre de girafes mesurées par Tarzan, et à donner la probabilité qu'il ait mesuré moins de 10 girafes. Pour cela, on considère que les observation sont des réalisations indépendantes, $(u_i)_{i\geq 1}$, de loi uniforme sur $(0,1)$ (correspondant aux tailles des girafes auxquelles on soustrait 0.5m). Soit $t \in (0,1)$.

1. Soit $k \geq 1$. On pose $y_k = \max_{i=1,\dots,k} u_i$. Montrer que

$$p(y_k \leq t) = t^k \,.$$

2. Soit $n \geq 1$. On observe le résultat du maximum d'un nombre inconnu et aléatoire, θ, de variables u_i. On suppose que θ est indépendant des u_i et, a priori, de loi uniforme sur $\{1, \ldots, n\}$. Déterminer la fonction de répartition de la variable y_θ. (Rappel : $\sum_{\ell=0}^{n} t^\ell = (1 - t^{n+1})/(1 - t)$.)
3. Tarzan observe la réalisation de l'événement $(y_\theta \leq t)$ pour $t = 3/4$. Montrer que la loi conditionnelle de θ sachant $(y_\theta \leq t)$ est donnée par

$$p(\theta = k | y_\theta \leq t) = \frac{t^{k-1}(1-t)}{1-t^n}, \quad k = 1, \ldots, n.$$

4. Démontrer que, pour tout $k = 1, \ldots, n$, nous avons

$$p(\theta = k | y_\theta \leq t) = p(\theta_\star = k | \theta_\star \leq n).$$

où $\theta_\star$ est une variable de loi géométrique de paramètre $1 - t$.
5. Déterminer la limite de la loi a posteriori de θ lorsque n tend vers l'infini. Quelle est l'espérance mathématique de la loi limite ?
6. Donner une estimation du nombre de girafes mesurées par Tarzan s'appuyant sur les questions précédentes. Calculer la probabilité que Tarzan ait mesuré moins de 10 girafes.

Exercice 4. Estimation de la fréquence revisitée par Laplace. Pierre-Simon observe qu'un événement (de probabilité inconnue) se produit $y = 11$ fois lors de la répétition de $n = 25$ épreuves indépendantes. Il souhaite estimer la probabilité que cet événement se réalise à nouveau lors de la 26^{e} épreuve. Il note θ la probabilité inconnue, et la traite comme variable aléatoire de loi uniforme sur $(0, 1)$.

1. Calculer la probabilité conditionnelle $p(y|\theta)$.

2. Déterminer la loi conditionnelle $p(\theta|y)$ (loi a posteriori).
3. Calculer l'espérance conditionnelle de θ sachant y. Que se passe-t-il lorsque $n \to \infty$? Interprétation ?
4. Programmer un algorithme en langage R permettant de simuler la loi conditionnelle $p(\theta|y)$.
5. Calculer la probabilité que l'événement se réalise lors d'une épreuve future théoriquement, puis donner une estimation numérique à l'aide de l'algorithme de simulation.

Exercice 5. Recherche d'un objet. Un objet a disparu dans une zone déterminée divisée en trois régions de même surface pour les recherches. La recherche commence par la fouille de la région 1. Soit q_1 la probabilité de ne pas trouver l'objet dans cette région conditionnellement à l'événement qu'il s'y trouve effectivement. Quelle est la probabilité que l'objet se trouve dans les régions 2 ou 3 sachant que les recherches dans la région 1 ont été infructueuses.

3 Séance 3 : Espérance conditionnelle – Loi prédictive

3.1 Introduction

Nous discutons dans cette séance de la valeur prédictive d'un modèle probabiliste et de la manière d'évaluer les prédictions d'un modèle.

3.2 Définitions

Nous considérons le couple (y, θ), où y est un vecteur de dimension n et θ un vecteur de dimension J, de loi jointe $p(y, \theta)$. L'**espérance conditionnelle** de θ sachant y est définie comme l'espérance de la loi conditionnelle $p(\theta|y)$

$$\mathrm{E}[\theta|y] = \int \theta p(\theta|y) d\theta \,.$$

Lorsque la dimension du paramètre θ est supérieure à 1, l'intégrale est calculée composante par composante, et l'espérance est définie comme un vecteur de taille J. Dans un sens à préciser, l'espérance conditionnelle représente la meilleure estimation que l'on peut faire du paramètre θ à l'issue de la phase d'apprentissage. Nous illustrons le calcul de l'espérance conditionnelle dans une section séparée.

Un concept particulièrement utile en matière de prédiction est la notion de **loi prédictive** a posteriori. Afin de définir cette loi, considérons un vecteur de données, y, et un modèle probabiliste pour ces données, décrit par la loi a priori, $p(\theta)$, et par la loi générative, $p(\theta|y)$. La loi prédictive est la loi d'une nouvelle donnée, y_{rep}, prédite par la loi générative suite à l'observation du vecteur y

$$p(y_{\mathrm{rep}}|y) = \int p(y_{\mathrm{rep}}|\theta) p(\theta|y) d\theta.$$

En condensé, cette intégrale se réécrit de la manière suivante

$$p(y_{\text{rep}}|y) = \text{E}[p(y_{\text{rep}}|\theta)|y],$$

où le calcul de la moyenne porte sur la variable aléatoire θ. L'intégrale précédente traduit le mécanisme de génération de données a posteriori par le modèle. Dans une première étape, les données, y, permettent de mettre à jour l'incertitude sur le paramètre θ. Cela est possible grâce à la formule de Bayes et au calcul de la loi a posteriori, $p(\theta|y)$. Dans une seconde étape, nous utilisons la loi a posteriori pour tirer de nouveaux paramètres, et nous créons des répliques des données selon la loi générative.

Notons que l'intégrale définissant la loi générative peut être très difficile à évaluer mathématiquement. Nous aurons le plus souvent recours à un algorithme de Monte Carlo pour échantillonner cette loi. Cet algorithme résume l'interprétation que nous venons de faire de la définition de la loi prédictive.

```
1. Simuler theta selon la loi p(theta|y)
2. Simuler y_rep selon la loi p(y_rep|theta)
```

La loi prédictive peut être utilisée pour évaluer la capacité du modèle à capturer les caractéristiques essentielles des données. Ces caractéristiques sont bien souvent spécifiques au problème considéré, et nous entrons ici dans le contexte de la vérification du modèle (model checking). Dans ce contexte, il faut définir avec précaution les aspects d'un modèle que l'on souhaite évaluer. Pour reprendre une citation du célèbre statisticien Cox, *tous les modèles sont faux, mais certains modèles faux peuvent être utiles*. En effet un modèle de la météo de Grenoble peut mal prévoir la météo à Nice. Nous devons donc définir avec soin les critères

(ou les statistiques) permettant l'évaluation d'un modèle.

3.3 Propriétés de l'espérance conditionnelle

Dans cette section, nous présentons deux propriétés importantes de l'espérance conditionnelle. La première propriété généralise la formule des probabilités totales, vue en première année. La deuxième propriété établit l'optimalité prédictive de l'espérance conditionnelle au sens des moindres carrés.

Propriété 1 : Formule de conditionnement. Nous avons

$$\mathrm{E}[\mathrm{E}[\theta|y]] = \mathrm{E}[\theta]$$

En d'autres termes, lorsque l'espérance conditionnelle de θ est moyennée sur l'ensemble des données, nous retrouvons l'espérance tout simplement

$$\mathrm{E}[\theta] = \int \mathrm{E}[\theta|y]p(y)dy.$$

La preuve de ce résultat résulte d'une inversion de l'ordre des intégrales intervenant dans le calcul de l'espérance

$$\int \mathrm{E}[\theta|y]p(y)dy = \int \left(\int \theta p(\theta|y)d\theta \right) p(y)dy$$

Sous réserve d'existence, inversons les symboles d'intégration

$$\int \mathrm{E}[\theta|y]p(y)dy = \int \theta \left(\int p(\theta|y)p(y)dy \right) d\theta \,.$$

Le résultat découle du fait que

$$p(\theta) = \int p(\theta|y)p(y)dy \,.$$

Propriété 2 : Espérance conditionnelle et prédiction. Dans ce paragraphe, nous établissons une propriété d'optimalité de l'espérance conditionnelle. Vue comme une fonction des données, y, **l'espérance conditionnelle**

$\mathrm{E}[\theta|y]$ est le prédicteur optimal du paramètre θ au sens des moindres carrés.

Supposons donc que l'on observe y, et que l'on souhaite prédire la valeur de θ, en s'appuyant sur l'observation de y. On note $g(y)$ la fonction servant de *prédicteur* pour θ. On suppose que $\mathrm{E}[g^2(y)] < \infty$. On dit q'un prédicteur, $g^\star(y)$ est optimal s'il est solution du problème de minimisation suivant

$$g^\star(y) = \arg\min_{g \in L^2} \{\mathrm{E}[(\theta - g(y))^2]\},$$

où L^2 représente l'ensemble des prédicteurs tels que $\mathrm{E}[g^2(y)] < \infty$.

Voyons pourquoi la solution de ce problème de minimisation est donnée par l'espérance conditionnelle. Tout d'abord, grâce à la propriété 1, nous avons

$$\mathrm{E}[(\theta - g(y))^2] = \mathrm{E}[\mathrm{E}[(\theta - g(y))^2|y]].$$

Il suffit donc de minimiser l'expression $\mathrm{E}[(\theta - g(y))^2|y]$. En développant le terme quadratique à l'intérieur de l'espérance, nous obtenons

$$\mathrm{E}[(\theta - g(y))^2|y] = \mathrm{E}[(\theta - \mathrm{E}[\theta|y])^2|y] + \mathrm{E}[(g(y) - \mathrm{E}[\theta|y])^2|y].$$

En remarquant que le terme $\mathrm{E}[(g(y) - \mathrm{E}[\theta|y])^2|y]$ est toujours positif ou nul, nous obtenons le résultat annoncé

$$\mathrm{E}[(\theta - g(y))^2|y] \geq \mathrm{E}[(\theta - \mathrm{E}[\theta|y])^2|y].$$

Puisque une variable aléatoire positive d'espérance nulle est nécessairement p.s. nulle, cela implique que $g^\star(y) = \mathrm{E}[\theta|y]$.

3.4 Exemple : le modèle d'erreur

Pour illustrer le concept d'apprentissage et le calcul de la loi prédictive, considérons un exemple très simple. Nous observons la valeur d'une mesure

unidimensionnelle continue, y, effectuée avec une erreur. Nous ne connaissons donc pas la véritable valeur de cette mesure, θ, et nous souhaitons l'estimer. Pour cela, nous disposons d'un modèle de l'erreur. En particulier, l'écart-type de l'erreur est connu exactement, et il est égal à σ.

Nous considérons le modèle défini par

$$\theta \sim p(\theta) = \mathrm{N}(0, \sigma_0^2)$$

et

$$y|\theta \sim N(\theta, \sigma^2)\,.$$

La constante σ_0^2 représente la **variance** de la loi a priori. Nous notons $\beta_0 = 1/\sigma_0^2$. La constante β_0 est appelée la **précision** de la loi a priori. Nous notons $\beta = 1/\sigma^2$ la précision de la loi générative. La loi $\mathrm{N}(\theta, \sigma^2)$ est la loi normale de moyenne θ et de variance σ^2. Elle admet pour densité

$$p(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-(y-\theta)^2/2\sigma^2)\,, \quad y \in \mathbb{R}.$$

Pour modéliser une incertitude a priori absolue sur la valeur de θ, il est tentant de choisir $\sigma_0^2 = \infty$. Cela correspond à une loi a priori dégénérée. Malgré le caractère non-défini de cette loi, nous verrons que les calculs ont tout de même du sens, loi a posteriori étant bien définie. Le modèle se résume donc de la manière suivante. La valeur de la mesure θ est inconnue, et supposée tirée d'une loi non-informative ou plate. On observe une version bruitée de θ, notée y, obtenue par l'ajout à θ d'une perturbation gaussienne d'écart-type σ.

La loi a posteriori s'obtient directement en effectuant le produit des densités

$$p(\theta|y) \propto \exp\left(-\frac{1}{2}(\beta_0\theta^2 + \beta(y-\theta)^2)\right)$$

En réorganisant les termes à l'intérieur de l'exponentielle, nous obtenons

$$p(\theta|y) \propto \exp\left(-\frac{1}{2}\beta_1(\theta - m_1)^2)\right)$$

où

$$\begin{cases} \beta_1 & = \beta_0 + \beta \\ m_1 & = \beta y/(\beta + \beta_0)\,. \end{cases}$$

La loi a posteriori est donc la loi normale de moyenne m_1 et $\sigma_1^2 = 1/\beta_1$. Dans le cas où l'incertitude a priori est totale ($\beta_0 = 0$), nous obtenons

$$p(\theta|y) = \mathrm{N}(y, \sigma^2)(\theta)\,.$$

En d'autres termes, le modèle d'erreur s'applique directement, et θ s'estime à l'aide de y en ajoutant un bruit gaussien centré d'écart-type σ. Dans le cas où la **loi a priori est informative** ($\beta_0 > 0$), nous obtenons une loi a posteriori plus précise que les lois générative et a priori. Lorsque $\beta_0 > 0$ et y est positif, nous obtenons que

$$\mathrm{E}[\theta|y] < y\,.$$

Dans le cas $y < 0$, l'inégalité est renversée. Nous observons donc un phénomène de rétrécissement (*shrinkage*), aussi appelé régularisation. L'introduction d'une loi a priori informative induit une modification de l'estimation de la valeur réelle qui la tire vers la valeur 0. La régularisation peut être un aspect important des modèles afin d'obtenir de meilleures qualités prédictives. Elle peut aussi jouer un rôle numérique lors de l'exécution des algorithmes de Monte Carlo que nous verrons plus loin dans ce cours.

Nous déterminons maintenant la loi prédictive en supposant que $\beta_0 = 0$. Le calcul de la loi prédictive pour $\beta_0 > 0$ est un exercice à faire en travaux dirigés.

Soit y^* (plutôt que y_{rep}) une réplique des données. Nous souhaitons calculer la loi

$$p(y^*|y) \propto \int \exp\left(-\frac{1}{2}(\beta(y^*-\theta)^2+\beta(y-\theta)^2)\right) d\theta\,.$$

En séparant les termes de l'exponentielle dépendant de θ, nous avons

$$(y^*-\theta)^2+(y-\theta)^2 = \frac{1}{2}(y-y^*)^2 + 2\left(\theta - \left(\frac{1}{2}(y+y^*)\right)\right)^2$$

Ainsi, nous avons,

$$p(y^*|y) \propto \exp\left(-\frac{1}{4}\beta(y-y^*)^2\right) \int \exp\left(-\beta\left(\theta - \left(\frac{1}{2}(y+y^*)\right)\right)^2\right) d\theta\,.$$

L'intégrale apparaissant dans le membre de droite est une constante que l'on peut calculer en effectuant un changement de variables affine. La valeur exacte est connue grâce à l'expression de la loi gaussienne (exercice). Il est important de remarquer que cette valeur est indépendante de y^*. Cela n'est pas utile de faire le calcul explicite, nous obtenons directement la loi prédictive

$$p(y^*|y) = \mathrm{N}(y, 2\sigma^2)(y^*)\,.$$

La loi prédictive s'interprète donc comme la somme de deux variables indépendantes de loi normale. Sachant y, la première variable correspond à θ, de loi $\mathrm{N}(y,\sigma^2)$ et la seconde variable correspond au modèle d'erreur $\mathrm{N}(0,\sigma^2)$ que l'on peut ajouter à θ.

3.5 Approche par simulation numérique

Nous montrons dans ce paragraphe comment il possible de générer un histogramme de la loi prédictive sans effectuer le calcul de l'intégrale. La simulation s'appuie sur le générateur aléatoire `rnorm` du langage `R`. Les commandes ci-

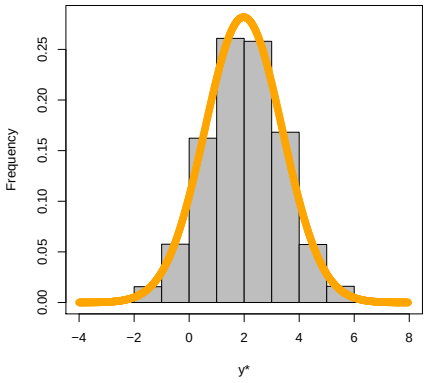


FIGURE 6 – *Loi prédictive a posteriori pour le modèle d'erreur gaussien. La donnée observée est $y = 2$ et la variance du bruit, supposée connue, est égale à 1. La courbe théorique obtenue grâce au calcul intégral est superposée à l'histogramme.*

dessous génèrent 10000 simulations de la loi prédictive pour la valeur observée $y = 2$ et pour la variance connue $\sigma^2 = 1$.

```
theta <- rnorm(10000, mean = y, sd = 1) #loi a posteriori

y.etoile <- rnorm(10000, mean = theta, sd= 1) #loi générative
```

Un résultat correspondant à cette simulation est représenté dans la figure 6. Nous observons une bonne adéquation à la courbe obtenue par le calcul théorique.

3.6 Données séquentielles et apprentissage

Nous supposons maintenant que nous pouvons répéter la mesure de la valeur inconnue θ. Nous n'observons plus une unique valeur continue, y, mais une suite de valeurs effectuées avec erreur. Nous notons cette suite d'observations $y_1, \ldots, y_n$, et nous cherchons à prédire de manière optimale la valeur de θ au

fur et à mesure que les observations arrivent. Cela revient à calculer l'espérance conditionnelle de θ sachant y_1, puis y_1, y_2, etc.

Les calculs s'effectuent en remarquant que les lois a posteriori calculées séquentiellement sont toujours des lois normales. On identifie l'espérance conditionnelle dans le terme quadratique en reconnaissant la moyenne de la loi normale.

Suite à la première observation, y_1, nous avons, d'après les calculs effectués précédemment, le résultat suivant

$$\mathrm{E}[\theta|y_1] = m_1 = \frac{\beta y_1}{\beta_0 + \beta}\,.$$

Pour calculer $\mathrm{E}[\theta|y_1, y_2]$, notons que

$$p(\theta|y_1, y_2) \propto p(y_2|\theta)p(\theta|y_1)$$

A l'aide d'un calcul similaire à celui effectué dans la section précédente (*cf* Exercice 2), nous obtenons

$$\mathrm{E}[\theta|y_1, y_2] = m_2 = \frac{\beta_1 \mathrm{E}[\theta|y_1] + \beta y_2}{\beta_1 + \beta}\,,$$

où $\beta_1 = \beta_0 + \beta$. Suite à n observations, $y_1, \ldots, y_n$, nous obtenons

$$\mathrm{E}[\theta|y_1, \ldots, y_n] = m_n = \frac{\beta_{n-1} \mathrm{E}[\theta|y_1, \ldots, y_{n-1}] + \beta y_n}{\beta_{n-1} + \beta}\,, \tag{1}$$

où $\beta_{n-1} = \beta_{n-2} + \beta$. Les valeurs obtenues pour des lois a priori différentes les unes des autres sont illustrées dans la figure 7.

Lorsque la loi a priori est non-informative ($\beta_0 = 0$), nous obtenons un résultat très simple

$$\mathrm{E}[\theta|y_1, \ldots, y_n] = m_n = \frac{\sum_{i=1}^{n} y_i}{n}\,.$$

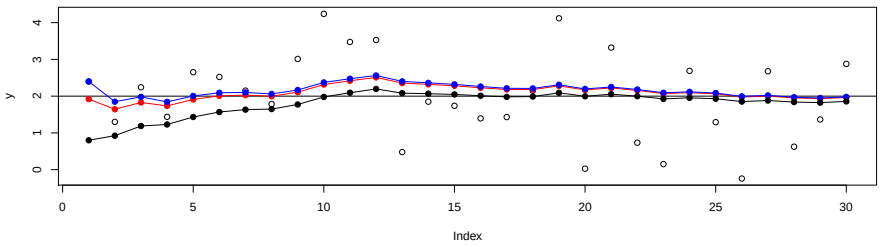


FIGURE 7 – *Espérance conditionnelle pour le modèle d'erreur gaussien. La valeur cachée est $\theta = 2$ et la variance du bruit est égale à 1. Les courbes montrent l'apprentissage de la valeur cachée pour différents a priori : $\beta_0 = 0$ (bleu), $\beta_0 = .25$ (rouge), $\beta_0 = 2$ (rouge). On observe le phénomène d'aplatissement (shrinkage) lorsque la précision β_0 grandit.*

Cela illumine l'intérêt de considérer la moyenne empirique. Lorsque n tend vers l'infini, nous savons grâce à la loi des grands nombres que la moyenne empirique converge vers la valeur inconnue. Lorsque l'on dispose de n données (n est fini), la moyenne empirique représente la meilleur prédiction que l'on peut faire de la valeur inconnue au sens des moindres carrés.

3.7 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

3.8 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. Simulation et calcul d'une loi prédictive.

Nous observons la valeur d'une mesure unidimensionnelle continue, y, effectuée avec une erreur. Nous ne connaissons donc pas la véritable valeur de cette mesure, θ, et nous souhaitons l'estimer. Pour cela, nous disposons d'un modèle de l'erreur. En particulier, l'écart-type de l'erreur est connu exactement, et il est égal à σ.

Nous considérons le modèle défini par

$$\theta \sim p(\theta) = \mathrm{N}(0, \sigma_0^2)$$

et

$$y|\theta \sim \mathrm{N}(\theta, \sigma^2)$$

On suppose que σ_0^2 est une valeur finie.

1. Retrouver l'expression de la loi a posteriori, reconnaître cette loi. Ecrire un algorithme de simulation de la loi prédictive du modèle pour la donnée y.
2. On suppose que $\sigma = 1$ et $\sigma_0 = 2$. On observe $y = 3$. Ecrire un algorithme de simulation de la loi prédictive, le programmer en langage `R` et générer 10000 tirages de cette loi.

3. Calculer la loi prédictive théoriquement et superposer le résultat à un histogramme obtenu par la méthode de simulation précédente.

Exercice 2. Données séquentielles et apprentissage.

1. Reprendre le modèle d'erreur gaussien et établir la formule (1) par récurrence.
2. On suppose que $\sigma = 1$ et $\beta_0 = 1/4$. Commenter brièvement la figure 7.
3. On suppose désormais un modèle d'erreur poissonnien. Dans ce modèle, la variable cachée, θ, est une variable positive. On supposera

$$p(\log(\theta)) \propto 1$$

et

$$p(y|\theta) \propto \theta^y \exp(-\theta)$$

Calculer l'espérance conditionnelle, $\mathrm{E}[\theta|y]$, lorsque l'on dispose d'une observation. Calculer l'espérance conditionnelle, $\mathrm{E}[\theta|y_1, \ldots, y_n]$, lorsque l'on dispose de n observations.

Exercice 3. Loi jointe du minimum et du maximum de deux variables de loi exponentielle. On considère un couple (y, θ) de densité définie par

$$\forall (x, y) \in \mathbb{R}^2\,, \quad p(y, \theta) = \begin{cases} 2\, e^{-\theta - y} & \text{si } 0 < y < \theta\,, \\ 0 & \text{sinon.} \end{cases}$$

1. Rappeler le principe de simulation d'un couple de variables aléatoires de densité $p(y, \theta)$.
2. Calculer la loi marginale, la loi a priori, la loi générative et la loi a posteriori pour le couple (y, θ).

3. Montrer que la loi a posteriori peut se représenter comme la loi de la somme $y + z$, où z est une variable de loi exponentielle de paramètre 1 pour tout $y > 0$.

4. Ecrire un algorithme de simulation du couple (y, θ) faisant explicitement appel à un changement de variables. Prouver la validité de cet algorithme.

5. Soit `unif(0,1)` un générateur aléatoire de loi uniforme. On considère l'algorithme suivant

```
Repeat
u = -log(unif(0,1)) ;
v = -log(unif(0,1)) ;
Jusqu'a (u < v) ;
 y <- u ; theta <- v ;
```

   Argumenter (sans calcul) du fait que la loi du couple (y, θ) en sortie de cet algorithme est identique à la loi du couple $(\min(u, v), \max(u, v))$, où u, v sont des variables indépendantes et de loi exponentielle de paramètre 1.

6. En déduire que la loi de y en sortie de l'algorithme précédent est la loi exponentielle de paramètre 2.

7. Soit $y > 0$ et soit $t > y$. En sortie de l'algorithme précédent, justifier que l'on a

$$p(\theta \leq t|y) = \frac{p(y < v \leq t)}{p(v > y)},$$

   et calculer la densité de la loi conditionnelle $p(\theta|y)$.

8. Démontrer que le couple (y, θ) en sortie de l'algorithme précédent admet

$p(y, \theta)$ pour densité jointe.

9. (Facultatif) Soit $y > 0$. En utilisant la question 3, montrer que $\mathrm{E}[\theta|y] = y + 1$. En déduire que $\mathrm{E}[y\theta|\theta] = \theta^2 + \theta$.

10. (Facultatif) Déduire de la question précédente l'espérance $\mathrm{E}[\theta]$ et la covariance $\mathrm{Cov}(y, \theta)$.

4 Séance 4 : Inférence probabiliste pour le modèle gaussien : moyenne et variance.

4.1 Introduction

Dans la séance précédente, nous avons étudié comment il est possible d'apprendre la valeur d'un paramètre caché à partir d'observations séquentielles erronées de cette valeur, mais ayant une erreur de variance connue. Dans cette séance, nous généralisons cette approche. Nous supposons que nous observons simultanément n répliques bruitées de ce paramètre inconnu et discutons le cas où le modèle d'erreur n'est pas connu.

Soit m le paramètre d'intérêt et σ^2 la variance du bruit, que nous supposons tour à tour connue ou inconnue. On dira que σ^2 est un paramètre de nuisance. Nous ne souhaitons pas particulièrement connaître sa valeur, mais nous devons modéliser son incertitude afin d'estimer ou d'apprendre m. Nous supposons que nous observons n réalisations indépendantes de la loi générative $\mathrm{N}(m, \sigma^2)$. Les observations sont notées $y = y_1, \ldots, y_n$.

Nous examinons tour à tour les cas suivants :

1. σ^2 connu, on cherche la loi de $\theta = m$ sachant y,
2. m connu, on cherche la loi de $\theta = \sigma^2$ sachant y,
3. m et σ^2 tout deux inconnus, on cherche la loi de $\theta = (m, \sigma^2)$ sachant y.

4.2 Quelques définitions : Matrice de covariance, loi gaussienne multivariée

Considérons un couple (x_1, x_2) de variables aléatoires réelles. Nous appelons **covariance** du couple (x_1, x_2) la grandeur définie par

$$\mathrm{Cov}(x_1, x_2) = \mathrm{E}[x_1 x_2] - \mathrm{E}[x_1]\mathrm{E}[x_2]\,.$$

La covariance du couple (x_1, x_2) est **nulle** lorsque les variables aléatoires x_1 et x_2 sont **indépendantes**. Nous voyons aussi que

$$\mathrm{Cov}(x_1, x_1) = \mathrm{Var}(x_1).$$

La covariance possède quelques propriétés intéressantes. Tout d'abord, elle est symétrique

$$\mathrm{Cov}(x_1, x_2) = \mathrm{Cov}(x_2, x_1).$$

De plus, la covariance est bi-linéaire

$$\mathrm{Cov}(ax_1 + by_1, cx_2 + dy_2) = ac\mathrm{Cov}(x_1, x_2) + \cdots + bd\mathrm{Cov}(y_1, y_2)\,.$$

De plus, elle vérifie l'inégalité de Cauchy-Schwarz

$$\mathrm{Cov}(x_1, x_2)^2 \leq \mathrm{Var}(x_1)\mathrm{Var}(x_2)\,.$$

Ainsi, on pourra définir le **coefficient de corrélation** entre x_1 et x_2 comme la grandeur comprise entre -1 et $+1$ suivante

$$\rho = \rho(x_1, x_2) = \frac{\mathrm{Cov}(x_1, x_2)}{\sqrt{\mathrm{Var}(x_1)\mathrm{Var}(x_2)}}$$

Matrice de covariance. Nous appelons **matrice de covariance** la matrice C dont le terme général est

$$c_{ij} = \mathrm{Cov}(x_i, x_j)$$

Nous supposons que les valeurs propres de la matrice de covariance sont non-nulles (strictement positives).

Couple gaussien. Nous dirons que le couple (x_1, x_2) est gaussien, de moyenne $m = (m_1, m_2)$ et de matrice de covariance C, si la densité de pro-

babilité de ce couple est donnée par la formule suivante

$$p(x_1, x_2) = \frac{1}{2\pi} \frac{1}{\sqrt{\det(C)}} \exp\left(-\frac{1}{2}(x-m)^T C^{-1}(x-m)\right) ,$$

pour tout $(x_1, x_2) \in \mathbb{R}^2$. Les propriétés des couples et des vecteurs gaussiens seront étudiées en détails plus loin dans le cours de MPA. Pour l'instant, nous admettrons (sans peine) que les lois marginales et conditionnelles d'un couple gaussien sont elles-aussi gaussiennes. Notons encore que la formule ci-dessus peut servir à définir un vecteur gaussien en dimension d, en remplaçant la constante par $1/(2\pi)^{d/2}$.

4.3 Apprentissage probabiliste de la moyenne (m) et de la variance (σ^2)

Nous supposons que nous observons n réalisations indépendantes de la loi générative $\mathrm{N}(m, \sigma^2)$. Ces observations sont notées $y = y_1, \ldots, y_n$. Soit m notre paramètre d'intérêt et σ^2 la variance du bruit génératif.

Cas σ^2 connu. Supposons dans un premier temps, que σ^2 est connu. Dans ce cas, nous posons $\theta = m$ et nous nous retrouvons dans une situation familière, déjà rencontrée lors de la dernière séance. Nous supposons que la loi de θ est non-informative ($\beta_0 = 0$)

$$p(\theta) \propto 1 .$$

Les observations $(y_1, \ldots, y_n)$ sont indépendantes, nous avons donc

$$p(y_1, \ldots, y_n|\theta) = \prod_{i=1}^{n} p(y_i|\theta)$$

et nous pouvons écrire l'expression de la loi générative

$$p(y_1, \ldots, y_n|\theta) \propto \prod_{i=1}^{n} \exp\left(-\frac{1}{2\sigma^2}(y_i - \theta)^2\right) .$$

En utilisant la propriété de l'exponentielle, nous voyons que la loi générative est une loi gaussienne multivariée, de matrice de covariance diagonale

$$p(y_1, \ldots, y_n|\theta) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\theta)^2\right).$$

Pour obtenir la loi a posteriori, considérons la moyenne empirique

$$\bar{y} = \sum_{i=1}^{n} y_i/n$$

et développons le carré à l'intérieur de l'exponentielle

$$\sum_{i=1}^{n}(y_i-\theta)^2 = \sum_{i=1}^{n}(y_i-\bar{y})^2 + n(\bar{y}-\theta)^2\,.$$

Le premier terme du second membre est indépendant de θ, nous avons donc

$$p(\theta|y) \propto p(y|\theta)p(\theta) \propto \exp\left(-\frac{n}{2\sigma^2}(\bar{y}-\theta)^2\right).$$

Ainsi, nous voyons que toute l'information utile pour apprendre le paramètre $\theta = m$ est résumée dans la statistique $\bar{y}$

$$p(\theta|y) = p(\theta|\bar{y})\,.$$

En conclusion, la loi a posteriori est la loi normale $\mathrm{N}(\bar{y}, \sigma^2/n)$.

Cas m connu. Nous traitons le cas symétrique au précédent où l'on cherche maintenant à apprendre $\theta = \sigma^2$ sachant la valeur de m. Ce cas n'est pas très réaliste, mais nous verrons que son étude est très utile ensuite.

Rappelons tout d'abord les définitions des lois χ^2_n et inverse χ^2_n à n degrés de liberté. Il s'agit de lois de variables positives. La densité de la loi χ^2_n s'écrit de la manière suivante :

$$p(x) \propto x^{n/2-1}e^{-x/2}\,, \quad x > 0\,.$$

La loi Inv-$\chi^2(\nu, s^2)$, est la loi de la variable définie par

$$x = \frac{\nu s^2}{\chi^2_\nu} \,.$$

La densité de la loi Inv-$\chi^2(\nu, s^2)$ est proposée en exercice :

$$p(x) \propto \frac{1}{x^{\nu/2+1}} e^{-\frac{\nu s^2}{2x}} \,, \quad x > 0 \,.$$

Le cas $\nu = 0$ est dégénéré. Il correspond à une loi non-informative

$$p(\log x) \propto 1 \quad \text{ou encore} \quad p(x) \propto 1/x \,.$$

Cela signifie que les valeurs de x sont uniformément réparties sur une échelle logarithmique.

Afin de traiter le cas de l'apprentissage de $\theta = \sigma^2$ (m connu), nous considérons un modèle dans lequel la loi a priori n'admet pas de densité

$$p(\theta) \propto \frac{1}{\theta} \,.$$

La loi d'échantillonnage est inchangée. On peut l'écrire de la manière suivante

$$p(y_1, \ldots, y_n|\theta) \propto \frac{1}{\theta^{n/2}} \exp\left(-\frac{n}{2\theta} s_n^2\right)$$

où

$$s_n^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - m)^2 \,.$$

En multipliant $p(y_1, \ldots, y_n|\theta)$ par l'inverse de θ, nous obtenons la loi a posteriori de la variance σ^2. Nous reconnaissons une loi de la famille Inv-χ^2, dont les paramètres sont faciles à identifier

$$\sigma^2|y \sim \text{Inv}{-}\chi^2(n, s_n^2) \,.$$

Apprentissage du couple (m, σ^2). Dans ce paragraphe, ni m ni σ^2 ne sont connus. Cela correspond à la situation la plus réaliste. Nous souhaitons donc apprendre le paramètre composite $\theta = (m, \sigma^2)$. Afin de spécifier un modèle pour le couple (θ, y), nous choisissons une loi a priori non-informative

$$p(m, \sigma^2) \propto \frac{1}{\sigma^2}\,.$$

Cette formulation implique que la loi a priori est uniforme pour m ($p(m) \propto 1$), et que la loi a priori est uniforme pour $\log(\sigma^2)$. Une loi uniforme sur une échelle logarithmique se justifie parce que la variance est un paramètre d'échelle, et la grandeur d'ordre de la variation est plus importante que la variation elle-même. Notons enfin qu'une loi uniforme sur σ^2 plutôt que sur son logarithme, conduirait à une loi a posteriori non définie.

Les calculs détaillés de la loi a posteriori seront effectués en exercice. Après arrangement des termes présents à l'intérieur de l'exponentielle, nous obtenons

$$p(m, \sigma^2|y) \propto \frac{1}{(\sigma^2)^{n/2+1}} \exp\left(-\frac{1}{2\sigma^2}((n-1)s_{n-1}^2 + n(\bar{y} - m)^2)\right)$$

où l'on a introduit une statistique très courante, correspondant à la **variance empirique** de l'échantillon y

$$s_{n-1}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2\,.$$

Contrairement aux cas précédents, nous nous trouvons en présence d'une loi multivariée, dont l'expression est explicite, mais qui ne correspond à **aucune loi connue**. Il est important, à ce niveau, d'insister que cela est la règle générale en apprentissage probabiliste. Les cas où l'on tombe sur des expressions connues restent exceptionnels.

Dans l'esprit général de ce cours, nous aurons recours à la simulation pour étudier la loi $p(m, \sigma^2|y)$. Etant donné l'échantillon y, nous utiliserons l'algorithme suivant :

1. Simuler σ^2 selon la loi $p(\sigma^2|y)$,
2. Sachant σ^2, simuler m selon la loi $p(m|\sigma^2, y)$.

Nous avons déjà résolu l'opération à effectuer lors de la seconde étape de cet algorithme. En effet, nous avons obtenu précédemment que la loi $p(m|\sigma^2, y)$ était une loi normale de moyenne $\bar{y}$ et de variance σ^2/n. En effet cette situation correspond au cas où σ^2 est connu.

Afin de simuler la loi $p(\sigma^2|y)$, nous devons identifier cette loi. Il s'agit de la marginale de la loi a posteriori dont l'expression est donnée plus haut dans ce paragraphe

$$p(\sigma^2|y) = \int p(m, \sigma^2|y) dm \,.$$

Nous devons donc effectuer le calcul d'une intégrale. C'est un exercice que l'on peut résoudre, et nous proposons de le faire en travaux dirigés. Nous trouvons que la loi de σ^2 est une loi Inv-χ^2

$$\sigma^2|y \sim \text{Inv}\chi^2(n-1, s^2_{n-1}) \,.$$

Finalement, la simulation de la loi a posteriori du couple (m, σ^2) se réalise de la manière suivante

1. $\sigma^2|y \sim (n-1)s^2_{n-1}/\chi^2_{n-1}$
2. $m|\sigma^2, y \sim \text{N}(\bar{y}, \sigma^2/n)$

En langage `R`, nous pouvons coder cet algorithme pour effectuer 10000 tirages selon la loi a posteriori de la manière suivante. Dans cette simulation, nous considérons que la variable cachée est égale à 0 et que la variance du modèle

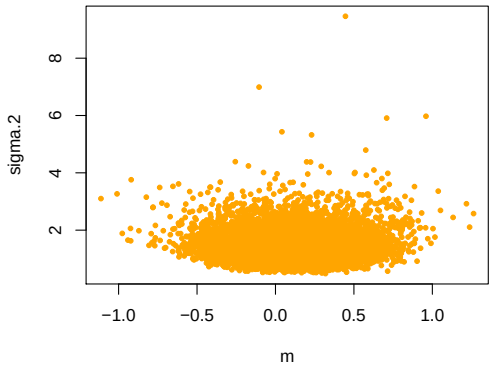


FIGURE 8 – *Echantillonnage de la loi a posteriori* $p(m, \sigma^2|y)$ *pour un échantillon de taille* $n = 20$. *La valeur cachée est égale à 0, et la variance du modèle d'erreur est égale à 1.*

d'erreur est égale à 1. Nous disposons de $n = 20$ répliques indépendantes. Dans ce cas, elles sont tirées au hasard

```
# simulated data

n = 20; y = rnorm(n)

# Posterior distribution sampling

sigma.2 = (n-1)*var(y)/rchisq(10000, n-1)

m = rnorm(10000, mean(y), sd = sqrt(sigma.2/n))
```

La figure 8 décrit un échantillonnage de la loi a posteriori. Toutes les grandeurs d'intérêt peuvent être calculées avec précision à partir de cet échantillon de grande taille. Par exemple, nous estimons que le coefficient de corrélation du couple (m, σ^2) est environ 0.001. Ce résultat confirme la théorie de RA Fisher indiquant l'indépendance de la moyenne et de la variance empirique dans un échantillon gaussien.

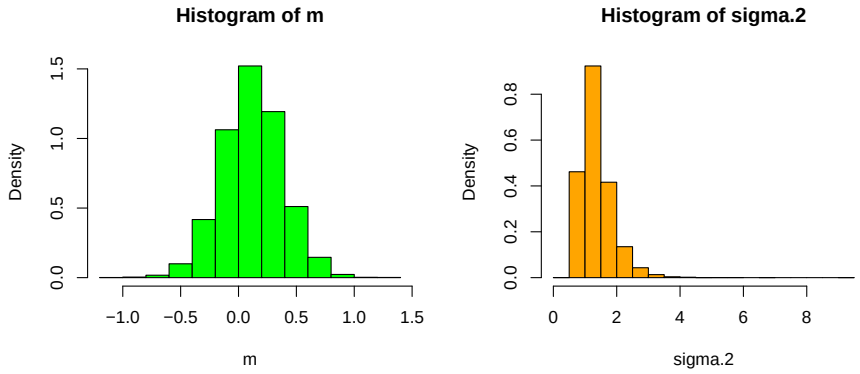


FIGURE 9 – *Histogrammes des lois a posteriori $p(m|y)$ et $p(\sigma^2|y)$ pour un échantillon de taille $n = 20$. La valeur cachée est égale à 0 et la variance du modèle d'erreur est égale à 1.*

Il est possible en particulier de calculer les histogrammes des lois marginales de m et σ^2 sachant y (figure 9). Remarquons que nous avons une expression théorique pour la loi $p(\sigma^2|y)$. Bien que cela ne soit pas nécessaire ici, il est intéressant de noter que nous pouvons aussi déterminer la loi de $p(m|y)$. En effet une série de calcul élémentaire conduit au résultat suivant

$$\sqrt{n}\frac{m-\bar{y}}{s_{n-1}}|y \sim t_{n-1}$$

où t_{n-1} est la loi de Student à $n-1$ degrés de liberté. Cette loi apparait notamment dans la théorie statistique pour tester une valeur moyenne lorsque la variance est inconnue (t-test).

4.4 Pertinence et qualité du modèle d'erreur

Lorsque l'on construit un modèle, nous faisons des hypothèses fortes concernant la nature des données. Par exemple, la loi générative du modèle d'erreur suppose que les données sont gaussiennes. En réalité, nous ne savons rien sur

la nature des données, et bien que la loi gaussienne est très répandue dans la nature, il est nécessaire de critiquer notre modèle pour l'affiner et obtenir de meilleures prédictions.

Considérons un exemple simple où nos observations ne sont pas issues du modèle que nous considérons. Sans le savoir, nous faisons donc une (inévitable) erreur de modélisation. Dans notre exemple, les $n = 20$ données que nous observons sont en fait issues de la loi de Cauchy. La loi de Cauchy à la particularité de n'admettre ni espérance ni écart-type. En langage `R`, nous pouvons créer des observations de la manière suivante

```
y = rcauchy(n)
```

Pour tester notre modèle d'erreur, nous pouvons choisir une statistique de test. Il y a un grand choix, et nous considérons une possibilité parmi tant d'autres. Nous testons notre modèle en utilisant le coefficient de dissymétrie de l'échantillon. Pour nos n observations, ce coefficient est égal à -1.23 (attention, ce résultat n'est pas reproductible !). L'algorithme de Monte Carlo suivant, programmé en langage `R` et disponible parmi les scripts associés au cours, simule la distribution prédictive du coefficient de dissymétrie (skewness) pour nos 20 observations.

```
post.pred = NULL
for (i in 1:1000) {
ind = sample(10000, 1)
post.pred[i] = skewness(rnorm(20, m[ind], sqrt(sigma.2[ind]))) }
hist(post.pred)
skewness(y)
```

Nous observons que la valeur observée du coefficient de dissymétrie (-1.23) se trouve dans les quantiles les plus faibles de la loi prédictive de ce coefficient. Un test rejette donc le modèle d'erreur gaussien pour notre jeu de données. Le rejet du modèle gaussien remet donc en question la validité des estimations faites pour la variable cachée et les calculs d'incertitude liés à cette variable. Nous devons, pour ces données, nous efforcer de rechercher un nouveau modèle d'erreur. Notons que pour exécuter le script précédent, on devra écrire la fonction `skewness` permettant de calculer le coefficient de dissymétrie car celle ci n'est pas disponible dans la bibliothèque de fonctions de base. D'autres choix de statistique sont bien entendu possibles et conseillés !

4.5 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

4.6 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. Modèle d'erreur gaussien. On considère un paramètre caché, θ de loi $p(\theta) = \mathrm{N}(0, \sigma_0^2 = 1/\beta_0)$, et une observation y générée par la loi $p(y|\theta) == \mathrm{N}(\theta, \sigma^2 = 1/\beta)$.

1. Montrer que l'on peut représenter le couple (θ, y) de la manière suivante

$$\begin{aligned} \theta &= x_0 \\ y &= x_0 + x_1 \end{aligned}$$

   où x_0 est de loi $\mathrm{N}(0, \sigma_0^2)$, et x_1 est indépendante de x_0, de loi $\mathrm{N}(0, \sigma^2)$.

2. Déduire de la question précédente, l'espérance et la matrice de covariance du couple (θ, y).

Exercice 2. On considère la loi générative suivante

$$p(y_1, \ldots, y_n|\theta) \propto \prod_{i=1}^{n} \exp\left(-\frac{1}{2\sigma^2}(y_i - \theta)^2\right), \quad y_i \in \mathbb{R}$$

où la moyenne, θ, est un paramètre inconnu. On suppose que la variance, σ^2, est connue, et que la loi a priori de θ est dégénérée (non-informative)

$$p(\theta) \propto 1\,.$$

On pose $\bar{y} = \sum_{i=1}^{n} y_i/n$.

1. Montrer que $p(\theta|y) = p(\theta|\bar{y})$.

2. Retrouver que la loi a posteriori du paramètre θ est donnée par

$$\theta|y \sim \mathrm{N}(\bar{y}, \sigma^2/n)\,.$$

Exercice 3. On considère le modèle gaussien de loi générative

$$p(y_1, \ldots, y_n|\theta) \propto \frac{1}{\theta^{n/2}} \exp\left(-\frac{n}{2\theta} s_n^2\right)$$

où

$$s_n^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - m)^2\,.$$

Le paramètre θ représente la variance, qui est inconnue. On suppose que la moyenne m est connue et que la loi de θ est dégénérée (uniforme sur une échelle log)

$$p(\theta) \propto \frac{1}{\theta}\,.$$

On pose $\bar{y} = \sum_{i=1}^{n} y_i/n$.

1. On considère une variable aléatoire de loi χ^2_ν, dont la densité est donnée par

   $$p(x) \propto x^{\nu/2-1}e^{-x/2}\,, \quad x > 0, \nu > 0\,.$$

   Soit $s^2 > 0$. Montrer que la loi de la variable $\nu s^2/\chi^2_\nu$ admet pour densité

   $$p(x) \propto \frac{1}{x^{\nu/2+1}} e^{-\nu s^2/2x}\,, \quad x > 0\,.$$

   Cette loi est notée Inv-$\chi^2(\nu, s^2)$.

2. Retrouver que la loi a posteriori du paramètre θ est donnée par

   $$\theta = \sigma^2|y \sim \mathrm{Inv}{-}\chi^2(n, s_n^2)$$

Exercice 4. Dans cet exercice, m et σ^2 sont des paramètres inconnus et aléatoires. On considère donc $\theta = (m, \sigma^2)$, de loi a priori non-informative

$$p(m, \sigma^2) \propto \frac{1}{\sigma^2}\,.$$

1. Retrouver l'expression de la loi a posteriori

$$p(m, \sigma^2|y) \propto \frac{1}{(\sigma^2)^{n/2+1}} \exp\left(-\frac{1}{2\sigma^2}((n-1)s_{n-1}^2 + n(\bar{y}-m)^2)\right)$$

où s_{n-1}^2 est *l'estimateur sans biais* de la variance :

$$s_{n-1}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2\,.$$

2. Démontrer que la loi marginale a posteriori de σ^2 est

$$\sigma^2|y \sim \text{Inv}\chi^2(n-1, s_{n-1}^2)\,.$$

3. Justifier la validité de l'algorithme de simulation de la loi a posteriori suivant `sigma.2 = (n-1)*var(y)/rchisq(10000, n-1)`

   `m = rnorm(10000, mean(y), sd = sqrt(sigma.2/n))`

4. Programmer cet algorithme en langage `R`, et l'utiliser pour estimer les paramètres de moyenne et variance des longueurs des sépales des iris de Fisher (`data(iris)`). Comparer les résultats obtenus aux résultats théoriques.

5. Que penser de la validité de l'algorithme de simulation de la loi a posteriori répétant les opérations suivantes depuis une condition initiale fixée

   `sigma.2 = sum((y -m)^ 2)/rchisq(1, n)`

   `m = rnorm(1, mean(y), sd = sqrt(sigma.2/n))`

6. Programmer cet algorithme en langage `R`, et comparer les résultats obtenus aux résultats théoriques pour le même jeu de données que précédemment.

5 Séance 5 : Méthodes de Monte-Carlo par chaînes de Markov

5.1 Introduction et quelques conventions

Dans cette séance, nous introduisons une classe de méthodes de Monte-Carlo très utile pour la simulation d'une loi de probabilité dont on ne connaît que le terme général et dont on ne sait pas calculer la constante de normalisation. Pour l'apprentissage probabiliste, cela concerne notamment la loi a posteriori du paramètre θ,

$$p(\theta|y) \propto p(y|\theta)p(\theta)\,,$$

dont la constante de normalisation est donnée par une intégrale qu'il est généralement impossible de calculer exactement

$$p(y) = \int p(y|\theta)p(\theta)d\theta\,.$$

Le principe des méthodes de Monte-Carlo que nous présentons ici repose sur l'utilisation d'un **processus itératif** permettant de générer de manière **approchée** des tirages d'une loi cible quelconque. Nous décrivons dans la section précédente la nature des itérations considérées pour la simulation de la loi cible. En théorie des probabilités, ces processus aléatoires rentrent dans le cadre très étudié des **chaînes de Markov**.

5.2 Notations et définitions

Nous appelons **chaîne de Markov** une suite de variables aléatoires, (θ^t), $t = 0, 1, 2, \ldots$, à valeurs dans un espace d'état E discret ou continu, vérifiant la propriété suivante

$$p(\theta^{t+1}|\theta^t, \cdots, \theta^0) = p(\theta^{t+1}|\theta^t)\,.$$

La chaîne est homogène dans le temps si, de plus,

$$p(\theta^{t+1}|\theta^t) = p(\theta^1|\theta^0)\,.$$

Pour une chaîne de Markov, la loi conditionnelle de la variable θ^{t+1} sachant le passé du processus au temps t ne dépend du passé qu'à travers la dernière observation du processus. Cette propriété est souvent appelée la *propriété de Markov*. La chaîne est homogène si le mécanisme de transition, aussi appelé **noyau de transition**, permettant de générer θ^{t+1} à partir de θ^t ne dépend pas de t.

Dans la suite, nous ne considérons que des chaînes de Markov homogènes dans le temps. Nous notons $k(\theta^0, \theta^1)$ le noyau (*kernel*) de transition

$$k(\theta^0, \theta^1) = p(\theta^1|\theta^0)\,.$$

Lorsque θ^0 est fixé, un noyau de transition définit donc une loi de probabilité pour la variable θ^1.

Lorsque E est un ensemble d'états finis, nous pouvons numéroter les états. La valeur $k(i, j)$ représente la probabilité que le processus effectue une transition vers l'état j lorsqu'il se trouve dans l'état i. Nous avons donc, pour tout i, j dans E,

$$0 \leq k(i, j) \leq 1 \quad \text{et} \quad \sum_{j \in E} k(i, j) = 1.$$

La matrice $K = (k(i, j))$ dont le terme général est donné par le noyau de transition s'appelle la **matrice de transition**. Nous étudions les propriétés des matrices de transition et illustrons quelques exemples simples correspondant à d'autres contextes d'applications que l'apprentissage dans une séance ultérieure du cours de MPA.

Lorsque E est un ensemble continu, $k(\theta^0, \theta^1)$ peut définir, pour chaque valeur connue θ^0, une loi de probabilité quelconque sur E . Dans ce cas, un noyau de transition est une fonction positive dont l'intégrale vaut 1

$$k(\theta^0, \theta^1) \geq 0 \quad \text{et} \quad \int_E k(\theta^0, \theta^1) d\theta^1 = 1.$$

La propriété de Markov permet de calculer les lois fini-dimensionnelles de la suite (θ^t). En effet, nous avons

$$p(\theta^0, \ldots, \theta^t) = p(\theta^t | \theta^0, \ldots, \theta^{t-1}) p(\theta^0, \ldots, \theta^{t-1}) \,.$$

Par définition, nous avons

$$p(\theta^t | \theta^0, \ldots, \theta^{t-1}) = k(\theta^{t-1}, \theta^t) \,.$$

Par une récurrence élémentaire, nous obtenons que

$$p(\theta^0, \ldots, \theta^t) = p(\theta^0) k(\theta^0, \theta^1) \cdots k(\theta^{t-1}, \theta^t) \,.$$

Par convention, nous notons $\pi(\theta^0) = p(\theta^0)$, la loi initiale de la chaîne de Markov. Le résultat précédent nous indique que le comportement probabiliste d'une chaîne de Markov homogène est entièrement spécifié par la donnée de la loi initiale $\pi(\theta^0)$ et du noyau de transition $k(\theta^0, \theta^1)$.

Nous disons qu'une loi $\pi(\theta)$ est stationnaire ou invariante si

$$p(\theta^1) = \pi(\theta^1) \,.$$

La loi $p(\theta^1)$ peut être calculée comme la loi marginale du couple (θ^0, θ^1)

$$p(\theta^1) = \int p(\theta^0, \theta^1) d\theta^0 = \int \pi(\theta^0) k(\theta^0, \theta^1) d\theta^0 \,.$$

Nous voyons que cette définition implique que la loi $\pi(\theta)$ est stationnaire si elle solution d'une équation de point fixe

$$\pi(\theta^1) = \int \pi(\theta^0) k(\theta^0, \theta^1) d\theta^0 \,.$$

Il existe un cas particulièrement intéressant ou l'on peut vérifier la stationnarité d'une loi sans trop d'effort. C'est le cas de la réversibilité. On dit qu'une chaîne de Markov est réversible si le sens du temps n'a pas d'influence sur la loi de la chaîne. En d'autres termes, une **chaîne de Markov est réversible** si

$$p(\theta^0, \theta^1) = p(\theta^1, \theta^0)$$

Cela se traduit par la condition

$$\pi(\theta^0) k(\theta^0, \theta^1) = \pi(\theta^1) k(\theta^1, \theta^0) \,.$$

Nous pouvons vérifier que la loi initiale d'une chaîne réversible sera invariante pour le noyau de transition. En effet, nous avons

$$\int \pi(\theta^0) k(\theta^0, \theta^1) d\theta^0 = \pi(\theta^1) \int k(\theta^1, \theta^0) d\theta^0 = \pi(\theta^1) \,.$$

Supposons que la chaîne de Markov (θ^t) admette une unique loi stationnaire $\pi^*(\theta)$. Nous admettrons que, sous des conditions faibles, la loi conditionnelle de la variable θ^t sachant la valeur initiale θ^0 converge vers $\pi^*(\theta^t)$

$$p(\theta^t | \theta^0) \to \pi^*(\theta^0)$$

L'idée de la démonstration de ce résultat résulte du phénomène de point fixe déjà évoqué un peu plus haut. En effet, nous avons

$$p(\theta^{t+1} | \theta^0) = \int p(\theta^t | \theta^0) k(\theta^t, \theta^{t+1}) d\theta^t$$

En passant à la limite dans chacun des termes de l'égalité ci-dessus, nous obtenons que

$$\pi^*(\theta^{t+1}) = \int \pi^*(\theta^t) k(\theta^t, \theta^{t+1}) d\theta^t$$

et par homogénéité des transitions

$$\pi^*(\theta^1) = \int \pi^*(\theta^0) k(\theta^0, \theta^1) d\theta^0 \,.$$

Nous voyons donc, comme dans une suite récurrente classique, que si la limite existe et est unique alors il s'agit nécessairement de la loi invariante de la chaîne.

5.3 Algorithme de Metropolis

D'après les résultats de la section précédente, une chaîne de Markov, décrite par une loi initiale et un noyau de transition, peut être vue comme **un algorithme itératif permettant d'échantillonner de manière approchée la loi invariante de la chaîne**, supposée unique.

L'algorithme de Metropolis ou *algorithme de Metropolis-Hastings* propose une solution au **problème inverse**. A partir d'une loi cible $\pi(\theta)$ donnée, l'algorithme de Metropolis définit un noyau de transition, $k(\theta^0, \theta^1)$, de sorte que la chaîne associée à ce noyau de transition converge vers $\pi(\theta)$. La propriété remarquable de cet algorithme est de permettre d'**échantillonner la loi** $\pi(\theta)$ **lorsque la constante de normalisation de cette loi est incalculable**.

L'algorithme de Metropolis s'appuie sur un algorithme de rejet appliqué itérativement à la variable θ^t. Pour définir un algorithme de Metropolis, une première étape consiste à choisir un noyau de transition instrumental, $q(\theta^t, \theta^*)$ (proposal kernel). Le noyau instrumental décrit la manière d'explorer l'espace d'état à partir de la valeur courante θ^t. Son rôle/objectif est de proposer des va-

leurs qui seront évaluées ensuite comme étant acceptable ou non par l'algorithme de rejet.

L'algorithme de Metropolis se définit à partir des étapes suivantes.

1. Initialiser θ^0, $t = 0$
2. A l'itération t, tirer θ^* selon la loi instrumentale $q(\theta^t, \theta^*)$
3. Calculer
$$r = \frac{\pi(\theta^*)q(\theta^*, \theta^t)}{\pi(\theta^t)q(\theta^t, \theta^*)}$$
4. Avec la probabilité $\min(1, r)$, faire $\theta^{t+1} \leftarrow \theta^*$, sinon $\theta^{t+1} \leftarrow \theta^t$.
5. Incrémenter t et retourner en 2.

L'algorithme décrit ci-dessus peut être modélisé par une chaîne de Markov de noyau de transition

$$k(\theta^0, \theta^1) = q(\theta^0, \theta^1) \min\left(1, \frac{\pi(\theta^1)q(\theta^1, \theta^0)}{\pi(\theta^0)q(\theta^0, \theta^1)}\right)$$

L'expression de $k(\theta^0, \theta^1)$ signifie que l'on utilise la loi instrumentale pour proposer une valeur θ^1, puis cette valeur est acceptée avec la probabilité $\min(1, r)$.

On vérifie que la chaîne initialisée par la loi cible $\pi(\theta)$ possède la propriété de réversibilité (Exercice)

$$\pi(\theta^0)k(\theta^0, \theta^1) = \pi(\theta^1)k(\theta^1, \theta^0)\,.$$

En conséquence, si le noyau instrumental est choisi de manière raisonnable et de sorte à visiter l'ensemble des valeurs à échantillonner, nous avons la garantie que l'algorithme de Metropolis simule des variables aléatoires dont la loi est très proche de la loi cible.

En pratique, le choix de la loi instrumentale détermine la qualité de l'approximation faite par l'algorithme de Metropolis. Il sera prudent de ne conserver que les valeurs générées après une phase de *chauffe* de l'algorithme, difficile à calibrer, car dépendante du choix du noyau $q(\theta^0, \theta^1)$. Cette phase, nécessaire pour "atteindre" la loi stationnaire est appelée *burn-in period* en anglais.

5.4 Apprentissage d'une fréquence : modèle bêta-binomial

Dans le contexte de l'apprentissage probabiliste, l'algorithme de Metropolis est très utile pour simuler la loi a posteriori $p(\theta|y)$ et évaluer les grandeurs d'intérêt (histogrammes, quantiles, prédiction) par la méthode de Monte-Carlo.

Afin d'illustrer le comportement de l'algorithme de Metropolis, nous l'appliquons à un cas où la loi a posteriori peut être déterminée explicitement : l'apprentissage d'une fréquence, vu lors de la séance 2. Nous considérons que le paramètre qui nous intéresse est une proportion θ, correspondant à la fréquence d'émission d'une source binaire. Nous supposons que la loi *a priori* est uniforme sur $(0, 1)$, c'est à dire, $p(\theta) = 1$. En supposant que nous observons y signaux égaux à 1 et $n - y$ signaux égaux à 0, la loi générative est la loi binomiale de paramètres n et θ

$$p(y|\theta) = \text{binom}(n, \theta)(y) \propto \theta^y (1-\theta)^{n-y} \,.$$

La loi a posteriori est une loi de la famille bêta

$$p(\theta|y) = \text{beta}(y+1, n+1-y)(\theta) \,,$$

dont la constante de normalisation est connue exactement, mais n'admet pas de forme explicite (fonction bêta).

De nombreux choix se présentent pour la loi instrumentale. Nous verrons en travaux dirigés qu'un choix raisonnable consiste à utiliser la loi a priori $q(\theta^0, \theta^1) = p(\theta^1)$. Dans cette section, nous étudions un autre choix, appelé la *marche aléatoire*. L'idée de la marche aléatoire est de proposer de nouvelles valeurs autour de la dernière valeur acceptée par l'algorithme de Metropolis. On entend par là que de telles valeurs auraient plus de chances d'être acceptées que des valeurs proposées au hasard. L'objectif de ce choix est donc d'augmenter le nombre d'acceptations, et donc la vitesse de convergence de l'algorithme de Metropolis.

Pour modéliser une marche aléatoire dans l'intervalle $(0, 1)$, nous pouvons utiliser la loi bêta. Par exemple, choisissons $b = 100$ et

$$q(\theta^0, \theta^1) = \text{beta}(b\theta^0/(1-\theta^0), b)(\theta^1)$$

En moyenne, l'espérance (conditionnelle) de θ^1 sachant θ^0 est égale à

$$\text{E}[\theta^1|\theta^0] = \frac{\theta^0/(1-\theta^0)}{\theta^0/(1-\theta^0)+1} = \theta^0$$

La valeur proposée est donc centrée autour de la valeur courante. L'écart-type peut être calculé exactement, nous retiendrons qu'il est faible. Cette loi instrumentale implante donc une recherche locale.

En langage `R`, le ratio r intervenant dans l'algorithme de Metropolis peut se calculer de la manière suivante

```
theta.1<-rbeta(1, b*theta.0/(1-theta.0), b)
ratio1<-(theta.1/theta.0)^  y*((1 - theta.1)/(1 - theta.0))^ (n-y)
ratio2<-dbeta(theta.0,b*theta.1/(1-theta.1),b)/dbeta(theta.1,
b*theta.0/(1-theta.0),b)
```

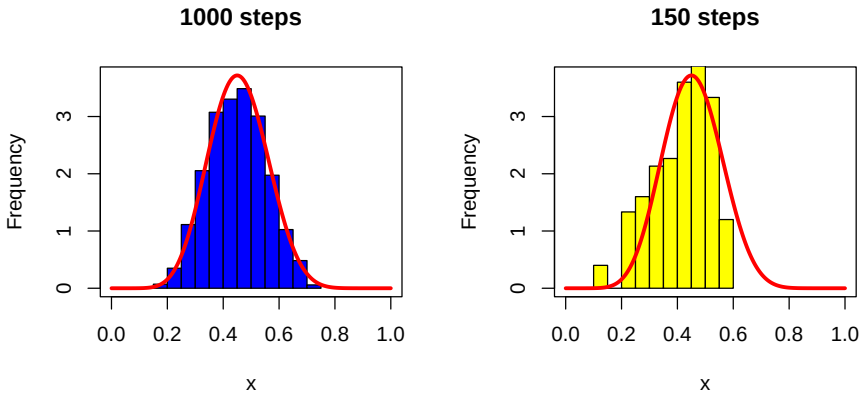


FIGURE 10 – *Histogrammes de la loi a posteriori obtenus par un algorithme de Metropolis pour la fréquence d'émission du source binaire* ($n = 20, y = 9$). *L'algorithme est initialisé par une valeur proche de* 0. *A droite, nous conservons 1000 tirages après élimination des 100 premiers. A droite, nous conservons les 150 premiers tirages.*

```
ratio <- ratio1*ratio2
```

Nous programmerons l'algorithme de Metropolis en séance de travaux dirigés. Par exemple, pour $n = 20$ et $y = 9$, la loi a posteriori est la loi bêta$(10, 12)$, dont la densité est représentée en rouge dans la figure 10. L'algorithme est initialisé par une valeur proche de 0. Lorsque nous conservons 1000 tirages après une période de chauffe de 100 itérations, nous obtenons une approximation de la loi bêta$(10, 12)$ de très bonne qualité. Si toutefois nous utilisons uniquement les 150 premiers tirages, nous observons que l'algorithme n'a pas convergé, et que l'approximation est de très mauvaise qualité. Cet exemple montre que la durée de la période de chauffe est importante pour la bonne convergence de l'algorithme de Metropolis.

5.5 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

5.6 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. Dans cette séance, on considère le modèle de vraisemblance binomiale et de loi a priori choisie dans la famille de lois bêta (modèle bêta-binomial). On note θ le paramètre correspondant à une probabilité inconnue d'un événement donné, et on suppose que l'on observe $y = 9$ réalisations de cet événement lors de la répétition de $n = 20$ épreuves indépendantes. L'objectif de cette séance est de programmer plusieurs algorithmes de Monte Carlo par chaînes de Markov et d'évaluer la convergence de ces algorithmes.

1. On suppose que la loi a priori est uniforme sur $(0, 1)$. Donner un argument de théorie de l'information pour justifier que ce choix exprime le maximum d'incertitude a priori sur θ. Rappeler les expressions mathématiques de la vraisemblance, de la loi Beta et de la loi a posterior du paramètre θ, aux constantes près.
2. Rappeler l'algorithme d'estimation de Metropolis-Hasting (MH) pour un noyau instrumental $q(\theta, \theta^*)$.
3. On choisit ce noyau de transition égal à la loi a priori (les tirages ne dépendent pas de la valeur en cours θ). Vérifier que le rapport de MH est égal à :
$$r = \left(\frac{\theta^*}{\theta^t}\right)^y \times \left(\frac{1-\theta^*}{1-\theta^t}\right)^{(n-y)}$$
4. Ecrire un script en `R` implantant la simulation de la loi a posteriori

```
theta.1 <- runif(1)
ratio <- (theta.1/theta.0)^ y *((1 - theta.1)/(1 - theta.0))^ (n-y)
if (runif(1) < ratio) theta.0 <- theta.1
```

5. Afficher un histogramme de la loi a posteriori, calculer numériquement un intervalle de crédibilité à 95% pour θ et afficher un histogramme de la loi predictive sachant y.

6. Testons maintenant l'algorithme implanté dans le script `R` suivant

```
#Metropolis-Hastings

#kernel = random walk

nit = 1000

theta.0 <- 0.1

theta.post <- theta.0

log.likelihood <- NULL


for (i in 1:nit) {

# proposal

b <- 100

theta.1 <- rbeta(1,b*theta.0/(1-theta.0), b)


# MH ratio

ratio1 <- (theta.1/theta.0)^(y)* ((1 - theta.1)/(1 - theta.0))^(n-y)

ratio2 <-  dbeta(theta.0, b*theta.1/(1-theta.1), b)

/dbeta(theta.1, b*theta.0/(1-theta.0), b)

ratio <- ratio1*ratio2


if (runif(1) < ratio) {theta.0 <- theta.1}

theta.post <- c(theta.post,theta.0)

log.likelihood <- c(log.likelihood, y*log(theta.0) + (n-y)*log(1 - theta.0))

}
```

```
# show results
u <- seq(0, 1, length= 200)
par(mfrow=c(1,3))
plot(u, dbeta(u, 10, 12), type="n", ylab="Frequency",
 main="Posterior distribution (burnin = 200)", xlab = "x")
hist(theta.post[200:1000], col = "blue" , prob = T, add = T)
lines(u, dbeta(u, 10, 12), col = 2, lwd = 3)
plot(u, dbeta(u, 10, 12), type="n", ylab="Frequency",
main="150 sweeps", xlab = "x")
hist(theta.post[1:150], col = "yellow" , prob = T, add = T)
lines(u, dbeta(u, 10, 12), col = 2, lwd = 3)
plot(log.likelihood[1:100], cex = .5, pch = 19, type = "l")
```

7. Quel est le choix de du noyau de transition instrumental $q(\theta^1|\theta^0)$ dans l'algorithme précédent ?

8. Donner l'espérance et la variance de θ^1 sachant la valeur en cours θ^0 (les valeurs pour la loi beta(a, b) sont $a/(a+b)$ et $ab/(a+b)^2(a+b+1)$).

9. Commenter les résultats affichés par le script `R`. L'algorithme est-il sensible au choix de la valeur initiale (la modifier) ? L'algorithme est-il sensible au choix de la loi instrumentale ? Comment vos observations se traduisent-elles pour le choix de la période de chauffe (*burn-in*) ?

6 Séance 6 : Modèles multi-paramétriques – Echantillonnage de Gibbs

6.1 Introduction

Dans la séance précédente, nous avons introduit un algorithme de Monte Carlo par chaîne de Markov – algorithme de Metropolis – permettant d'échantillonner la loi d'un paramètre. L'algorithme de Metropolis est très utile lorsque les constantes de normalisation des lois sont inconnues. Il possède toutefois un inconvénient majeur. Il demande de choisir une loi instrumentale, et si le choix de cette loi n'est pas pertinent, le taux d'acceptation de l'algorithme de rejet peut être faible.

Dans cette section, nous présentons un nouvel algorithme de simulation par chaîne de Markov, s'adressant plus particulièrement à la situation d'un paramètre composite (ou multi-dimensionnel), $\theta = (\theta_1, \theta_2)$, de loi cible $\pi(\theta)$. Cet algorithme est appelé échantillonneur de Gibbs. Il repose sur le calcul des lois conditionnelles $\pi(\theta_1|\theta_2)$ et $\pi(\theta_2|\theta_1)$, utilisées à tour de rôle comme lois instrumentales. L'échantillonneur de Gibbs n'effectue pas de rejet.

6.2 Echantillonnage de Gibbs

Nous souhaitons simuler un paramètre composite $\theta = (\theta_1, \theta_2)$ de loi cible $\pi(\theta_1, \theta_2)$. Appliqué à l'apprentissage probabiliste, la loi cible $\pi(\theta)$ que nous considérons est, le plus souvent, la loi a posteriori, $p(\theta|y)$. Rappelons que pour simuler un paramètre composite $\theta = (\theta_1, \theta_2)$, un algorithme élémentaire comporte les deux étapes suivantes

1. Simuler θ_1 selon la loi marginale $\pi(\theta_1)$
2. Etant donné θ_1, simuler θ_2 selon la loi conditionnelle $\pi(\theta_2|\theta_1)$.

Afin d'implanter cet algorithme, il est nécessaire de connaître la loi marginale et de savoir la simuler. Cette loi est définie par

$$\pi(\theta_1) = \int \pi(\theta_1, \theta_2) d\theta_2 \, .$$

Nous voyons qu'intervient le calcul d'une intégrale, peut être impossible à mener à terme.

L'algorithme d'échantillonnage de Gibbs (ou *Gibbs Sampler*), est un algorithme itératif mettant à jour le paramètre $\theta^t = (\theta_1^t, \theta_2^t)$ en répétant le cycle suivant :

GS1. Etant donné θ_2^t, simuler θ_1^{t+1} à partir de la loi conditionnelle $\pi(\theta_1|\theta_2^t)$.

GS2. Etant donné θ_1^{t+1}, simuler θ_2^{t+1} à partir de la loi conditionnelle $\pi(\theta_2|\theta_1^{t+1})$.

Dans cet algorithme, la définition d'un cycle peut être récursive. Si θ_1 est lui-même un paramètre composite, alors nous pouvons mettre à jour ce paramètre en effectuant un sous-cycle du cycle principal. De même, nous pouvons généraliser l'algorithme à des paramètres composites de la forme $\theta = (\theta_1, \ldots, \theta_J)$, en considérant des cycles de longueur $J \geq 2$.

Convergence de l'échantillonneur de Gibbs. L'échantillonneur de Gibbs peut être décrit par une chaîne de Markov dont le noyau de transition est donné par

$$k(\theta^0, \theta^1) = \pi(\theta_2^1|\theta_1^0)\pi(\theta_1^1|\theta_2^1)$$

En effet, le produit des deux lois conditionnelles traduit le fait que nous simulons des variables de manière cyclique à partir de la composante courante du paramètre θ. Notons que seule la valeur initiale de la première composante, θ_1^0, intervient dans la définition du noyau de transition.

L'algorithme se justifie par le fait que la loi cible, $\pi(\theta_1, \theta_2)$, est stationnaire (invariante) pour la chaîne de Markov de noyau de transition $k(\theta^0, \theta^1)$. Cela signifie que la condition suivante est réalisée

$$\pi(\theta_1^1, \theta_2^1) = \int\int \pi(\theta_1^0, \theta_2^0) k(\theta^0, \theta^1) d\theta_1^0 d\theta_2^0 \,.$$

Malgré l'aspect laborieux des notations (simplifiées au maximum), cela n'est pas difficile à vérifier. En effet, l'intégrale

$$\int\int \pi(\theta_1^0, \theta_2^0) \pi(\theta_2^1 | \theta_1^0) \pi(\theta_1^1 | \theta_2^1) d\theta_1^0 d\theta_2^0$$

est égale à

$$\int\int \pi(\theta_1^0, \theta_2^0) \frac{\pi(\theta_1^0, \theta_2^1)}{\pi(\theta_1^0)} \frac{\pi(\theta_1^1, \theta_2^1)}{\pi(\theta_2^1)} d\theta_1^0 d\theta_2^0 \,.$$

Dans cette expression, il suffit de décaler les dénominateurs vers la gauche pour obtenir le résultat suivant

$$\left(\int\int \frac{\pi(\theta_1^0, \theta_2^0)}{\pi(\theta_1^0)} \frac{\pi(\theta_1^0, \theta_2^1)}{\pi(\theta_2^1)} d\theta_1^0 d\theta_2^0 \right) \pi(\theta_1^1, \theta_2^1) \,.$$

Puisque nous avons

$$\int \pi(\theta_2^0 | \theta_1^0) \int \pi(\theta_1^0 | \theta_2^1) d\theta_1^0 d\theta_2^0 = 1 \,.$$

L'équation de stationnarité est prouvée.

Remarquons finalement que, pour justifier complètement la validité de l'algorithme, il est nécessaire que celui-ci converge vers la loi cible. L'invariance de la loi cible ne garantit pas nécessairement la convergence, et il est possible d'exhiber des cas pathologiques où l'échantillonneur de Gibbs ne converge pas. Nous retiendrons que ces cas sont rares, et qu'en pratique nous pouvons les détecter en simulant l'algorithme.

6.3 Modèles gaussiens

Reprenons le modèle d'erreur gaussien vu dans la séance précédente. Dans ce modèle, nous cherchons la loi a posteriori du paramètre composite $\theta = (m, \sigma^2)$ étant donné un vecteur de n observations, y. Rappelons qu'un l'algorithme de simulation exact de cette loi est donné par les deux étapes suivantes :

```
sigma.2 = (n-1)*var(y)/rchisq(10000, n-1)
m = rnorm(10000, mean(y), sd = sqrt(sigma.2/n))
```

Pour ce paramètre, nous avons aussi calculé la loi conditionnelle de la variance σ^2 sachant la moyenne m et y. Nous avons obtenu

$$\sigma^2 | m, y \sim n s_n^2 / \chi_n^2$$

où s_n^2 est l'estimateur sans biais de la variance lorsque la moyenne est connue. Pour échantillonner la loi a posteriori dans le modèle d'erreur gaussien, il suffit donc d'itérer le cycle suivant

GS1. $\sigma^2 | m, y \sim n s_n^2 / \chi_n^2$

GS2. $m | \sigma^2, y \sim \mathrm{N}(\bar{y}, \sigma^2/n)$

où $\bar{y}$ correspond à la moyenne empirique. En langage `R`, la boucle principale de l'échantillonneur de Gibbs correspond aux commandes suivantes :

```
sigma.2 = sum((y -m)^ 2)/rchisq(1, n)
m = rnorm(1, mean(y), sd = sqrt(sigma.2/n))
```

Après 10100 cycles et après avoir écarté les 100 premières simulations, nous voyons, dans la figure 11 que le résultat est très similaire à celui obtenu par l'algorithme exact. Dans ce cas, l'algorithme de Gibbs possède de très bonnes propriétés de convergence liées à la très faible dépendance entre m et σ^2.

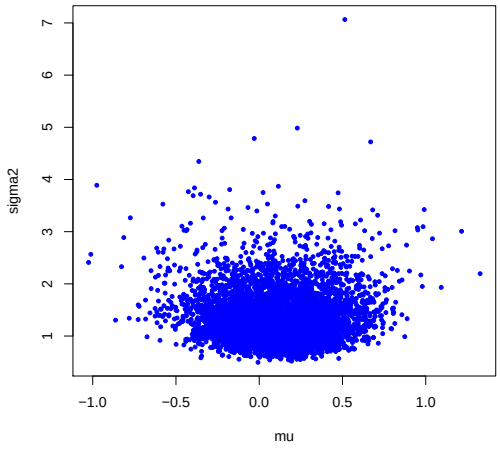


FIGURE 11 – *Simulation de l'échantillonneur de Gibbs pour le couple* (m, σ^2) *pour* $n = 20$ *observations de la loi* N*(0,1).*

6.4 Reconstruction d'image

Nous traiterons l'exemple de la reconstruction d'image lors de l'exercice 4 des travaux dirigés. Dans ce cas, le paramètre à apprendre est une image binaire, θ, composée de $K \times L$ pixels à valeurs -1 ou $+1$. La variable observée est à nouveau une image binaire, y, que nous supposons générée à partir d'un modèle d'erreur connu.

L'originalité de la modélisation repose sur une forme de loi a priori très particulière, prenant en compte les dépendances locales qui peuvent exister au sein d'une image binaire. En effet, deux pixels proches de l'image observée ont plus de chance de présenter des niveaux d'intensité identiques que deux pixels éloignés. Cette propriété de mémoire locale est très similaire à la propriété de Markov vue pour un processus itératif.

La reconstruction d'image consiste à échantillonner θ selon la loi a posteriori, $\mathrm{p}(\theta|y)$, dont les valeurs les plus probables représentent une version débruitée de l'observation y.

6.5 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

6.6 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. Echantillonnage de Gibbs pour la loi uniforme. Décrire l'algorithme d'échantillonnage de Gibbs pour simuler la loi uniforme sur le disque unité, puis sur un sous-ensemble borné de $\mathbb{R}^2$.

Exercice 2. Iris de Fisher. Pour les longueurs des sépales des iris de Fisher, donner un histogramme des lois a posteriori ainsi que des intervalles de crédibilité à 95% pour les paramètres m et σ^2. Le modèle d'erreur gaussien vous semble-t-il approprié pour l'analyse de ces données ?

Exercice 3. Echantillonnage de Gibbs pour la loi normale. On considère un couple (θ_1, θ_2) de loi gaussienne de moyenne nulle et de matrice de covariance

$$\Lambda = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

On pose $\rho = .99$.

1. Ecrire un algorithme de simulation exact pour le couple (θ_1, θ_2). Le programmer en langage `R`. Que peut-on dire des résultats.
2. Rappeler le principe de l'algorithme d'échantillonnage de Gibbs.
3. Montrer que le script `R` donné ci-dessous implante cet algorithme et commenter sa sortie.

```
rho <- .99
nit <- 2000
theta1 <- -3; theta2 <- 3
theta1.post <- theta1
theta2.post <- theta2

for (i in 1:nit){
theta1 <- rnorm(1, rho*theta2, sd = sqrt(1 - rho^2))
theta2 <- rnorm(1, rho*theta1, sd = sqrt(1 - rho^2))
theta1.post <- c(theta1.post, theta1)
theta2.post <- c(theta2.post, theta2)
}

# show results
qqnorm(theta2.post[100:200])
abline(0,1)
plot(theta1.post[-1], theta2.post[-1])
cor(theta1.post, theta2.post)
hist(theta2.post)
```

4. La vitesse de convergence et les propriétés de mélange de la chaîne de Markov sont-elles dépendantes de ρ ?

Exercice 4 : Reconstruction d'image (d'après Geman et Geman) On considère un signal binaire de longueur n codé sous la forme d'un vecteur (θ_i),

où $\theta_i \in \{-1, +1\}$. Ce signal représente par exemple une image dont les pixels ont deux niveaux de couleur, noir et blanc. Dans ce cas, le codage en vecteur correspond à une matrice binaire ($n = K \times L$).

On observe une réalisation bruitée (y_i) du signal telle que $y_i \in \{-1, +1\}$ pour tout i. On suppose que le modèle probabiliste du bruit est décrit par

$$p(y|\theta) = \prod_{i=1}^{n} p(y_i|\theta_i)$$

où

$$p(y_i = +1|\theta_i = -1) = p(y_i = -1|\theta_i = +1) = (1 + \exp(2\alpha))^{-1}$$

et α est un paramètre réel supposé connu. On suppose que la loi de θ est décrite par un modèle d'Ising

$$p(\theta) \propto \exp(\beta \sum_{i \sim j} \theta_i \theta_j), \quad \beta > 0$$

où la notation $i \sim j$ signifie que i et j sont voisins en un sens à préciser. Pour une image par exemple, on peut préalablement définir un voisinage pour chaque pixel. Dans ce cas, cela signifie que l'on somme sur les paires de pixels voisins dans l'image. On suppose aussi que i n'est pas voisin de lui-même.

1. Démontrer la propriété suivante, pour tout $i = 1, \ldots, n$,

$$p(\theta_i|\theta_{-i}) = p(\theta_i|\theta_j,\, j \sim i) \propto \frac{\exp(\beta\theta_i \sum_{j\sim i} \theta_j)}{\exp(-\beta \sum_{j\sim i} \theta_j) + \exp(\beta \sum_{j\sim i} \theta_j)}$$

   Pour quel type d'image le modèle vous paraît-il adapté ?

2. Montrer que la loi a posteriori s'écrit de la manière suivante

$$p(\theta|y) \propto \prod_{i=1}^{n} \exp(\theta_i(\frac{\beta}{2} \sum_{j\sim i} \theta_j + \alpha y_i))$$

3. Ecrire un algorithme d'échantillonnage de Gibbs pour simuler la loi $p(\theta|y)$.

4. Télécharger l'image

   `http://membres-timc.imag.fr/Olivier.Francois/imagetd7.txt`

   et vérifier que l'algorithme suivant, utilisant des conditions de bord 'cycliques' et les 4 pixels les plus proches implante l'algorithme d'échantillonnage de Gibbs (les commandes `moins, plus` définissant les voisinages sont à écrire en langage `R`)

```
image(y<-as.matrix(read.table("imagetd7.txt")))
alpha=log(2);beta=10
theta=y
for (iter in 1:100){
for(i in 1:100)
for(j in 1:100)
{
sij =  theta[moins(i),moins(j)]+ theta[moins(i),plus(j)]+
        theta[plus(i),moins(j)]+ theta[plus(i),plus(j)]
pij = exp(beta*sij+alpha*y[i,j])
p   = exp(beta*sij+alpha*y[i,j])+exp(-beta*sij-alpha*y[i,j])
theta[i,j]=sample(c(-1,+1),1, prob=c(1-pij/p, pij/p)) }
image(theta)}
```

   Vérifier que $\alpha = \log 2$ correspond à environ 20% de pixels bruités. Faire varier la constante β vers des valeurs plus grandes que 10 et plus petites que 1. Qu'observez-vous ?

7 Séance 7 : Modèles hiérarchiques – Modèles de mélange.

7.1 Introduction et Définitions

Dans cette séance, nous rentrons au cœur des techniques d'apprentissage probabiliste en considérant des paramètres composites de grande dimension, dans des modèles structurés hiérarchiquement. Dans ces modèles, les paramètres sont inter-dépendants, et nous modélisons la structure de dépendance des paramètres de sorte à créer des cascades de dépendance. Les hiérarchies considérées sont particulièrement propices à l'utilisation de la formule de Bayes et aux algorithmes d'échantillonnage vus dans les séances précédentes.

En particulier nous abordons un domaine très important de l'apprentissage probabiliste : la **classification non-supervisée**. Ce domaine consiste à grouper des observations en classes séparées, sans avoir préalablement défini les classes et sans disposer d'exemples précis de classement. L'objectif de la classification probabiliste est de quantifier l'incertitude liée à chaque classement établi de cette manière. Pour cela, il est important de pouvoir calculer la probabilité conditionnelle qu'une observation donnée provienne de la classe k connaissant toutes les observations.

Considérons à nouveau en le complexifiant un peu le modèle d'erreur gaussien. Nous supposons désormais que nous observons des données issues en réalité de 2 sources distinctes et non d'une unique source. Par cohérence avec la figure 12, nous appelons les sources source *jaune* et source *bleue*. La valeur cachée émise par la source jaune est égale à $m_1 = -1$ et la valeur cachée émise par la source bleue est égale à $m_2 = 2$. Les écarts types des erreurs produites par

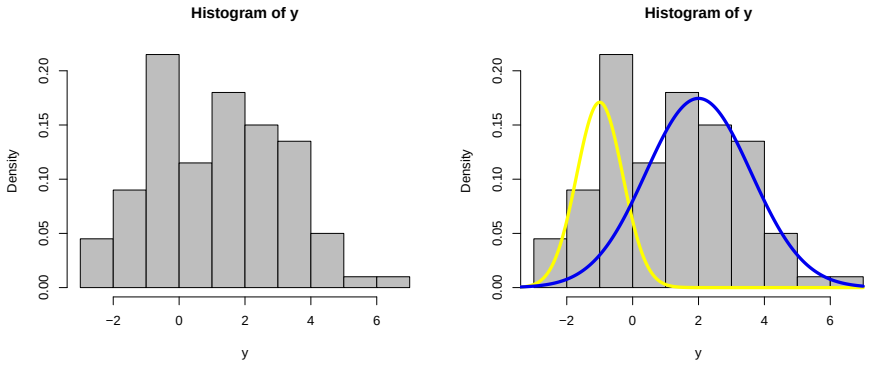


FIGURE 12 – *Observations d'un mélange de deux sources gaussiennes de paramètres respectifs* $m_1 = -1$ *et* $m_2 = 2$. *La partie droite de la figure représente les composantes cachées du mélange.*

les 2 sources sont aussi propres à chacune d'entre elles. De plus les fréquences d'émission des sources ne sont pas égales. La source jaune émet 30% du temps alors que la source bleue émet 70% du temps. Les sources sont aussi appelées *composantes* de mélange.

Bien entendu, les informations précédentes ne sont pas accessibles à un observateur. Ce dernier ne dispose que de données issues de l'observation des deux sources, représentées dans l'histogramme gris de la figure 12 (200 observations). En observant les données, nous ignorons la nature des sources. Le nombre de sources est aussi inconnu. Nous souhaitons donc apprendre les paramètres des sources, tout en nous posant la question suivante. Doit on expliquer les observations à l'aide d'un modèle ayant 1, 2 ou 3 sources ?

En résumé, la classification probabiliste cherche à répondre aux questions suivantes :

– Combien de groupes distincts peut on détecter au sein des observations ?

– Que valent les moyennes intra-groupes et les paramètres de dispersion des groupes détectés ?

– Pour une observation donnée, quelle est la probabilité de provenir de l'un ou l'autre de chaque groupe détecté ?

La classification probabiliste (ou *clustering*) répond à ce cahier des charges en considérant la classe associé à l'observation i, comme une variable non-observée ou latente, notée z_i. Aux paramètres z_i s'ajoutent les paramètres statistiques du modèle, incluant moyennes et variances des classes, résumés dans la variable composite θ. L'apprentissage probabiliste consiste à calculer ou simuler la loi a posteriori

$$p(\theta, z|y) \propto p(y|\theta, z)p(\theta)p(z)$$

et d'en déduire les lois marginales correspondant aux classes latentes $p(z_i|y)$. Notons enfin qu'un modèle à K classes possède $2K + n$ variables cachées, c'est à dire plus que le nombre des observations.

7.2 Exemple simple de modèle hiérarchique

La construction de modèle hiérarchique est particulièrement adaptée au développement d'algorithmes d'apprentissage. Un modèle hiérarchique décrit une loi de probabilité pour les données y et un paramètre composite que nous notons (θ, ψ) afin de mettre en évidence la structure hiérarchique. Le modèle est structuré de la manière suivante. Sans changement, les données y sont échantillonnées selon la loi générative

$$y|\theta, \psi \sim p(y|\theta)$$

Notons l'absence de dépendance de ψ dans cette définition. La loi de θ est définie conditionnellement à ψ

$$\theta|\psi \sim p(\theta|\psi)\,.$$

Le paramètre ψ est appelé *hyper-prior*. Il est échantillonné selon la loi

$$\psi \sim p(\psi)\,.$$

Ainsi, la loi a priori du paramètre composite (θ, ψ) est définie par $p(\theta, \psi) = p(\theta|\psi)p(\psi)$, et la loi a posteriori est donnée par la formule de chainage suivante

$$p(\theta, \psi|y) \propto p(y|\theta)p(\theta|\psi)p(\psi).$$

Les modèles hiérarchiques se prêtent bien à l'implantation de l'algorithme d'échantillonnage de Gibbs. En effet, nous avons les simplifications suivantes

$$p(\theta|\psi, y) \propto p(y|\theta)p(\theta|\psi)\,,$$

et

$$p(\psi|\theta, y) \propto p(\theta|\psi)p(\psi)\,.$$

Partant d'une condition initiale (θ^0, ψ^0), l'algorithme de Gibbs peut s'appuyer sur les cycles suivants :

Pour $t = 1, \ldots, T$

GS1 : Simuler θ^t selon $p(\theta|\psi^{t-1}, y)$,

GS2 : Simuler ψ^t selon $p(\psi|\theta^t, y)$

FinPour

7.3 Modèles de mélanges

Dans cette section, nous définissons les modèles de mélanges pour la classification non-supervisée, puis nous décrivons l'algorithme pour l'apprentissage

des variables cachées de ces modèles.

Mélange de lois univariées. Considérons une unique observation, $y \in \mathbb{R}$. Une loi de mélange est définie comme la combinaison convexe de lois élémentaires, appelées **composantes du mélange**. Les **proportions** ou **coefficients** de mélange, $(p_k)_{k=1,\ldots,K}$, sont des nombres strictement positifs tels que

$$\sum_{k=1}^{K} p_k = 1\,.$$

Les composantes du mélange peuvent être vues comme des lois génératives de paramètres θ_k, $k = 1, \ldots, K$. Nous définissons une loi de mélange de la manière suivante

$$p(y|\theta) = \sum_{k=1}^{K} p_k p(y|\theta_k)$$

Les paramètres des mélanges de gaussiennes correspondent à la moyenne et la variance des composantes, $\theta_k = (m_k, \sigma_k^2)$, et nous avons dans ce cas

$$p(y|\theta_k) = \mathrm{N}(m_k, \sigma_k^2)(y)\,.$$

Les mélanges de lois gaussiennes sont facile à simuler. L'exemple représenté dans la figure 12 correspond à un mélange de lois gaussiennes, dans lequel nous avons $K = 2$ et les proportions de mélange sont égales à .3 et .7. La simulation de la figure 12 a été effectuée en échantillonnant exactement 30% des données depuis la source jaune. Pour un échantillon de taille totale $n = 200$, nous avons utilisé les commandes du langage `R` suivantes

```
z.truth = c(rep(1,60), rep(2, 140))
m.truth = c(-1, 2)
s.truth = c(.7, 1.6)
```

```
for (i in 1:200){
y[i]=rnorm(1,m.truth[z.truth[i]],sd=s.truth[z.truth[i]]) }
```

Pour un échantillon de taille n, $y = (y_1, \ldots, y_n)$, une approche naïve de l'apprentissage des paramètres consiste à écrire directement la loi générative du modèle

$$p(y|\theta) = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} p_k p(y_i|\theta_k) \right)$$

Cette approche n'est pas inexacte, mais elle comporte de nombreux inconvénients algorithmiques. Pour l'apprentissage probabiliste dans les modèles de mélange, nous utilisons une technique d'**augmentation** des données, qui consiste à introduire une variable de classe cachée, $z_i \in \{1, \ldots, K\}$, associée à chacune des observations y_i. Cela nous amène à considérer, pour tout $i = 1, \ldots, n$, la loi générative suivante

$$p(y_i|\theta, z_i) = p(y_i|\theta_{z_i}).$$

Nous voyons que cette représentation nous conduit bien à un modèle de mélange :

$$p(y_i|\theta) = \sum_{k=1}^{K} p(y_i|\theta_k) p(z_i = k)\,.$$

Pour construire un modèle de mélange de lois gaussiennes, nous définissons donc un vecteur de variables cachées $z = (z_1, \ldots, z_n)$. Notre modèle (y, z, θ) s'écrit de la manière suivante

$$y_i|\theta, z_i = k \sim \mathrm{N}(m_k, \sigma_k^2)(y_i)$$

$$p(z) \propto 1$$

$$p(\theta) \propto \tfrac{1}{\sigma_1^2} \tfrac{1}{\sigma_2^2} \cdots \tfrac{1}{\sigma_K^2}\,.$$

Dans cette expression, la loi uniforme $p(z)$ signifie simplement que $p(z_i = k) = 1/K$ pour i et pour tout k.

7.4 Echantillonnage de Gibbs pour le modèle de mélange de gaussiennes

Afin d'écrire l'algorithme de Gibbs pour simuler la loi a posteriori du modèle de mélange de lois gaussiennes, nous devons considérer le paramètre composite suivant

$$(z, \theta) = (z_1, \ldots, z_n, m_1, \ldots, m_K, \sigma_1^2, \ldots, \sigma_K^2)$$

Ce paramètre comporte exactement $n + 2K$ dimensions. La loi a posteriori est proportionnelle à

$$p(z, \theta|y) \propto p(y|\theta, z)p(z)p(\theta)\,.$$

Les calculs peuvent se détailler de la manière suivante

$$\begin{aligned} p(z, \theta|y) &\propto \left(\prod_{i=1}^{n} p(y_i|\theta, z_i)p(z_i)\right) p(\theta) \\ &\propto \prod_{i=1}^{n} p(y_i|\theta, z_i) \prod_{k=1}^{K} \frac{1}{\sigma_k^2} \\ &\propto \prod_{i=1}^{n} \frac{1}{\sqrt{\sigma_{z_i}^2}} \exp(-\frac{1}{2\sigma_{z_i}^2}(y_i - m_{z_i})^2) \prod_{k=1}^{K} \frac{1}{\sigma_k^2} \end{aligned}$$

L'échantillonnage de Gibbs s'appuie sur les itérations d'un cycle que nous décomposons de la manière suivante

GS1. Pour $i = 1, \ldots, n$, mettre à jour $z_i \sim p(z_i|y_i, \theta)$ (mise à jour simultanée)

GS2. Pour $k = 1, \ldots, K$, mettre à jour $m_k \sim p(m_k|z, m_{-k}, \sigma^2)$ où le vecteur m_{-k} est privé de la composante m_k

$$m_{-k} = (\ldots, m_{k-1}, m_{k+1}, \ldots)$$

GS3. Pour $k = 1, \ldots, K$, mettre à jour $\sigma_k^2 \sim p(\sigma_k^2 | z, m, \sigma_{-k}^2)$ où le vecteur σ_{-k}^2 est privé de la composante σ_k^2

$$\sigma_{-k}^2 = (\ldots, \sigma_{k-1}^2, \sigma_{k+1}^2, \ldots) .$$

Dans la suite de cette section, nous détaillons chacune des 3 étapes du cycle de l'échantillonnage de Gibbs.

Gibbs sampler GS1. Soit $z_i \in \{1, \ldots, K\}$. D'après la formule de Bayes, nous avons

$$p(z_i | y, \theta) = \frac{p(y | z_i, \theta) p(z_i, \theta)}{p(y, \theta)}$$

Or, nous avons

$$p(y | z_i, \theta) = p(y_i | z_i, \theta) \prod_{j \neq i} p(y_j | \theta)$$

et

$$p(z_i, \theta) \propto p(\theta) .$$

En reportant ces équations dans la formule de Bayes nous obtenons

$$p(z_i | y_i, \theta) = \frac{p(y_i | z_i, \theta)}{\sum_k p(y_i | z_i = k, \theta)} .$$

En utilisant l'expression des densités gaussiennes, nous obtenons finalement

$$p(z_i | y_i, \theta) \propto \frac{\frac{1}{\sqrt{\sigma_{z_i}^2}} \exp(-\frac{1}{2\sigma_{z_i}^2}(y_i - m_{z_i})^2)}{\sum_{k=1}^K \frac{1}{\sqrt{\sigma_k^2}} \exp(-\frac{1}{2\sigma_k^2}(y_i - m_k)^2)} .$$

Le code R implantant cette phase du cycle utilise la commande `sample`

```
for (i in 1:n) {
p = exp(-(y[i] - m)^ 2/2/sigma2 )/sqrt(sigma2)
z[i] = sample(1:K, 1, prob = p ) }
```

Gibbs sampler GS2. Les étapes GS2 et GS3 découlent de considérations vues dans les séances précédentes. Nous détaillerons les résultats en travaux dirigés. La phase GS2 concerne la simulation de la loi conditionnelle des moyennes intra-groupes sachant les variances (et les données). Soit n_k l'effectif courant de la classe k,

$$n_k = \#\{i : z_i = k\},$$

supposé non-nul. Nous avons alors

$$p(m_k|y, z, m_{-k}, \sigma^2) = \mathrm{N}(\bar{y}_k, \frac{\sigma_k^2}{n_k})$$

où $\bar{y}_k$ est la moyenne empirique calculée pour la classe k :

$$\bar{y}_k = \frac{1}{n_k} \sum_{i:z_i=k} y_i.$$

Le code `R` implantant cette phase du cycle utilise la commande `rnorm`

```
nk[k]=sum(z==k)
m[k]=rnorm(1,mean(y[z==k]),sd=sqrt(sigma2[k]/nk[k]))
```

Gibbs sampler GS3. La phase GS3 concerne la simulation de la loi conditionnelle des variances intra-groupes sachant les moyennes (et les données). Nous avons

$$p(\sigma_k^2|y, z, m, \sigma_{-k}^2) = \mathrm{Inv}{-}\chi^2(n_k, s_k^2)$$

où s_k^2 représente la variance empirique au sein de la classe k lorsque la moyenne est connue

$$s_k^2 = \frac{1}{n_k} \sum_{i:z_i=k} (y_i - m_k)^2.$$

Le code `R` implantant cette phase du cycle utilise la commande `rchisq`.

```
sigma2[k] = sum((y[z==k]-m[k])^ 2)/rchisq(1, nk[k])
```

Finalement nous pouvons écrire une fonction en langage R permettant de simuler l'échantillonneur de Gibbs pour les mélanges de lois gaussiennes.

```
mcmc.mix=function(y,niter=1000,m.o= c(0,1),sigma2.o=c(1,1)) {
n=length(y);K=length(m.o);m=m.o;s2=sigma2.o
z=NULL;p=NULL;nk = NULL
p.mcmc=NULL;m.mcmc=NULL;sigma2.mcmc=NULL;z.mcmc=NULL
for(nit in 1:niter) {
for(i in 1:n) {
p=exp(-(y[i]-m)^ 2/2/s2)/sqrt(s2)
z[i]=sample(1:K,1,prob = p) }
z.mcmc=rbind(z.mcmc,z)
for(k in 1:K) {
nk[k]=sum(z==k)
m[k]=rnorm(1,mean(y[z==k]),sd=sqrt(s2[k]/nk[k]))
s2[k]=sum((y[z==k]-m[k])^ 2)/rchisq(1,nk[k]) }
m.mcmc=rbind(m.mcmc,m)
sigma2.mcmc=rbind(sigma2.mcmc,s2) }
return(list(z=z.mcmc,m=m.mcmc,sigma2=sigma2.mcmc)) }
```

7.5 Quelques résultats

Nous pouvons utiliser l'algorithme de Gibbs pour analyser les données présentées dans la figure 12. Nous effectuons 200 cycles de l'algorithme, précédés de 100 cycles de chauffe (burn-in). Il est pratique de représenter les lois a posteriori des

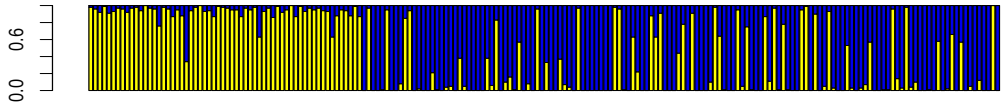


FIGURE 13 – *Diagramme en barres pour les lois a posteriori $p(z_i|y)$ correspondant à l'histogramme simulé au début de la section. Résultats pour $K = 2$ classes.*

classes individuelles ($p(z_i|y)$) en utilisant un diagramme en barres colorées. Dans ce diagramme, chaque segment vertical correspond à l'une des classes cachées, et la longueur du segment jaune ou bleu correspond à la probabilité a posteriori que le classement soit jaune ou bleu. En langage `R`, nous pouvons obtenir les probabilités a posteriori de chaque classe en comptant l'occurrence cette classe le long de la chaîne de Markov associée à l'échantillonnage de Gibbs. La représentation graphique est obtenue grâce à la commande `barplot`. Les résultats sont reportés dans les figures 13 et 14.

Pour $K = 2$, concernant la classe jaune, le taux de faux positifs est de l'ordre de 20% et le taux de fausse découverte est de l'ordre de 40% (résultats obtenus en considérant la valeur la plus probable a posteriori). Pour $K = 3$, nous observons que l'algorithme conserve la classe jaune. Il partitionne la seconde classe en attribuant des probabilités sensiblement égales aux classes bleue et rouge. Le résultat semble indiquer que $K = 2$ est un meilleur choix que $K = 3$.

Nous utilisons la loi prédictive de deux statistiques pour tester le modèle à trois classes : les coefficients d'aplatissement (kurtosis) et d'asymétrie (skew-

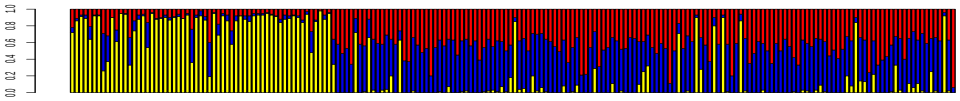


FIGURE 14 – *Diagramme en barres pour les lois a posteriori $p(z_i|y)$ correspondant à l'histogramme simulé au début de la section. Résultats pour $K = 3$ classes.*

ness). Dans le script suivant, l'objet `obj` est obtenu après application de la fonction `mcmc.mix` aux données de mélange.

```
post.stat = NULL

for (i in 1:100){

zz = obj$z[i+100,];mm = obj$m[i+100,];ss2 = obj$sigma2[i+100,]

yy = c(rnorm(sum(zz==1),mm[1],sqrt(ss2[1])),

rnorm(sum(zz==2),mm[2],sqrt(ss2[2])),

rnorm(sum(zz==3),mm[3],sqrt(ss2[3])))

post.stat=c(post.stat,kurtosis(yy))}
```

La figure 15 décrit un histogramme de la loi prédictive pour la statistique d'aplatissement (kurtosis) dans le modèle avec $K = 3$ classes. Nous voyons que cette statistique ne permet pas de rejeter fermement un modèle à trois classes pour nos données simulées. Cela montre que le choix de modèle peut être très difficile à réaliser. Dans ce cas, nous savons qu'il y a deux sources, mais un modèles à trois sources peut donner des résultats que l'on ne peut pas critiquer.

7.6 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts principaux de la séance.

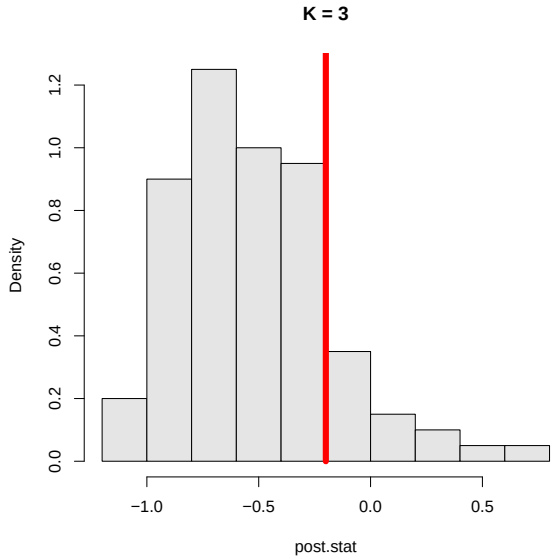


FIGURE 15 – *Loi prédictive pour la statistique d'aplatissement (kurtosis, $K = 3$).*

7.7 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1 : Mélange de 2 gaussiennes. On considère un échantillon $y = y_1, \ldots, y_n$ un échantillon constitué de n données réelles. On suppose les données indépendantes et provenant de deux sources gaussiennes de moyennes et de variances inconnues $\theta_1 = (m_1, \sigma_1^2)$ et $\theta_2 = (m_2, \sigma_2^2)$. On définit un vecteur $z = (z_1, \ldots, z_n)$, $z_i \in \{1, 2\}$, (correspondant aux classes non-observées) et un

modèle de la manière suivante

$$y_i|z_i = k, \theta \sim \mathrm{N}(y_i|\theta_k), \quad k = 1, 2$$

$$p(z) \propto 1$$

$$p(\theta) \propto 1/\sigma_1^2\sigma_2^2$$

1. Montrer que la loi a posteriori vérifie

$$p(z, \theta|y) \propto \prod_{k=1,2} \prod_{i \in I_k} \frac{1}{(\sigma_k^2)^{(3/2)}} \exp\left(-\frac{1}{2\sigma_k^2}(y_i - m_k)^2\right)$$

   où I_k est l'ensemble des indices tels que $z_i = k$.

2. Pour tout $k = 1, 2$, montrer que la loi conditionnelle de z_i sachant (y, θ) vérifie

$$p(z_i = k|y, \theta) = \frac{\exp\left(-\frac{1}{2\sigma_k^2}(y_i - m_k)^2\right)/\sqrt{\sigma_k^2}}{\sum_{\ell=1}^{2} \exp(-\frac{1}{2\sigma_\ell^2}(y_i - m_\ell)^2)/\sqrt{\sigma_\ell^2}}$$

3. En vous appuyant sur les résultats d'une séance de Tds précédente, justifier que, pour tout $k = 1, 2$,

$$m_k|y, z, \theta_{-m_k} \sim \mathrm{N}(m_k|\bar{y}_k, \sigma_k^2/n_k)$$

   où n_k est le nombre d'éléments dans I_k (supposé non nul) et $\bar{y}_k = \sum_{i \in I_k} y_i/n_k$.

4. De même justifier que, pour tout $k = 1, 2$,

$$\sigma_k^2|y, z, \theta_{-\sigma_k^2} \sim \mathrm{Inv}{-}\chi^2(\sigma_k^2|n_k, s_k^2)$$

   où $s_k^2 = \sum_{i \in I_k}(y_i - m_k)^2/n_k$ est l'estimateur de la variance dans la classe k.

5. Ecrire un algorithme d'échantillonnage de Gibbs pour la loi a posteriori de ce modèle en langage `R`. On pourra utiliser la fonction `mcmc.mix` du script téléchargeable à l'adresse suivante

`http://membres-timc.imag.fr/Olivier.Francois/script_R.R`

6. À quoi correspondent les lois marginales a posteriori, $p(z_i|y)$, $i = 1, \ldots, n$, pour l'analyse des donnés y ? Comment peut-on les visualiser sous `R` ? (aide : `barplot`)

7. Charger le jeu de données des iris de Fisher et extraire les longueurs des sépales. Simuler la loi a posteriori du modèle précédent et comparer les classifications obtenues pour chaque plante à l'appartenance de chaque plante à l'espèce *I. setosa*. On pourra utiliser la commande `barplot` pour visualiser les lois marginales a posteriori, $p(z_i|y)$, pour tout $i = 1, \ldots, n$.

Exercice 2. Modèle hiérarchique, effet aléatoire et *shrinkage*. Dans un modèle hiérarchique, les observations sont modélisées conditionnellement à certains paramètres, et ces paramètres sont eux-mêmes décrits par des lois de probabilités dépendant d'autres paramètres, appelés *hyper-paramètres*.

L'objectif de cet exercice est d'estimer le risque pour un groupe de rats de décéder d'une tumeur cancéreuse à l'issue d'un traitement thérapeutique spécifique (les rats ont été sélectionnés pour développer la tumeur). Nous disposons pour cela de 71 groupes de rats, dont le nombre d'individus varie entre 14 et 52. Pour tenir compte de différences dans les échantillons considérés, nous souhaitons utiliser un modèle hiérarchique pour le risque. On note θ_i , i variant de 1 à 71, la probabilité qu'un rat du groupe i développe une tumeur, et on considère le vecteur $\theta = (\theta_i)$. Dans un groupe de rats de taille n_i, on modélise le nombre y_i de rats avec une tumeur par une loi binomiale de paramètres n_i et θ_i.

A priori, on modélise θ_i par une loi Beta de paramètres α et β. Les réels positifs α et β sont les hyperparamètres du modèle. La figure ci dessous représente la structure du modèle hiérarchique que l'on souhaite ajuster aux données.

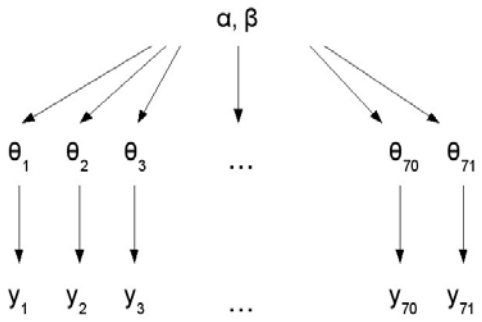


Figure 1: *Structure du modèle hiérarchique*

Il reste maintenant à spécifier les paramètres α et β. Afin d'effectuer un traitement bayésien du modèle, c'est-à-dire apprendre la loi a posteriori jointe de tous les paramètres du modèle, il est important de spécifier une loi a priori pour α et β. Comme nous n'avons pas d'information sur l'ensemble des paramètres, il est naturel de chercher à décrire une loi a priori non informative. Une loi uniforme ne convient pas, car elle conduit à une loi a posteriori non intégrable. Pour des raisons que l'on admettra ici, il est raisonnable de choisir une loi a priori qu'on voudra uniforme pour le couple

$$(\mu, \nu) = \left(\frac{\alpha}{\alpha + \beta}, \frac{1}{\sqrt{\alpha + \beta}} \right).$$

Afin d'apprendre la loi a posteriori de tous les paramètres du modèle, nous

souhaitons programmer une méthode de Monte-Carlo par chaîne de Markov hybride. Il s'agit de mettre à jour les paramètres du modèle de manière séquentielle, pour un nombre fixé de pas. Voici une description synthétique de la méthode de MCMC à programmer :

Etape 1. Partir de valeurs arbitraires α_0 , β_0 et θ_0 et fixer un nombre de pas pour l'algorithme, N. Pour $t = 1, \ldots, N$, répéter les opérations suivantes :

Etape 2. Mettre à jour les paramètres α_t , β_t à partir de leur valeur au pas $t-1$ à l'aide de l'algorithme de Métropolis-Hasting.

Etape 3. Mettre à jour le vecteur de paramètres θ_t à partir de la valeur précédente θ_{t-1} par échantillonage de Gibbs.

Etape 4. Après N balayages de l'ensemble des paramètres, choisir un nombre $b < N$ et conserver les valeurs de θ_t pour $t > b$ (b pour la période de *burn-in*).

1. Télécharger le fichier de données `rats.asc` à l'adresse suivante

   `http://www.stat.columbia.edu/~gelman/book/data/`

2. Calculer analytiquement le terme général de la densité $p(\alpha, \beta)$ par un changement de variables. Nous pouvons remarquer que cette densité est impropre (la densité n'est pas intégrable).

3. Donner l'expression de la vraisemblance $p(y|\theta)$ et de la loi conditionnelle $p(\theta|\alpha, \beta, y)$.

4. Calculer le rapport de Metropolis permettant la mise à jour du couple d'hyper-paramètres (α, β) (étape 2). Pour cette question, on pourra choisir

librement la loi de transition instrumentale servant à proposer de nouvelles valeurs des paramètres.

5. Rappeler brièvement le principe de l'échantillonnage de Gibbs. Décrire l'algorithme de mise à jour du paramètre θ (étape 3).

6. Programmer l'algorithme MCMC dans le langage `R`. Donner une estimation ponctuelle (moyenne conditionnelle, médiane et mode a posteriori) pour chacun des paramètres θ_i. Calculer les intervalles de crédibilité à 95% de ces paramètres .

7. Comparer ces estimations avec les estimations de maximum de vraisemblance $\hat{\theta}_i = y_i/n_i$. On pourra trier les valeurs y_i/n_i par ordre croissant, et afficher les moyennes a posteriori des lois marginales (les θ_i) en utilisant le même ordre. Commenter le phénomène de régularisation des risques observé dans cette courbe.

8 Séance 8 : Lois gaussiennes

8.1 Introduction

Dans cette séance, nous définissons les lois gaussiennes multivariées à partir des *composantes principales*. Nous établissons la forme générale de la densité de cette famille de lois, puis, dans le cas bi-varié, nous calculons l'espérance et la variance conditionnelle d'une coordonnée sachant l'autre coordonnée. Nous montrons que le calcul de l'espérance conditionnelle est similaire au calcul de la *régression linéaire* et justifions que le carré du coefficient de corrélation – ou coefficient de détermination – représente exactement le pourcentage de variance expliquée par la variable observée. Nous discutons ensuite la simulation de variables gaussiennes multivariées ainsi que l'échantillonnage de Gibbs dans ce contexte.

8.2 Définitions

On dit qu'un vecteur aléatoire $y \in \mathbb{R}^d$ d'espérance nulle est **un vecteur gaussien** s'il existe une matrice de rotation U, vérifiant $UU^T = \mathrm{Id}$, telle que les coordonnées du vecteur $z = Uy$ sont indépendantes et de loi normale : $z_j \sim \mathrm{N}(0, \lambda_j)$ pour tout $j = 1, \ldots, d$.

La matrice de covariance d'un vecteur aléatoire, y, est la matrice C dont le terme général c_{ij} correspond à la covariance du couple (y_i, y_j). Elle peut s'écrire sous la forme suivante

$$C = \mathrm{E}[(y - \mathrm{E}[y])(y - \mathrm{E}[y])^T] \,.$$

Pour une transformation linéaire d'un vecteur aléatoire y de matrice de cova-

riance C, notée $x = Ay$, la matrice de covariance de x, notée $\tilde{C}$, est égale à

$$\tilde{C} = ACA^T \, .$$

En remplaçant y par $U^T z$ dans l'expression précédente, nous obtenons en toute généralité que la matrice covariance de y est égale à

$$C = U^T \Lambda U \, ,$$

où Λ est une matrice diagonale dont le j^{e} terme est égal à λ_j. Nous voyons ainsi que le déterminant de C, égal à celui de Λ, est strictement positif. Les variances λ_j correspondent aux valeurs propres de C.

Un vecteur aléatoire gaussien $y \in \mathbb{R}^d$ d'espérance m est défini comme la somme d'un vecteur gaussien d'espérance nulle et du vecteur m. Dans ce cas, cela revient à prendre $z = U(y - m)$ dans la définition précédente.

Nous pouvons remarquer que toute combinaison linéaire des coordonnées d'un vecteur gaussien suit une loi normale. Plus précisément, soit z_a une combinaison linéaire des coordonnées y_j, telle que a représente les coefficients de cette combinaison

$$z_a = a^T y = \sum_{j=1}^{d} a_j y_j \, .$$

Alors, z_a suit la loi normale de moyenne $m_a = a^T m$ et de variance $\sigma_a^2 = a^T C a$. En effet, nous avons

$$z_a = (Ua)^T z = b^T z = \sum_{j=1}^{d} b_j z_j \, .$$

où $b = Ua$ Le résultat découle alors du fait que les variables $b_j z_j$ sont des variables de loi normale indépendantes entre elles. Nous savons, en effet, que la somme de variables aléatoires indépendantes de loi normale est encore une

variable aléatoire de loi normale (le vérifier à l'aide de l'expression des densités). Cela implique qu'une transformation matricielle linéaire, $x = Ay$, du vecteur gaussien y est un vecteur gaussien à condition que le déterminant de la matrice de covariance ne soit pas nul.

Une combinaison particulière des coordonnées d'un vecteur gaussien consiste à considérer la première variable, y_1, extraite du vecteur y. Dans ce cas, le vecteur a est le vecteur unitaire correspondant à la première coordonnée, $a = (1, 0, \ldots, 0)$. Le résultat précédent nous indique que y_1 est de loi normale. Généralement, **les lois marginales d'un vecteur gaussien sont toujours gaussiennes**.

8.3 Densité de probabilité d'un vecteur gaussien.

Dans ce paragraphe, nous décrivons la forme générale de la loi de probabilité d'un vecteur gaussien de moyenne m et de matrice de covariance C, en supposant que le déterminant de C est strictement positif.

Expression de la densité d'un vecteur gaussien. Soit $y \in \mathbb{R}^d$ un vecteur aléatoire ayant d coordonnées. On dit que la loi de y est une loi gaussienne de moyenne m et de matrice de covariance C si

$$p(y) = \frac{1}{(2\pi)^{d/2}} \frac{1}{(\det C)^{1/2}} \exp(-\frac{1}{2}(y-m)^T C^{-1}(y-m))$$

On note $\Delta^2 = (y-m)^T C^{-1}(y-m)$ la distance de Mahanalobis entre m et y. La densité est notée $\mathrm{N}(y|m, C)$ ou $\mathrm{N}(m, C)(y)$.

Il est important de bien comprendre la forme de cette loi de probabilité. Pour cela, notons que C est une matrice symétrique dont les valeurs propres sont positives. D'après le théorème spectral et la définition donnée au début de

cette séance, nous pouvons considérer les vecteurs propres orthonormés, (u_j), associés aux valeurs propres, (λ_j), de C

$$Cu_j = \lambda_j u_j \, .$$

Ces vecteurs propres s'appellent les **composantes principales** de la matrice de covariance. Nous pouvons écrire les représentations spectrales associées à la matrice de covariance

$$C = \sum_{j=1}^{d} \lambda_j u_j u_j^T \, ,$$

ainsi qu'à son inverse

$$C^{-1} = \sum_{j=1}^{d} \frac{1}{\lambda_j} u_j u_j^T \, .$$

Nous voyons que la distance de Mahanalobis entre m et y s'écrit de la manière suivante

$$\Delta^2 = \sum_{j=1}^{d} \frac{1}{\lambda_j} z_j^T z_j = \sum_{j=1}^{d} \frac{1}{\lambda_j} z_j^2$$

où $z_j = u_j^T (y - m)$. En écriture matricielle, nous venons d'effectuer le changement de variable linéaire suivant

$$z = U(y - m) = \varphi^{-1}(y) \, .$$

La valeur absolue du jacobien de ce changement de variable linéaire est égale au déterminant de U, c'est à dire, égale à 1. En rappelant que le déterminant d'une matrice carrée correspond au produit des valeurs propres de cette matrice, nous obtenons par la formule de changement de variable l'expression de la densité de la variable z

$$p(z) = \frac{1}{(2\pi)^{d/2}} \frac{1}{(\lambda_1 \dots \lambda_d)^{1/2}} \exp(-\sum_{j=1}^{d} \frac{1}{\lambda_j} z_j^2).$$

Cette densité peut se factoriser de la manière suivante

$$p(z) = \prod_{j=1}^{d} \mathrm{N}(z_j|0, \lambda_j)\,.$$

Les variables z_j sont donc indépendantes et de loi normale $\mathrm{N}(0, \lambda_j)$. Nous vérifions ainsi la définition d'un vecteur gaussien, et au passage, que la matrice covariance de ce vecteur est bien égale à C.

Une conséquence importante de la définition d'un vecteur gaussien est que pour un tel vecteur, les **coordonnées, $y_1, \ldots, y_d$, sont indépendantes si et seulement si la matrice covariance du vecteur y est diagonale**.

Exemple. Considérons le vecteur $y = (y_1, y_2)$ de moyenne nulle et de matrice de covariance C égale à

$$C = \frac{1}{2}\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}$$

Le déterminant de C est égal à 2 et nous avons

$$4\Delta^2 = 3y_1^2 + 2y_1y_2 + 3y_2^2\,.$$

Nous pouvons réécrire cette expression sous la forme d'une somme de carrés

$$\Delta^2 = z_1^2 + \frac{z_2^2}{2}$$

où

$$\begin{cases} z_1 &= (y_1 + y_2)/\sqrt{2} \\ z_2 &= (y_2 - y_1)/\sqrt{2} \end{cases}$$

Ainsi, nous venons de mettre en évidence la rotation permettant le changement de base $z = Uy$ (angle $\pi/4$). D'après l'expression de Δ^2, nous pouvons vérifier que z_1 suit la loi N(0,1) et que z_2 suit la loi N(0,2). De plus les deux variables

sont indépendantes. Nous pouvons de plus vérifier que $C = U^T \Lambda U$ où Λ est la matrice diagonale dont les valeurs propres sont égales à $(1, 2)$.

Bien que cette méthode n'est pas la plus aisée, il est enfin possible grâce à la décomposition spectrale que nous venons d'effectuer, d'écrire un algorithme de simulation du couple (y_1, y_2). En langage `R`, nous pouvons effectuer 1000 tirages de cette loi bi-variée de la manière suivante

```
z1 = rnorm(1000) ; z2 = rnorm(1000, sd = sqrt(2))
y1 = (z1 - z2)/sqrt(2) ; y2 = (z1 + z2)/sqrt(2)
```

8.4 Lois conditionnelles d'un vecteur gaussien – Régression linéaire

Dans cette section, nous calculons l'espérance et la variance conditionnelle au sein d'un couple de variables gaussiennes, et nous établissons une connexion avec la régression linéaire. Le terme **régression** est un synonyme du terme **espérance conditionnelle**. Cette grandeur représente la meilleure prédiction d'une variable au sens des moindres carrés sachant une autre variable. Dans cette section, nous établissons deux résultats importants : **dans le cas gaussien, la régression est linéaire, et le coefficient de corrélation représente la fraction de la variance d'une variable expliquée par l'autre variable**. Comme il est simple de le vérifier grâce à l'expression des densités, nous admettons que les **lois conditionnelles d'un vecteur gaussien sont elles mêmes gaussiennes**.

Pour bien comprendre le conditionnement gaussien, considérons un couple (y, θ) où θ est la variable cachée que nous cherchons à prédire et y est une observation bruitée résultant d'un plan d'expérience gaussien. Nous supposons

que le couple (y, θ) est gaussien, d'espérance m et de matrice de covariance C. Afin de calculer l'espérance conditionnelle $\mathrm{E}[\theta|y]$, nous cherchons une combinaison linéaire de θ et y indépendante de θ. Pour cela, considérons les variables suivantes

$$\begin{cases} z_1 &= y \\ z_2 &= y + a\theta \end{cases}$$

où a est un scalaire que choisissons de sorte que

$$\mathrm{Cov}(y, y + a\theta) = 0\,.$$

Nous obtenons

$$a = -\left[\frac{\mathrm{Cov}(y,\theta)}{\mathrm{Var}(y)}\right]^{-1}\,.$$

Nous pouvons donc calculer facilement $\mathrm{E}[z_2|y]$. Par indépendance, cette espérance ne dépend pas de y. Elle est égale à

$$\mathrm{E}[z_2|y] = \mathrm{E}[z_2] = \mathrm{E}[y] + a\mathrm{E}[\theta].$$

D'autre part, nous avons

$$\mathrm{E}[z_2|y] = \mathrm{E}[y + a\theta|y] = y + a\mathrm{E}[\theta|y]\,.$$

En comparant les deux expressions de $\mathrm{E}[z_2|y]$, nous obtenons une formule générale donnant $\mathrm{E}[\theta|y]$

$$\mathrm{E}[\theta|y] = \mathrm{E}[\theta] + \frac{\mathrm{Cov}(\theta, y)}{\mathrm{Var}(y)}(y - \mathrm{E}[y])\,.$$

Cette formule correspond à la formule classique permettant de calculer la **régression linéaire**. On pourrait retrouver cette formule directement par la résolution d'un problème de moindres carrés. Nous nous passerons de ce calcul, car nous avons justifié le résultat lors d'une séance précédente.

Nous pouvons maintenant calculer la variance conditionnelle de θ sachant y. En utilisant les résultats précédents, nous avons

$$\mathrm{Var}(z_2|y) = \mathrm{Var}(y + a\theta)\,.$$

En développant cette expression nous obtenons que

$$\mathrm{Var}(z_2|y) = \left(\frac{\mathrm{Var}(y)\mathrm{Var}(\theta)}{\mathrm{Cov}(y,\theta)^2}\right) - \mathrm{Var}(y) = \rho^{-2} - \mathrm{Var}(y)\,,$$

alors qu'en effectuant le calcul direct, nous avons

$$\mathrm{Var}(z_2|y) = \mathrm{Var}(a\theta|y) = a^2\mathrm{Var}(\theta|y)\,.$$

Nous pouvons comparer les deux expressions de $\mathrm{Var}(z_2|y)$, et cela conduit au résultat suivant

$$\mathrm{Var}(\theta|y) = \mathrm{Var}(\theta) - \frac{\mathrm{Cov}(y,\theta)^2}{\mathrm{Var}(y)}\,.$$

En d'autres termes, **la part de variance de θ expliquée par l'observation y est égale au carré du coefficient de corrélation**

$$\frac{\mathrm{Var}(\theta) - \mathrm{Var}(\theta|y)}{\mathrm{Var}(\theta)} = \rho^2\,.$$

A nouveau, il s'agit d'un résultat classique pour la régression linéaire, justifiant l'utilisation du carré du coefficient de corrélation dans ce contexte.

8.5 Simulation d'un vecteur gaussien

Nous souhaitons simuler un couple gaussien de moyenne nulle et de matrice de covariance C. Les techniques évoquées ici se généralisent pour des vecteurs de dimension d. Pour simplifier, nous nous limitons à décrire le cas $d = 2$.

D'après la définition d'un vecteur gaussien, nous avons la possibilité d'effectuer la simulation de ce vecteur en utilisant les composantes principales de

la matrice de covariance. Une méthode élémentaire pour simuler un vecteur gaussien consiste à calculer les vecteurs propres orthonormés de la matrice de covariance C (matrice U), de simuler des variables z_j de loi normale $\mathrm{N}(0, \lambda_j)$ où λ_j est la j^{e} valeur propre de C, puis enfin de transformer z en $y = U^T z$. Dans cette section, nous présentons deux méthodes différentes. La première méthode, la plus utile, s'appuie sur la **décomposition de Cholesky de la matrice de covariance**. La seconde méthode s'appuie sur l'échantillonnage de Gibbs, donné ici comme illustration pédagogique du cours de MPA.

Décomposition de Cholesky. La matrice C est une matrice symétrique semi-definie positive. D'après Cholesky, il existe une matrice L, triangulaire inférieure telle que

$$C = LL^T \,.$$

La matrice L peut être obtenue grâce à l'algorithme de **factorisation de Cholesky**. L'algorithme de simulation consiste alors à tirer les variables z_j selon la loi $\mathrm{N}(0, 1)$ et de manière indépendante, puis d'effectuer la transformation

$$y = Lz \,.$$

Nous voyons que cette méthode est très similaire à l'algorithme classique de simulation d'un couple de variable aléatoire pour les lois gaussiennes. En effet, en choisissant la transformation

$$\left\{ \begin{array}{ll} z_1 & = y_1 \\ z_2 & = y_1 + a y_2 \end{array} \right.$$

nous avons défini une transformation linéaire inverse triangulaire inférieure

$$\left\{ \begin{array}{ll} y_1 & = z_1 \\ y_2 & = (z_2 - z_1)/a \end{array} \right.$$

Nous pouvons choisir a afin que z_1 et z_2 soient indépendantes et normaliser ces deux variables afin d'obtenir la transformation L. En dimension $d = 2$, on se contente de vérifier que les calculs précédents nous on permis d'obtenir la loi de la variable y_1

$$y_1 \sim \mathrm{N}(0, \sigma_1^2)\,,$$

où σ_1^2 est la variance de y_1, et la loi conditionnelle de y_2 sachant y_1

$$y_2|y_1 \sim \mathrm{N}(\frac{\mathrm{Cov}(y_1, y_2)}{\sigma_1^2} y_1, \sigma_2^2(1-\rho^2)).$$

L'implantation de l'algorithme classique et exact de la simulation d'un couple de variable aléatoire est donc facile à réaliser. A une normalisation près, il est similaire à l'algorithme de factorisation de Cholesky.

Echantillonnage de Gibbs. L'algorithme d'échantillonnage de Gibbs pour un couple de loi gaussienne consiste à alterner la simulation des lois conditionnelles,

$$y_2|y_1 \sim \mathrm{N}(\frac{\mathrm{Cov}(y_1, y_2)}{\sigma_1^2} y_1, \sigma_2^2(1-\rho^2))$$

et

$$y_1|y_2 \sim \mathrm{N}(\frac{\mathrm{Cov}(y_1, y_2)}{\sigma_2^2} y_2, \sigma_1^2(1-\rho^2)).$$

Afin de comparer l'échantillonneur de Gibbs à l'algorithme exact décrit dans le paragraphe précédent, nous souhaitons simuler une loi gaussienne de moyenne $(0, 0)$ et de matrice de covariance

$$C = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}.$$

Dans cette matrice de covariance, le coefficient ρ est le coefficient de corrélation des deux variables simulées, θ_1 et θ_2. L'algorithme de simulation exact de cette loi gaussienne peut s'écrire en langage `R` de la manière suivante

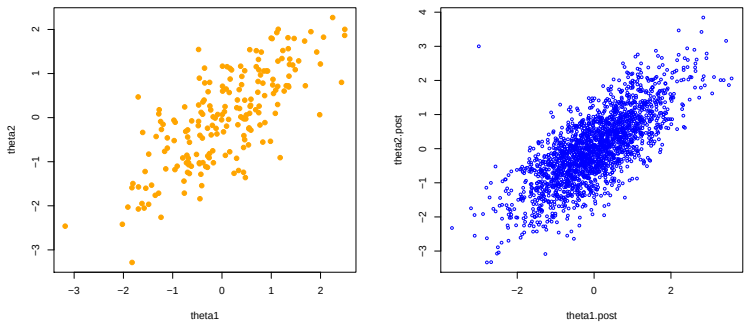


FIGURE 16 – *Simulation d'une loi gaussienne par l'algorithme exact et par échantillonnage de Gibbs.*

```
theta1 = rnorm(1000)

theta2 = rnorm(1000, rho*theta1, sd = sqrt(1 - rho^ 2))
```

Dans ce cas l'algorithme d'échantillonnage de Gibbs consiste à itérer le cycle suivant

```
theta1 = rnorm(1, rho*theta2, sd = sqrt(1 - rho^ 2))

theta2 = rnorm(1, rho*theta1, sd = sqrt(1 - rho^ 2))
```

Pour la valeur $\rho = 70\%$, les résultats des deux algorithmes sont quasiment identiques (figure 16). Nous verrons, en expérimentant cet algorithme lors de la séance de travaux dirigés que la vitesse de convergence de l'algorithme d'échantillonnage de Gibbs dépend de la corrélation entre les composantes du paramètre à simuler. Plus la valeur de ρ est grande, plus il devient difficile d'obtenir un échantillon de la loi cible tels que les paramètres simulés peuvent être considérés indépendants les uns des autres.

8.6 Résumé

Résumer les points à retenir et donner quelques exemples illustrant les concepts

principaux de la séance.

8.7 Exercices

Pour les exercices suivants, il est nécessaire d'avoir un ordinateur sur lequel on aura installé préalablement le logiciel `R`. Ce logiciel libre est disponible sur le site `http://cran.r-project.org/`.

Exercice 1. Résumé On considère un couple gaussien (y_1, y_2) de moyenne $m = (1, 0)$ et de matrice de covariance

$$C = \begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix}.$$

1. Décrire la densité du couple (y_1, y_2) et ses lois marginales.

2. Soit c une constante non nulle. Décrire la loi du vecteur (z_1, z_2) ci-dessous

$$\begin{cases} z_1 = y_1 \\ z_2 = y_1 + cy_2 \end{cases}$$

3. Démontrer que les variables z_1 et z_2 sont indépendantes si et seulement si $c = 3$.

4. Soit $y_1 \in \mathbb{R}$ et $c = 3$. Calculer l'espérance de z_1 et montrer qu'elle est aussi égale à

$$\mathrm{E}[z_1] = y_1 + 3\,\mathrm{E}[y_2|y_1]\,.$$

   En déduire l'espérance conditionnelle de y_2 sachant y_1.

5. Calculer l'espérance de z_1^2 et, par un argument similaire à la question précédente, calculer la variance conditionnelle de y_2 sachant y_1.

6. On note

$$R^2 = \frac{\mathrm{Var}(y_2) - \mathrm{Var}(y_2|y_1)}{\mathrm{Var}(y_2)}$$

   la part de variance de y_2 expliquée par y_1. Montrer que R^2 est égal au carré du coefficient de corrélation et le calculer.

7. Décrire la loi conditionnelle de y_2 sachant y_1.

8. Proposer un algorithme de simulation du couple (y_1, y_2) à partir d'un simulateur `rnorm` de la loi $\mathrm{N}(0, 1)$, et effectuer des simulations pour vérifier les résultats précédents à l'aide du logiciel `R`, et en utilisant la commande `lm`.

Exercice 2. Soit (x, y) un couple de variables aléatoires de densité

$$p(x, y) \propto \exp(-\frac{4x^2 - 2xy + y^2}{6}).$$

1. Montrer que le couple (x, y) est gaussien, de moyenne $m = (0, 0)$ et de matrice de covariance

$$C = \begin{pmatrix} 1 & 1 \\ 1 & 4 \end{pmatrix} .$$

En déduire la valeur de la constante de proportionnalité de la loi $p(x, y)$.

2. Déterminer la loi de x, de y. Quelle est la valeur de $cov(x, y)$? Les variables x et y sont-elles indépendantes ?

3. On pose

$$\begin{cases} z_1 &= x - y \\ z_2 &= x \end{cases}$$

Montrer que le couple (z_1, z_2) est gaussien et caractériser sa loi. Les variables z_1 et z_2 sont-elles indépendantes ?

4. Ecrire un algorithme de simulation pour le couple (x, y). (On prendra soin de démontrer que le couple en sortie de cet algorithme est bien de densité $p(x, y)$.)

5. Déterminer la loi conditionnelle de y sachant x .

6. À la suite de la question précédente, écrire un nouvel algorithme de simulation du couple (x, y).

7. Ecrire un algorithme d'échantillonnage de Gibbs pour simuler la loi de ce vecteur.

8. programmer les algorithmes précédents en langage `R` et comparer les résultats des simulations.

Exercice 3. On considère la matrice symétrique suivante

$$\Lambda = \begin{pmatrix} 3 & c \\ c & 4 \end{pmatrix},$$

où c est un scalaire quelconque.

1. Pour quelles valeurs de la constante c, la matrice Λ est-elle une matrice de covariance ? *Lorsque la condition précédente est satisfaite, on définit un couple gaussien* (θ_1, θ_2) *de moyenne nulle et de matrice de covariance* Λ. *On pose* $\rho = c/2\sqrt{3}$.

2. Écrire la densité du couple (θ_1, θ_2). Caractériser les lois marginales et les lois conditionnelles en fonction de ρ à partir des formules du cours.

3. Proposer un algorithme de simulation exacte du couple (θ_1, θ_2) à partir de la commande `rnorm` du langage `R`.

4. Rappeler le principe de l'algorithme d'échantillonnage de Gibbs pour le couple (θ_1, θ_2) et écrire cet algorithme en langage `R`.

5. On initialise l'algorithme d'échantillonnage de Gibbs avec la valeur $\theta_2^0 = 0$ et le tirage de θ_1^0. Montrer qu'au cycle t de l'algorithme, on peut écrire

$$\begin{aligned} \theta_1^{t-1} &= \frac{\sqrt{3}}{2}\rho\,\theta_2^{t-1} + \epsilon_2^{t-1} \\ \theta_2^{t} &= \frac{2}{\sqrt{3}}\rho\,\theta_1^{t-1} + \epsilon_1^{t} \end{aligned}$$

   où les variables ϵ_i^t sont des variables gaussiennes indépendantes de moyenne et de variance à préciser.

6. Montrer, à l'aide de la question précédente, que l'on peut écrire pour $t \geq 1$

$$\theta_2^t = \epsilon^t + \rho^2\epsilon^{t-1} + \rho^4\epsilon^{t-2} + \cdots + (\rho^2)^{t-1}\epsilon^1$$

où les variables ϵ^t sont indépendantes de loi normale de moyenne 0 et de variance $4(1-\rho^4)$.

7. En déduire que θ_2^t est une variable aléatoire gaussienne de moyenne nulle et de variance égale à

$$\mathrm{Var}(\theta_2^t) = 4(1-\rho^{4t})$$

Que peut-on dire du comportement de l'algorithme lorsque ρ est proche de la valeur 1 ?

Exercice 4. On considère un vecteur gaussien x en dimension 3 de moyenne nulle et de matrice de covariance

$$C = \begin{pmatrix} \sigma_1^2 & 0 & c_{13} \\ 0 & \sigma_2^2 & c_{23} \\ c_{13} & c_{23} & \sigma_3^2 \end{pmatrix}$$

où $\det C > 0$. Les coordonnées de x sont notées x_1, x_2 et x_3. On définit le vecteur y de la manière suivante

$$\begin{cases} y_1 = x_1 \\ y_2 = x_2 \\ y_3 = x_3 - c_{13}x_1/\sigma_1^2 - c_{23}x_2/\sigma_2^2 \end{cases}$$

1. Montrer que le vecteur y est gaussien et de moyenne nulle.
2. Calculer $\mathrm{Cov}(y_1, y_3)$ et $\mathrm{Cov}(y_2, y_3)$. Montrer que les coordonnées y_1, y_2 et y_3 sont indépendantes.
3. Montrer que

$$\mathrm{Var}(x_3) = \mathrm{Var}(y_3) + c_{13}^2/\sigma_1^2 + c_{23}^2/\sigma_2^2$$

En déduire la loi de y.

4. On dispose d'un générateur aléatoire `rnorm()` retournant des variables indépendantes de loi $\mathrm{N}(0,1)$. Écrire un algorithme de simulation exact

d'un vecteur gaussien de moyenne $m = (0, 1, 0)$ et de matrice de covariance

$$K = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 9 & 2 \\ 0 & 2 & 1 \end{pmatrix}$$

5. Ecrire un algorithme d'échantillonnage de Gibbs pour simuler la loi de ce vecteur.

6. Programmer les algorithmes précédents en langage `R` et comparer les résultats des simulations.

TPs de MPA

Les TPs peuvent être effectués en binôme, et donnent lieu à un compte-rendu noté à rendre **par courrier électronique à son enseignant de TD**. Toute journée de retard dans la réception du courrierélectronique par l'enseignant de TD concerné entraînera un décompte de 4 points sur la note finale. Le compte-rendu ne dépassera pas le nombre de pages indiqué ci-dessous incluant les formules, figures, tableaux et les principales commandes de `R` (format doc ou PDF acceptés). Le barème de notation prend en compte l'exactitude, la qualité de la présentation, les commentaires et la discussion des résultats. Les légendes des figures et de leurs axes devront être explicites sans référence systématique au texte. La description des algorithmes devra garantir que l'on puisse les programmer en `R` sans ambiguité. La qualité de la rédaction, en français ou en anglais, sera aussi notée.

Exercice 0. Simulation et programmation en `R` : A remettre avant de lundi de la semaine 4 (19h).

On considère la loi de probabilité définie sur $\mathbb{R}^2$ par

$$p(y,\theta) = \frac{1}{16\pi} \exp\left(-\frac{1}{32}(8y^2 - 4y\theta + \theta^2)\right) .$$

1. Calculer les lois marginales et conditionnelles du couple (y,θ). Identifier précisement la nature de ces lois.
2. Ecrire un programme en langage `R` permettant de simuler des tirages de loi $p(y,\theta)$.

3. Calculer l'espérance de la loi conditionnelle $p(\theta|y)$.

4. Effectuer 1000 tirages de loi $p(y, \theta)$. Représenter graphiquement les tirages en couleur grise, et superposer en couleur bleue la courbe qui, à y, associe l'espérance $\mathrm{E}[\theta|y]$.

5. Superposer en couleur orange la droite de régression linéaire de la variable θ par la variable y (commande `lm`).

6. Effectuer 100000 tirages de la loi $p(y, \theta)$. Programmer en langage `R` un algorithme rejettant les valeurs de la variable θ telles que la variable y se trouve en dehors de l'intervalle $1.99, 2.01$.

7. Tracer l'histogramme des valeurs retenues par l'algorithme. Interpréter l'histogramme ainsi obtenu.

Exercice 1. Modèle de Poisson. A remettre avant de lundi de la semaine 9 (19h) On souhaite estimer le nombre moyen d'occurrences d'un phénomène donné, correspondant par exemple au nombre de clics journaliers sur un type de produit spécifique dans un site de vente en ligne. Pour cela, on dispose de $n = 19$ observations entières positives ou nulles, notées $y_1, \ldots, y_n$.

y :	9	5	7	15	9	0	22	1	9	11	9	13	12	11	2	1	26	7	0

1. On suppose les observations indépendantes et de loi de Poisson de paramètre $\theta > 0$. Déterminer la vraisemblance du paramètre dans ce modèle.

2. On suppose que la loi a priori est non informative

$$p(\theta) \propto \frac{1}{\theta}, \quad \theta > 0$$

Déterminer la loi a posteriori du paramètre θ. Quelle est l'espérance de la loi a posteriori ?

3. Rappeler le principe de l'algorithme de Metropolis-Hasting. Ecrire dans un programme en `R` une version de cet algorithme pour simuler la loi a posteriori du paramètre θ. On pourra, par exemple, choisir une loi instrumentale exponentielle.

4. Fournir une estimation ponctuelle du paramètre θ (moyenne et médiane a posteriori). Donner un intervalle de crédibilité à 95% pour ce paramètre.

5. Représenter graphiquement l'histogramme de la loi a posteriori du paramètre θ obtenu suivant l'algorithme de Metropolis-Hasting. Superposer la loi obtenue à la question 2.

6. Déterminer la loi prédictive *a posteriori* d'une nouvelle donnée $\tilde{y}$. Ecrire un algorithme de simulation de cette loi et comparer l'histogramme des résultats simulés aux données. Quelles critiques pouvez-vous faire du modèle proposé pour les données et le paramètre θ.

Exercice 2. Modèles de mélanges. A remettre avant de lundi de la semaine 11 (19h) Soit $y = y_1, \ldots, y_n$ un échantillon constitué de n données de comptage, entiers positifs ou nuls. On suppose les données indépendantes et provenant de K sources poissonniennes de moyennes inconnues $\theta_1, \ldots, \theta_K$. On définit un vecteur de variables non-observées $z = (z_1, \ldots, z_n)$, $z_i \in \{1, \ldots, K\}$,

et un modèle pour (y, z, θ) de la manière suivante

$$p(y_i|z_i, \theta) = (\theta_{z_i})^{y_i} e^{-\theta_{z_i}} / y_i!$$

$$p(z) \propto 1$$

$$p(\theta) \propto 1/\theta_1 \dots \theta_K \, .$$

1. Décrire la loi $p(y|z, \theta)$ en séparant les produits faisant intervenir les ensembles d'indices $I_k = \{i : z_i = k\}$, $k = 1, \dots, K$.
2. Décrire la loi a posteriori du vecteur de variables (z, θ).
3. Soit $i \in \{1, \dots, n\}$, calculer la probabilité conditionnelle $p(z_i = k|y, \theta)$.
4. Soit n_k le nombre d'éléments dans I_k et $\bar{y}_k$ la moyenne empirique des données y_i pour $i \in I_k$. Montrer que

$$p(\theta_k|y, z) \propto \theta_k^{n_k \bar{y}_k - 1} \exp(-n_k \theta_k).$$

5. Décrire l'implantation d'un cycle de l'algorithme d'échantillonnage de Gibbs pour la loi a posteriori en langage `R`. On pourra utiliser la commande `rgamma` pour simuler des réalisations de la loi gamma.
6. À l'issue d'une exécution de l'algorithme précédent, comment peut-on estimer l'espérance de θ_k sachant y ? Comment peut-on estimer les proportions de chacune des composantes du mélange ?
7. On considère l'échantillon $y = (9, 5, 7, 15, 9, 0, 22, 1, 9, 11, 9, 13, 12, 11, 2, 1, 26, 7, 0)$. Pour cet échantillon, représenter un histogramme, et superposer la loi de mélange obtenue par l'algorithme d'échantillonnage de Gibbs pour diverses valeurs de K. Quelle valeur de K vous semble-t-elle la plus appropriée pour modéliser l'échantillon de données ? Proposer une ou plusieurs statistiques pour vérifier le modèle et en calculer les lois prédictives (code `R` à donner). Critiquer la modélisation.

R pour les débutants

Emmanuel Paradis

Institut des Sciences de l'Évolution
Université Montpellier II
F-34095 Montpellier cédex 05
France

E-mail : *paradis@isem.univ-montp2.fr*

Je remercie Julien Claude, Christophe Declercq, Élodie Gazave, Friedrich Leisch, Louis Luangkesron, François Pinard et Mathieu Ros pour leurs commentaires et suggestions sur des versions précédentes de ce document. J'exprime également ma reconnaissance à tous les membres du *R Development Core Team* pour leurs efforts considérables dans le développement de R et dans l'animation de la liste de discussion électronique « r-help ». Merci également aux utilisateurs de R qui par leurs questions ou commentaires m'ont aidé à écrire *R pour les débutants*. Mention spéciale à Jorge Ahumada pour la traduction en espagnol.

Table des matières

1 Préambule

Le but du présent document est de fournir un point de départ pour les novices intéressés par R. J'ai fait le choix d'insister sur la compréhension du fonctionnement de R, bien sûr dans le but d'une utilisation de niveau débutant plutôt qu'expert. Les possibilités offertes par R étant très vastes, il est utile pour le débutant d'assimiler certaines notions et concepts afin d'évoluer plus aisément par la suite. J'ai essayé de simplifier au maximum les explications pour les rendre accessibles à tous, tout en donnant les détails utiles, parfois sous forme de tableaux.

R est un système d'analyse statistique et graphique créé par Ross Ihaka et Robert Gentleman[1]. R est à la fois un logiciel et un langage qualifié de dialecte du langage S créé par AT&T Bell Laboratories. S est disponible sous la forme du logiciel S-PLUS commercialisé par la compagnie Insightful[2]. Il y a des différences importantes dans la conception de R et celle de S : ceux qui veulent en savoir plus sur ce point peuvent se reporter à l'article de Ihaka & Gentleman (1996) ou au R-FAQ[3] dont une copie est également distribuée avec R.

R est distribué librement sous les termes de la *GNU General Public Licence*[4] ; son développement et sa distribution sont assurés par plusieurs statisticiens rassemblés dans le *R Development Core Team*.

R est disponible sous plusieurs formes : le code (écrit principalement en C et certaines routines en Fortran), surtout pour les machines Unix et Linux, ou des exécutables précompilés pour Windows, Linux et Macintosh. Les fichiers pour installer R, à partir du code ou des exécutables, sont distribués à partir du site internet du *Comprehensive R Archive Network* (CRAN)[5] où se trouvent aussi les instructions à suivre pour l'installation sur chaque système. En ce qui concerne les distributions de Linux (Debian, ...), les exécutables sont généralement disponibles pour les versions les plus récentes ; consultez le site du CRAN si besoin.

R comporte de nombreuses fonctions pour les analyses statistiques et les graphiques ; ceux-ci sont visualisés immédiatement dans une fenêtre propre et peuvent être exportés sous divers formats (jpg, png, bmp, ps, pdf, emf, pictex, xfig ; les formats disponibles peuvent dépendre du système d'exploitation). Les résultats des analyses statistiques sont affichés à l'écran, certains résultats partiels (valeurs de P, coefficients de régression, résidus, ...) peuvent être sauvés à part, exportés dans un fichier ou utilisés dans des analyses ultérieures.

[1] Ihaka R. & Gentleman R. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5 : 299–314.

[2] voir http://www.insightful.com/products/splus/default.asp pour plus d'information

[3] http://cran.r-project.org/doc/FAQ/R-FAQ.html

[4] pour plus d'infos : http://www.gnu.org/

[5] http://cran.r-project.org/

Le langage R permet, par exemple, de programmer des boucles qui vont analyser successivement différents jeux de données. Il est aussi possible de combiner dans le même programme différentes fonctions statistiques pour réaliser des analyses plus complexes. Les utilisateurs de R peuvent bénéficier des nombreux programmes écrits pour S et disponibles sur internet[6], la plupart de ces programmes étant directement utilisables avec R.

De prime abord, R peut sembler trop complexe pour une utilisation par un non-spécialiste. Ce n'est pas forcément le cas. En fait, R privilégie la flexibilité. Alors qu'un logiciel classique affichera directement les résultats d'une analyse, avec R ces résultats sont stockés dans un "objet", si bien qu'une analyse peut être faite sans qu'aucun résultat ne soit affiché. L'utilisateur peut être déconcerté par ceci, mais cette facilité se révèle extrêmement utile. En effet, l'utilisateur peut alors extraire uniquement la partie des résultats qui l'intéresse. Par exemple, si l'on doit faire une série de 20 régressions et que l'on veuille comparer les coefficients des différentes régressions, R pourra afficher uniquement les coefficients estimés : les résultats tiendront donc sur une ligne, alors qu'un logiciel plus classique pourra ouvrir 20 fenêtres de résultats. On verra d'autres exemples illustrant la flexibilité d'un système comme R vis-à-vis des logiciels classiques.

[6] par exemple : http://stat.cmu.edu/S/

2 Quelques concepts avant de démarrer

Une fois R installé sur votre ordinateur, il suffit de lancer l'exécutable correspondant pour démarrer le programme. L'attente de commandes (par défaut le symbole '>') apparait alors indiquant que R est prêt à exécuter les commandes. Sous Windows en utilisant le programme Rgui.exe, certaines commandes (accès à l'aide, ouverture de fichiers, ...) peuvent être exécutées par les menus. L'utilisateur novice a alors toutes les chances de se demander « Je fais quoi maintenant ? » Il est en effet très utile d'avoir quelques idées sur le fonctionnement de R lorsqu'on l'utilise pour la première fois : c'est ce que nous allons voir maintenant.

Nous allons dans un premier temps voir schématiquement comment R travaille. Ensuite nous décrirons l'opérateur « assigner » qui permet de créer des objets, puis comment gérer les objets en mémoire, et finalement comment utiliser l'aide en ligne qui est extrêmement utile dans une utilisation courante.

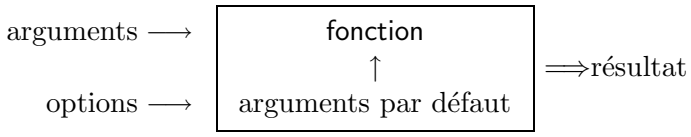
2.1 Comment R travaille

Le fait que R soit un langage peut effrayer plus d'un utilisateur potentiel pensant « Je ne sais pas programmer ». Cela ne devrait pas être le cas pour deux raisons. D'abord, R est un langage interprété et non compilé, c'est-à-dire que les commandes tapées au clavier sont directement exécutées sans qu'il soit besoin de construire un programme complet comme cela est le cas pour la plupart des langages informatiques (C, Fortran, Pascal, ...).

Ensuite, la syntaxe de R est très simple et intuitive. Par exemple, une régression linéaire pourra être faite avec la commande `lm(y ~ x)`. Avec R, une fonction, pour être exécutée, s'écrit *toujours* avec des parenthèses, même si elles ne contiennent rien (par exemple `ls()`). Si l'utilisateur tape le nom de la fonction sans parenthèses, R affichera le contenu des instructions de cette fonction. Dans la suite de ce document, les noms des fonctions sont généralement écrits avec des parenthèses pour les distinguer des autres objets sauf si le texte indique clairement qu'il s'agit d'une fonction.

Quand R est utilisé, les variables, les données, les fonctions, les résultats, etc, sont stockés dans la mémoire de l'ordinateur sous forme d'*objets* qui ont chacun un *nom*. L'utilisateur va agir sur ces objets avec des *opérateurs* (arithmétiques, logiques, de comparaison, ...) et des *fonctions* (qui sont elles-mêmes des objets).

L'utilisation des opérateurs est relativement intuitive, on en verra les détails plus loin (p. 27). Une fonction de R peut être schématisée comme suit :



Les arguments peuvent être des objets (« données », formules, expressions, ...) dont certains peuvent être définis par défaut dans la fonction ; ces valeurs par défaut peuvent être modifiées par l'utilisateur avec les options. Une fonction de R peut ne nécessiter aucun argument de la part de l'utilisateur : soit tous les arguments sont définis par défaut (et peuvent être changés avec les options), ou soit aucun argument n'est défini. On verra plus en détail l'utilisation et la construction des fonctions (p. 72). La présente description est pour le moment suffisante pour comprendre comment R opère.

Toutes les actions de R sont effectuées sur les objets présents dans la mémoire vive de l'ordinateur : aucun fichier temporaire n'est utilisé (Fig. 1). Les lectures et écritures de fichiers sont utilisées pour la lecture et l'enregistrement des données et des résultats (graphiques, ...). L'utilisateur exécute des fonctions par l'intermédiaire de commandes. Les résultats sont affichés directement à l'écran, ou stockés dans un objet, ou encore écrits sur le disque (en particulier pour les graphiques). Les résultats étant eux-mêmes des objets, ils peuvent être considérés comme des données et être analysés à leur tour. Les fichiers de données peuvent être lus sur le disque de l'ordinateur local ou sur un serveur distant via internet.

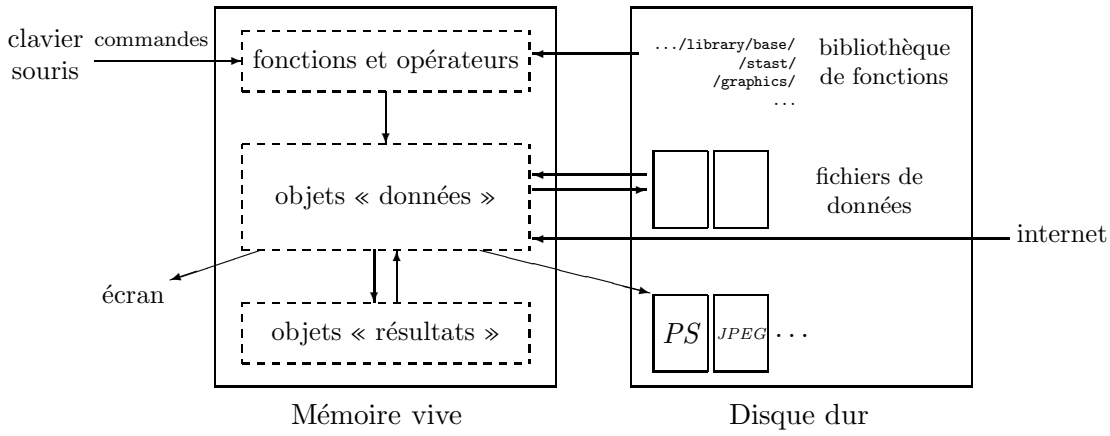


Fig. 1 – Une vue schématique du fonctionnement de R.

Les fonctions disponibles sont stockées dans une bibliothèque localisées sur le disque dans le répertoire R_HOME/library (R_HOME désignant le répertoire où R est installé). Ce répertoire contient des *packages* de fonctions, eux-mêmes présents sur le disque sous forme de répertoires. Le package nommé base est en quelque sorte le cœur de R et contient les fonctions de base du lan-

gage, en particulier pour la lecture et la manipulation des données. Chaque package a un répertoire nommé R avec un fichier qui a pour nom celui du package (par exemple, pour base, ce sera le fichier R_HOME/library/base/R/base). Ce fichier contient les fonctions du package.

Une des commandes les plus simples consiste à taper le nom d'un objet pour afficher son contenu. Par exemple, si un objet `n` contient la valeur 10 :

```
> n
[1] 10
```

Le chiffre 1 entre crochets indique que l'affichage commence au premier élément de `n`. Cette commande est une utilisation implicite de la fonction `print` et l'exemple ci-dessus est identique à `print(n)` (dans certaines situations, la fonction `print` doit être utilisée de façon explicite, par exemple au sein d'une fonction ou d'une boucle).

Le nom d'un objet doit obligatoirement commencer par une lettre (A–Z et a–z) et peut comporter des lettres, des chiffres (0–9), des points (.) et des 'espaces soulignés' (_). Il faut savoir aussi que R distingue, pour les noms des objets, les majuscules des minuscules, c'est-à-dire que `x` et `X` pourront servir à nommer des objets distincts (même sous Windows).

2.2 Créer, lister et effacer les objets en mémoire

Un objet peut être créé avec l'opérateur « assigner » qui s'écrit avec une flèche composée d'un signe moins accolé à un crochet, ce symbole pouvant être orienté dans un sens ou dans l'autre :

```
> n <- 15
> n
[1] 15
> 5 -> n
> n
[1] 5
> x <- 1
> X <- 10
> x
[1] 1
> X
[1] 10
```

Si l'objet existe déjà, sa valeur précédente est effacée (la modification n'affecte que les objets en mémoire vive, pas les données sur le disque). La valeur ainsi donnée peut être le résultat d'une opération et/ou d'une fonction :

```
> n <- 10 + 2
> n
```

```
[1] 12
> n <- 3 + rnorm(1)
> n
[1] 2.208807
```

La fonction `rnorm(1)` génère une variable aléatoire normale de moyenne zéro et variance unité (p. 19). On peut simplement taper une expression sans assigner sa valeur à un objet, le résultat est alors affiché à l'écran mais n'est pas stocké en mémoire :

```
> (10 + 2) * 5
[1] 60
```

Dans nos exemples, on omettra l'assignement si cela n'est pas nécessaire à la compréhension.

La fonction `ls` permet d'afficher une liste simple des objets en mémoire, c'est-à-dire que seuls les noms des objets sont affichés.

```
> name <- "Carmen"; n1 <- 10; n2 <- 100; m <- 0.5
> ls()
[1] "m"    "n1"   "n2"   "name"
```

Notons l'usage du point-virgule pour séparer des commandes distinctes sur la même ligne. Si l'on veut lister uniquement les objets qui contiennent un caractère donné dans leur nom, on utilisera alors l'option `pattern` (qui peut s'abréger avec `pat`) :

```
> ls(pat = "m")
[1] "m"    "name"
```

Pour restreindre la liste aux objets dont le nom commence par le caractère en question :

```
> ls(pat = "^m")
[1] "m"
```

La fonction `ls.str` affiche des détails sur les objets en mémoire :

```
> ls.str()
m :  num 0.5
n1 :  num 10
n2 :  num 100
name :  chr "Carmen"
```

L'option `pattern` peut également être utilisée comme avec `ls`. Une autre option utile de `ls.str` est `max.level` qui spécifie le niveau de détails de l'affichage des objets composites. Par défaut, `ls.str` affiche les détails de tous les objets contenus en mémoire, y compris les colonnes des jeux de données, matrices et listes, ce qui peut faire un affichage très long. On évite d'afficher tous les détails avec l'option `max.level = -1` :

```
> M <- data.frame(n1, n2, m)
> ls.str(pat = "M")
M : 'data.frame':       1 obs. of  3 variables:
 $ n1: num 10
 $ n2: num 100
 $ m : num 0.5
> ls.str(pat="M", max.level=-1)
M : 'data.frame':       1 obs. of  3 variables:
```

Pour effacer des objets de la mémoire, on utilise la fonction `rm` : `rm(x)` pour effacer l'objet `x`, `rm(x, y)` pour effacer les objets `x` et `y`, `rm(list=ls())` pour effacer tous les objets en mémoire ; on pourra ensuite utiliser les mêmes options citées pour `ls()` pour effacer sélectivement certains objets : `rm(list=ls(pat = "^m"))`.

2.3 L'aide en ligne

L'aide en ligne de R est extrêment utile pour l'utilisation des fonctions. L'aide est disponible directement pour une fonction donnée, par exemple :

```
> ?lm
```

affichera, dans R, la page d'aide pour la fonction `lm()` (*linear model*). Les commandes `help(lm)` et `help("lm")` auront le même effet. C'est cette dernière qu'il faut utiliser pour accéder à l'aide avec des caractères non-conventionnels :

```
> ?*
Error: syntax error
> help("*")
Arithmetic              package:base              R Documentation

Arithmetic Operators
...
```

L'appel de l'aide ouvre une page (le comportement exact dépend du système d'exploitation) avec sur la première ligne des informations générales dont le nom du package où se trouvent la (ou les) fonction(s) ou les opérateurs documentés. Ensuite vient un titre suivi de paragraphes qui chacun apporte une information bien précise.

Description: brève description.

Usage: pour une fonction donne le nom avec tous ses arguments et les éventuelles options (et les valeurs par défaut correspondantes) ; pour un opérateur donne l'usage typique.

Arguments: pour une fonction détaille chacun des arguments.

Details: description détaillée.

Value: le cas échéant, le type d'objet retourné par la fonction ou l'opérateur.

See Also: autres rubriques d'aide proches ou similaires à celle documentée.

Examples: des exemples qui généralement peuvent être exécutés sans ouvrir l'aide avec la fonction `example`.

Pour un débutant, il est conseillé de regarder le paragraphe **Examples**. En général, il est utile de lire attentivement le paragraphe **Arguments**. D'autres paragraphes peuvent être rencontrés, tel **Note**, **References** ou **Author(s)**.

Par défaut, la fonction `help` ne recherche que dans les packages chargés en mémoire. L'option `try.all.packages`, dont le défaut est `FALSE`, permet de chercher dans tous les packages si sa valeur est `TRUE` :

```
> help("bs")
No documentation for 'bs' in specified packages and libraries:
you could try 'help.search("bs")'
> help("bs", try.all.packages = TRUE)
Help for topic 'bs' is not in any loaded package but
can be found in the following packages:

  Package               Library
  splines               /usr/lib/R/library
```

Notez que dans ce cas la page d'aide de la fonction `bs` n'est pas ouverte. L'utilisateur peut ouvrir des pages d'aide d'un package non chargé en mémoire en utilisant l'option `package` :

```
> help("bs", package = "splines")
bs                package:splines                R Documentation

B-Spline Basis for Polynomial Splines

Description:

     Generate the B-spline basis matrix for a polynomial spline.
...
```

On peut ouvrir l'aide au format html (qui sera lu avec Netscape, par exemple) en tapant :

```
> help.start()
```

Une recherche par mots-clefs est possible avec cette aide html. La rubrique **See Also** contient ici des liens hypertextes vers les pages d'aide des autres fonctions. La recherche par mots-clefs est également possible depuis R avec la fonction `help.search`. Cette dernière recherche un thème, spécifié par une chaîne de caractère, dans les pages d'aide de tous les packages installés. Par exemple, `help.search("tree")` affichera une liste des fonctions dont les pages

d'aide mentionnent « tree ». Notez que si certains packages ont été installés récemment, il peut être utile de rafraîchir la base de données utilisée par `help.search` en utilisant l'option `rebuild` (`help.search("tree", rebuild = TRUE)`).

La fonction `apropos` trouve les fonctions qui contiennent dans leur nom la chaîne de caractère passée en argument ; seuls les packages chargés en mémoire sont cherchés :

```
> apropos(help)
[1] "help"         ".helpForCall" "help.search"
[4] "help.start"
```

3 Les données avec R

3.1 Les objects

Nous avons vu que R manipule des objets : ceux-ci sont caractérisés bien sûr par leur nom et leur contenu, mais aussi par des *attributs* qui vont spécifier le type de données représenté par un objet. Afin de comprendre l'utilité de ces attributs, considérons une variable qui prendrait les valeurs 1, 2 ou 3 : une telle variable peut représenter une variable entière (par exemple, le nombre d'œufs dans un nid), ou le codage d'une variable catégorique (par exemple, le sexe dans certaines populations de crustacés : mâle, femelle ou hermaphrodite).

Il est clair que le traitement statistique de cette variable ne sera pas le même dans les deux cas : avec R, les attributs de l'objet donnent l'information nécessaire. Plus techniquement, et plus généralement, l'action d'une fonction sur un objet va dépendre des attributs de celui-ci.

Les objets ont tous deux attributs *intrinsèques* : le *mode* et la *longueur*. Le mode est le type des éléments d'un objet ; il en existe quatre principaux : numérique, caractère, complexe[7], et logique (`FALSE` ou `TRUE`). D'autres modes existent qui ne représentent pas des données, par exemple fonction ou expression. La longueur est le nombre d'éléments de l'objet. Pour connaître le mode et la longueur d'un objet on peut utiliser, respectivement, les fonctions `mode` et `length` :

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
```

Quelque soit le mode, les valeurs manquantes sont représentées par `NA` (*not available*). Une valeur numérique très grande peut être spécifiée avec une notation exponentielle :

```
> N <- 2.1e23
> N
[1] 2.1e+23
```

[7] Le mode complexe ne sera pas discuté dans ce document.

R représente correctement des valeurs numériques qui ne sont pas finies, telles que $\pm\infty$ avec `Inf` et `-Inf`, ou des valeurs qui ne sont pas des nombres avec `NaN` (*not a number*).

```
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

Une valeur de mode caractère est donc entrée entre des guillemets doubles `"`. Il est possible d'inclure ce dernier caractère dans la valeur s'il suit un antislash \. L'ensemble des deux caractères \" sera traité de façon spécifique par certaines fonctions telle que `cat` pour l'affichage à l'écran, ou `write.table` pour écrire sur le disque (p. 16, l'option `qmethod` de cette fonction).

```
> x <- "Double quotes \" delimitate R's strings."
> x
[1] "Double quotes \" delimitate R's strings."
> cat(x)
Double quotes " delimitate R's strings.
```

Une autre possibilité est de délimiter les variables de mode caractère avec des guillemets simples (') ; dans ce cas il n'est pas nécessaire d'échapper les guillemets doubles avec des antislash (mais les guillemets simples doivent l'être !) :

```
> x <- 'Double quotes " delimitate R\'s strings.'
> x
[1] "Double quotes \" delimitate R's strings."\
```

Le tableau suivant donne un aperçu des objets représentant des données.

objet	modes	plusieurs modes possibles dans le même objet ?
vecteur	numérique, caractère, complexe *ou* logique	Non
facteur	numérique *ou* caractère	Non
tableau	numérique, caractère, complexe *ou* logique	Non
matrice	numérique, caractère, complexe *ou* logique	Non
tableau de données	numérique, caractère, complexe *ou* logique	Oui
ts	numérique, caractère, complexe *ou* logique	Non
liste	numérique, caractère, complexe, logique, fonction, expression, ...	Oui

Un vecteur est une variable dans le sens généralement admis. Un facteur est une variable catégorique. Un tableau (*array*) possède k dimensions, une matrice étant un cas particulier de tableau avec $k = 2$. À noter que les éléments d'un tableau ou d'une matrice sont tous du même mode. Un tableau de données (*data frame*) est composé de un ou plusieurs vecteurs et/ou facteurs ayant tous la même longueur mais pouvant être de modes différents. Un « ts » est un jeu de données de type séries temporelles (*time series*) et comporte donc des attributs supplémentaires comme la fréquence et les dates. Enfin, une *liste* peut contenir n'importe quel type d'objet, y compris des listes !

Pour un vecteur, le mode et la longueur suffisent pour décrire les données. Pour les autres objets, d'autres informations sont nécessaires et celles-ci sont données par les attributs dits *non-intrinsèques*. Parmi ces attributs, citons *dim* qui correspond au nombre de dimensions d'un objet. Par exemple, une matrice composée de 2 lignes et 2 colonnes aura pour `dim` le couple de valeurs [2, 2] ; par contre sa longueur sera de 4.

3.2 Lire des données dans un fichier

Pour les lectures et écritures dans les fichiers, R utilise le répertoire de travail. Pour connaître ce répertoire on peut utiliser la commande `getwd()` (*get working directory*), et on peut le modifier avec, par exemple, `setwd("C:/data")` ou `setwd("/home/paradis/R")`. Il est nécessaire de préciser le chemin d'accès au fichier s'il n'est pas dans le répertoire de travail.[8]

R peut lire des données stockées dans des fichiers texte (ASCII) à l'aide des fonctions suivantes : `read.table` (qui a plusieurs variantes, *cf.* ci-dessous), `scan` et `read.fwf`. R peut également lire des fichiers dans d'autres formats (Excel, SAS, SPSS, ...) et accéder à des bases de données de type SQL, mais les fonctions nécessaires ne sont pas dans le package `base`. Ces fonctionnalités

[8] Sous Windows, il est pratique de créer un raccourci de Rgui.exe, puis éditer ses propriétés et modifier le répertoire dans le champ « Démarrer en : » sous l'onglet « Raccourci » : ce répertoire sera ensuite le répertoire de travail en démarrant R depuis ce raccourci.

sont très utiles pour une utilisation un peu plus avancée de R, mais on se limitera ici à la lecture de fichiers au format ASCII.

La fonction `read.table` a pour effet de créer un tableau de données et est donc le moyen principal pour lire des fichiers de données. Par exemple, si on a un fichier nommé data.dat, la commande :

```
> mydata <- read.table("data.dat")
```

créera un tableau de données nommé `mydata`, et les variables, par défaut nommées `V1`, `V2` ..., pourront être accédées individuellement par `mydata$V1`, `mydata$V2`, ..., ou par `mydata ["V1"]`, `mydata["V2"]`, ..., ou encore par `mydata[, 1]`, `mydata[, 2]`, ...[9] Il y a plusieurs options dont voici les valeurs par défaut (c'est-à-dire celles utilisées par R si elles sont omises par l'utilisateur) et les détails dans le tableau qui suit :

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".",
           row.names, col.names, as.is = FALSE, na.strings = "NA",
           colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#")
```

`file`	le nom du fichier (entre `""` ou une variable de mode caractère), éventuellement avec son chemin d'accès (le symbole \ est interdit et doit être remplacé par /, même sous Windows), ou un accès distant à un fichier de type URL (http://...)
`header`	une valeur logique (`FALSE` ou `TRUE`) indicant si le fichier contient les noms des variables sur la 1ère ligne
`sep`	le séparateur de champ dans le fichier, par exemple `sep="\t"` si c'est une tabulation
`quote`	les caractères utilisés pour citer les variables de mode caractère
`dec`	le caractère utilisé pour les décimales
`row.names`	un vecteur contenant les noms des lignes qui peut être un vecteur de mode character, ou le numéro (ou le nom) d'une variable du fichier (par défaut : `1`, `2`, `3`, ...)
`col.names`	un vecteur contenant les noms des variables (par défaut : `V1`, `V2`, `V3`, ...)
`as.is`	contrôle la conversion des variables caractères en facteur (si `FALSE`) ou les conserve en caractères (`TRUE`) ; `as.is` peut être un vecteur logique, numérique ou caractère précisant les variables conservées en caractère
`na.strings`	indique la valeur des données manquantes (sera converti en `NA`)
`colClasses`	un vecteur de caractères donnant les classes à attribuer aux colonnes
`nrows`	le nombre maximum de lignes à lire (les valeurs négatives sont ignorées)

[9] Il y a toutefois une différence : `mydata$V1` et `mydata[, 1]` sont des vecteurs alors que `mydata["V1"]` est un tableau de données. On verra plus loin (p. 20) des détails sur la manipulation des objets.

`skip`	le nombre de lignes à sauter avant de commencer la lecture des données
`check.names`	si `TRUE`, vérifie que les noms des variables sont valides pour R
`fill`	si `TRUE` et que les lignes n'ont pas tous le même nombre de variables, des "blancs" sont ajoutés
`strip.white`	(conditionnel à `sep`) si `TRUE`, efface les espaces (= blancs) avant et après les variables de mode caractère
`blank.lines.skip`	si `TRUE`, ignore les lignes « blanches »
`comment.char`	un caractère qui définit des commentaires dans le fichier de données, la lecture des données passant à la ligne suivante (pour désactiver cet option, utiliser `comment.char = ""`)

Les variantes de `read.table` sont utiles car elles ont des valeurs par défaut différentes :

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, ...)
read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
          fill = TRUE, ...)
read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".",
           fill = TRUE, ...)
read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=",",
            fill = TRUE, ...)
```

La fonction `scan` est plus flexible que `read.table`. Une différence est qu'il est possible de spécifier le mode des variables, par exemple :

```
> mydata <- scan("data.dat", what = list("", 0, 0))
```

lira dans le fichier data.dat trois variables, la première de mode caractère et les deux suivantes de mode numérique. Une autre distinction importante est que `scan()` peut être utilisée pour créer différents objets, vecteurs, matrices, tableaux de données, listes, ... Dans l'exemple ci-dessus, `mydata` est une liste de trois vecteurs. Par défaut, c'est-à-dire si `what` est omis, `scan()` crée un vecteur numérique. Si les données lues ne correspondent pas au(x) mode(s) attendu(s) (par défaut ou spécifiés par `what`), un message d'erreur est retourné. Les options sont les suivantes.

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
     quote = if (sep=="\n") "" else "'\"", dec = ".",
     skip = 0, nlines = 0, na.strings = "NA",
     flush = FALSE, fill = FALSE, strip.white = FALSE, quiet = FALSE,
     blank.lines.skip = TRUE, multi.line = TRUE, comment.char = "",
     allowEscapes = TRUE)
```

`file`	le nom du fichier (entre `""`), éventuellement avec son chemin d'accès (le symbole \ est interdit et doit être remplacé par /, même sous Windows), ou un accès distant à un fichier de type URL (http://...) ; si `file=""`, les données sont entrées au clavier (l'entrée étant terminée par une ligne blanche)
`what`	indique le(s) mode(s) des données lues (numérique par défaut)
`nmax`	le nombre de données à lire, ou, si `what` est une liste, le nombre de lignes lues (par défaut, `scan` lit jusqu'à la fin du fichier)
`n`	le nombre de données à lire (par défaut, pas de limite)
`sep`	le séparateur de champ dans le fichier
`quote`	les caractères utilisés pour citer les variables de mode caractère
`dec`	le caractère utilisé pour les décimales
`skip`	le nombre de lignes à sauter avant de commencer la lecture des données
`nlines`	le nombre de lignes à lire
`na.string`	indique la valeur des données manquantes (sera converti en `NA`)
`flush`	si `TRUE`, `scan` va à la ligne suivante une fois que le nombre de colonnes est atteint (permet d'ajouter des commentaires dans le fichier de données)
`fill`	si `TRUE` et que les lignes n'ont pas tous le même nombre de variables, des "blancs" sont ajoutés
`strip.white`	(conditionnel à `sep`) si `TRUE`, efface les espaces (= blancs) avant et après les variables de mode character
`quiet`	si `FALSE`, `scan` affiche une ligne indiquant quels champs ont été lus
`blank.lines.skip`	si `TRUE`, ignore les lignes « blanches »
`multi.line`	si `what` est une liste, précise si les variables du même individu sont sur une seule ligne dans le fichier (`FALSE`)
`comment.char`	un caractère qui définit des commentaires dans le fichier de données, la lecture des données passant à la ligne suivante (par défaut les commentaires ne sont pas permis)
`allowEscapes`	spécifie si les caractères échappés (par ex. \t) doivent être interprétés (le défaut) ou laissés tels-quels

La fonction `read.fwf` sert à lire dans un fichier où les données sont dans un format à largeur fixée (*fixed width format*) :

```
read.fwf(file, widths, header = FALSE, sep = "\t",
         as.is = FALSE, skip = 0, row.names, col.names,
         n = -1, buffersize = 2000, ...)
```

Les options sont les mêmes que pour `read.table()` sauf `widths` qui spécifie la largeur des champs (`buffersize` est le nombre maximum de lignes lues en même temps). Par exemple, si on a un fichier nommé data.txt dont le contenu est indiqué ci-contre, on pourra lire les données avec la commande suivante :

```
A1.501.2
A1.551.3
B1.601.4
B1.651.5
C1.701.6
C1.751.7
```

```
> mydata <- read.fwf("data.txt", widths=c(1, 4, 3))
> mydata
  V1   V2  V3
1  A 1.50 1.2
```

```
2  A 1.55 1.3
3  B 1.60 1.4
4  B 1.65 1.5
5  C 1.70 1.6
6  C 1.75 1.7
```

3.3 Enregistrer les données

La fonction `write.table` écrit dans un fichier un objet, typiquement un tableau de données mais cela peut très bien être un autre type d'objet (vecteur, matrice, ...). Les arguments et options sont :

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"))
```

`x`	le nom de l'objet à écrire
`file`	le nom du fichier (par défaut l'objet est affiché à l'écran)
`append`	si `TRUE` ajoute les données sans effacer celles éventuellement existantes dans le fichier
`quote`	une variable logique ou un vecteur numérique : si `TRUE` les variables de mode caractère et les facteurs sont écrits entre `""`, sinon le vecteur indique les numéros des variables à écrire entre `""` (dans les deux cas les noms des variables sont écrits entre `""` mais pas si `quote = FALSE`)
`sep`	le séparateur de champ dans le fichier
`eol`	le caractère imprimé à la fin de chaque ligne (`"\n"` correspond à un retour-charriot)
`na`	indique le caractère utilisé pour les données manquantes
`dec`	le caractère utilisé pour les décimales
`row.names`	une variable logique indiquant si les noms des lignes doivent être écrits dans le fichier
`col.names`	idem pour les noms des colonnes
`qmethod`	spécifie, si `quote=TRUE`, comment sont traitées les guillemets doubles `"` incluses dans les variables de mode caractère : si `"escape"` (ou `"e"`, le défaut) chaque `"` est remplacée par `\"`, si `"d"` chaque `"` est remplacée par `""`

Pour écrire de façon plus simple un objet dans un fichier, on peut utiliser la commande `write(x, file="data.txt")` où `x` est le nom de l'objet (qui peut être un vecteur, une matrice ou un tableau). Il y a deux options : `nc` (ou `ncol`) qui définit le nombre de colonnes dans le fichier (par défaut `nc=1` si `x` est de mode caractère, `nc=5` pour les autres modes), et `append` (un logique) pour ajouter les données sans effacer celles éventuellement déjà existantes dans le fichier (`TRUE`) ou les effacer si le fichier existe déjà (`FALSE`, le défaut).

Pour enregistrer des objets, cette fois de n'importe quel type, on utilisera la commande `save(x, y, z, file="xyz.RData")`. Pour faciliter l'échange de fichiers entre machines et systèmes d'exploitation, on peut utiliser l'option `ascii=TRUE`. Les données (qui sont alors nommées *workspace* dans le jargon de

R) peuvent ultérieurement être chargées en mémoire avec `load("xyz.RData")`. La fonction `save.image` est un raccourci pour `save(list=ls (all=TRUE), file=".RData")`.

3.4 Générer des données

3.4.1 Séquences régulières

Une séquence régulière de nombres entiers, par exemple de 1 à 30, peut être générée par :

```
> x <- 1:30
```

On a ainsi un vecteur `x` avec 30 éléments. Cet opérateur ‘`:`’ est prioritaire sur les opérations arithmétiques au sein d'une expression :

```
> 1:10-1
 [1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
```

La fonction `seq` peut générer des séquences de nombres réels de la manière suivante :

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

où le premier nombre indique le début de la séquence, le second la fin, et le troisième l'incrément utilisé dans la progression de la séquence. On peut aussi utiliser :

```
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

On peut aussi taper directement les valeurs désirées en utilisant la fonction `c` :

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Il est aussi possible si l'on veut taper des données au clavier d'utiliser la fonction `scan` avec tout simplement les options par défaut :

```
> z <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
10:
Read 9 items
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

La fonction `rep` crée un vecteur qui aura tous ses éléments identiques :

```
> rep(1, 30)
 [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 1 1
```

La fonction `sequence` va créer une suite de séquences de nombres entiers qui chacune se termine par les nombres donnés comme arguments à cette fonction :

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

La fonction `gl` (*generate levels*) est très utile car elle génère des séries régulières dans un facteur. Cette fonction s'utilise ainsi `gl(k, n)` où `k` est le nombre de niveaux (ou classes) du facteur, et `n` est le nombre de réplications pour chaque niveau. Deux options peuvent être utilisées : `length` pour spécifier le nombre de données produites, et `labels` pour indiquer les noms des niveaux du facteur. Exemples :

```
> gl(3, 5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels:  1 2 3
> gl(3, 5, length=30)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels:  1 2 3
> gl(2, 6, label=c("Male", "Female"))
 [1] Male   Male   Male   Male   Male   Male
 [7] Female Female Female Female Female Female
Levels:  Male Female
> gl(2, 10)
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels:  1 2
> gl(2, 1, length=20)
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels:  1 2
> gl(2, 2, length=20)
 [1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels:  1 2
```

Enfin, `expand.grid()` sert à créer un tableau de données avec toutes les combinaisons des vecteurs ou facteurs donnés comme arguments :

```
> expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
   h   w    sex
1 60 100   Male
```

```
2 80 100   Male
3 60 300   Male
4 80 300   Male
5 60 100 Female
6 80 100 Female
7 60 300 Female
8 80 300 Female
```

3.4.2 Séquences aléatoires

Il est utile en statistique de pouvoir générer des données aléatoires, et R peut le faire pour un grand nombre de fonctions de densité de probabilité. Ces fonctions sont de la forme `rfunc(n, p1, p2, ...)`, où *func* indique la loi de probabilité, `n` le nombre de données générées et `p1`, `p2`, ... sont les valeurs des paramètres de la loi. Le tableau suivant donne les détails pour chaque loi, et les éventuelles valeurs par défaut (si aucune valeur par défaut n'est indiquée, c'est que le paramètre doit être spécifié).

loi	**fonction**
Gauss (normale)	`rnorm(n, mean=0, sd=1)`
exponentielle	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomiale	`rbinom(n, size, prob)`
multinomiale	`rmultinom(n, size, prob)`
géométrique	`rgeom(n, prob)`
hypergéométrique	`rhyper(nn, m, n, k)`
logistique	`rlogis(n, location=0, scale=1)`
lognormale	`rlnorm(n, meanlog=0, sdlog=1)`
binomiale négative	`rnbinom(n, size, prob)`
uniforme	`runif(n, min=0, max=1)`
statistiques de Wilcoxon	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

La plupart de ces fonctions ont des compagnes obtenues en remplaçant la lettre `r` par `d`, `p` ou `q` pour obtenir, dans l'ordre, la densité de probabilité (`dfunc(x, ...)`), la densité de probabilité cumulée (`pfunc(x, ...)`), et la valeur de quantile (`qfunc(p, ...)`, avec $0 < p < 1$).

Les deux dernières séries de fonctions peuvent être utilisées pour trouver les valeurs critiques ou les valeurs de P de tests statistiques. Par exemple, les

valeurs critiques au seuil de 5% pour un test bilatéral suivant une loi normale sont :

```
> qnorm(0.025)
[1] -1.959964
> qnorm(0.975)
[1] 1.959964
```

Pour la version unilatérale de ce test, `qnorm(0.05)` ou `1 - qnorm(0.95)` sera utilisé dépendant de la forme de l'hypothèse alternative.

La valeur de P d'un test, disons $\chi^2 = 3.84$ avec $ddl = 1$, est :

```
> 1 - pchisq(3.84, 1)
[1] 0.05004352
```

3.5 Manipuler les objets

3.5.1 Création d'objets

On a vu différentes façons de créer des objets en utilisant l'opérateur assigner ; le mode et le type de l'objet ainsi créé sont généralement déterminés de façon implicite. Il est possible de créer un objet en précisant de façon explicite son mode, sa longueur, son type, etc. Cette approche est intéressante dans l'idée de manipuler les objets. On peut, par exemple, créer un vecteur 'vide' puis modifier successivement ses éléments, ce qui est beaucoup plus efficace que de rassembler ces éléments avec `c()`. On utilisera alors l'indexation comme on le verra plus loin (p. 28).

Il peut être aussi extrêment pratique de créer des objets à partir d'autres objets. Par exemple, si l'on veut ajuster une série de modèles, il sera commode de mettre les formules correspondantes dans une liste puis d'extraire successivement chaque élément de celle-ci qui sera ensuite inséré dans la fonction `lm`.

À ce point de notre apprentissage de R, l'intérêt d'aborder les fonctionnalités qui suivent n'est pas seulement pratique mais aussi didactique. La construction explicite d'objets permet de mieux comprendre leur structure et d'approfondir certaines notions vues précédemment.

Vecteur. La fonction `vector`, qui a deux arguments `mode` et `length`, va servir à créer un vecteur dont la valeur des éléments sera fonction du mode spécifié : 0 si numérique, `FALSE` si logique, ou `""` si caractère. Les fonctions suivantes ont exactement le même effet et ont pour seul argument la longueur du vecteur créé : `numeric()`, `logical()`, et `character()`.

Facteur. Un facteur inclue non seulement les valeurs de la variable catégorique correspondante mais aussi les différents niveaux possibles de cette variable (même ceux qui ne sont pas représentés dans les données). La fonction `factor` crée un facteur avec les options suivantes :

```
factor(x, levels = sort(unique(x), na.last = TRUE),
       labels = levels, exclude = NA, ordered = is.ordered(x))
```

`levels` spécifie quels sont les niveaux possibles du facteur (par défaut les valeurs uniques du vecteur `x`), `labels` définit les noms des niveaux, `exclude` les valeurs de `x` à ne pas inclure dans les niveaux, et `ordered` est un argument logique spécifiant si les niveaux du facteur sont ordonnés. Rappelons que `x` est de mode numérique ou caractère. En guise d'exemples :

```
> factor(1:3)
[1] 1 2 3
Levels:  1 2 3
> factor(1:3, levels=1:5)
[1] 1 2 3
Levels:  1 2 3 4 5
> factor(1:3, labels=c("A", "B", "C"))
[1] A B C
Levels:  A B C
> factor(1:5, exclude=4)
[1] 1  2  3  NA 5
Levels:  1 2 3 5
```

La fonction `levels` sert à extraire les niveaux possibles d'un facteur :

```
> ff <- factor(c(2, 4), levels=2:5)
> ff
[1] 2 4
Levels:  2 3 4 5
> levels(ff)
[1] "2" "3" "4" "5"
```

Matrice. Une matrice est en fait un vecteur qui possède un argument supplémentaire (dim) qui est lui-même un vecteur numérique de longueur 2 et qui définit les nombres de lignes et de colonnes de la matrice. Une matrice peut être créée avec la fonction `matrix` :

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

L'option `byrow` indique si les valeurs données par `data` doivent remplir successivement les colonnes (le défaut) ou les lignes (si `TRUE`). L'option `dimnames` permet de donner des noms aux lignes et colonnes.

```
> matrix(data=5, nr=2, nc=2)
     [,1] [,2]
[1,]    5    5
[2,]    5    5
> matrix(1:6, 2, 3)
     [,1] [,2] [,3]
```

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

Une autre façon de créer une matrice est de donner les valeurs voulues à l'attribut dim d'un vecteur (attribut qui est initialement `NULL`) :

```
> x <- 1:15
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> dim(x)
NULL
> dim(x) <- c(5, 3)
> x
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

Tableau de données. On a vu qu'un tableau de données est créé de façon implicite par la fonction `read.table`; on peut également créer un tableau de données avec la fonction `data.frame`. Les vecteurs inclus dans le tableau doivent être de même longueur, ou si un de ces éléments est plus court il est alors « recyclé » un nombre entier de fois :

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x, n)
  x  n
1 1 10
2 2 10
3 3 10
4 4 10
> data.frame(x, M)
  x  M
1 1 10
2 2 35
3 3 10
4 4 35
> data.frame(x, y)
Error in data.frame(x, y) :
    arguments imply differing number of rows: 4, 3
```

Si un facteur est inclus dans le tableau de données, il doit être de même longueur que le(s) vecteur(s). Il est possible de changer les noms des

colonnes avec, par exemple, `data.frame(A1=x, A2=n)`. On peut aussi donner des noms aux lignes avec l'option `row.names` qui doit, bien sûr, être un vecteur de mode caractère et de longueur égale au nombre de lignes du tableau de données. Enfin, notons que les tableaux de données ont un attribut dim de la même façon que les matrices.

Liste. Une liste est créée de la même façon qu'un tableau de données avec la fonction `list`. Il n'y a aucune contrainte sur les objets qui y sont inclus. À la différence de `data.frame()`, les noms des objets ne sont pas repris par défaut ; en reprenant les vecteurs `x` et `y` de l'exemple précédant :

```
> L1 <- list(x, y); L2 <- list(A=x, B=y)
> L1
[[1]]
[1] 1 2 3 4

[[2]]
[1] 2 3 4

> L2
$A
[1] 1 2 3 4

$B
[1] 2 3 4

> names(L1)
NULL
> names(L2)
[1] "A" "B"
```

Série temporelle. La fonction `ts` va créer un objet de classe `"ts"` à partir d'un vecteur (série temporelle simple) ou d'une matrice (série temporelle multiple), et des options qui caractérisent la série. Les options, avec les valeurs par défaut, sont :

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
   deltat = 1, ts.eps = getOption("ts.eps"), class, names)
```

`data`	un vecteur ou une matrice
`start`	le temps de la 1ère observation, soit un nombre, ou soit un vecteur de deux entiers (*cf.* les exemples ci-dessous)
`end`	le temps de la dernière observation spécifié de la même façon que `start`
`frequency`	nombre d'observations par unité de temps
`deltat`	la fraction de la période d'échantillonnage entre observations successives (ex. `1/12` pour des données mensuelles) ; seulement un de `frequency` ou `deltat` doit être précisé

`ts.eps` tolérance pour la comparaison de séries. Les fréquences sont considérées égales si leur différence est inférieure à `ts.eps`

`class` classe à donner à l'objet ; le défaut est `"ts"` pour une série simple, et `c("mts", "ts")` pour une série multiple

`names` un vecteur de mode caractère avec les noms des séries individuelles dans le cas d'une série multiple ; par défaut les noms des colonnes de `data`, ou `Series 1`, `Series 2`, etc.

Quelques exemples de création de séries temporelles avec `ts` :

```
> ts(1:10, start = 1959)
Time Series:
Start = 1959
End = 1968
Frequency = 1
 [1]  1  2  3  4  5  6  7  8  9 10
> ts(1:47, frequency = 12, start = c(1959, 2))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1959       1   2   3   4   5   6   7   8   9  10  11
1960  12  13  14  15  16  17  18  19  20  21  22  23
1961  24  25  26  27  28  29  30  31  32  33  34  35
1962  36  37  38  39  40  41  42  43  44  45  46  47
> ts(1:10, frequency = 4, start = c(1959, 2))
     Qtr1 Qtr2 Qtr3 Qtr4
1959         1    2    3
1960    4    5    6    7
1961    8    9   10
> ts(matrix(rpois(36, 5), 12, 3), start=c(1961, 1), frequency=12)
         Series 1 Series 2 Series 3
Jan 1961        8        5        4
Feb 1961        6        6        9
Mar 1961        2        3        3
Apr 1961        8        5        4
May 1961        4        9        3
Jun 1961        4        6       13
Jul 1961        4        2        6
Aug 1961       11        6        4
Sep 1961        6        5        7
Oct 1961        6        5        7
Nov 1961        5        5        7
Dec 1961        8        5        2
```

Expression. Les objets de mode expression ont un rôle fondamental dans R. Une expression est une suite de caractères qui ont un sens pour R. Toutes les commandes valides sont des expressions. Lorsque la commande est

tapée directement au clavier, elle est alors *évaluée* par R qui l'exécute si elle est valide. Dans bien des circonstances, il est utile de construire une expression sans l'évaluer : c'est le rôle de la fonction `expression`. On pourra, bien sûr, évaluer l'expression ultérieurement avec `eval()`.

```
> x <- 3; y <- 2.5; z <- 1
> exp1 <- expression(x / (y + exp(z)))
> exp1
expression(x/(y + exp(z)))
> eval(exp1)
[1] 0.5749019
```

Les expressions servent aussi, entre autres, à inclure des équations sur les graphiques (p. 44). Une expression peut être créée à partir d'une variable de mode caractère. Certaines fonctions utilisent des expressions en tant qu'argument, par exemple `D` qui calcule des dérivées partielles :

```
> D(exp1, "x")
1/(y + exp(z))
> D(exp1, "y")
-x/(y + exp(z))^2
> D(exp1, "z")
-x * exp(z)/(y + exp(z))^2
```

3.5.2 Conversion d'objets

Le lecteur aura sûrement réalisé que les différences entre certains objets sont parfois minces ; il est donc logique de pouvoir convertir un objet en un autre en changeant certains de ces attributs. Une telle conversion sera effectuée avec une fonction du genre `as.`*`something`*. R (version 2.1.0) comporte, dans les packages base et utils, 98 de ces fonctions, aussi nous ne rentrerons pas dans les détails ici.

Le résultat d'une conversion dépend bien sûr des attributs de l'objet converti. En général, la conversion suit des règles intuitives. Pour les conversions de modes, le tableau suivant résume la situation.

Conversion en	Fonction	Règles
numérique	`as.numeric`	`FALSE` → 0
		`TRUE` → 1
		`"1"`, `"2"`, ... → 1, 2, ...
		`"A"`, ... → `NA`
logique	`as.logical`	0 → `FALSE`
		autres nombres → `TRUE`
		`"FALSE"`, `"F"` → `FALSE`
		`"TRUE"`, `"T"` → `TRUE`
		autres caractères → `NA`
caractère	`as.character`	1, 2, ... → `"1"`, `"2"`, ...
		`FALSE` → `"FALSE"`
		`TRUE` → `"TRUE"`

Il existe des fonctions pour convertir les types d'objets (`as.matrix`, `as.ts`, `as.data.frame`, `as.expression`, ...). Ces fonctions vont agir sur des attributs autres que le mode pour la conversion. Là encore les résultats sont généralement intuitifs. Une situation fréquemment rencontrée est la conversion de facteur en vecteur numérique. Dans ce cas, R convertit avec le codage numérique des niveaux du facteur :

```
> fac <- factor(c(1, 10))
> fac
[1] 1 10
Levels:  1 10
> as.numeric(fac)
[1] 1 2
```

Cela est logique si l'on considère un facteur de mode caractère :

```
> fac2 <- factor(c("Male", "Female"))
> fac2
[1] Male   Female
Levels: Female Male
> as.numeric(fac2)
[1] 2 1
```

Notez que le résultat n'est pas `NA` comme on aurait pu s'attendre d'après le tableau ci-dessus.

Pour convertir un facteur de mode numérique en conservant les niveaux tels qu'ils sont spécifiés, on convertira d'abord en caractère puis en numérique.

```
> as.numeric(as.character(fac))
[1] 1 10
```

Cette procédure est très utile si, dans un fichier, une variable numérique contient (pour une raison ou une autre) également des valeurs non-numériques. On a vu que `read.table()` dans ce genre de situation va, par défaut, lire cette colonne comme un facteur.

3.5.3 Les opérateurs

Nous avons vu précédemment qu'il y a trois principaux types d'opérateurs dans R[10]. En voici la liste.

Opérateurs					
Arithmétique		Comparaison		Logique	
`+`	addition	`<`	inférieur à	`! x`	NON logique
`-`	soustraction	`>`	supérieur à	`x & y`	ET logique
`*`	multiplication	`<=`	inférieur ou égal à	`x && y`	idem
`/`	division	`>=`	supérieur ou égal à	`x | y`	OU logique
`^`	puissance	`==`	égal	`x || y`	idem
`%%`	modulo	`!=`	différent	`xor(x, y)`	OU exclusif
`%/%`	division entière				

Les opérateurs arithmétiques ou de comparaison agissent sur deux éléments (`x + y`, `a < b`). Les opérateurs arithmétiques agissent non seulement sur les variables de mode numérique ou complexe, mais aussi sur celles de mode logique ; dans ce dernier cas, les valeurs logiques sont converties en valeurs numériques. Les opérateurs de comparaison peuvent s'appliquer à n'importe quel mode : ils retournent une ou plusieurs valeurs logiques.

Les opérateurs logiques s'appliquent à un (`!`) ou deux objets de mode logique et retournent une (ou plusieurs) valeurs logiques. Les opérateurs « ET » et « OU » existent sous deux formes : la forme simple opére sur chaque élément des objets et retourne autant de valeurs logiques que de comparaisons effectuées ; la forme double opére sur le premier élément des objets.

On utilisera l'opérateur « ET » pour spécifier une inégalité du type $0 < x < 1$ qui sera codée ainsi : `0 < x & x < 1`. L'expression `0 < x < 1` est valide mais ne donnera pas le résultat escompté : les deux opérateurs de cette expression étant identiques, ils seront exécutés successivement de la gauche vers la droite. L'opération `0 < x` sera d'abord réalisée retournant une valeur logique qui sera ensuite comparée à 1 (`TRUE` ou `FALSE < 1`) : dans ce cas la valeur logique sera convertie implicitement en numérique (`1` ou `0 < 1`).

```
> x <- 0.5
> 0 < x < 1
[1] FALSE
```

Les opérateurs de comparaison opèrent sur *chaque* élément des deux objets qui sont comparés (en recyclant éventuellement les valeurs si l'un est plus court), et retournent donc un objet de même taille. Pour effectuer une comparaison « globale » de deux objets, deux fonctions sont disponibles : `identical` et `all.equal`.

[10] Les caractères suivants sont en fait aussi des opérateurs pour R : `$`, `@`, `[`, `[[`, `:`, `?`, `<-`, `<<-`, `=`, `::`. Un tableau des opérateurs décrivant les règles de priorité peut être trouvé avec `?Syntax`.

```
> x <- 1:3; y <- 1:3
> x == y
[1] TRUE TRUE TRUE
> identical(x, y)
[1] TRUE
> all.equal(x, y)
[1] TRUE
```

`identical` compare la représentation interne des données et retourne `TRUE` si les objets sont strictement identiques, sinon `FALSE`. `all.equal` compare « l'égalité approximative » des deux objets, et retourne `TRUE` ou affiche un résumé des différences. Cette dernière fonction prend en compte l'approximation des calculs dans la comparaison des valeurs numériques. La comparaison de valeurs numériques sur un ordinateur est parfois surprenante !

```
> 0.9 == (1 - 0.1)
[1] TRUE
> identical(0.9, 1 - 0.1)
[1] TRUE
> all.equal(0.9, 1 - 0.1)
[1] TRUE
> 0.9 == (1.1 - 0.2)
[1] FALSE
> identical(0.9, 1.1 - 0.2)
[1] FALSE
> all.equal(0.9, 1.1 - 0.2)
[1] TRUE
> all.equal(0.9, 1.1 - 0.2, tolerance = 1e-16)
[1] "Mean relative  difference: 1.233581e-16"
```

3.5.4 Accéder aux valeurs d'un objet : le système d'indexation

L'indexation est un moyen efficace et flexible d'accéder de façon sélective aux éléments d'un objet ; elle peut être *numérique* ou *logique*. Pour accéder à, par exemple, la $3^{\text{ème}}$ valeur d'un vecteur `x`, on tape `x[3]` qui peut être utilisé aussi bien pour extraire ou changer cette valeur :

```
> x <- 1:5
> x[3]
[1] 3
> x[3] <- 20
> x
[1]  1  2 20  4  5
```

L'indice lui-même peut être un vecteur de mode numérique :

```
> i <- c(1, 3)
```

```
> x[i]
[1]  1 20
```

Si `x` est une matrice ou un tableau de données, on accédera à la valeur de la i$^{\text{ème}}$ ligne et j$^{\text{ème}}$ colonne par `x[i, j]`. Pour accéder à toutes les valeurs d'une ligne ou d'une colonne donnée, il suffit simplement d'omettre l'indice approprié (sans oublier la virgule !) :

```
> x <- matrix(1:6, 2, 3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[, 3] <- 21:22
> x
     [,1] [,2] [,3]
[1,]    1    3   21
[2,]    2    4   22
> x[, 3]
[1] 21 22
```

Vous avez certainement noté que le dernier résultat est un vecteur et non une matrice. Par défaut, R retourne un objet de la plus petite dimension possible. Ceci peut être modifié avec l'option `drop` dont le défaut est `TRUE` :

```
> x[, 3, drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

Ce système d'indexation se généralise facilement pour les tableaux, on aura alors autant d'indices que le tableau a de dimensions (par exemple pour une tableau à trois dimensions : `x[i, j, k]`, `x[, , 3]`,`x[, , 3, drop = FALSE]`, etc). Il peut être utile de se souvenir que l'indexation se fait à l'aide de crochets, les parenthèses étant réservées pour les arguments d'une fonction :

```
> x(1)
Error: couldn't find function "x"
```

L'indexation peut aussi être utilisée pour supprimer une ou plusieurs lignes ou colonnes en utilisant des valeurs négatives. Par exemple, `x[-1, ]` supprimera la 1$^{\text{ère}}$ ligne, ou `x[-c(1, 15), ]` fera de même avec les 1$^{\text{ère}}$ et 15$^{\text{ème}}$ lignes. En utilisant la matrice définies ci-dessus :

```
> x[, -1]
     [,1] [,2]
[1,]    3   21
```

```
[2,]    4   22
> x[, -(1:2)]
[1] 21 22
> x[, -(1:2), drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

Pour les vecteurs, matrices et tableaux il est possible d'accéder aux valeurs de ces éléments à l'aide d'une expression de comparaison en guise d'indice :

```
> x <- 1:10
> x[x >= 5] <- 20
> x
 [1]  1  2  3  4 20 20 20 20 20 20
> x[x == 1] <- 25
> x
 [1] 25  2  3  4 20 20 20 20 20 20
```

Une utilisation pratique de cette indexation logique est, par exemple, la possibilité de sélectionner les éléments pairs d'une variable entière :

```
> x <- rpois(40, lambda=5)
> x
 [1]  5  9  4  7  7  6  4  5 11  3  5  7  1  5  3  9  2  2  5  2
[21]  4  6  6  5  4  5  3  4  3  3  3  7  7  3  8  1  4  2  1  4
> x[x %% 2 == 0]
 [1] 4 6 4 2 2 2 4 6 6 4 4 8 4 2 4
```

Ce système d'indexation utilise donc des valeurs logiques retournées dans ce cas par les opérateurs de comparaison. Ces valeurs logiques peuvent être calculées au préalable, elles seront éventuellement recyclées :

```
> x <- 1:40
> s <- c(FALSE, TRUE)
> x[s]
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

L'indexation logique peut également être utilisée avec des tableaux de données, mais avec la difficulté que les différentes colonnes peuvent être de modes différents.

Pour les listes, l'accès aux différents éléments (qui peuvent être n'importe quel objet) se fait avec des crochets simples ou doubles : la différence étant qu'avec les crochets simples une liste est retournée, alors qu'avec les crochets doubles *extraient* l'objet de la liste. Par exemple, si le $3^{\text{ème}}$ élément d'une liste est un vecteur, le $i^{\text{ème}}$ élément de ce vecteur peut être accédé avec `my.list[[3]][i]`, ou bien avec `my.list[[3]][i, j, k]` s'il s'agit d'un tableau à trois dimensions, etc. Une autre différence est que `my.list[1:2]` retournera une liste avec le premier et le second élément de la liste originale, alors que `my.list[[1:2]]` ne donnera pas le résultat escompté.

3.5.5 Accéder aux valeurs d'un objet avec les noms

Les *noms* sont les étiquettes des éléments d'un objet, et sont donc de mode caractère. Ce sont généralement des attributs optionnels ; il en existe plusieurs sortes (*names*, *colnames*, *rownames*, *dimnames*).

Les *noms* d'un vecteur sont stockés dans un vecteur de même longueur, et peuvent accédés avec la fonction `names`.

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("a", "b", "c")
> x
a b c
1 2 3
> names(x)
[1] "a" "b" "c"
> names(x) <- NULL
> x
[1] 1 2 3
```

Pour les matrices et les tableaux de données, *colnames* and *rownames* sont les étiquettes des lignes et des colonnes. Elles peuvent être accédées avec leurs fonctions respectives, ou avec `dimnames` qui retourne une liste avec les deux vecteurs.

```
> X <- matrix(1:4, 2)
> rownames(X) <- c("a", "b")
> colnames(X) <- c("c", "d")
> X
  c d
a 1 3
b 2 4
> dimnames(X)
[[1]]
[1] "a" "b"

[[2]]
[1] "c" "d"
```

Pour les tableaux, les noms des dimensions peuvent être accédés avec `dimnames`.

```
> A <- array(1:8, dim = c(2, 2, 2))
> A
, , 1
```

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

> dimnames(A) <- list(c("a", "b"), c("c", "d"), c("e", "f"))
> A
, , e

  c d
a 1 3
b 2 4

, , f

  c d
a 5 7
b 6 8
```

Si les éléments d'un objet ont des noms, ils peuvent être extraits en les utilisant en guise d'indices. En fait, cela doit être appelè *subdivision* (*subsetting*) plutôt qu'extraction car les attributs de l'objet d'origine sont conservés. Par exemple, si un tableau de données `DF` comporte les variables `x`, `y`, et `z`, la commande `DF["x"]` donnera un tableau de données avec juste `x`; `DF[c("x", "y")]` donnera un tableau de données avec les deux variables correspondantes. Ce système marche aussi avec une liste si ses éléments ont des noms.

Comme on le constate, l'index ainsi utilisé est un vecteur de mode caractère. Comme pour les vecteurs logiques ou numériques vus précédemment, ce vecteur peut être établi au préalable et ensuite inséré pour l'extraction.

Pour extraire un vecteur ou un facteur d'un tableau de données on utilisera l'opérateur `$` (par exemple `DF$x`). Cela marche également avec les listes.

3.5.6 L'éditeur de données

Il est possible d'utiliser un éditeur graphique de style tableur pour éditer un objet contenant des données. Par exemple, si on a une matrice `X`, la commande `data.entry(X)` ouvrira l'éditeur graphique et l'on pourra modifier les valeurs en cliquant sur les cases correspondantes ou encore ajouter des colonnes ou des lignes.

La fonction `data.entry` modifie directement l'objet passé en argument sans avoir à assigner son résultat. Par contre la fonction `de` retourne une

liste composée des objets passés en arguments et éventuellement modifiés. Ce résultat est affiché à l'écran par défaut mais, comme pour la plupart des fonctions, peut être assigné dans un objet.

Les détails de l'utilisation de cet éditeur de données dépendent du système d'exploitation.

3.5.7 Calcul arithmétique et fonctions simples

Il existe de nombreuses fonctions dans R pour manipuler les données. La plus simple, on l'a vue plus haut, est `c` qui concatène les objets énumérés entre parenthèses. Par exemple :

```
> c(1:5, seq(10, 11, 0.2))
 [1]  1.0  2.0  3.0  4.0  5.0 10.0 10.2 10.4 10.6 10.8 11.0
```

Les vecteurs peuvent être manipulés selon des expressions arithmétiques classiques :

```
> x <- 1:4
> y <- rep(1, 4)
> z <- x + y
> z
[1] 2 3 4 5
```

Des vecteurs de longueurs différentes peuvent être additionnés, dans ce cas le vecteur le plus court est recyclé. Exemples :

```
> x <- 1:4
> y <- 1:2
> z <- x + y
> z
[1] 2 4 4 6
> x <- 1:3
> y <- 1:2
> z <- x + y
Warning message:
longer object length
 is not a multiple of shorter object length in: x + y
> z
[1] 2 4 4
```

On notera que R a retourné un message d'avertissement et non pas un message d'erreur, l'opération a donc été effectuée. Si l'on veut ajouter (ou multiplier) la même valeur à tous les éléments d'un vecteur :

```
> x <- 1:4
> a <- 10
```

```
> z <- a * x
> z
[1] 10 20 30 40
```

Les fonctions disponibles dans R pour les manipulations de données sont trop nombeuses pour être énumérées ici. On trouve toutes les fonctions mathématiques de base (`log`, `exp`, `log10`, `log2`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `abs`, `sqrt`, ...), des fonctions spéciales (`gamma`, `digamma`, `beta`, `besselI`, ...), ainsi que diverses fonctions utiles en statistiques. Quelques-unes sont indiquées dans le tableau qui suit.

`sum(x)`	somme des éléments de `x`
`prod(x)`	produit des éléments de `x`
`max(x)`	maximum des éléments de `x`
`min(x)`	minimum des éléments de `x`
`which.max(x)`	retourne l'indice du maximum des éléments de `x`
`which.min(x)`	retourne l'indice du minimum des éléments de `x`
`range(x)`	idem que `c(min(x), max(x))`
`length(x)`	nombre d'éléments dans `x`
`mean(x)`	moyenne des éléments de `x`
`median(x)`	médiane des éléments de `x`
`var(x)` ou `cov(x)`	variance des éléments de `x` (calculée sur $n-1$) ; si `x` est une matrice ou un tableau de données, la matrice de variance-covariance est calculée
`cor(x)`	matrice de corrélation si `x` est une matrice ou un tableau de données (1 si `x` est un vecteur)
`var(x, y)` ou `cov(x, y)`	covariance entre `x` et `y`, ou entre les colonnes de `x` et de `y` si ce sont des matrices ou des tableaux de données
`cor(x, y)`	corrélation linéaire entre `x` et `y`, ou matrice de corrélations si ce sont des matrices ou des tableaux de données

Ces fonctions retournent une valeur simple (donc un vecteur de longueur 1), sauf `range` qui retourne un vecteur de longueur 2, et `var`, `cov` et `cor` qui peuvent retourner une matrice. Les fonctions suivantes retournent des résultats plus complexes.

`round(x, n)`	arrondit les éléments de `x` à `n` chiffres après la virgule
`rev(x)`	inverse l'ordre des éléments de `x`
`sort(x)`	trie les éléments de `x` dans l'ordre ascendant ; pour trier dans l'ordre descendant : `rev(sort(x))`
`rank(x)`	rangs des éléments de `x`
`log(x, base)`	calcule le logarithme à base `base` de `x`
`scale(x)`	si `x` est une matrice, centre et réduit les données ; pour centrer uniquement ajouter l'option `center=FALSE`, pour réduire uniquement `scale=FALSE` (par défaut `center=TRUE, scale=TRUE`)
`pmin(x,y,...)`	un vecteur dont le $i^{\text{ème}}$ élément est le minimum entre `x[i]`, `y[i]`, ...

`pmax(x,y,...)`	idem pour le maximum
`cumsum(x)`	un vecteur dont le i$^{\text{ème}}$ élément est la somme de `x[1]` à `x[i]`
`cumprod(x)`	idem pour le produit
`cummin(x)`	idem pour le minimum
`cummax(x)`	idem pour le maximum
`match(x, y)`	retourne un vecteur de même longueur que `x` contenant les éléments de `x` qui sont dans `y` (`NA` sinon)
`which(x == a)`	retourne un vecteur des indices de `x` pour lesquels l'opération de comparaison est vraie (`TRUE`), dans cet exemple les valeurs de `i` telles que `x[i] == a` (l'argument de cette fonction doit être une variable de mode logique)
`choose(n, k)`	calcule les combinaisons de k événements parmi n répétitions $= n!/[(n-k)!k!]$
`na.omit(x)`	supprime les observations avec données manquantes (`NA`) (supprime la ligne correspondante si `x` est une matrice ou un tableau de données)
`na.fail(x)`	retourne un message d'erreur si `x` contient au moins un `NA`
`unique(x)`	si `x` est un vecteur ou un tableau de données, retourne un objet similaire mais avec les éléments dupliqués supprimés
`table(x)`	retourne un tableau des effectifs des différentes valeurs de `x` (typiquement pour des entiers ou des facteurs)
`table(x, y)`	tableau de contingence de `x` et `y`
`subset(x, ...)`	retourne une sélection de `x` en fonction de critères (`...`, typiquement des comparaisons : `x$V1 < 10`) ; si `x` est un tableau de données, l'option `select` permet de préciser les variables à sélectionner (ou à éliminer à l'aide du signe moins)
`sample(x, size)`	ré-échantillonne aléatoirement et sans remise `size` éléments dans le vecteur `x`, pour ré-échantillonner avec remise on ajoute l'option `replace = TRUE`

3.5.8 Calcul matriciel

R offre des facilités pour le calcul et la manipulation de matrices. Les fonctions `rbind` et `cbind` juxtaposent des matrices en conservant les lignes ou les colonnes, respectivement :

```
> m1 <- matrix(1, nr = 2, nc = 2)
> m2 <- matrix(2, nr = 2, nc = 2)
> rbind(m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
> cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    2
[2,]    1    1    2    2
```

L'opérateur pour le produit de deux matrices est '`%*%`'. Par exemple, en reprenant les deux matrices `m1` et `m2` ci-dessus :

```
> rbind(m1, m2) %*% cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    2    2    4    4
[2,]    2    2    4    4
[3,]    4    4    8    8
[4,]    4    4    8    8
> cbind(m1, m2) %*% rbind(m1, m2)
     [,1] [,2]
[1,]   10   10
[2,]   10   10
```

La transposition d'une matrice se fait avec la fonction `t` ; cette fonction marche aussi avec un tableau de données.

La fonction `diag` sert à extraire, modifier la diagonale d'une matrice, ou encore à construire une matrice diagonale.

```
> diag(m1)
[1] 1 1
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
     [,1] [,2]
[1,]   10    1
[2,]    1   10
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> v <- c(10, 20, 30)
> diag(v)
     [,1] [,2] [,3]
[1,]   10    0    0
[2,]    0   20    0
[3,]    0    0   30
> diag(2.1, nr = 3, nc = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.1  0.0  0.0    0    0
[2,]  0.0  2.1  0.0    0    0
[3,]  0.0  0.0  2.1    0    0
```

R a également des fonctions spéciales pour le calcul matriciel. Citons `solve` pour l'inversion d'une matrice, `qr` pour la décomposition, `eigen` pour le cal-

cul des valeurs et vecteurs propres, et `svd` pour la décomposition en valeurs singulières.

4 Les graphiques avec R

R offre une variété de graphiques remarquable. Pour avoir une petite idée des possibilités offertes, il suffit de taper la commande `demo(graphics)` ou `demo(persp)`. Il n'est pas possible ici de détailler toutes les possibilités ainsi offertes, en particulier chaque fonction graphique a beaucoup d'options qui rendent la production de graphiques extrêment flexible.

Le fonctionnement des fonctions graphiques dévie substantiellement du schéma dressé au début de ce document. Notamment, le résultat d'une fonction graphique ne peut pas être assigné à un objet[11] mais est envoyé à un *périphérique graphique* (*graphical device*). Un périphérique graphique est matérialisé par une fenêtre graphique ou un fichier.

Il existe deux sortes de fonctions graphiques : *principales* qui créent un nouveau graphe, et *secondaires* qui ajoutent des éléments à un graphe déjà existant. Les graphes sont produits en fonction de *paramètres graphiques* qui sont définis par défaut et peuvent être modifiés avec la fonction `par`.

Nous allons dans un premier temps voir comment gérer les graphiques, ensuite nous détaillerons les fonctions et paramètres graphiques. Nous verrons un exemple concret de l'utilisation de ces fonctionnalités pour la production de graphes. Enfin, nous verrons les packages grid et lattice dont le fonctionnement est différent de celui résumé ci-dessus.

4.1 Gestion des graphiques

4.1.1 Ouvrir plusieurs dispositifs graphiques

Lorsqu'une fonction graphique est exécutée, si aucun périphérique graphique n'est alors ouvert, R ouvrira une fenêtre graphique et y affichera le graphe. Un périphérique graphique peut être ouvert avec une fonction appropriée. La liste des périphériques graphiques disponibles dépend du système d'exploitation. Les fenêtres graphiques sont nommées `X11` sous Unix/Linux et `windows` sous Windows. Dans tous les cas, on peut ouvrir une fenêtre avec la commande `x11()` qui marche même sous Windows grâce à un alias vers la commande `windows()`. Un périphérique graphique de type fichier sera ouvert avec une fonction qui dépend du format : `postscript()`, `pdf()`, `png()`, ... Pour connaître la liste des périphériques disponibles pour votre installation, tapez `?device`.

Le dernier périphérique ouvert devient le périphérique graphique actif sur lequel seront affichés les graphes suivants. La fonction `dev.list()` affiche la liste des périphériques ouverts :

[11] Il y a quelques exceptions notables : `hist()` et `barplot()` produisent également des résultats numériques sous forme de liste ou de matrice.

```
> x11(); x11(); pdf()
> dev.list()
X11 X11 pdf
  2   3   4
```

Les chiffres qui s'affichent correspondent aux numéros des périphériques qui doivent être utilisés si l'on veut changer le périphérique actif. Pour connaître le périphérique actif :

```
> dev.cur()
pdf
  4
```

et pour changer le périphérique actif :

```
> dev.set(3)
X11
  3
```

La fonction `dev.off()` ferme un périphérique graphique : par défaut le périphérique actif est fermé sinon c'est celui dont le numéro est donné comme argument à la fonction. R affiche le numéro du périphérique actif :

```
> dev.off(2)
X11
  3
> dev.off()
pdf
  4
```

Deux spécificités de la version Windows de R sont à signaler : la fonction `win.metafile` qui accède à un fichier au format Windows Metafile, et un menu « History » affiché lorsque la fenêtre graphique est sélectionnée qui permet d'enregistrer tous les graphes produits au cours d'une session (par défaut l'enregistrement n'est pas activé, l'utilisateur l'active en cliquant sur « Enregistrer » dans ce menu).

4.1.2 Partitionner un graphique

La fonction `split.screen` partitionne le graphique actif. Par exemple :

```
> split.screen(c(1, 2))
```

va diviser le graphique en deux parties qu'on sélectionnera avec `screen(1)` ou `screen(2)` ; `erase.screen()` efface le graphe dernièrement dessiné. Une partie peut être elle-même divisée avec `split.screen()` donnant la possibilité de faire des arrangements complexes.

Ces fonctions sont incompatibles avec d'autres (tel `layout` ou `coplot`) et ne doivent pas être utilisées avec des périphériques graphiques multiples. Leur

utilisation doit donc être limitée par exemple pour l'exploration visuelle de données.

La fonction `layout` partitionne le graphique actif en plusieurs parties sur lesquelles sont affichés les graphes successivement ; son argument principal est une matrice avec des valeurs entières qui indiquent les numéros des sous-fenêtres. Par exemple, si l'on veut diviser la fenêtre en quatre parties égales :

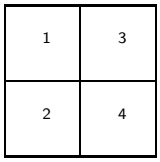
```
> layout(matrix(1:4, 2, 2))
```

On pourra bien sûr créer cette matrice au préalable ce qui permettra de mieux voir comment est divisé le graphique :

```
> mat <- matrix(1:4, 2, 2)
> mat
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> layout(mat)
```

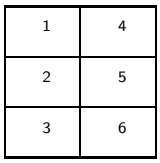
Pour visualiser concrètement la partition créée, on utilisera la fonction `layout.show` avec en argument le nombre de sous-fenêtres (ici 4). Avec cet exemple on aura :

```
> layout.show(4)
```

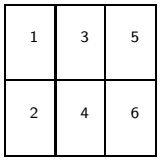


Les exemples qui suivent montrent certaines des possibilités ainsi offertes.

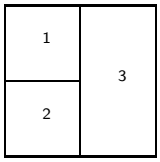
```
> layout(matrix(1:6, 3, 2))
> layout.show(6)
```




```
> layout(matrix(1:6, 2, 3))
> layout.show(6)
```



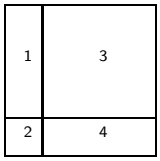
```
> m <- matrix(c(1:3, 3), 2, 2)
> layout(m)
> layout.show(3)
```



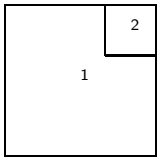
Dans tous ces exemples, nous n'avons pas utilisé l'option `byrow` de `matrix`, les sous-fenêtres sont donc numérotées par colonne ; il suffit bien sûr de spécifier `matrix(..., byrow = TRUE)` pour que les sous-fenêtres soient numérotées par ligne. On peut aussi donner les numéros dans la matrice dans l'ordre que l'on veut avec, par exemple, `matrix(c(2, 1, 4, 3), 2, 2)`.

Par défaut, `layout()` va partitionner le graphique avec des hauteurs et largeurs régulières : ceci peut être modifié avec les options `widths` et `heights`. Ces dimensions sont données relativement[12]. Exemples :

```
> m <- matrix(1:4, 2, 2)
> layout(m, widths=c(1, 3),
          heights=c(3, 1))
> layout.show(4)
```

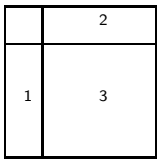



```
> m <- matrix(c(1,1,2,1),2,2)
> layout(m, widths=c(2, 1),
          heights=c(1, 2))
> layout.show(2)
```



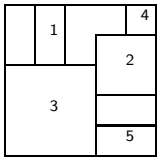
Enfin, les numéros dans la matrice peuvent inclure des 0 donnant la possibilité de construire des partitions complexes (voire ésotériques).

```
> m <- matrix(0:3, 2, 2)
> layout(m, c(1, 3), c(1, 3))
> layout.show(3)
```



[12] Elles peuvent aussi être données en centimètres, *cf.* `?layout`.

```
> m <- matrix(scan(), 5, 5)
1: 0 0 3 3 3 1 1 3 3 3
11: 0 0 3 3 3 0 2 2 0 5
21: 4 2 2 0 5
26:
Read 25 items
> layout(m)
> layout.show(5)
```



4.2 Les fonctions graphiques

Voici un aperçu des fonctions graphiques principales de R.

`plot(x)`	graphe des valeurs de `x` (sur l'axe des y) ordonnées sur l'axe des x
`plot(x, y)`	graphe bivarié de `x` (sur l'axe des x) et `y` (sur l'axe des y)
`sunflowerplot(x, y)`	idem que `plot()` mais les points superposés sont dessinés en forme de fleurs dont le nombre de pétales représente le nombre de points
`pie(x)`	graphe en camembert
`boxplot(x)`	graphe boites et moustaches
`stripchart(x)`	graphe des valeurs de `x` sur une ligne (une alternative à `boxplot()` pour des petits échantillons)
`coplot(x~y | z)`	graphe bivarié de `x` et `y` pour chaque valeur (ou intervalle de valeurs) de `z`
`interaction.plot (f1, f2, y)`	si `f1` et `f2` sont des facteurs, graphe des moyennes de `y` (sur l'axe des y) en fonction des valeurs de `f1` (sur l'axe des x) et de `f2` (différentes courbes) ; l'option `fun` permet de choisir la statistique résumée de `y` (par défaut `fun=mean`)
`matplot(x,y)`	graphe bivarié de la $1^{\text{ère}}$ colonne de `x` contre la $1^{\text{ère}}$ de `y`, la $2^{\text{ème}}$ de `x` contre la $2^{\text{ème}}$ de `y`, etc.
`dotchart(x)`	si `x` est un tableau de données, dessine un graphe de Cleveland (graphes superposés ligne par ligne et colonne par colonne)
`fourfoldplot(x)`	visualise, avec des quarts de cercles, l'association entre deux variables dichotomiques pour différentes populations (`x` doit être un tableau avec `dim=c(2, 2, k)` ou une matrice avec `dim=c(2, 2)` si $k = 1$)
`assocplot(x)`	graphe de Cohen–Friendly indiquant les déviations de l'hypothèse d'indépendance des lignes et des colonnes dans un tableau de contingence à deux dimensions
`mosaicplot(x)`	graphe en 'mosaïque' des résidus d'une régression log-linéaire sur une table de contingence
`pairs(x)`	si `x` est une matrice ou un tableau de données, dessine tous les graphes bivariés entre les colonnes de `x`
`plot.ts(x)`	si `x` est un objet de classe `"ts"`, graphe de `x` en fonction du temps, `x` peut être multivarié mais les séries doivent avoir les mêmes fréquence et dates

`ts.plot(x)`	idem mais si `x` est multivarié les séries peuvent avoir des dates différentes et doivent avoir la même fréquence
`hist(x)`	histogramme des fréquences de `x`
`barplot(x)`	histogramme des valeurs de `x`
`qqnorm(x)`	quantiles de `x` en fonction des valeurs attendues selon une loi normale
`qqplot(x, y)`	quantiles de `y` en fonction des quantiles de `x`
`contour(x, y, z)`	courbes de niveau (les données sont interpolées pour tracer les courbes), `x` et `y` doivent être des vecteurs et `z` une matrice telle que `dim(z)=c(length(x), length(y))` (`x` et `y` peuvent être omis)
`filled.contour (x, y, z)`	idem mais les aires entre les contours sont colorées, et une légende des couleurs est également dessinée
`image(x, y, z)`	idem mais les données sont représentées avec des couleurs
`persp(x, y, z)`	idem mais en perspective
`stars(x)`	si `x` est une matrice ou un tableau de données, dessine un graphe en segments ou en étoile où chaque ligne de `x` est représentée par une étoile et les colonnes par les longueurs des branches
`symbols(x, y, ...)`	dessine aux coordonnées données par `x` et `y` des symboles (cercles, carrés, rectangles, étoiles, thermomètres ou « box-plots ») dont les tailles, couleurs, etc, sont spécifiées par des arguments supplémentaires
`termplot(mod.obj)`	graphe des effets (partiels) d'un modèle de régression (`mod.obj`)

Pour chaque fonction, les options peuvent être trouvées via l'aide-en-ligne de R. Certaines de ces options sont identiques pour plusieurs fonctions graphiques ; voici les principales (avec leurs éventuelles valeurs par défaut) :

`add=FALSE`	si `TRUE` superpose le graphe au graphe existant (s'il y en a un)
`axes=TRUE`	si `FALSE` ne trace pas les axes ni le cadre
`type="p"`	le type de graphe qui sera dessiné, `"p"` : points, `"l"` : lignes, `"b"` : points connectés par des lignes, `"o"` : idem mais les lignes recouvrent les points, `"h"` : lignes verticales, `"s"` : escaliers, les données étant représentées par le sommet des lignes verticales, `"S"` : idem mais les données étant représentées par le bas des lignes verticales
`xlim=, ylim=`	fixe les limites inférieures et supérieures des axes, par exemple avec `xlim=c(1, 10)` ou `xlim=range(x)`
`xlab=, ylab=`	annotations des axes, doivent être des variables de mode caractère
`main=`	titre principal, doit être une variable de mode caractère
`sub=`	sous-titre (écrit dans une police plus petite)

4.3 Les fonctions graphiques secondaires

Il y a dans R un ensemble de fonctions graphiques qui ont une action sur un graphe déjà existant (ces fonctions sont appelées *low-level plotting commands*

dans le jargon de R, alors que les fonctions précédentes sont nommées *high-level plotting commands*). Voici les principales :

`points(x, y)`	ajoute des points (l'option `type=` peut être utilisée)
`lines(x, y)`	idem mais avec des lignes
`text(x, y, labels, ...)`	ajoute le texte spécifié par `labels` au coordonnées (`x`,`y`) ; un usage typique sera : `plot(x, y, type="n") ; text(x, y, names)`
`mtext(text, side=3, line=0, ...)`	ajoute le texte spécifié par `text` dans la marge spécifiée par `side` (*cf.* `axis()` plus bas) ; `line` spécifie la ligne à partir du cadre de traçage
`segments(x0, y0, x1, y1)`	trace des lignes des points (`x0`,`y0`) aux points (`x1`,`y1`)
`arrows(x0, y0, x1, y1, angle=30, code=2)`	idem avec des flèches aux points (`x0`,`y0`) si `code=2`, aux points (`x1`,`y1`) si `code=1`, ou aux deux si `code=3` ; `angle` contrôle l'angle de la pointe par rapport à l'axe
`abline(a,b)`	trace une ligne de pente `b` et ordonnée à l'origine `a`
`abline(h=y)`	trace une ligne horizontale sur l'ordonnée `y`
`abline(v=x)`	trace une ligne verticale sur l'abcisse `x`
`abline(lm.obj)`	trace la droite de régression donnée par `lm.obj` (*cf.* section 5)
`rect(x1, y1, x2, y2)`	trace un rectangle délimité à gauche par `x1`, à droite par `x2`, en bas par `y1` et en haut par `y2`
`polygon(x, y)`	trace un polygone reliant les points dont les coordonnées sont données par `x` et `y`
`legend(x, y, legend)`	ajoute la légende au point de coordonnées (`x`,`y`) avec les symboles donnés par `legend`
`title()`	ajoute un titre et optionnellement un sous-titre
`axis(side, vect)`	ajoute un axe en bas (`side=1`), à gauche (`2`), en haut (`3`) ou à droite (`4`) ; `vect` (optionnel) indique les abcisses (ou ordonnées) où les graduations seront tracées
`box()`	ajoute un cadre autour du graphe
`rug(x)`	dessine les données `x` sur l'axe des x sous forme de petits traits verticaux
`locator(n, type="n", ...)`	retourne les coordonnées (x, y) après que l'utilisateur ait cliqué `n` fois sur le graphe avec la souris ; également trace des symboles (`type="p"`) ou des lignes (`type="l"`) en fonction de paramètres graphiques optionnels (`...`) ; par défaut ne trace rien (`type="n"`)

À noter la possibilité d'ajouter des expressions mathématiques sur un graphe à l'aide de `text(x, y, expression(...))`, où la fonction `expression` transforme son argument en équation mathématique. Par exemple,

```
> text(x, y, expression(p == over(1, 1+e^-(beta*x+alpha))))
```

va afficher, sur le graphe, l'équation suivante au point de coordonnées (x, y) :

$$\mathrm{p} = \frac{1}{1 + \mathrm{e}^{-(\beta \mathrm{x} + \alpha)}}$$

Pour inclure dans une expression une variable numérique on utilisera les fonctions `substitute` et `as.expression` ; par exemple pour inclure une valeur de R^2 (précédemment calculée et stockée dans un objet nommé `Rsquared`) :

```
> text(x, y, as.expression(substitute(R^2==r, list(r=Rsquared))))
```

qui affichera sur le graphe au point de coordonnées (x, y) :

$$\mathrm{R}^2 = 0.9856298$$

Pour ne conserver que trois chiffres après la virgule on modifiera le code comme suit :

```
> text(x, y, as.expression(substitute(R^2==r,
+                           list(r=round(Rsquared, 3)))))
```

qui affichera :

$$\mathrm{R}^2 = 0.986$$

Enfin, pour obtenir le R en italique :

```
> text(x, y, as.expression(substitute(italic(R)^2==r,
+                           list(r=round(Rsquared, 3)))))
```

$$R^2 = 0.986$$

4.4 Les paramètres graphiques

En plus des fonctions graphiques secondaires, la présentation des graphiques peut être améliorée grâce aux paramètres graphiques. Ceux-ci s'utilisent soit comme des options des fonctions graphiques principales ou secondaires (mais cela ne marche pas pour tous), soit à l'aide de la fonction `par` qui permet d'enregistrer les changements des paramètres graphiques de façon permanente, c'est-à-dire que les graphes suivants seront dessinés en fonction des nouveaux paramètres spécifiés par l'utilisateur. Par exemple, l'instruction suivante :

```
> par(bg="yellow")
```

résultera en un fond jaune pour tous les graphes. Il y a 73 paramètres graphiques, dont certains ont des rôles proches. La liste détaillée peut être obtenue avec `?par` ; je me limite ici à ceux qui sont les plus couramment utilisés.

`adj`	contrôle la justification du texte par rapport au bord gauche du texte : `0` à gauche, `0.5` centré, `1` à droite, les valeurs > 1 déplacent le texte vers la gauche, et les valeurs négatives vers la droite ; si deux valeurs dont données (ex. `c(0, 0)`) la seconde contrôle la justification verticale par rapport à la ligne de base du texte
`bg`	spécifie la couleur de l'arrière-plan (ex. `bg="red"`, `bg="blue"` ; la liste des 657 couleurs disponibles est affichée avec `colors()`)

`bty`	contrôle comment le cadre est tracé, valeurs permises : `"o"`, `"l"`, `"7"`, `"c"`, `"u"` ou `"]"` (le cadre ressemblant au caractère correspondant) ; `bty="n"` supprime le cadre
`cex`	une valeur qui contrôle la taille des caractères et des symboles par rapport au défaut ; les paramètres suivants ont le même contrôle pour les nombres sur les axes, `cex.axis`, les annotations des axes, `cex.lab`, le titre, `cex.main`, le sous-titre, `cex.sub`
`col`	contrôle la couleur des symboles ; comme pour `cex` il y a : `col.axis`, `col.lab`, `col.main`, `col.sub`
`font`	un entier qui contrôle le style du texte (`1` : normal, `2` : italique, `3` : gras, `4` : gras italique) ; comme pour `cex` il y a : `font.axis`, `font.lab`, `font.main`, `font.sub`
`las`	un entier qui contrôle comment sont disposées les annotations des axes (`0` : parallèles aux axes, `1` : horizontales, `2` : perpendiculaires aux axes, `3` : verticales)
`lty`	contrôle le type de ligne tracée, peut être un entier (`1` : continue, `2` : tirets, `3` : points, `4` : points et tirets alternés, `5` : tirets longs, `6` : tirets courts et longs alternés), ou un ensemble de 8 caractères maximum (entre `"0"` et `"9"`) qui spécifie alternativement la longueur, en points ou pixels, des éléments tracés et des blancs, par exemple `lty="44"` aura le même effet que `lty=2`
`lwd`	une valeur numérique qui contrôle la largeur des lignes
`mar`	un vecteur de 4 valeurs numériques qui contrôle l'espace entre les axes et le bord de la figure de la forme `c(bas, gauche, haut, droit)`, les valeurs par défaut sont `c(5.1, 4.1, 4.1, 2.1)`
`mfcol`	un vecteur de forme `c(nr,nc)` qui partitionne la fenêtre graphique en une matrice de `nr` lignes et `nc` colonnes, les graphes sont ensuite dessinés en colonne (*cf.* section 4.1.2)
`mfrow`	idem mais les graphes sont ensuite dessinés en ligne (*cf.* section 4.1.2)
`pch`	contrôle le type de symbole, soit un entier entre 1 et 25, soit n'importe quel caractère entre guillements (Fig. 2)
`ps`	un entier qui contrôle la taille en points du texte et des symboles
`pty`	un caractère qui spécifie la forme du graphe, `"s"` : carrée, `"m"` : maximale
`tck`	une valeur qui spécifie la longueur des graduations sur les axes en fraction du plus petit de la largeur ou de la hauteur du graphe ; si `tck=1` une grille est tracée
`tcl`	idem mais en fraction de la hauteur d'une ligne de texte (défaut `tcl=-0.5`)
`xaxt`	si `xaxt="n"` l'axe des x est défini mais pas tracé (utile avec `axis(side=1, ...)`)
`yaxt`	si `yaxt="n"` l'axe des y est défini mais pas tracé (utile avec `axis(side=2, ...)`)

4.5 Un exemple concret

Afin d'illustrer l'utilisation des fonctionnalités graphiques de R, considérons un cas concret et simple d'un graphe bivarié de 10 paires de valeurs aléatoires. Ces valeurs ont été générées avec :

```
> x <- rnorm(10)
> y <- rnorm(10)
```

Le graphe voulu sera obtenu avec `plot()` ; on tapera la commande :

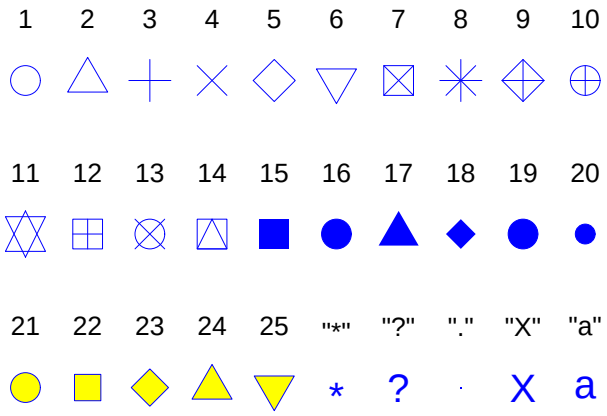


FIG. 2 – Les symboles pour tracer des points avec R (`pch=1:25`). Les couleurs ont été obtenues avec les options `col="blue", bg="yellow"`, la seconde option n'a un effet que pour les symboles 21 à 25. N'importe quel caractère peut être utilisé (`pch="*"`, `"?"`, `"."`, ...).

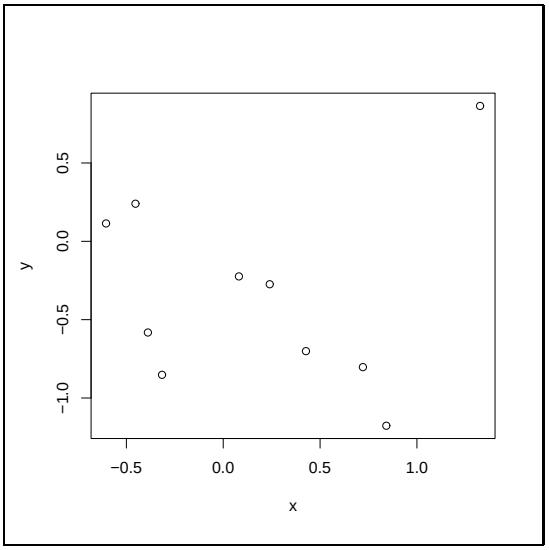


FIG. 3 – La fonction `plot` utilisée sans options.

```
> plot(x, y)
```

et le graphique sera dessiné sur le périphérique actif. Le résultat est représenté Fig. 3. Par défaut, R dessine les graphiques de façon « intelligente » : l'espacement entre les graduations sur les axes, la disposition des annotations, etc, sont calculés afin que le graphique obtenu soit le plus intelligible possible.

L'utilisateur peut toutefois vouloir changer l'allure du graphe, par exemple, pour conformer ses figures avec un style éditorial prédéfini ou les personnaliser pour un séminaire. La façon la plus simple de changer la présentation d'un graphe est d'ajouter des options qui modifieront les arguments par défaut. Dans notre cas, nous pouvons modifier de façon appréciable notre figure de la façon suivante :

```
plot(x, y, xlab="Ten random values", ylab="Ten other values",
     xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red",
     bg="yellow", bty="l", tcl=0.4,
     main="How to customize a plot with R", las=1, cex=1.5)
```

Le résultat est la Fig. 4. Voyons en détail chacune des options utilisée. D'abord, `xlab` et `ylab` vont changer les annotations sur les axes qui, par défaut, étaient les noms des variables. Ensuite, `xlim` et `ylim` nous permettent de définir les limites sur les deux axes[13]. Le paramètre graphique `pch` a été ici utilisé comme option : `pch=22` spécifie un carré dont la couleur du contour et celle de l'intérieur peuvent être différentes et qui sont données, respectivement, par `col` et `bg`. On se reportera au tableau sur les paramètres graphiques pour comprendre les modifications apportées par `bty`, `tcl`, `las` et `cex`. Enfin, un titre a été ajouté par l'option `main`.

Les paramètres graphiques et les fonctions graphiques secondaires permettent d'aller plus loin dans la présentation d'un graphe. Comme vu précédemment, certains paramètres graphiques ne peuvent pas être passés comme arguments dans une fonction comme `plot`. Nous allons maintenant modifier certains de ces paramètres avec `par()`, il est donc nécessaire cette fois de taper plusieurs commandes. Quand les paramètres graphiques sont modifiés, il est utile de sauver les valeurs initiales de ces paramètres au préalable afin de pouvoir les rétablir par la suite. Voici les commandes pour obtenir la Fig. 5.

```
opar <- par()
par(bg="lightyellow", col.axis="blue", mar=c(4, 4, 2.5, 0.25))
plot(x, y, xlab="Ten random values", ylab="Ten other values",
     xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red", bg="yellow",
     bty="l", tcl=-.25, las=1, cex=1.5)
title("How to customize a plot with R (bis)", font.main=3, adj=1)
par(opar)
```

[13] Par défaut, R ajoute 4% de part et d'autre des limites des axes. Ce comportement peut être supprimé en mettant les paramètres graphiques `xaxs="i"` et `yaxs="i"` (ceux-ci peuvent être passés comme options à `plot()`).

FIG. 4 – La fonction `plot` utilisée avec options.

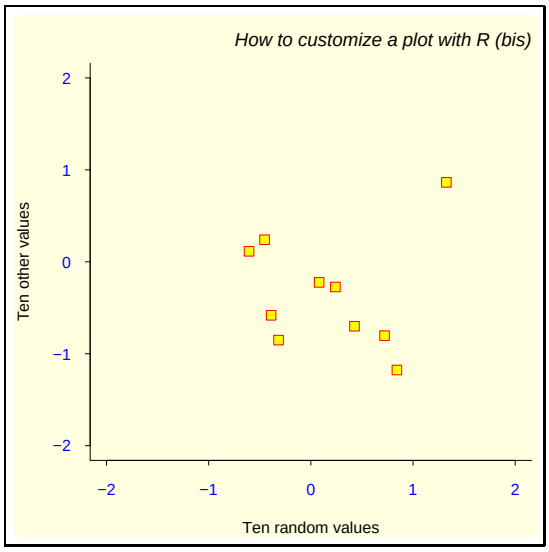


FIG. 5 – Les fonctions `par`, `plot` et `title`.

Détaillons les actions provoquées par ces commandes. Tout d'abord, les paramètres graphiques par défaut sont sauvés dans une liste qui est nommée, par exemple, `opar`. Trois paramètres vont être modifiés ensuite : `bg` pour la couleur de l'arrière-plan, `col.axis` pour la couleur des chiffres sur les axes et `mar` pour les dimensions des marges autour du cadre de traçage. Le graphe est tracé de façon presque similaire que pour la Fig. 4. On voit que la modification des marges a permis d'utiliser de l'espace libre autour du cadre de traçage. Le titre est ajouté cette fois avec la fonction graphique secondaire `title` ce qui permet de passer certains paramètres en arguments sans altérer le reste du graphique. Enfin, les paramètres graphiques initiaux sont restaurés avec la dernière commande.

Maintenant, le contrôle total ! Sur la Fig. 5, R détermine encore certaines choses comme le nombre de graduations sur les axes ou l'espace entre le titre et le cadre de traçage. Nous allons maintenant contrôler totalement la présentation du graphique. L'approche utilisée ici est de tracer le graphe « à blanc » avec `plot(..., type="n")`, puis d'ajouter les points, les axes, les annotations, etc, avec des fonctions graphiques secondaires. On se permettra aussi quelques fantaisies, comme de changer la couleur de fond du cadre de traçage. Les commandes suivent, et le graphe produit est la Fig. 6.

```
opar <- par()
par(bg="lightgray", mar=c(2.5, 1.5, 2.5, 0.25))
plot(x, y, type="n", xlab="", ylab="", xlim=c(-2, 2),
     ylim=c(-2, 2), xaxt="n", yaxt="n")
rect(-3, -3, 3, 3, col="cornsilk")
points(x, y, pch=10, col="red", cex=2)
axis(side=1, c(-2, 0, 2), tcl=-0.2, labels=FALSE)
axis(side=2, -1:1, tcl=-0.2, labels=FALSE)
title("How to customize a plot with R (ter)",
      font.main=4, adj=1, cex.main=1)
mtext("Ten random values", side=1, line=1, at=1, cex=0.9, font=3)
mtext("Ten other values", line=0.5, at=-1.8, cex=0.9, font=3)
mtext(c(-2, 0, 2), side=1, las=1, at=c(-2, 0, 2), line=0.3,
      col="blue", cex=0.9)
mtext(-1:1, side=2, las=1, at=-1:1, line=0.2, col="blue", cex=0.9)
par(opar)
```

Comme précédemment, les paramètres graphiques par défaut sont enregistrés et la couleur de l'arrière-plan est changé ainsi que les marges. Le graphe est ensuite dessiné avec `type="n"` pour ne pas tracer les points, `xlab="", ylab=""` pour ne pas marquer les noms des axes et `xaxt="n", yaxt="n"` pour ne pas tracer les axes. Le résultat est de tracer uniquement le cadre de traçage et de définir les axes en fonction de `xlim` et `ylim`. Notez qu'on aurait pu utiliser l'option `axes=FALSE` mais dans ce cas ni les axes ni le cadre n'auraient été tracés.

Les éléments sont ensuite ajoutés dans le cadre ainsi défini avec des fonctions graphiques secondaires. Avant d'ajouter les points, on va changer la

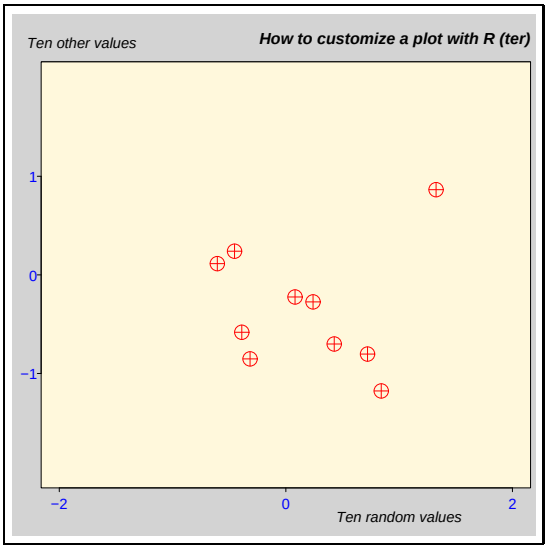


FIG. 6 – Un graphe fait « sur mesure ».

couleur dans le cadre avec `rect()` : les dimensions du rectangle sont choisies afin de dépasser largement celles du cadre.

Les points sont tracés avec `points()` ; on a cette fois changé de symbole. Les axes sont ajoutés avec `axis()` : le vecteur qui est passé en second argument donne les coordonnées des graduations qui doivent être tracées. L'option `labels=FALSE` spécifie qu'aucune annotation n'est ajoutée avec les graduations. Cette option accepte aussi un vecteur de mode caractère, par exemple `labels=c("A", "B", "C")`.

Le titre est ajouté avec `title()`, mais on a changé légèrement la police. Les annotations des axes sont mises avec `mtext()` (*marginal text*). Le premier argument de cette fonction est un vecteur de mode caractère qui donne le texte à afficher. L'option `line` indique la distance à partir du cadre de traçage (par défaut `line=0`), et `at` la coordonnée. Le second appel à `mtext()` utilise la valeur par défaut de `side` (3). Les deux autres appels de `mtext()` passent un vecteur numérique en premier argument : celui-ci sera converti en mode caractère.

4.6 Les packages grid et lattice

Les packages grid et lattice implémentent les systèmes « grid» et « lattice». Grid est un nouveau mode graphique avec son propre systéme de paramètres graphiques qui sont distincts de ceux vus ci-dessus. Les deux distinctions principales entre grid et le mode graphique de base sont :

- plus de flexibilité pour diviser les périphériques graphiques à l'aide des *vues* (*viewports*) qui peuvent être chevauchantes (les objets graphiques peuvent même être partagés entre vues, par exemple des flèches) ;
- les objets graphiques (*grob*) peuvent être modifiès ou effacés d'un graphe

sans avoir à le re-dessiner (comme doit être fait avec le mode graphique de base).

Les graphiques obtenus avec grid ne peuvent habituellement pas être combinés ou mélangés avec ceux produits par le mode graphique de base (le package gridBase doit être utilisé à cette fin). Les deux modes graphiques peuvent cependant être utilisés dans la même session sur le même périphérique graphique.

Lattice est essentiellement l'implémentation dans R des graphiques de type Trellis de S-PLUS. Trellis est une approche pour la visualisation de données multivariées particulièrement appropriée pour l'exploration de relations ou d'interactions entre variables[14]. L'idée principale derrière lattice (tout comme Trellis) est celle des graphes multiples conditionnés : un graphe bivarié entre deux variables sera découpé en plusieurs graphes en fonction des valeurs d'une troisième variable. La fonction `coplot` utilise une approche similaire, mais lattice offre des fonctionnalités plus vastes. Lattice utilise grid comme mode graphique.

La plupart des fonctions de lattice prennent pour argument principal une formule[15], par exemple `y ~ x`. La formule `y ~ x | z` signifie que le graphe de `y` en fonction de `x` sera dessiné en plusieurs sous-graphes en fonction des valeurs de `z`.

Le tableau ci-dessous indique les principales fonctions de lattice. La formule donnée en argument est la formule type nécessaire, mais toutes ces fonctions acceptent une formule conditionnelle (`y ~ x | z`) comme argument principal ; dans ce cas un graphe multiple, en fonction des valeurs de `z`, est dessiné comme nous le verrons dans les exemples ci-dessous.

`barchart(y ~ x)`	histogramme des valeurs de `y` en fonction de celles de `x`
`bwplot(y ~ x)`	graphe 'boites et moustaches'
`densityplot(~ x)`	graphe de fonctions de densité
`dotplot(y ~ x)`	graphe de Cleveland (graphes superposés ligne par ligne et colonne par colonne)
`histogram(~ x)`	histogrammes des fréquences de `x`
`qqmath(~ x)`	quantiles de `x` en fonction des valeurs attendues selon une distribution théorique
`stripplot(y ~ x)`	graphe unidimensionnel, `x` doit être numérique, `y` peut être un facteur
`qq(y ~ x)`	quantiles pour comparer deux distributions, `x` doit être numérique, `y` peut être numérique, caractère ou facteur mais doit avoir deux 'niveaux'
`xyplot(y ~ x)`	graphes bivariés (avec de nombreuses fonctionnalités)

[14] http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/index.html

[15] `plot()` accepte également une formule en argument principal : si `x` et `y` sont deux vecteurs de même longueur, `plot(y ~ x)` et `plot(x, y)` donneront des graphiques identiques.

`levelplot(z ~ x*y)` `contourplot(z ~ x*y)`	graphe en couleur des valeurs de `z` aux coordonnées fournies par `x` et `y` (`x`, `y` et `z` sont tous de même longueur)
`cloud(z ~ x*y)`	graphe 3-D en perspective (points)
`wireframe(z ~ x*y)`	idem (surface)
`splom(~ x)`	matrice de graphes bivariés
`parallel(~ x)`	graphe de coordonnées parallèles

Voyons maintenant quelques exemples afin d'illustrer quelques aspects de lattice. Il faut au préalable charger le package en mémoire avec la commande `library(lattice)` afin d'accéder aux fonctions.

D'abord, les graphes de fonctions de densité. Un tel graphe peut être dessiné simplement avec `densityplot(~ x)` qui tracera une courbe de densité empirique ainsi que les points correspondants aux observations sur l'axe des x (comme `rug()`). Notre exemple sera un peu plus compliqué avec la superposition, sur chaque graphe, des courbes de densité empirique et de densité estimée avec une loi normale. Il nous faut à cette fin utiliser l'argument `panel` qui définit ce qui doit être tracé dans chaque graphe. Les commandes sont :

```
n <- seq(5, 45, 5)
x <- rnorm(sum(n))
y <- factor(rep(n, n), labels=paste("n =", n))
densityplot(~ x | y,
            panel = function(x, ...) {
                panel.densityplot(x, col="DarkOliveGreen", ...)
                panel.mathdensity(dmath=dnorm,
                                  args=list(mean=mean(x), sd=sd(x)),
                                  col="darkblue")
            })
```

Les trois premières lignes génèrent un échantillon de variables normales que l'on divise en sous-échantillons d'effectif égal à 5, 10, 15, ... et 45. Ensuite vient l'appel de `densityplot` qui produit un graphe par sous-échantillon. `panel` prend pour argument une fonction. Dans notre exemple, nous avons défini une fonction qui fait appel à deux fonctions prédéfinies dans lattice : `panel.densityplot` qui trace la fonction de densité empirique et `panel.mathdensity` qui trace la fonction de densité estimée avec une loi normale. La fonction `panel.densityplot` est appellée par défaut si aucun argument n'est donné à `panel` : la commande `densityplot(~ x | y)` aurait donné le même graphe que sur la Fig. 7 mais sans les courbes bleues.

Les exemples suivants sont pris, plus ou modifiés, des pages d'aide de lattice et utilisent des données disponibles dans R : les localisations de 1000 séismes près des îles Fidji et des données biométriques sur des fleurs de trois espèces d'iris.

La Fig. 8 représente la localisation géographique des séismes en fonction de la profondeur. Les commandes nécessaires pour ce graphe sont :

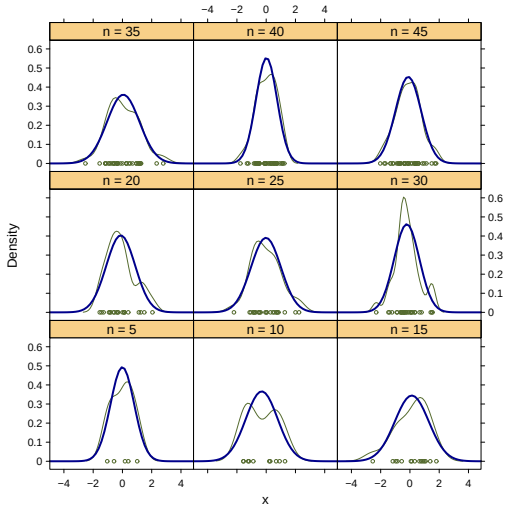


FIG. 7 – La fonction `densityplot`.

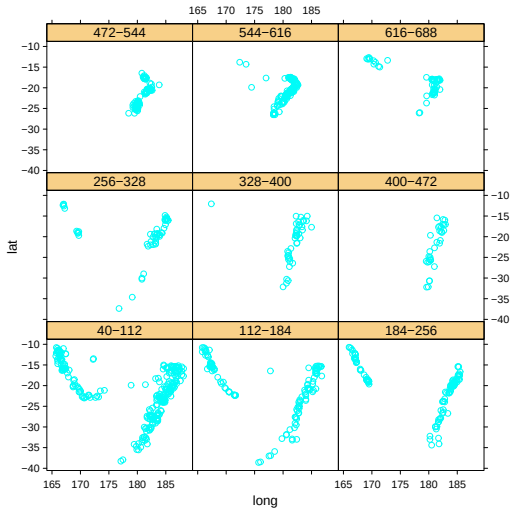


FIG. 8 – La fonction `xyplot` avec les données « quakes ».

```
data(quakes)
mini <- min(quakes$depth)
maxi <- max(quakes$depth)
int <- ceiling((maxi - mini)/9)
inf <- seq(mini, maxi, int)
quakes$depth.cat <- factor(floor(((quakes$depth - mini) / int)),
                      labels=paste(inf, inf + int, sep="-"))
xyplot(lat ~ long | depth.cat, data = quakes)
```

La première commande charge le jeu de données `quakes` en mémoire. Les cinq commandes suivantes créent un facteur en divisant la profondeur (variable `depth`) en neuf intervalles d'étendues égales : les niveaux de ce facteur sont nommés avec les bornes inférieures et supérieures de ces intervalles. Il suffit ensuite d'appeller la fonction `xyplot` avec la formule appropriée et un argument `data` qui indique où `xyplot` doit chercher les variables[16].

Avec les données `iris`, le chevauchement entre les différentes espèces est suffisament faible pour les représenter ensemble sur la même figure (Fig. 9). Les commandes correspondantes sont :

```
data(iris)
xyplot(
  Petal.Length ~ Petal.Width, data = iris, groups=Species,
  panel = panel.superpose,
  type = c("p", "smooth"), span=.75,
  auto.key = list(x = 0.15, y = 0.85)
)
```

L'appel de la fonction `xyplot` est ici un peu plus complexe que dans l'exemple précédent et utilise plusieurs options que nous allons détailler. L'option `groups`, comme son nom l'indique, définit des groupes qui seront utilisés par les autres options. On a déjà vu l'option `panel` qui définit comment les différents groupes vont être représentés sur la graphe : on utilise ici une fonction prédéfinie `panel.superpose` afin de superposer les groupes sur le même graphe. Aucune option n'étant passée à `panel.superpose`, les couleurs par défaut seront utilisées pour distinguer les groupes. L'option `type`, comme dans `plot()`, précise le type de traçage, sauf qu'ici on peut donner plusieurs arguments sous forme d'un vecteur : `"p"` pour tracer les points et `"smooth"` pour tracer une courbe de lissage dont le degré est donné par `span`. L'option `auto.key` ajoute la légende au graphe ; il est seulement nécessaire de donner, dans une liste, les coordonnées où la légende doit être tracée. Notez que ces coordonnées sont relatives (c'est-à-dire dans l'intervalle [0, 1]).

Nous allons voir maintenant la fonction `splom` avec les mêmes données sur les iris. Les commandes suivantes ont servi à produire la Fig. 10 :

[16] `plot()` ne peut pas prendre d'argument `data`, la localisation des variables doit être donnée explicitement, par exemple `plot(quakes$long ~ quakes$lat)`.

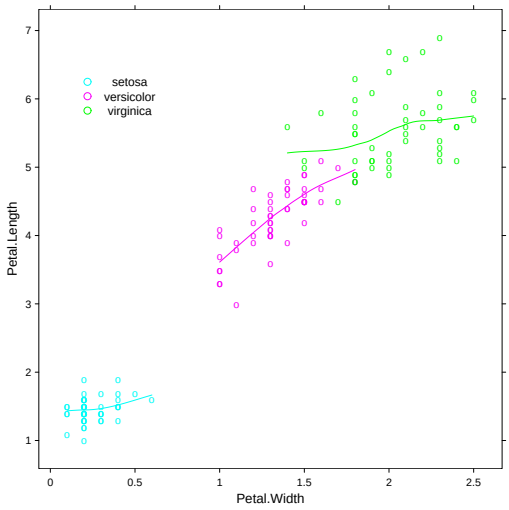


FIG. 9 – La fonction `xyplot` avec les données « iris ».

```
splom(
  ~iris[1:4], groups = Species, data = iris, xlab = "",
  panel = panel.superpose,
  auto.key = list(columns = 3)
)
```

L'argument principal est cette fois une matrice (les quatre premières colonnes d'`iris`). Le résultat est l'ensemble des graphes bivariés possibles entre les variables de la matrice, tout comme la fonction standard `pairs`. Par défaut, `splom` ajoute le texte « Scatter Plot Matrix » sous l'axe des x : pour l'éviter on a précisé `xlab=""`. Le reste des options est similaire à l'exemple précédent, sauf qu'on a précisé `columns = 3` pour `auto.key` afin que la légende soit disposée sur trois colonnes.

La FIG. 10 aurait pu être faite avec `pairs()`, mais cette fonction ne peut pas produire des graphes conditionnés comme sur la FIG. 11. Le code utilisé est relativement simple :

```
splom(~iris[1:3] | Species, data = iris, pscales = 0,
      varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"))
```

Les sous-graphes étant assez petits, on a ajouté deux options pour améliorer la lisibilité de la figure : `pscales = 0` supprime les graduations des axes (tous les sous-graphes sont à la même échelle), et on a redéfini les noms des variables pour les faire tenir sur deux lignes (`"\n"` code pour un saut de ligne dans une chaine de caractères).

Le dernier exemple utilise la méthode des coordonnées parallèles pour l'analyse exploratoire de données multivariées. Les variables sont alignées sur un axe (par exemple sur l'axe des y) et les valeurs observées sont représentées sur l'autre axe (les variables étant mises à la même échelle, par exemple en les

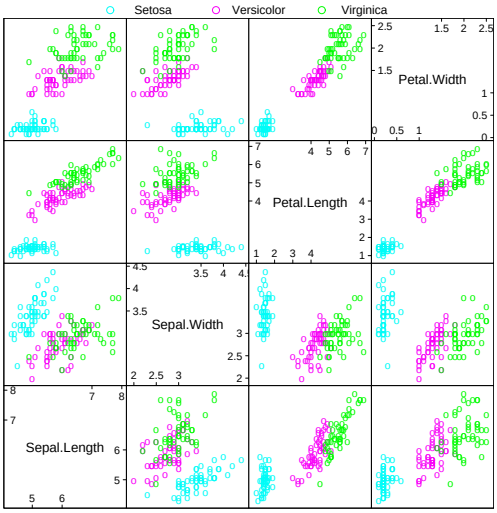


FIG. 10 – La fonction `splom` avec les données « iris » (1).

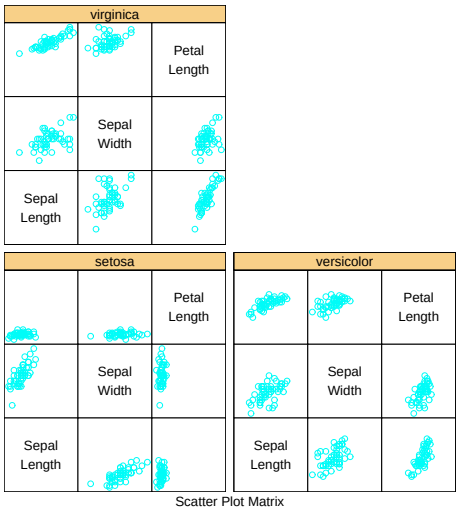


FIG. 11 – La fonction `splom` avec les données « iris » (2).

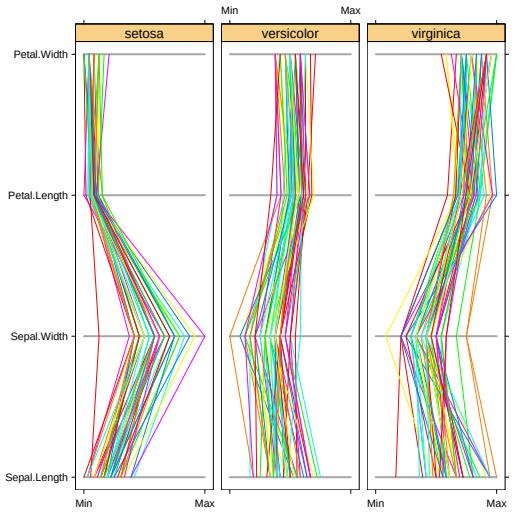


FIG. 12 – La fonction `parallel` avec les données « iris ».

réduisant). Les valeurs correspondant au même individu sont reliées par une ligne. Avec les données `iris` on obtient la FIG. 12 avec le code suivant :

```
parallel(~iris[, 1:4] | Species, data = iris, layout = c(3, 1))
```

5 Les analyses statistiques avec R

Encore plus que pour les graphiques, il est impossible ici d'aller dans les détails sur les possibilités offertes par R pour les analyses statistiques. Mon but est ici de donner des points de repère afin de se faire une idée sur les caractéristiques de R pour conduire des analyses de données.

Le package stats inclut des fonctions pour un large évental d'analyses statistiques : tests classiques, modèles linéaires (y compris régression par les moindres carrés, modèles linéaires généralisés et analyse de variance), lois de distribution, résumés statistiques, classifications hiérarchiques, analyses de séries-temporelles, moindres carrés non-linéaires, et analyses multivariées. D'autres méthodes statistiques sont disponibles dans un grand nombre de packages. Certains sont distribués avec une installation de base de R et sont *recommandés*, et de nombreux autres sont *contribués* et doivent être installés par l'utilisateur.

Nous commencerons par un exemple simple, qui ne nécessite aucun package autre que stats, afin de présenter l'approche générale pour analyser des données avec R. Puis nous détaillerons certaines notions qui sont utiles en général quelque soit le type d'analyse que l'on veut conduire tel les *formules* et les *fonctions génériques*. Ensuite, nous dresserons une vue d'ensemble sur les packages.

5.1 Un exemple simple d'analyse de variance

La fonction pour l'analyse de variance dans stats est `aov`. Pour l'essayer, prenons un jeu de données disponible dans R : `InsectSprays`. Six insecticides ont été testés en culture, la réponse observée étant le nombre d'insectes. Chaque insecticide ayant été testé 12 fois, on a donc 72 observations. Laissons de côté l'exploration graphique de ces données pour se consacrer à une simple analyse de variance de la réponse en fonction de l'insecticide. Après avoir chargé les données en mémoire à l'aide de la fonction `data`, l'analyse est faite après transformation en racine carrée de la réponse :

```
> data(InsectSprays)
> aov.spray <- aov(sqrt(count) ~ spray, data = InsectSprays)
```

L'argument principal (et obligatoire) d'`aov` est une formule qui précise la réponse à gauche du signe `~` et le prédicteur à droite. L'option `data = InsectSprays` précise que les variables doivent être prises dans le tableau de données `InsectSprays`. Cette syntaxe est équivalente à :

```
> aov.spray <- aov(sqrt(InsectSprays$count) ~ InsectSprays$spray)
```

ou encore (si l'on connait les numéros de colonne des variables) :

```
> aov.spray <- aov(sqrt(InsectSprays[, 1]) ~ InsectSprays[, 2])
```

La première syntaxe est préférable car plus claire.

Les résultats ne sont pas affichés car ceux-ci sont copiés dans un objet nommé `aov.spray`. Certaines fonctions sont utilisées pour extraire les résultats désirés, par exemple `print` pour afficher un bref résumé de l'analyse (essentiellement les paramètres estimés) et `summary` pour afficher plus de détails (dont les tests statistiques) :

```
> aov.spray
Call:
   aov(formula = sqrt(count) ~ spray, data = InsectSprays)

Terms:
                    spray Residuals
Sum of Squares  88.43787  26.05798
Deg. of Freedom        5        66

Residual standard error: 0.6283453
Estimated effects may be unbalanced
> summary(aov.spray)
            Df Sum Sq Mean Sq F value    Pr(>F)
spray        5 88.438  17.688  44.799 < 2.2e-16 ***
Residuals   66 26.058   0.395
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Rappelons que de taper le nom de l'objet en guise de commande équivaut à la commande `print(aov.spray)`. Une représentation graphique des résultats peut être obtenue avec `plot()` ou `termplot()`. Avant de taper la commande `plot(aov.spray)`, le graphique dera divisé en quatre afin que les quatre graphes diagnostiques soient dessinés sur le même graphe. Les commandes sont :

```
> opar <- par()
> par(mfcol = c(2, 2))
> plot(aov.spray)
> par(opar)
> termplot(aov.spray, se=TRUE, partial.resid=TRUE, rug=TRUE)
```

et les graphes obtenus sont représentés Fig. 13 et Fig. 14.

5.2 Les formules

Les formules sont un élément-clef des analyses statistiques avec R : la notation utilisée est la même pour (presque) toutes les fonctions. Une formule est

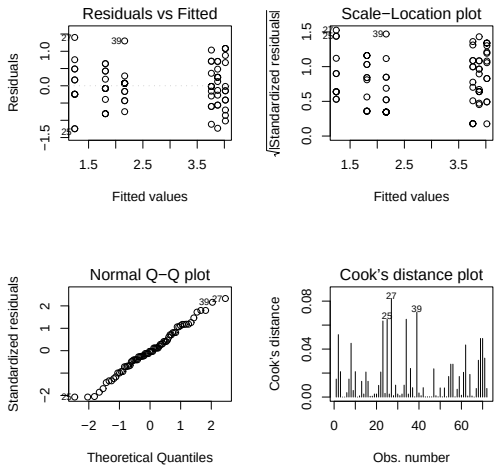


FIG. 13 – Représentation graphique des résultats de la fonction `aov` avec `plot()`.

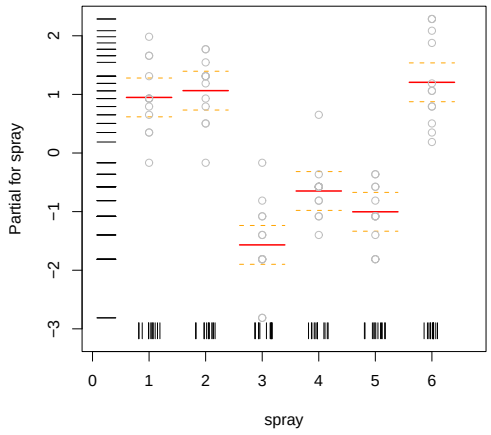


FIG. 14 – Représentation graphique des résultats de la fonction `aov` avec `termplot()`.

typiquement de la forme `y ~ model` où `y` est la réponse analysée et *`model`* est un ensemble de termes pour lesquels les paramètres sont estimés. Ces termes sont séparés par des symboles arithmétiques mais qui ont ici une signification particulière.

`a+b`	effets additifs de `a` et de `b`
`X`	si `X` est une matrice, ceci équivaut à un effet additif de toutes ses colonnes, c'est-à-dire `X[,1]+X[,2]+...+X[,ncol(X)]`; certaines de ces colonnes peuvent être sélectionnées avec l'indexation numérique (ex. : `X[,2:4]`)
`a:b`	effet interactif entre `a` et `b`
`a*b`	effets additifs et interactifs (identique à `a+b+a:b`)
`poly(a, n)`	polynome de `a` jusqu'au degré n
`^n`	inclue toutes les interactions jusqu'au niveau n, c'est-à-dire `(a+b+c)^2` est identique à `a+b+c+a:b+a:c+b:c`
`b %in% a`	les effets de `b` sont hiérarchiquement inclus dans `a` (identique à `a+a:b` ou `a/b`)
`-b`	supprime l'effet de `b`, par exemple : `(a+b+c)^2-a:b` est identique à `a+b+c+a:c+b:c`
`-1`	`y~x-1` force la régression à passer par l'origine (idem pour `y~x+0` ou `0+y~x`)
`1`	`y~1` ajuste un modèle sans effets (juste l'« intercept »)
`offset(...)`	ajoute un effet au modèle sans estimer de paramètre (par ex. `offset(3*x)`)

On voit que les opérateurs arithmétiques de R ont dans une formule un sens différent de celui qu'ils ont dans une expression classique. Par exemple, la formule `y~x1+x2` définira le modèle $y = \beta_1 x_1 + \beta_2 x_2 + \alpha$, et non pas (si l'opérateur `+` avait sa fonction habituelle) $y = \beta(x_1 + x_2) + \alpha$. Pour inclure des opérations arithmétiques dans une formule, on utilisera la fonction `I` : la formule `y~I(x1+x2)` définira alors le modèle $y = \beta(x_1 + x_2) + \alpha$. De même, pour définir le modèle $y = \beta_1 x + \beta_2 x^2 + \alpha$ on utilisera la formule `y ~ poly(x, 2)` (et non pas `y ~ x + x^2`). Cependant, il est possible d'inclure une fonction dans une formule afin de transformer une variable comme nous l'avons vu ci-dessus avec l'analyse de variance des données d'insecticides.

Pour les analyses de variance, `aov()` accepte une syntaxe particulière pour spécifier les effets aléatoires. Par exemple, `y ~ a + Error(b)` signifie effets additifs d'un terme fixe (`a`) et d'un terme aléatoire (`b`).

5.3 Les fonctions génériques

On se souvient que les fonctions de R agissent en fonction des attributs des objets éventuellement passés en arguments. La *classe* est un attibut qui mérite une certaine attention ici. Il est très fréquent que les fonctions statistiques de R retournent un objet de classe emprunté au nom de la fonction (par exemple,

`aov` retourne un objet de classe `"aov"`, `lm` retourne un de classe `"lm"`). Les fonctions que nous pourrons utiliser par la suite pour extraire les résultats agiront spécifiquement en fonction de la classe de l'objet. Ces fonctions sont dites *génériques*.

Par exemple, la fonction la plus utilisée pour extraire des résultats d'analyse est `summary` qui permet d'afficher les résultats détaillés. Selon que l'objet qui est passé en argument est de classe `"lm"` (modèle linéaire) ou `"aov"` (analyse de variance), il est clair que les informations à afficher ne seront pas les mêmes. L'avantage des fonctions génériques est d'avoir une syntaxe unique pour tous les cas.

Un objet qui contient les résultats d'une analyse est généralement une liste dont l'affichage est déterminée par sa classe. Nous avons déjà vu cette notion que les fonctions de R agissent spécifiquement en fonction de la nature des objets qui sont donnés en arguments. C'est un caractère général de R[17]. Le tableau suivant donne les principales fonctions génériques qui permettent d'extraire des informations d'un objet qui résulte d'une analyse. L'usage typique de ces fonctions étant :

```
> mod <- lm(y ~ x)
> df.residual(mod)
[1] 8
```

`print`	retourne un résumé succint
`summary`	retourne un résumé détaillé
`df.residual`	retourne le nombre de degrés de liberté résiduel
`coef`	retourne les coefficients estimés (avec parfois leurs erreurs-standards)
`residuals`	retourne les résidus
`deviance`	retourne la déviance
`fitted`	retourne les valeurs ajustées par le modèle
`logLik`	calcule le logarithme de la vraisemblance et le nombre de paramètre d'un modèle
`AIC`	calcule le critère d'information d'Akaike ou AIC (dépend de `logLik()`)

Une fonction comme `aov` ou `lm` produit donc une liste dont les différents éléments correspondent aux résultats de l'analyse. Si l'on reprend l'exemple de l'analyse de variance sur les données `InsectSprays`, on peut regarder la structure de l'objet créé par `aov` :

```
> str(aov.spray, max.level = -1)
List of 13
 - attr(*, "class")= chr [1:2] "aov" "lm"
```

Une autre façon de regarder cette structure est d'afficher les noms des éléments de l'objet :

[17] Il y a plus de 100 fonctions génériques dans R.

```
> names(aov.spray)
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "contrasts"
[10] "xlevels"       "call"          "terms"
[13] "model"
```

Les éléments peuvent ensuite être extraits comme vu précédemment :

```
> aov.spray$coefficients
(Intercept)      sprayB      sprayC      sprayD
  3.7606784   0.1159530  -2.5158217  -1.5963245
     sprayE      sprayF
 -1.9512174   0.2579388
```

`summary()` crée également une liste, qui dans le cas d'`aov()` se limite à un tableau de tests :

```
> str(summary(aov.spray))
List of 1
 $ :Classes anova  and 'data.frame':    2 obs. of  5 variables:
  ..$ Df     : num [1:2] 5 66
  ..$ Sum Sq : num [1:2] 88.4 26.1
  ..$ Mean Sq: num [1:2] 17.688  0.395
  ..$ F value: num [1:2] 44.8   NA
  ..$ Pr(>F) : num [1:2] 0 NA
 - attr(*, "class")= chr [1:2] "summary.aov" "listof"
> names(summary(aov.spray))
NULL
```

Les fonctions génériques n'agissent généralement pas sur les objets : elles appèlent la fonction appropriée en fonction de la classe de l'argument. Une fonction appelée par une générique est une *méthode* (*method*) dans le jargon de R. De façon schématique, une méthode est contruite selon *`generic.cls`*, où *`cls`* désigne la classe de l'objet. Dans le cas de `summary`, on peut afficher les méthodes correspondantes :

```
> apropos("^summary")
 [1] "summary"               "summary.aov"
 [3] "summary.aovlist"       "summary.connection"
 [5] "summary.data.frame"    "summary.default"
 [7] "summary.factor"        "summary.glm"
 [9] "summary.glm.null"      "summary.infl"
[11] "summary.lm"            "summary.lm.null"
[13] "summary.manova"        "summary.matrix"
[15] "summary.mlm"           "summary.packageStatus"
[17] "summary.POSIXct"       "summary.POSIXlt"
[19] "summary.table"
```

On peut visualiser les particularités de cette générique dans le cas de la régression linéaire comparée à l'analyse de variance avec un petit exemple simulé :

```
> x <- y <- rnorm(5)
> lm.spray <- lm(y ~ x)
> names(lm.spray)
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
> names(summary(lm.spray))
 [1] "call"          "terms"         "residuals"
 [4] "coefficients"  "sigma"         "df"
 [7] "r.squared"     "adj.r.squared" "fstatistic"
[10] "cov.unscaled"
```

Le tableau suivant indique certaines fonctions génériques qui font des analyses supplémentaires à partir d'un objet qui résulte d'une analyse faite au préalable, l'argument principal étant cet objet, mais dans certains cas un argument supplémentaire est nécessaire comme pour `predict` ou `update`.

`add1`	teste successivement tous les termes qui peuvent être ajoutés à un modèle
`drop1`	teste successivement tous les termes qui peuvent être enlevés d'un modèle
`step`	sélectionne un modèle par AIC (fait appel à `add1` et `drop1`)
`anova`	calcule une table d'analyse de variance ou de déviance pour un ou plusieurs modèles
`predict`	calcule les valeurs prédites pour de nouvelles données à partir d'un modèle
`update`	ré-ajuste un modèle avec une nouvelle formule ou de nouvelles données

Il y a également diverses fonctions utilitaires qui extraient des informations d'un objet modèle ou d'une formule, comme `alias` qui trouve les termes linéairement dépendants dans un modèle linéaire spécifié par une formule.

Enfin, il y a bien sûr les fonctions graphiques comme `plot` qui affiche divers diagnostiques ou `termplot` (*cf.* l'exemple ci-dessus) ; cette dernière fonction n'est pas vraiment générique mais fait appel à `predict`.

5.4 Les packages

Le tableau suivant liste les packages *standards* distribués avec une installation de base de R.

Package	Description
base	fonctions de base de R
datasets	jeux de données de base
grDevices	périphériques graphiques pour modes base et grid
graphics	graphiques base
grid	graphiques grid
methods	définition des méthodes et classes pour les objets R ainsi que des utilitaires pour la programmation
splines	régression et classes utilisant les représentations polynomiales
stats	fonctions statistiques
stats4	fonctions statistiques utilisant les classes S4
tcltk	fonctions pour utiliser les éléments de l'interface graphique Tcl/Tk
tools	utilitaires pour le développement de package et l'administration
utils	fonctions utilitaires de R

Certains de ces packages sont chargés en mémoire quand R est démarré ; ceci peut être affiché avec la fonction `search` :

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

Les autres packages peuvent être utilisés après chargement :

```
> library(grid)
```

La liste des fonctions d'un package peut être affichée avec :

```
> library(help = grid)
```

ou en parcourant l'aide au format html. Les informations relatives à chaque fonction peuvent être accédées comme vu précédemment (p. 7).

De nombreux packages *contribués* allongent la liste des analyses possibles avec R. Ils sont distribués séparément, et doivent être installés et chargés en mémoire sous R. Une liste complète de ces packages contribués, accompagnée d'une description, se trouve sur le site Web du CRAN[18]. Certains de ces packages sont regroupés parmi les packages *recommandés* car ils couvrent des méthodes souvent utilsées en analyse des données. Les packages recommandés sont souvent distribués avec une installation de base de R. Ils sont brièvement décrits dans le tableau ci-dessous.

[18] http://cran.r-project.org/src/contrib/PACKAGES.html

Package	Description
boot	méthodes de ré-échantillonnage et de bootstrap
class	méthodes de classification
cluster	méthodes d'aggrégation
foreign	fonctions pour importer des données enregistrés sous divers formats (S3, Stata, SAS, Minitab, SPSS, Epi Info)
KernSmooth	méthodes pour le calcul de fonctions de densité (y compris bivariées)
lattice	graphiques Lattice (Trellis)
MASS	contient de nombreuses fonctions, utilitaires et jeux de données accompagnant le livre « Modern Applied Statistics with S » par Venables & Ripley
mgcv	modèles additifs généralisés
nlme	modèles linéaires ou non-linéaires à effets mixtes
nnet	réseaux neuronaux et modèles log-linéaires multinomiaux
rpart	méthodes de partitionnement récursif
spatial	analyses spatiales (« kriging », covariance spatiale, ...)
survival	analyses de survie

Il y a deux autres dépôts principaux de packages pour R : le Projet Omegahat pour le Calcul Statistique[19] centré sur les applications basés sur le web et les interfaces entre programmes et langages, et le Projet Bioconductor[20] spécialisé dans les applications bioinformatiques (en particulier pour l'analyse des données de 'micro-arrays').

La procédure pour installer un package dépend du système d'exploitation et si vous avez installé R à partir des sources ou des exécutables précompilés. Dans ce dernier cas, il est recommandé d'utiliser les packages précompilés disponibles sur le site du CRAN. Sous Windows, l'exécutable Rgui.exe a un menu « Packages » qui permet d'installer un ou plusieurs packages via internet à partir du site Web de CRAN ou des fichiers '.zip' sur le disque local.

Si l'on a compilé R, un package pourra être installé à partir de ses sources qui sont distribuées sous forme de fichiers '.tar.gz'. Par exemple, si l'on veut installer le package `gee`, on téléchargera dans un permier temps le fichier gee_4.13-6.tar.gz (le numéro 4.13-6 désigne la version du package ; en général une seule version est disponible sur CRAN). On tapera ensuite à partir du système (et non pas de R) la commande :

```
R CMD INSTALL gee_4.13-6.tar.gz
```

Il y a plusieurs fonctions utiles pour gérer les packages comme `CRAN.packages`, `installed.packages` ou `download.packages`. Il est utile également de taper régulièrement la commande :

```
> update.packages()
```

[19] http://www.omegahat.org/R/
[20] http://www.bioconductor.org/

qui vérifie les versions des packages installés en comparaison à celles disponibles sur CRAN (cette commande peut être appelée du menu « Packages » sous Windows). L'utilisateur peut ensuite mettre à jour les packages qui ont des versions plus récentes que celles installées sur son système.

6 Programmer avec R en pratique

Maintenant que nous avons fait un tour d'ensemble des fonctionnalités de R, revenons au langage et à la programmation. Nous allons voir des idées simples susceptibles d'être mises en pratique.

6.1 Boucles et vectorisation

Le point fort de R par rapport à un logiciel à menus déroulants est dans la possibilité de programmer, de façon simple, une suite d'analyses qui seront exécutées successivement. Cette possibilité est propre à tout langage informatique, mais R possède des particularités qui rendent la programmation accessible à des non-spécialistes.

Comme les autres langages, R possède des *structures de contrôle* qui ne sont pas sans rappeler celles du langage C. Supposons qu'on a un vecteur `x`, et pour les éléments de `x` qui ont la valeur `b`, on va donner la valeur 0 à une autre variable `y`, sinon 1. On crée d'abord un vecteur `y` de même longueur que `x` :

```
y <- numeric(length(x))
for (i in 1:length(x)) if (x[i] == b) y[i] <- 0 else y[i] <- 1
```

On peut faire exécuter plusieurs instructions si elles sont encadrées dans des accolades :

```
for (i in 1:length(x)) {
    y[i] <- 0
    ...
}

if (x[i] == b) {
    y[i] <- 0
    ...
}
```

Une autre situation possible est de vouloir faire exécuter une instruction tant qu'une condition est vraie :

```
while (myfun > minimum) {
    ...
}
```

Les boucles et structures de contrôle peuvent cependant être évitées dans la plupart des situations et ce grâce à une caractéristique du langage R : la *vectorisation*. La structure vectorielle rend les boucles implicites dans les expressions et nous en avons vu de nombreux cas. Considérons l'addition de deux vecteurs :

```
> z <- x + y
```

Cette addition pourrait être écrite avec une boucle comme cela se fait dans la plupart de langages :

```
> z <- numeric(length(x))
> for (i in 1:length(z)) z[i] <- x[i] + y[i]
```

Dans ce cas il est nécessaire de créer le vecteur `z` au préalable à cause de l'utilisation de l'indexation. On réalise que cette boucle explicite ne fonctionnera que si `x` et `y` sont de même longueur : elle devra être modifiée si cela n'est pas le cas, alors que la première expression marchera quelque soit la situation.

Les exécutions conditionnelles (`if ... else`) peuvent être évitées avec l'indexation logique ; en reprenant l'exemple plus haut :

```
> y[x == b] <- 0
> y[x != b] <- 1
```

Les expressions vectorisées sont non seulement plus simples, mais aussi plus efficaces d'un point de vue informatique, particulièrement pour les grosses quantités de données.

Il y a également les fonctions du type 'apply' qui évitent d'écrire des boucles. `apply` agit sur les lignes et/ou les colonnes d'une matrice, sa syntaxe est `apply(X, MARGIN, FUN, ...)`, où `X` est la matrice, `MARGIN` indique si l'action doit être appliquée sur les lignes (`1`), les colonnes (`2`) ou les deux (`c(1, 2)`), `FUN` est la fonction (ou l'opérateur mais dans ce cas il doit être spécifié entre guillemets doubles) qui sera utilisée, et `...` sont d'éventuels arguments supplémentaires pour `FUN`. Un exemple simple suit.

```
> x <- rnorm(10, -5, 0.1)
> y <- rnorm(10, 5, 2)
> X <- cbind(x, y) # les colonnes gardent les noms "x" et "y"
> apply(X, 2, mean)
        x         y
-4.975132  4.932979
> apply(X, 2, sd)
        x         y
0.0755153 2.1388071
```

`lapply()` va agir sur une liste : la syntaxe est similaire à celle d'`apply` et le résultat retourné est une liste.

```
> forms <- list(y ~ x, y ~ poly(x, 2))
> lapply(forms, lm)
[[1]]

Call:
FUN(formula = X[[1]])

Coefficients:
(Intercept)            x
     31.683        5.377


[[2]]

Call:
FUN(formula = X[[2]])

Coefficients:
(Intercept)  poly(x, 2)1  poly(x, 2)2
     4.9330       1.2181      -0.6037
```

`sapply()` est une variante plus flexible de `lapply()` qui peut prendre un vecteur ou une matrice en argument principal, et retourne ses résultats sous une forme plus conviviale, en général sous forme de tableau.

6.2 Écrire un programme en R

Typiquement, un programme en R sera écrit dans un fichier sauvé au format ASCII et avec l'extension '.R'. La situation typique où un programme se révèle utile est lorsque l'on veut exécuter plusieurs fois une tâche identique. Dans notre premier exemple, nous voulons tracer le même graphe pour trois espèces d'oiseaux différentes, les données se trouvant dans trois fichiers distincts. Nous allons procéder pas-à-pas en voyant différentes façons de construire un programme pour ce problème très simple.

D'abord, construisons notre programme de la façon la plus intuitive en faisant exécuter successivement les différentes commandes désirées, en prenant soin au préalable de partitionner le graphique.

```
layout(matrix(1:3, 3, 1))         # partitionne le graphique
data <- read.table("Swal.dat")    # lit les données
plot(data$V1, data$V2, type="l")
title("swallow")                  # ajoute le titre
data <- read.table("Wren.dat")
plot(data$V1, data$V2, type="l")
title("wren")
data <- read.table("Dunn.dat")
```

```
plot(data$V1, data$V2, type="l")
title("dunnock")
```

Le caractère '#' sert à ajouter des commentaires dans le programme : R passe alors à la ligne suivante.

Le problème de ce premier programme est qu'il risque de s'allonger sérieusement si l'on veut ajouter d'autres espèces. De plus, certaines commandes sont répétées plusieurs fois, elles peuvent être regroupées et exécutées en modifiant les arguments qui changent. Les noms de fichier et d'espèce sont donc utilisés comme des variables. La stratégie utilisée ici est de mettre ces noms dans des vecteurs de mode caractère, et d'utiliser ensuite l'indexation pour accéder à leurs différentes valeurs.

```
layout(matrix(1:3, 3, 1))             # partitionne le graphique
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
for(i in 1:length(species)) {
    data <- read.table(file[i])       # lit les données
    plot(data$V1, data$V2, type="l")
    title(species[i])                 # ajoute le titre
}
```

On notera qu'il n'y a pas de guillemets autour de `file[i]` dans `read.table` puisque cet argument est de mode caractère.

Notre programme est maintenant plus compact. Il est plus facile d'ajouter d'autres espèces car les deux vecteurs qui contiennent les noms d'espèces et de fichiers sont définis au début du programme.

Les programmes ci-dessus pourront marcher si les fichiers '.dat' sont placés dans le répertoire de travail de R, sinon il faut soit changer ce répertoire de travail, ou bien spécifier le chemin d'accés dans le programme (par exemple : `file <- "/home/paradis/data/Swal.dat"`). Si les instructions sont écrites dans un fichier Mybirds.R, on peut appeler le programme en tapant :

```
> source("Mybirds.R")
```

Comme pour toute lecture dans un fichier, il est nécessaire de préciser le chemin d'accès au fichier s'il n'est pas dans le répertoire de travail.

6.3 Écrire ses fonctions

L'essentiel du travail de R se fait à l'aide de fonctions dont les arguments sont indiqués entre parenthèses. L'utilisateur peut écrire ses propres fonctions qui auront les mêmes propriétés que les autres fonctions de R.

Écrire ses propres fonctions permet une utilisation efficace, flexible et rationnelle de R. Reprenons l'exemple ci-dessus de la lecture de données dans un fichier suivi d'un graphe. Si l'on veut répéter cette opération quand on le veut, il peut être judicieux d'écrire une fonction :

```
myfun <- function(S, F)
{
    data <- read.table(F)
    plot(data$V1, data$V2, type="l")
    title(S)
}
```

Pour pouvoir être exécutée, cette fonction doit être chargée en mémoire ce qui peut se faire de plusieurs façons. On peut entrer les lignes de la fonction au clavier comme n'importe quelle commande, ou les copier/coller à partir d'un éditeur. Si la fonction a été enregistrée dans un fichier au format texte, on peut la charger avec `source()` comme un autre programme. Si l'utilisateur veut que ses fonctions soient chargées au démarrage de R, il peut les enregistrer dans un workspace `.RData` qui sera chargé en mémoire s'il est localisé dans le répertoire de travail de démarrage. Une autre possibilté est de configurer le fichier '.Rprofile' ou 'Rprofile' (voir `?Startup` pour les détails). Enfin, il est possible de créer un package mais ceci ne sera pas abordé ici (on se reportera au manuel « Writing R Extensions »).

On pourra par la suite, par une seule commande, lire les données et dessiner le graphe, par exemple `myfun("swallow", "Swal.dat")`. Nous arrivons donc à une troisième version de notre programme :

```
layout(matrix(1:3, 3, 1))
myfun("swallow", "Swal.dat")
myfun("wren", "Wrenn.dat")
myfun("dunnock", "Dunn.dat")
```

On peut également utiliser `sapply()` aboutissant à une quatrième version du programme :

```
layout(matrix(1:3, 3, 1))
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
sapply(species, myfun, file)
```

Avec R, il n'est pas nécessaire de déclarer les variables qui sont utilisées dans une fonction. Quand une fonction est exécutée, R utilise une règle nommée *étendue lexiquale* (*lexical scoping*) pour décider si un objet désigne une variable locale à la fonction ou un objet global. Pour comprendre ce mécanisme, considérons la fonction très simple ci-dessous :

```
> foo <- function() print(x)
> x <- 1
> foo()
[1] 1
```

Le nom `x` n'a pas été utilisé pour créer un objet au sein de `foo()`, R va donc chercher dans l'environnement *immédiatement* supérieur si un objet nommé `x` existe et affichera sa valeur (sinon un message d'erreur est affiché et l'exécution est terminée).

Si l'on utilise `x` comme nom d'objet au sein de notre fonction, la valeur de `x` dans l'environnement global n'est pas utilisée.

```
> x <- 1
> foo2 <- function() { x <- 2; print(x) }
> foo2()
[1] 2
> x
[1] 1
```

Cette fois `print()` a utilisé l'objet `x` qui a été défini dans son environnement, c'est-à-dire celui de la fonction `foo2`.

Le mot « *immédiatement* » ci-dessus est important. Dans les deux exemples que nous venons de voir, il y a *deux* environnements : celui global et celui de la fonction `foo` ou `foo2`. S'il y avait trois ou plus environnements emboîtés, la recherche des objets se fait par « paliers » d'un environnement à l'environnement immédiatement supérieur, ainsi de suite jusqu'à l'environnement global.

Il y a deux façons de spécifier les arguments à une fonction : par leurs positions ou par leurs noms. Par exemple, considérons une fonction qui prendrait trois arguments :

```
foo <- function(arg1, arg2, arg3) {...}
```

On peut exécuter `foo()` sans utiliser les noms `arg1`, ..., si les objets correspondants sont placés dans l'ordre, par exemple : `foo(x, y, z)`. Par contre, l'ordre n'a pas d'importance si les noms des arguments sont utilisés, par exemple : `foo(arg3 = z, arg2 = y, arg1 = x)`. Une autre particularité des fonctions dans R est la possibilité d'utiliser des valeurs par défaut dans la définition. Par exemple :

```
foo <- function(arg1, arg2 = 5, arg3 = FALSE) {...}
```

Les commandes `foo(x)`, `foo(x, 5, FALSE)` et `foo(x, arg3 = FALSE)` auront exactement le même résultat. L'utilisation de valeurs par défaut dans la définition d'une fonction est très utile, particulièrement en conjonction avec les arguments nommés (notamment pour changer une seule valeur par défaut : `foo(x, arg3 = TRUE)`).

Pour conclure cette partie, nous allons voir un exemple de fonction n'est pas purement statistique mais qui illustre bien la flexibilité de R. Considérons que l'on veuille étudier le comportement d'un modèle non-linéaire : le modèle de Ricker défini par :

$$N_{t+1} = N_t \exp\left[r\left(1 - \frac{N_t}{K}\right)\right]$$

Ce modèle est très utilisé en dynamique des populations, en particulier de poissons. On voudra à l'aide d'une fonction simuler ce modèle en fonction du taux de croissance r et de l'effectif initial de la population N_0 (la capacité du milieu K est couramment prise égale à 1 et cette valeur sera prise par défaut) ; les résultats seront affichés sous forme de graphique montrant les changements d'effectifs au cours du temps. On ajoutera une option qui permettra de réduire l'affichage des résultats aux dernières générations (par défaut tous les résultats seront affichés). La fonction ci-dessous permet de faire cette analyse numérique du modèle de Ricker.

```
ricker <- function(nzero, r, K=1, time=100, from=0, to=time)
{
    N <- numeric(time+1)
    N[1] <- nzero
    for (i in 1:time) N[i+1] <- N[i]*exp(r*(1 - N[i]/K))
    Time <- 0:time
    plot(Time, N, type="l", xlim=c(from, to))
}
```

Essayez vous-mêmes avec :

```
> layout(matrix(1:3, 3, 1))
> ricker(0.1, 1); title("r = 1")
> ricker(0.1, 2); title("r = 2")
> ricker(0.1, 3); title("r = 3")
```

7 Littérature sur R

Manuels. Plusieurs manuels sont distribués avec R dans R_HOME/doc/manual/ :

- *An Introduction to R* [R-intro.pdf],
- *R Installation and Administration* [R-admin.pdf],
- *R Data Import/Export* [R-data.pdf],
- *Writing R Extensions* [R-exts.pdf],
- *R Language Definition* [R-lang.pdf].

Les fichiers correspondants peuvent être sous divers formats (pdf, html, texi, ...) en fonction du type d'installation.

FAQ. R est également distribué avec un FAQ (*Frequently Asked Questions*) localisé dans le répertoire R_HOME/doc/html/. Une version de ce R-FAQ est régulièrement mise à jour sur le site Web du CRAN :

http://cran.r-project.org/doc/FAQ/R-FAQ.html.

Ressources en-ligne. Le site Web du CRAN accueille plusieurs documents et ressources bibliographiques ainsi que des liens vers d'autres sites. On peut y trouver une liste de publications (livres et articles) liées à R ou aux méthodes statistiques[21] et des documents et manuels écrits par des utilisateurs de R[22].

Listes de discussion. Il y a quatre listes de discussion électronique sur R ; pour s'inscrire, envoyer un message ou consulter les archives voir :

http://www.R-project.org/mail.html

La liste de discussion générale 'r-help' est une source intéressante d'information pour les utilisateurs (les trois autres listes sont consacrées aux annonces de nouvelles versions, et aux développeurs). De nombreux utilisateurs ont envoyé sur 'r-help' des fonctions ou des programmes qui peuvent donc être trouvés dans les archives. Il est donc important si l'on a un problème avec R de procéder dans l'ordre avant d'envoyer un message à 'r-help' et de :

1. consulter attentivement l'aide-en-ligne (éventuellement avec le moteur de recherche) ;
2. consulter le R-FAQ ;
3. chercher dans les archives de 'r-help' à l'adresse ci-dessus ou en consultant un des moteurs de recherche mis en place sur certains sites Web [23] ;

[21] http://www.R-project.org/doc/bib/R-publications.html
[22] http://cran.r-project.org/other-docs.html
[23] Les adresses de ces sites sont répertoriées sur celui du CRAN à http://cran.r-project.org/search.html

4. lire le « posting guide »[24] avant d'envoyer vos questions.

R News. La revue électronique *R News* a pour but de combler l'espace entre les listes de discussion électroniques et les publications scientifiques traditionnelles. Le premier numéro a été publié en janvier 2001[25].

Citer R dans une publication. Enfin, si vous mentionnez R dans une publication, il faut citer la référence suivante :

> R Development Core Team (2005). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL: http://www.R-project.org.

[24] http://www.r-project.org/posting-guide.html
[25] http://cran.r-project.org/doc/Rnews/