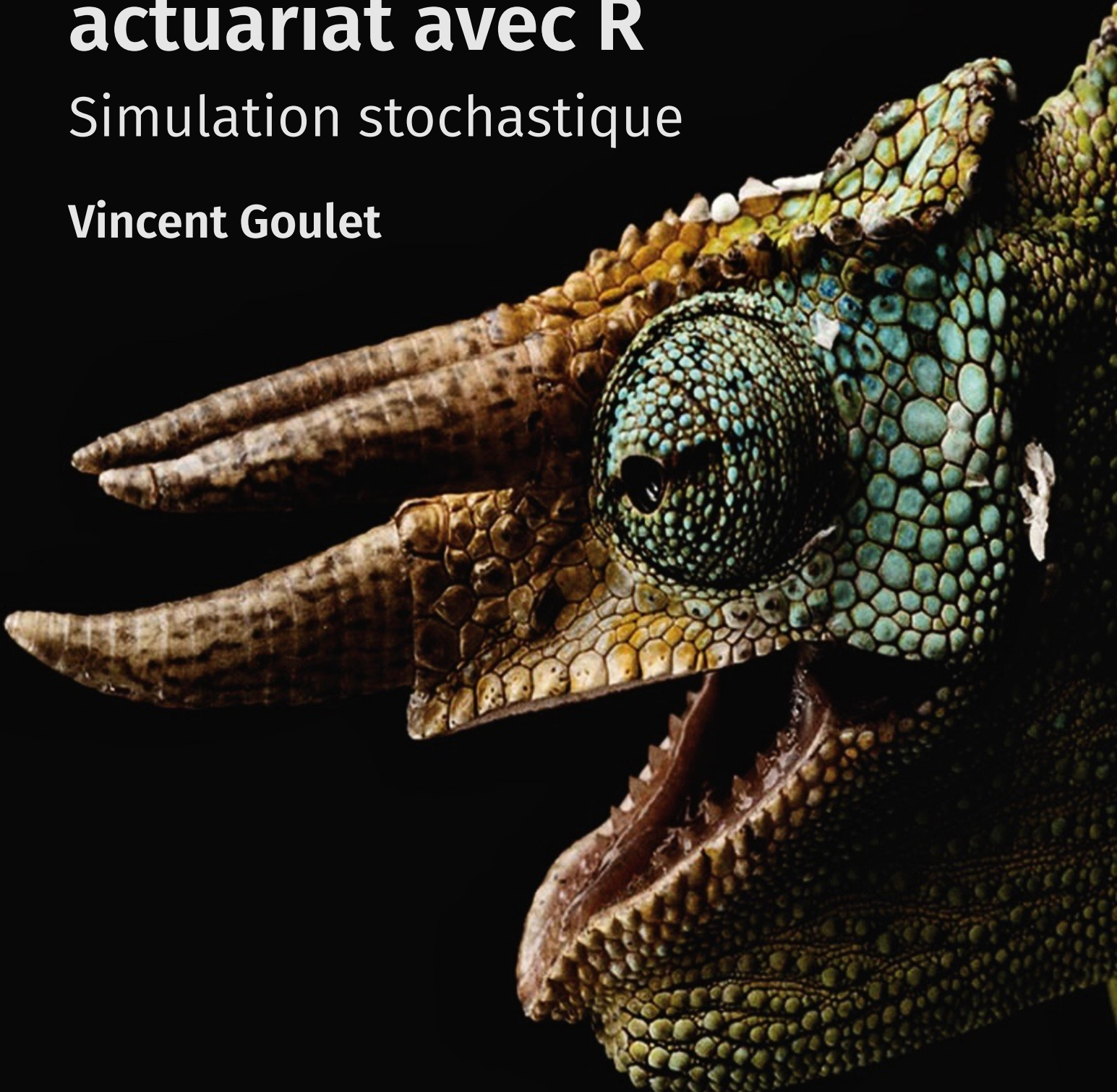


Méthodes numériques en actuariat avec R

Simulation stochastique

Vincent Goulet



UNIVERSITÉ
LAVAL

Méthodes numériques en actuariat avec R

Simulation stochastique

Vincent Goulet

Professeur titulaire

École d'actuariat, Université Laval

Avec la collaboration de

Laurent Caron

Édition 2020.02



Vincent Goulet, 2020

© 2020 par Vincent Goulet. « Méthodes numériques en actuariat avec R — Simulation stochastique » est mis à disposition sous licence **Attribution-Partage dans les mêmes conditions 4.0 International** de Creative Commons. En vertu de cette licence, vous êtes autorisé à :

- ▶ **partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats;
- ▶ **adapter** — remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.

L'Offrant ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence.

Selon les conditions suivantes :



Attribution — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.



Partage dans les mêmes conditions — Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'œuvre originale, vous devez diffuser l'œuvre modifiée dans les mêmes conditions, c'est-à-dire avec la même licence avec laquelle l'œuvre originale a été diffusée.

Code source

 [Voir sur GitLab](#)

Couverture

Le reptile en couverture est un caméléon de Jackson (*Chamaeleo jacksonii*) ou caméléon à trois cornes. On le rencontre en Afrique de l'Est, au Kenya et en Tanzanie, ainsi qu'aux États-Unis, plus précisément à Hawaï.

Crédit photo : Michael Nichols, National Geographic Society

Introduction

La simulation stochastique est une technique utilisée dans un grand nombre de domaines. Pensons seulement aux simulations boursières qui font l'objet d'un concours annuel, aux voitures qui sont d'abord conçues sur ordinateur et soumises à des tests de collisions virtuels, ou encore aux prévisions météo qui ne sont en fait que les résultats de simulations de systèmes climatiques d'une grande complexité.

Toute simulation stochastique repose sur une source de nombres aléatoires de qualité. Comment en générer un grand nombre rapidement et, surtout, comment s'assurer que les nombres produits sont bien aléatoires ? C'est un sujet d'une grande importance, mais aussi fort complexe. Nous nous contenterons donc de l'effleurer en étudiant les techniques de base dans le [chapitre 1](#).

En actuariat, nous avons habituellement besoin de nombres aléatoires provenant d'une loi de probabilité non uniforme. Le [chapitre 2](#) présente quelques algorithmes pour transformer des nombres aléatoires uniformes en nombres non uniformes. Évidemment, des outils informatiques sont aujourd'hui disponibles pour générer facilement et rapidement des nombres aléatoires de diverses lois de probabilité. Nous passons en revue les fonctionnalités de R et d'Excel à cet égard.

Enfin, l'ouvrage se termine au [chapitre 3](#) par une application à première vue inusitée de la simulation, soit le calcul d'intégrales définies par la méthode dite « Monte Carlo ».

Utilisation de l'ouvrage

Chaque chapitre de cet ouvrage propose un problème à résoudre au fil du texte. L'énoncé du problème, les indices en cours de chapitre et la solution complète se présentent dans des sections marquées des symboles ⚙️, 🔍 et ⚡.


L'étude de l'ouvrage implique des allers-retours entre le texte et le code R à la fin de chaque chapitre. Ce code informatique et les commentaires qui l'accompagnent visent à enrichir vos apprentissages. Assurez-vous donc de lire attentivement tant les commentaires que le code, d'exécuter le code pas-à-pas et de bien comprendre ses effets.

Le code informatique est distribué avec l'ouvrage sous forme de fichiers de script. De plus, à chaque fichier `.R` correspond un fichier `.Rout` contenant les résultats de son évaluation non interactive.


Vous trouverez également en annexe un bref exposé sur la planification d'une simulation en R, des rappels sur la transformation de variables aléatoires, de même que les solutions complètes des exercices.

Fonctionnalités interactives

En consultation électronique, ce document se trouve enrichi de plusieurs fonctionnalités interactives.

- ▶ Intraliens du texte vers une ligne précise d'une section de code informatique et, en sens inverse, du numéro de la ligne vers le point de la référence dans le texte. Ces intraliens sont marqués par la couleur ■.
- ▶ Intraliens entre le numéro d'un exercice et sa solution, et vice versa. Ces intraliens sont aussi marqués par la couleur ■.
- ▶ Intraliens entre les citations dans le texte et leur entrée dans la bibliographie. Ces intraliens sont marqués par la couleur ■.
- ▶ Hyperliens vers des ressources externes marqués par le symbole  et la couleur ■.
- ▶ Table des matières, liste des tableaux, liste des figures et liste des vidéos permettant d'accéder rapidement à des ressources du document.

Blocs signalétiques

Le document est parsemé de divers types de blocs signalétiques inspirés de **AsciiDoc**  qui visent à attirer votre attention sur une notion. Vous pourrez rencontrer l'un ou l'autre des blocs suivants.



Astuce! Ces blocs contiennent un truc, une astuce, ou tout autre type d'information utile.



Avertissement! Ces blocs mettent l'accent sur une notion ou fournissent une information importante pour la suite.



Attention! Vous risquez de vous brûler — métaphoriquement s'entend — si vous ne suivez pas les recommandations de ces blocs.



Important! Ces blocs contiennent les remarques les plus importantes. Veillez à en tenir compte.



Ces blocs contiennent des remarques additionnelles sur la matière ou des informations amusantes, mais non essentielles.



Ces blocs contiennent des liens vers des vidéos dans ma [chaîne YouTube](#) dédiée à ce document de référence. Les vidéos sont répertoriées dans la liste des vidéos.



Ces blocs vous invitent à interrompre la lecture du texte pour passer à l'étude du code R des sections d'exemples.



Remarques spécifiques à macOS.

Document libre

Tout comme R et l'ensemble des outils présentés dans ce document, le projet « Méthodes numériques en actuariat avec R » s'inscrit dans le mouvement de l'[informatique libre](#). Vous pouvez accéder à l'ensemble du code source en format \LaTeX en suivant le lien dans la page de copyright. Vous trouverez dans le fichier `README.md` toutes les informations utiles pour composer le document.

Votre contribution à l'amélioration du document est également la bienvenue; consultez le fichier `CONTRIBUTING.md` fourni avec ce document et voyez votre nom ajouté au fichier `COLLABORATEURS`.

Remerciements

Je tiens à souligner la précieuse collaboration de MM. Mathieu Boudreault, Sébastien Auclair et Louis-Philippe Pouliot lors de la rédaction des exercices et des solutions.

Table des matières

Introduction **vii**

Table des matières **xi**

Liste des tableaux **xiii**

Liste des figures **xv**

Liste des vidéos **xvii**

1 Génération de nombres aléatoires uniformes 1

- 1.1 Énoncé du problème **1**
- 1.2 Pourquoi faire de la simulation ? **2**
- 1.3 Indice pour le problème **3**
- 1.4 Générateurs de nombres aléatoires **4**
- 1.5 Congruence et modulo **5**
- 1.6 Générateurs congruentiels linéaires **6**
- 1.7 Indice pour le problème **8**
- 1.8 Générateurs utilisés dans Excel et R **8**
- 1.9 Indice pour le problème **10**
- 1.10 Solution du problème **10**
- 1.11 Code informatique **11**
- 1.12 Exercices **13**

2 Simulation de nombres aléatoires non uniformes 15

- 2.1 Énoncé du problème **16**
- 2.2 Méthode de l'inverse **16**
- 2.3 Indice pour le problème **22**
- 2.4 Méthode acceptation-rejet **22**
- 2.5 Indice pour le problème **28**

2.6	Fonctions de simulation dans Excel et R	28
2.7	Modèles actuariels	32
2.8	Solution du problème	38
2.9	Code informatique	40
2.10	Exercices	48
3	Intégration Monte Carlo	55
3.1	Énoncé du problème	55
3.2	Contexte	56
3.3	Méthode Monte Carlo	56
3.4	Indice pour le problème	60
3.5	Intégrales multiples	61
3.6	Solution du problème	62
3.7	Code informatique	63
3.8	Exercices	66
A	Planification d'une simulation avec R	67
A.1	Contexte	67
A.2	Première approche : avec une boucle	68
A.3	Seconde approche : avec <code>sapply</code>	68
A.4	Variante de la seconde approche	72
A.5	Gestion des fichiers	72
A.6	Exécution en lot	73
A.7	Conclusion	74
B	Transformations de variables aléatoires	75
B.1	Technique de la fonction de répartition	75
B.2	Technique du changement de variable univariée	79
B.3	Technique du changement de variable multivariée	81
B.4	Technique de la fonction génératrice des moments	85
C	Solutions des exercices	87
	Chapitre 1	87
	Chapitre 2	89
	Chapitre 3	105
	Bibliographie	109

Liste des tableaux

- 2.1 Liste des fonctions relatives à des lois de probabilité depuis Excel 2010. 30
- 2.2 Lois de probabilité pour lesquelles il existe des fonctions dans le système R de base 31

Liste des figures

1.1	Enveloppe convexe d'un ensemble de points	2
2.1	Représentation graphique de la méthode de l'inverse	17
2.2	Illustration de la méthode de l'inverse pour une distribution discrète	20
2.3	Fonctions de densité de probabilité et de répartition d'une variable aléatoire mixte	21
2.4	Illustration de la méthode acceptation-rejet	24
2.5	Interprétation alternative de la méthode acceptation-rejet	25
2.6	Fonction de densité de probabilité d'une loi Bêta(3, 2) enveloppée d'un triangle	27
2.7	Simulation de points sur un disque par acceptation-rejet	29
2.8	Fonctions de densité de probabilité de deux log-normales et de leur mélange discret	34
2.9	Code d'une fonction pour simuler par la méthode d'acceptation-rejet des nombres uniformément sur un disque de rayon 1 centré à l'origine	39
2.10	Code d'une fonction pour simuler des nombres uniformément sur un disque de rayon 1 centré à l'origine en passant par les coordonnées polaires	40
2.11	Fonction de simulation d'une loi Bêta(α, β)	52
3.1	Approximation de l'aire sous une fonction par les deux sommes de Riemann R_n^- (gauche) et R_n^+ (droite)	57
3.2	Fonction $h(x)$ de l'exemple 3.1 et points choisis aléatoirement où la fonction est évaluée pour l'intégration Monte Carlo	60
3.3	Fonction $h(x)$ de l'exemple 3.2 et points choisis aléatoirement où la fonction est évaluée pour l'intégration Monte Carlo	63
A.1	Code pour la simulation utilisant une boucle for	69

A.2	Définition de la fonction <code>simul1</code>	70
A.3	Définitions des fonction <code>fun1</code> et <code>simul2</code>	71
A.4	Définitions des fonction <code>fun2</code> et <code>simul3</code>	73
C.1	Code de la fonction <code>rand</code>	87
C.2	Paires de valeurs du générateur congruentiel multiplicatif avec $m = 2^{31} - 1$ et $a = 17$	89
C.3	Paires de valeurs du générateur congruentiel multiplicatif avec $m = 2^{31} - 1$ et $a = 85$	90
C.4	Démonstration graphique du fonctionnement de la transformation de Box-Muller	92
C.5	Fonction de simulation d'une distribution $\text{Gamma}(\alpha, 1)$ avec $\alpha > 1$	94
C.6	Démonstration graphique du fonctionnement de la transformation de l'exercice 2.8	97
C.7	Démonstration graphique du fonctionnement de la transformation de l'exercice 2.9	99
C.8	Code R de la fonction <code>rbeta.ar2</code>	100

Liste des vidéos

Le numéro indiqué à gauche est celui de la section dans laquelle se trouve le bloc signalétique.

- 2.4 [Méthode d'acceptation-rejet](#) 24
- 2.7 [Simulation de mélanges discrets](#) 35

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

Tiré de [XKCD.com](https://xkcd.com)

1 Génération de nombres aléatoires uniformes

Objectifs du chapitre

- ▶ Identifier des applications actuarielles de la simulation stochastique.
- ▶ Énoncer les caractéristiques d'un bon générateur de nombre pseudo-aléatoires.
- ▶ Utiliser l'opérateur mathématique modulo.
- ▶ Générer des nombres pseudo-aléatoires à l'aide d'un générateur congruentiel linéaire.
- ▶ Établir les caractéristiques d'un générateur congruentiel linéaire, notamment sa période.
- ▶ Utiliser les générateurs de nombres aléatoires de Excel et R.

Ce chapitre traite de la simulation de nombres (pseudo) aléatoires distribués uniformément sur l'intervalle $(0, 1)$. La transformation de ces nombres uniformes en nombres provenant d'autres distributions statistiques fera l'objet du prochain chapitre.



1.1 Énoncé du problème

L'enveloppe convexe (*convex hull*) d'un ensemble de points dans le plan est l'ensemble des points qui, lorsque reliés entre eux, forment un polygone convexe contenant tous les autres points. La [figure 1.1](#) en présente un exemple.

Dans le problème des quatre points de [Sylvester \(1865\)](#), on cherche la probabilité que l'enveloppe convexe de quatre points placés dans une région R forme un triangle (voir l'article de [Weisstein](#) dans MathWorld pour plus de détails).

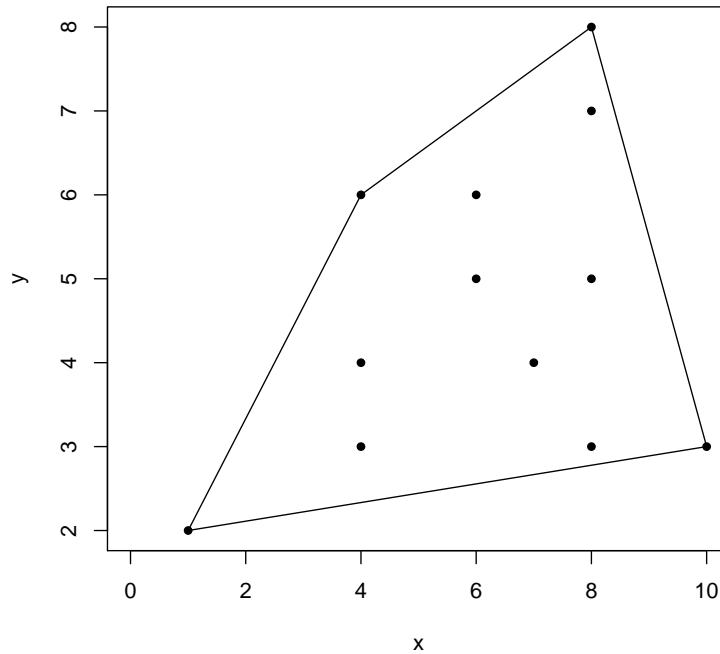


FIG. 1.1 – Enveloppe convexe d'un ensemble de points

Nous souhaitons vérifier que lorsque les quatre points sont distribués uniformément sur un carré, la probabilité que leur enveloppe convexe forme un triangle est $\frac{11}{36} \approx 0,30556$.

1.2 Pourquoi faire de la simulation ?

La simulation stochastique est une technique de plus en plus utilisée en actuariat comme dans la plupart des sciences appliquées, en génie, en finance, etc. Les modèles mathématiques et la simulation stochastique comportent plusieurs avantages par rapport à l'expérimentation directe, dont, entre autres :

- la simulation est non destructrice et peu coûteuse ;
- le système considéré n'a pas besoin d'exister ;

- ▶ la simulation est facile à répéter ;
- ▶ l'évolution dans la simulation peut être plus rapide que dans la réalité ;
- ▶ la simulation permet de considérer des modèles très complexes impossibles à traiter analytiquement.

En revanche, au rayon des inconvénients, on note :

- ▶ le coût (en temps et en argent) de modélisation et de programmation s'avère parfois important ;
- ▶ le temps d'exécution peut devenir excessif ;
- ▶ la simulation ne fournit que des estimations ;
- ▶ l'analyse statistique des résultats peut ne pas toujours être simple.

À la base, toute étude de simulation requiert une source de nombres aléatoires. Or, ces nombres aléatoires ne sont pas toujours facile à obtenir — surtout en grande quantité — et la qualité de la source est primordiale pour que l'étude soit fiable. En effet, un générateur qui ne fournirait pas des nombres suffisamment aléatoires, ou alors qui se répètent trop rapidement, peut corrompre les résultats d'une étude jusqu'à rendre ses conclusions invalides.

Les nombres aléatoires sont également beaucoup utilisés en cryptographie. Ici encore, un générateur de mauvaise qualité peut avoir des conséquences fâcheuses. Par exemple, si la période du générateur est trop courte, il devient relativement facile de percer la sécurité d'un système en découvrant le mot de passe par une attaque en force.



1.3 Indice pour le problème

Nous pourrions effectuer la vérification demandée à l'aide de la simulation. L'idée consiste à générer les coordonnées de quatre points dans le plan et de vérifier si trois de ceux-ci forment l'enveloppe convexe — autrement dit, l'enveloppe est un triangle. En répétant cette procédure un grand nombre de fois, nous pourrions calculer la *proportion* d'enveloppes triangulaires et ainsi obtenir une estimation de la *probabilité* que l'enveloppe convexe de quatre points dans le plan forme un triangle.

Pour nous aider, la fonction `chull` de R calcule l'enveloppe convexe d'un ensemble de points. Plus précisément, la fonction retourne les indices des points faisant partie de l'enveloppe. Par exemple, les coordonnées cartésiennes des points de la [figure 1.1](#) sont :


```
> x <- c(8, 1, 7, 6, 4, 4, 8, 8, 4, 6, 8, 10)
> y <- c(8, 2, 4, 5, 3, 4, 5, 3, 6, 6, 7, 3)
```

Ceux formant l'enveloppe se trouvent aux positions 12, 2, 9 et 1 des vecteurs de coordonnées :

```
> ch <- chull(x, y)
```

1.4 Générateurs de nombres aléatoires

Nous souhaitons obtenir des nombres issus d'une distribution uniforme sur un intervalle quelconque, en général $(0, 1)$. Comment procéder ?

1. Nous pouvons utiliser les résultats de processus physiques aléatoires en apparence comme, par exemple :
 - ▶ le lancer d'une pièce de monnaie ou d'un dé ;
 - ▶ des nombres pris au hasard dans des tableaux de rapports ou dans un annuaire ;
 - ▶ la roulette ;
 - ▶ le bruit électronique (tableaux RAND) ;
 - ▶ les intervalles de temps dans un processus de décroissance radioactive sont considérés parfaitement aléatoires ; le site [HotBits](#)  fournit des nombres issus d'un tel processus.

L'utilisation de listes de nombres aléatoires ou de sources physiques est toutefois peu pratique avec un ordinateur, surtout lorsque l'on a besoin de milliers ou de millions de nombres aléatoires.

2. Une ancienne technique est celle des carrés centraux de von Neumann : on prend un nombre à quatre chiffres, on l'élève au carré, puis on extrait les quatre chiffres du milieu, et ainsi de suite. Par exemple :

$$8653^2 = 74874409$$

$$8744^2 = 76457536$$

$$4575^2 = 20930625$$

3. Nous pouvons construire des générateurs basés sur la suite de Fibonacci, des générateurs chaotiques, etc.

En fait, les générateurs couramment utilisés aujourd'hui dans les ordinateurs sont des évolutions des générateurs dits *congruentiels*. Ils sont particulièrement utiles parce qu'aisément *reproductibles*. De plus, nous pouvons

généralement en connaître les propriétés — notamment la période — par une analyse mathématique poussée. Knuth (1997, section 3.1) fournit un exemple éloquent de l'importance de pouvoir démontrer mathématiquement les propriétés d'un générateur de nombres aléatoires. Cette référence de quelques pages seulement est fournie dans le site du cours; la lire avant d'aller plus loin.

C'est fait ? Bien. Intéressant, n'est-ce pas ?

Dans la suite, nous nous concentrerons sur les générateurs de nombres pseudo-aléatoires. En général, nous exigerons d'un générateur de ce type qu'il :

1. produise des nombres distribués approximativement uniformément ;
2. produise des nombres approximativement indépendants dans un bon nombre de dimensions ;
3. possède une période suffisamment longue (au moins 2^{60}) ;
4. soit facilement reproductible à partir d'un point de départ donné, mais qu'il soit autrement impossible à prédire.

1.5 Congruence et modulo

Les générateurs congruentiels utilisent l'arithmétique modulo. Une propriété de base de cette arithmétique est l'équivalence, ou congruence, modulo m .

Définition 1.1. Deux nombres a et b sont dits *équivalents*, ou *congruents*, modulo m si la différence entre a et b est un entier divisible par m . Mathématiquement,

$$a \equiv b \pmod{m} \quad \Leftrightarrow \quad \frac{a-b}{m} = k, \quad k \in \mathbb{Z}.$$

En d'autres mots, deux nombres sont équivalents modulo m si la distance entre ceux-ci est un multiple de m . La notion d'équivalence partitionne donc l'ensemble des nombres (ici, les réels).

Exemple 1.1. On a

1. $5 \equiv 14 \pmod{3}$ car $\frac{14-5}{3} = 3$; 14 et 5 sont distants de 9, un multiple de 3 ;
2. $-1 \equiv 5 \pmod{3}$ car $\frac{5+1}{3} = 2$; -1 et 5 sont distants de 6, un multiple de 3 ;

3. $0,33 \equiv 1,33 \bmod 1$; on notera que le calcul en modulo 1 équivaut à retourner la partie fractionnaire d'un nombre ;
4. la minute dans l'heure est donnée en modulo 60 : 00h15, 1h15, 2h15, ... sont des heures équivalentes modulo 60.

□

De la définition de congruence découle celle de *réduction modulo* ou *résidu modulo* : si $a \equiv b \bmod m$ et $0 \leq a < m$, alors a est le résidu de la division de b par m , ou le résidu de b modulo m , c'est-à-dire

$$a = b \bmod m \quad \Leftrightarrow \quad a = b - \left\lfloor \frac{b}{m} \right\rfloor m,$$

où $\lfloor x \rfloor$ est le plus grand entier inférieur ou égal à x .



La plupart des langages de programmation et logiciels à connotation mathématique comportent un opérateur ou une fonction modulo. Dans R il s'agit de l'opérateur %%; dans Excel, de la fonction MOD().

1.6 Générateurs congruentiels linéaires

Dans un générateur congruentiel linéaire, tout nombre dans la suite générée détermine le nombre suivant par la formule

$$x_i = (ax_{i-1} + c) \bmod m,$$

où $0 \leq x_i < m$ et

- ▶ a est appelé le *multiplicateur* ;
- ▶ c est appelé l'*incrément* ;
- ▶ m est appelé le *module* ;
- ▶ x_0 (un nombre quelconque) est l'*amorce* (« *seed* »).

Un générateur est dit *multiplicatif* lorsque l'incrément est $c = 0$, et *mixte* dans le cas contraire.

Pour obtenir des nombres uniformes sur $[0, 1]$ ou $(0, 1)$, il suffit de définir

$$u_i = \frac{x_i}{m}.$$



La méthode de génération des nombres est entièrement déterministe, c'est pourquoi il convient mieux de parler de nombres *pseudo*-aléatoires.

Un générateur congruentiel est forcément périodique puisqu'il ne peut prendre, au mieux, que les valeurs

- ▶ $0, 1, 2, \dots, m - 1$ pour un générateur mixte;
- ▶ $1, 2, \dots, m - 1$ pour un générateur multiplicatif.

C'est pourquoi nous cherchons donc à avoir la période la plus longue possible, tout en obtenant des suites en apparence aléatoires.

Pour les générateurs multiplicatifs ($c = 0$), la période maximale $m - 1$ est atteinte si :

- ▶ m est un nombre premier (on en choisira un grand);
- ▶ a est une *racine primitive* de m , c'est à dire que le plus petit entier k satisfaisant

$$1 = a^k \bmod m$$

est $k = m - 1$.

Des valeurs populaires sont $m = 2^{31} - 1$ (nombre premier de Mersenne) et $a = 7^5$.

Exemple 1.2. Soit un générateur congruentiel multiplicatif avec $a = 7$ et $m = 31$. Les quatre premiers nombres pseudo-aléatoires avec l'amorce $x_0 = 19$ sont :

$$\begin{aligned} (7 \times 19) \bmod 31 &= 133 \bmod 31 = 9 \rightarrow x_1 \\ (7 \times 9) \bmod 31 &= 63 \bmod 31 = 1 \rightarrow x_2 \\ (7 \times 1) \bmod 31 &= 7 \bmod 31 = 7 \rightarrow x_3 \\ (7 \times 7) \bmod 31 &= 49 \bmod 31 = 18 \rightarrow x_4. \end{aligned}$$

□



Étudiez les lignes 13-106 du fichier de script `generation.R` reproduit à la [section 1.11](#). Le code illustre l'effet des différents paramètres d'un générateur congruentiel sur la qualité des nombres pseudo-aléatoires produits. Il démontre également qu'un générateur apparemment de qualité en une dimension peut rapidement se révéler médiocre dans les dimensions supérieures. L'[exercice 1.3](#) reprend ces idées en trois dimensions avec un générateur longtemps considéré standard.



1.7 Indice pour le problème

Un générateur de nombres pseudo-aléatoires distribués uniformément sur $(0, 1)$ pourrait nous permettre de générer les coordonnées en abscisse et en ordonnée de quatre points sur un carré 1×1 .

1.8 Générateurs utilisés dans Excel et R

Avant d'utiliser pour quelque tâche moins importante un générateur de nombres aléatoires inclus dans un logiciel, il importe de s'assurer de la qualité de celui-ci. Vous trouverez en général relativement facilement l'information pertinente dans Internet.

Nous présentons ici, sans entrer dans les détails, les générateurs utilisés dans Excel et R.

1.8.1 Générateur de Excel

La fonction à utiliser dans Microsoft Excel pour obtenir un nombre aléatoire dans l'intervalle $(0, 1)$ est `ALEA()` (dans la version française) ou `RAND()` (dans la version anglaise).

L'historique de Microsoft n'est pas reluisant lorsqu'il s'agit de mise en œuvre de générateurs de nombres aléatoires pour Excel. Au fil des ans, plusieurs articles scientifiques ont démontré les lacunes de la fonction `RAND()`.

Par exemple, dans les versions 2003 et 2007, Microsoft Excel utilisait le générateur de nombres aléatoires Whichmann-Hill. Ce générateur a longtemps été considéré comme le meilleur disponible, mais il a été supplanté ces dernières années. Microsoft prétendait que la période du générateur Whichmann-Hill est 10^{13} , mais omettait de tenir compte de littérature scientifique démontrant qu'elle est plutôt de $6,95 \times 10^{12} \approx 2^{43}$, une période désormais considérée trop courte.

De plus, la mise en œuvre du générateur Whichmann-Hill dans Excel 2003 avait le fâcheux défaut de pouvoir générer des nombres négatifs. Ce défaut a plus tard été corrigé par un *Service Pack* 1 de Office 2003 ainsi que dans la version 2007.

L'article de [McCullough et Heiser \(2008b\)](#) demeure une référence sur les lacunes observées dans la génération de nombres aléatoires et les procédures statistiques dans Excel 2007. De plus, [McCullough et Heiser \(2008a\)](#) démontrent que le générateur de Excel ne saurait être véritablement celui de Whichmann-Hill. Les auteurs écrivent en conclusion :

Twice Microsoft has attempted to implement the dozen lines of code that define the Wichmann and Hill (1982) RNG¹, and twice Microsoft has failed, apparently not using standard methods for verifying that an RNG has been correctly implemented. Consequently, users of Excel's "rand" function have been using random numbers from an unknown and undocumented RNG of unknown period that is not known to pass any standard tests of randomness.

Les critiques relevées dans les articles ci-dessus ont poussé Microsoft à améliorer la précision des procédures statistiques depuis Excel 2010 ([Microsoft Office Blog, 2009](#)). Microsoft affirme également utiliser un nouveau générateur de nombres aléatoires pour la fonction `RAND()`. Seulement, plusieurs années après le lancement du produit, il demeure difficile d'obtenir des détails qui permettraient de dissiper tout doute sur la qualité des nombres aléatoires fournis par Excel.




Si vous programmez en VBA, peut-être serez-vous tenté d'utiliser la fonction `RND()` pour générer un nombre aléatoire. Or, le générateur utilisé par `RND()` est un simple générateur congruentiel linéaire. C'est tout à fait archaïque. De plus, le générateur utilise toujours la même amorce et, par conséquent, les suites de nombres aléatoires sont toujours les mêmes. Pour toute utilisation moindrement sérieuse de nombres aléatoires, vous devez donc à tout prix éviter la fonction `RND()` et de lui préférer un appel à la fonction `RAND()` de Excel.²

1.8.2 Générateurs de R

On obtient des nombres uniformes sur un intervalle quelconque avec la fonction `runif` dans R. La fonction `set.seed` permet de spécifier la valeur de l'amorce du générateur aléatoire, ce qui est utile si nous voulons répéter une simulation absolument à l'identique.

R offre la possibilité de choisir entre plusieurs générateurs de nombres aléatoires différents, ou encore de spécifier son propre générateur. Par dé-

1. *Random Number Generator*

2. L'information concernant le générateur utilisé par VBA se trouvait dans l'article 231847 de la base de connaissances Microsoft, mais ce lien est aujourd'hui obsolète. Vous trouverez l'information équivalente — avec une analyse exhaustive — dans une [entrée de bloque](#)  de la firme australienne de services informatiques Noesis.

faut, R utilise le générateur Marsenne-Twister, considéré comme le plus avancé en ce moment. La période de ce générateur est $2^{19937} - 1$ (rien de moins!) et la distribution des nombres est uniforme dans 623 dimensions consécutives sur toute la période.

Pour de plus amples détails et les dernières informations sur les générateurs disponibles et la procédure de réglage de l'amorce, consulter les rubriques d'aide des fonctions `.Random.seed` et `set.seed`.

❁ 1.9 Indice pour le problème

Générer les coordonnées uniformes des points sera plus simple et plus rapide avec la fonction `runif` de R.

⚡ 1.10 Solution du problème

En premier lieu, écrivons une fonction `is.triangle` pour déterminer si l'enveloppe convexe de quatre points forme un triangle. La fonction prendra en arguments un vecteur `x` des coordonnées en abscisse des quatre points et un vecteur `y` des coordonnées en ordonnée. La fonction retournera `TRUE` ou `FALSE` selon que l'enveloppe convexe est un triangle ou non.

```
> is.triangle <- function(x, y) 3 == length(chull(x, y))
> is.triangle(c(8, 1, 7, 10), c(8, 2, 4, 3))
[1] TRUE
> is.triangle(c(7, 5, 2, 1), c(9, 3, 10, 7))
[1] FALSE
```

L'expression suivante permet de placer quatre points au hasard dans un carré 1×1 , puis de vérifier si leur enveloppe convexe est un triangle.

```
> is.triangle(runif(4), runif(4))
```

Pour répéter cette expression un grand nombre de fois, nous aurons recours, tel qu'expliqué à l'[annexe A](#), à la fonction `replicate`. Puisque chaque expression exécutée par `replicate` retourne `TRUE` ou `FALSE`, le résultat sera un vecteur booléen. La valeur moyenne du vecteur booléen correspond donc à la proportion d'enveloppes convexes triangulaires au fil des 100 000 simulations.

```
> mean(replicate(1E5,
+   is.triangle(x = runif(4), y = runif(4))))
```

```
[1] 0.30438
```

La valeur ci-dessus est suffisamment proche de $\frac{11}{36} \approx 0,30556$ pour que nous jugions l'expérience concluante.

1.11 Code informatique

📄 Fichier d'accompagnement generation.R

```
12 ###
13 ### EFFET DES PARAMÈTRES D'UN GÉNÉRATEUR
14 ###
15
16 ## Définissons tout d'abord une petite fonction pour calculer
17 ## les valeurs successives d'un générateur congruentiel
18 ## linéaire général.
19 rand <- function(n, a, c, m, seed)
20 {
21   x <- numeric(n + 1)
22   x[1] <- seed
23   for (i in seq(n))
24     x[i + 1] <- (a * x[i] + c) %% m
25   x[-1]
26 }
27
28 ## Nous pouvons toujours obtenir une période maximale de m
29 ## avec un générateur congruentiel en posant a = c = 1, mais
30 ## l'aspect aléatoire de la suite de nombres obtenus en prend
31 ## alors pour son rhume...
32 rand(17, a = 1, c = 1, m = 16, seed = 0)
33
34 ## Avec un meilleur choix de multiplicateur et d'incrément, la
35 ## période est tout aussi longue, mais la suite davantage
36 ## aléatoire.
37 rand(17, a = 5, c = 1, m = 16, seed = 0)
38
39 ## Un tout petit changement et la période est beaucoup plus
40 ## courte. Même la valeur de l'amorce a une influence sur la
41 ## période.
42 rand(17, a = 5, c = 4, m = 16, seed = 0)
43 rand(17, a = 5, c = 4, m = 16, seed = 1)
44 rand(17, a = 5, c = 4, m = 16, seed = 2)
45
46 ## Le générateur multiplicatif de l'exemple 1.2 ne satisfait
```

```

47 ## pas les conditions pour que la période soit maximale (7
48 ## n'est pas une racine primitive de 31).
49 rand(32, a = 7, c = 0, m = 31, seed = 19)
50 length(unique(rand(32, a = 7, c = 0, m = 31, seed = 19)))
51
52 ## Avec a = 3, on atteint la période de 30 car 3 est une
53 ## racine primitive de 31.
54 length(unique(rand(32, a = 3, c = 0, m = 31, seed = 19)))
55
56 ###
57 ### ANALYSE DE LA QUALITÉ D'UN GÉNÉRATEUR
58 ###
59
60 ## Un générateur avec une période de 30. Les valeurs obtenues
61 ## semblent assez aléatoires.
62 (x <- rand(30, a = 3, c = 0, m = 31, seed = 19))
63
64 ## Un graphique des valeurs successives ne montre pas de
65 ## structure particulière.
66 par(pch = 19) # disque plein comme symbole
67 plot(x)
68
69 ## Par contre, si nous traçons un graphique des paires de
70 ## valeurs successives (c'est-à-dire x[1] en fonction de x[2],
71 ## x[2] en fonction de x[3], etc.), un portrait dérangent
72 ## apparaît.
73 plot(head(x, -1), tail(x, -1))
74
75 ## Augmenter la valeur du multiplicateur améliore la
76 ## situation. Si nous utilisons plutôt a = 12 (une racine
77 ## primitive), alors la période est toujours de 30 et donc les
78 ## valeurs de la suite sont les mêmes que ci-dessus, dans un
79 ## ordre différent. Cependant, les valeurs sont mieux
80 ## distribuées.
81 x <- rand(30, a = 12, c = 0, m = 31, seed = 19)
82 plot(head(x, -1), tail(x, -1))
83
84 ## Illustrons par six autres graphiques comment le
85 ## choix des paramètres d'un générateur congruentiel
86 ## peut avoir une importance majeure sur la qualité des
87 ## nombres générés.
88 par(mfrow = c(3, 2)) # 6 graphiques sur 3 lignes, 2 colonnes
89
90 x <- rand(2048, a = 65, c = 1, m = 2048, seed = 0)
91 plot(tail(x, -1), head(x, -1), main = "a = 65, c = 1")

```

```

92
93 x <- rand(2048, a = 1365, c = 1, m = 2048, seed = 0)
94 plot(tail(x, -1), head(x, -1), main = "a = 1365, c = 1")
95
96 x <- rand(2048, a = 1229, c = 1, m = 2048, seed = 0)
97 plot(tail(x, -1), head(x, -1), main = "a = 1229, c = 1")
98
99 x <- rand(2048, a = 157, c = 1, m = 2048, seed = 0)
100 plot(tail(x, -1), head(x, -1), main = "a = 157, c = 1")
101
102 x <- rand(2048, a = 45, c = 0, m = 2048, seed = 1234)
103 plot(tail(x, -1), head(x, -1), main = "a = 45, c = 0")
104
105 x <- rand(2048, a = 43, c = 0, m = 2048, seed = 1234)
106 plot(tail(x, -1), head(x, -1), main = "a = 43, c = 0")

```

1.12 Exercices

L'exercice 1.3 requiert d'installer le paquetage `rgl`, disponible dans CRAN;?, chapitre 7 explique la procédure d'installation d'un paquetage R.



L'utilisation de `rgl` sous macOS requiert l'application `X11.app`, qui n'est plus livrée avec le système depuis la version 10.8 (Mountain Lion). Le projet XQuartz fournit un serveur et des bibliothèques client X11 compatibles avec les versions de macOS 10.6 et suivantes. Il suffit de télécharger l'application depuis le [site du projet](#).

- 1.1** Calculer cinq nombres pseudo-aléatoires avec chacun des générateurs congruentiels ci-dessous. Dans tous les cas, $m = 64$. Choisir l'amorce.
- a) $a = 29, c = 17$
 - b) $a = 9, c = 1$
 - c) $a = 13, c = 0$
 - d) $a = 11, c = 0$
- 1.2** a) Composer une fonction R faisant la mise en œuvre du générateur congruentiel multiplicatif avec $m = 2^{13} - 1$ et $a = 17$. Générer 500 nombres pseudo-aléatoire, puis faire un graphique des paires (x_i, x_{i+1}) . Sur combien de lignes les points sont-ils alignés?
- b) Répéter la partie a) avec $a = 85$.

- 1.3 Le générateur RANDU, qui a longtemps été le plus populaire générateur de nombres pseudo-aléatoires, est défini ainsi :

$$x_i = 65\,539x_{i-1} \bmod 2^{31}.$$

Les nombres aléatoires obtenus avec ce générateur présentent une excellente structure aléatoire en une et en deux dimensions. On peut toutefois démontrer mathématiquement qu'en trois dimensions, les triplets (x_i, x_{i+1}, x_{i+2}) se retrouvent sur quinze plans ou moins, rendant ainsi assez prévisible la valeur de x_{i+2} étant donné les deux autres valeurs. Contentons-nous, ici, de constater cet état de fait graphiquement.

- a) Générer une suite $\{x_i\}$ de longueur 20 002 avec le générateur RANDU et poser $u_i = 2^{-31}x_i$. Pour tous les triplets (u_i, u_{i+1}, u_{i+2}) , sélectionner les cas où $0,5 \leq u_{i+1} \leq 0,51$ et faire un graphique de u_{i+2} en fonction de u_i . Commenter le graphique obtenu.
- b) Générer une suite $\{x_i\}$ de longueur 1 002 avec le générateur RANDU. À l'aide de R, placer les triplets (u_i, u_{i+1}, u_{i+2}) dans un graphique en trois dimensions avec la fonction `plot3d` du paquetage `rgl`, puis faire pivoter le graphique pour voir les quinze plans sur lesquels se trouvent répartis les points.

2 Simulation de nombres aléatoires non uniformes

Objectifs du chapitre

- ▶ Développer un algorithme de simulation de nombres non uniformes à partir de la méthode de l'inverse.
- ▶ Développer un algorithme de simulation de nombres non uniformes à partir de la méthode d'acceptation-rejet.
- ▶ Calculer des nombres pseudo-aléatoires non uniformes en suivant un algorithme donné.
- ▶ Utiliser les outils de Excel et R pour la simulation de nombres non uniformes.

Habituellement, les applications de la simulation requièrent des nombres aléatoires provenant non pas d'une distribution $U(0, 1)$, mais plutôt d'une distribution avec fonction de répartition $F_X(x)$. Cela nécessite donc de transformer les nombres uniformes en des nombres provenant de distributions non-uniformes.

Il existe de très nombreux algorithmes pour générer des nombres aléatoires de différentes distributions; voir, par exemple, [Devroye \(1986\)](#). Nous n'en étudierons que deux en détail, soit la méthode de l'inverse et la méthode d'acceptation-rejet. Par ailleurs, la [section 2.7](#) présente des procédures de simulation pour les modèles actuariels les plus courants.

Plusieurs algorithmes de simulation de nombres non uniformes reposent sur des résultats de transformations de variables aléatoires. Par exemple :

- ▶ la somme de α exponentielles est une loi gamma avec paramètre de forme α entier;
- ▶ la loi géométrique est la partie entière de l'exponentielle;
- ▶ la loi F est un ratio de deux khi-carré; etc.

Le lecteur qui ne serait pas à l'aise avec les notions de transformation de variables aléatoires trouvera à l'[annexe B](#) des rappels sur les principales techniques étudiées dans les cours d'analyse probabiliste et d'analyse statistique.



2.1 Énoncé du problème

Au chapitre précédent, nous nous sommes penchés sur le problème des quatre points de Sylvester lorsque la région R est un carré.

Cette fois, nous devons vérifier par simulation que lorsque la région R est un disque, la probabilité que l'enveloppe convexe forme un triangle est $\frac{35}{12\pi^2} \approx 0,29552$.

2.2 Méthode de l'inverse

La méthode de l'inverse repose sur une idée toute simple, soit que nous pouvons transformer des nombres uniformes sur $(0, 1)$ en des nombres provenant de la distribution avec fonction de répartition $F_X(x)$ en utilisant le théorème suivant.

Théorème 2.1. Soit X une variable aléatoire avec fonction de répartition $F_X(x)$. Alors

$$F_X(X) \sim U(0, 1).$$

Démonstration. Soit la transformation $U = F_X(X)$. Alors,

$$\begin{aligned} F_U(u) &= \Pr[U \leq u] \\ &= \Pr[F_X(X) \leq u] \\ &= \Pr[X \leq F_X^{-1}(u)] \\ &= F_X(F_X^{-1}(u)) \\ &= u, \end{aligned}$$

d'où $U \sim U(0, 1)$. □

Par conséquent, si $U \sim U(0, 1)$, alors

$$F_X^{-1}(U) \sim X.$$

La fonction de répartition inverse, F_X^{-1} , est aussi appelée *fonction de quantile*.

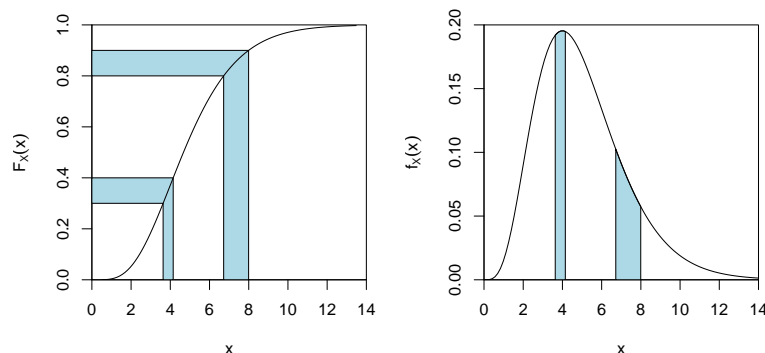


FIG. 2.1 – Représentation graphique de la méthode de l'inverse. À des intervalles égaux en ordonnée correspondent des intervalles différents en abscisse selon la forme de la distribution. La fonction de répartition à gauche correspond à la densité de droite.

Algorithme 2.1 (Méthode de l'inverse). Soit X une variable aléatoire avec fonction de répartition $F_X(x)$. Les étapes ci-dessous permettent de générer une observation x de cette variable aléatoire.

1. Obtenir un nombre u d'une loi $U(0, 1)$.
2. Poser $x = F_X^{-1}(u)$.

La méthode de l'inverse consiste à choisir un nombre uniformément sur l'axe des ordonnées d'une fonction de répartition (donc un nombre entre 0 et 1) et à trouver la valeur correspondante sur l'axe des abscisses telle que donnée par la fonction de quantile. Comme les valeurs en x seront plus concentrées là où la pente de la fonction de répartition est la plus grande, et vice versa, la distribution en abscisse ne sera pas uniforme. Voir la [figure 2.1](#) pour une représentation graphique de ce phénomène.

La méthode de l'inverse en est une bonne si la fonction de quantile est facile à calculer. S'il n'existe pas de forme explicite pour $F_X^{-1}(\cdot)$, résoudre numériquement

$$F_X(x) - u = 0$$

peut s'avérer aussi efficace que bien d'autres méthodes.



Dans Excel, il faut nécessairement utiliser la méthode de l'inverse. Plusieurs fonctions de quantiles sont disponibles; voir la [section 2.6](#).

2.2.1 Distributions continues

La méthode de l'inverse est, en principe du moins, simple à utiliser avec les distributions continues : il suffit de connaître la fonction de quantile et de l'appliquer à des nombres uniformes pour obtenir des nombres de la distribution souhaitée.

Dans les faits, il y a peu de lois de probabilité continues dont la fonction de répartition est simple à inverser (exponentielle, Pareto, Weibull). Il faut parfois utiliser d'autres méthodes.

Exemple 2.1. Nous voulons obtenir un échantillon aléatoire d'une distribution exponentielle de paramètre λ avec fonction de densité de probabilité

$$f(x) = \lambda e^{-\lambda x}, \quad x > 0$$

et fonction de répartition

$$F(x) = 1 - e^{-\lambda x}, \quad x > 0.$$

Or,

$$F^{-1}(u) = -\frac{1}{\lambda} \ln(1 - u),$$

donc

$$X = -\frac{1}{\lambda} \ln(1 - U) \sim \text{Exponentielle}(\lambda),$$

où $U \sim U(0, 1)$. En fait, puisque $U \sim U(0, 1) \Leftrightarrow 1 - U \sim U(0, 1)$, nous pouvons nous contenter de la relation

$$X = -\frac{1}{\lambda} \ln U \sim \text{Exponentielle}(\lambda).$$

Par conséquent, l'algorithme pour simuler des nombres provenant d'une exponentielle de paramètre λ est :

1. Obtenir un nombre u d'une $U(0, 1)$;
2. Poser $x = -\lambda^{-1} \ln u$.



Complétez cet exemple en étudiant les lignes 13-37 du fichier de script `simulation.R` reproduit à la [section 2.9](#).

□

2.2.2 Distributions discrètes

Nous pouvons aussi utiliser la méthode de l'inverse avec les distributions discrètes. Cependant, puisque la fonction de répartition comporte des sauts, son inverse n'existe pas formellement. Par conséquent, il n'existe pas de solution de $u = F_X(x)$ pour certaines valeurs de u , ou alors une infinité de solutions.

Supposons une distribution avec un saut en x_0 et

$$\begin{aligned} F_X(x_0^-) &= a \\ F_X(x_0) &= b > a. \end{aligned}$$

Si $a < u < b$, nous posons $x = x_0$. Ainsi, nous simulerons x_0 dans une proportion $b - a$ du temps, ce qui correspond à $\Pr[X = x_0]$. Voir la [figure 2.2](#) pour une illustration.

Que faire si $u = a$ ou $u = b$? Nous prenons la plus grande valeur de l'intervalle où $F_X(x)$ est constante :

$$\begin{aligned} u = a &\Rightarrow x = x_0 \\ u = b &\Rightarrow x = x_1. \end{aligned}$$

Il faut procéder ainsi parce que plusieurs générateurs produisent des nombres uniformes sur $[0, 1)$.

Exemple 2.2. Soit X une variable aléatoire avec fonction de densité de probabilité

$$f(x) = \begin{cases} 0,5, & 0 \leq x < 1 \\ 0, & 1 \leq x < 2 \\ 0,5, & 2 \leq x < 3. \end{cases}$$

Il s'agit d'une variable aléatoire *mixte* (en partie continue et en partie discrète) dont la fonction de répartition est

$$F(x) = \begin{cases} 0,5x, & 0 \leq x < 1 \\ 0,5, & 1 \leq x < 2 \\ 0,5x - 0,5, & 2 \leq x < 3. \end{cases}$$

Cette fonction est représentée à la [figure 2.3](#).

Un algorithme pour simuler des nombres aléatoires de cette distribution est donc :

1. Obtenir un nombre u d'une loi $U(0, 1)$.

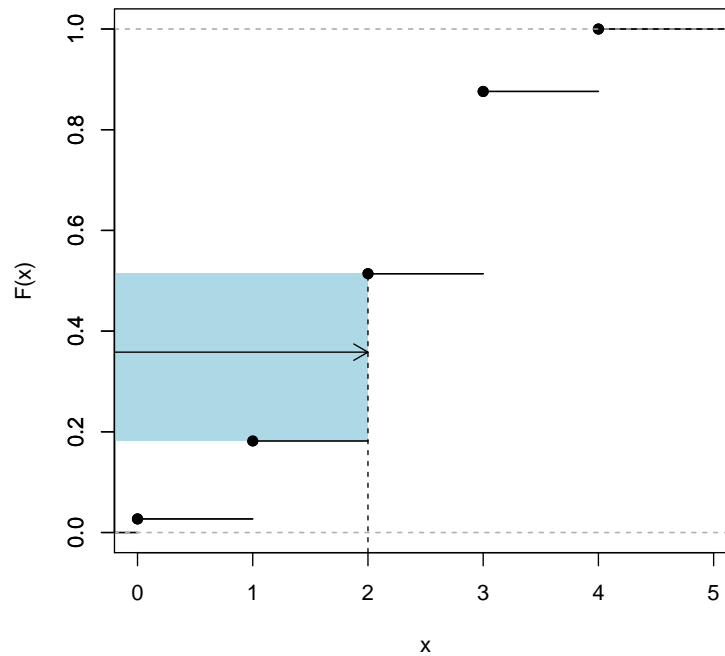


FIG. 2.2 - Illustration de la méthode de l'inverse pour une distribution discrète. Tout nombre uniforme tiré dans la zone colorée est converti en la même valeur, ce qui correspond à la probabilité d'obtenir cette valeur (la hauteur du saut).

2. Poser

$$x = \begin{cases} 2u, & \text{si } 0 \leq u < 0,5 \\ 2, & \text{si } u = 0,5 \\ 2u + 1 & \text{si } 0,5 < u < 1. \end{cases}$$

□

Exemple 2.3. Nous voulons simuler des observations d'une distribution binomiale de paramètres n et θ . Nous pourrions, pour chaque x simulé, faire n expériences de Bernoulli et compter le nombre de succès. Par la méthode de l'inverse pour la loi de Bernoulli, on a un succès si $u \leq \theta$. Par conséquent, un algorithme serait :

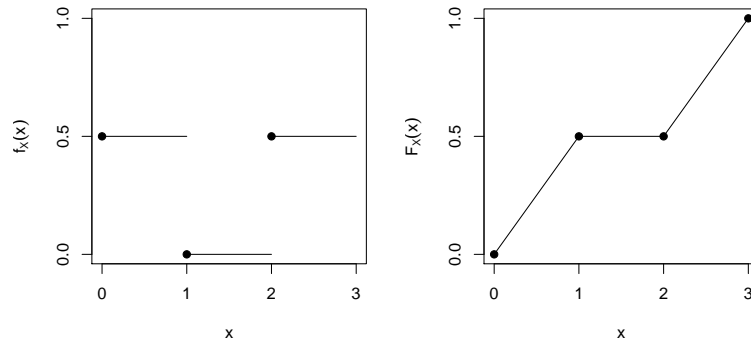


FIG. 2.3 – Fonction de densité de probabilité (gauche) et fonction de répartition (droite) de l'exemple 2.2

1. Simuler n nombres uniformes indépendants u_1, \dots, u_n d'une loi $U(0, 1)$.
2. Poser

$$x = \sum_{i=1}^n I\{u_i \leq \theta\}.$$

Cette technique requiert toutefois de simuler n nombres uniformes pour chaque valeur de x .

Il est plus efficace d'utiliser la méthode de l'inverse directement avec la distribution binomiale. Par exemple, si $n = 4$ et $\theta = 0,5$, on a que

$$\Pr[X = x] = \begin{cases} 0,0625, & x = 0 \\ 0,25, & x = 1 \\ 0,375, & x = 2 \\ 0,25, & x = 3 \\ 0,0625, & x = 4 \end{cases}$$

et

$$\Pr[X \leq x] = \begin{cases} 0, & x < 0 \\ 0,0625, & 0 \leq x < 1 \\ 0,3125, & 1 \leq x < 2 \\ 0,6875, & 2 \leq x < 3 \\ 0,9375, & 3 \leq x < 4 \\ 1, & x \geq 4, \end{cases}$$

d'où l'algorithme suivant :

1. Simuler un nombre u d'une distribution $U(0, 1)$.
2. Déterminer x selon le tableau suivant :

u dans l'intervalle	Valeur de x
$[0, 0,0625)$	0
$[0,0625, 0,3125)$	1
$[0,3125, 0,6875)$	2
$[0,6875, 0,9375)$	3
$[0,9375, 1)$	4

□



La méthode de l'inverse pour les distributions discrètes à support fini est facile à mettre en œuvre dans Excel à l'aide de la fonction RECHERCHEV().



2.3 Indice pour le problème

Au cœur du problème que nous tentons de résoudre, il y a la simulation de points dans un cercle. L'interprétation géométrique est claire.

Il serait également possible d'en faire une interprétation probabiliste : nous voulons simuler des valeurs d'une distribution bidimensionnelle uniforme sur un disque — sans perte de généralité, de rayon 1 et centré à l'origine — dont la fonction de densité conjointe est

$$f_{XY}(x, y) = \frac{1}{\pi}, \quad -1 < x < 1, x^2 + y^2 < 1.$$

Cependant, les méthodes de simulation pour les distributions multidimensionnelles deviennent rapidement complexes, surtout lorsque les variables ne sont pas indépendantes, comme c'est le cas ici. Nous tâcherons donc de trouver une manière plus intuitive d'un point de vue géométrique pour simuler des points sur un disque.

2.4 Méthode acceptation-rejet

Supposons qu'il est compliqué de simuler des réalisations de la variable aléatoire X avec fonction de densité de probabilité $f_X(x)$. Si nous pouvons

trouver une variable aléatoire Y avec fonction de densité de probabilité $g_Y(x)$ pour laquelle la simulation est simple (uniforme, triangle, exponentielle, etc.) et qu'il est possible d'« envelopper » la densité f par un multiple de g , c'est-à-dire que

$$cg_Y(x) \geq f_X(x)$$

pour tout x , alors nous pouvons utiliser l'algorithme d'acceptation-rejet ci-dessous pour générer des observations de la variable aléatoire X .

Algorithme 2.2 (Méthode d'acceptation-rejet). *Soit X une variable aléatoire avec fonction de densité de probabilité $f_X(x)$ et Y une variable aléatoire avec fonction de densité de probabilité $g_Y(x)$. Les étapes ci-dessous permettent de générer un nombre x issu de la distribution f .*

1. Obtenir une réalisation y de distribution avec densité $g_Y(\cdot)$.
2. Obtenir un nombre u d'une $U(0, 1)$.
3. Si

$$u \leq \frac{f_X(y)}{cg_Y(y)},$$

poser $x = y$. Sinon, retourner à l'étape 1.



Puisque, par définition, l'aire sous les densités f et g vaut 1 dans les deux cas, la relation $cg_Y(x) \geq f_X(x)$ ne peut être vraie pour tout x que si $c > 1$.

L'idée de la méthode d'acceptation-rejet consiste à accepter la « bonne proportion » des réalisations de Y comme provenant de X . La principale difficulté avec la méthode d'acceptation-rejet consiste à trouver la fonction enveloppante $cg_Y(x)$.

Dans l'illustration de la [figure 2.4](#), la densité f à support fini est facilement enveloppée par un rectangle. Un nombre y simulé de cette distribution (à toutes fins pratiques une uniforme, ici) est accepté comme provenant de f dans une proportion correspondant au ratio entre la valeur de $f_X(y)$ (les segments pointillés dans la figure) et la valeur de $cg_Y(y)$ (les segments pleins). Vous constaterez aisément que certaines valeurs y seront acceptées plus souvent que d'autres en conformité avec la forme de la densité.

Une autre interprétation est possible. La méthode dit d'accepter la valeur y simulée de la densité g comme provenant de la densité f si

$$ucg_Y(y) \leq f_X(y),$$

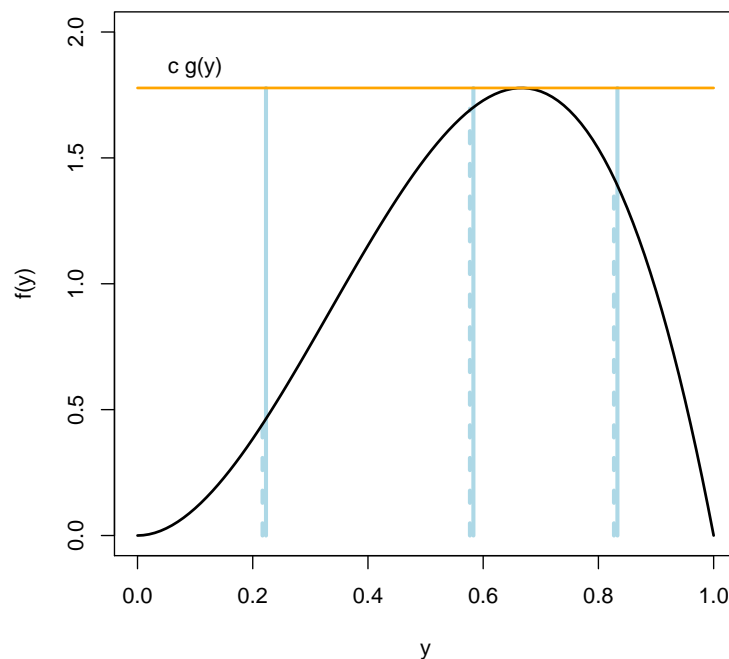


FIG. 2.4 – Illustration de la méthode acceptation-rejet

où u est un nombre issu d'une loi $U(0, 1)$. Graphiquement, cela signifie que nous acceptons la valeur y si le point $(y, u c g_Y(y))$ se trouve sous la courbe f en y et que nous la rejetons s'il se trouve entre f et l'enveloppe. La [figure 2.5](#) illustre cette interprétation.



Évidemment, plus l'enveloppe est près de $f_X(x)$, plus l'algorithme est performant puisque nous rejetons alors moins de valeurs.



Toujours perplexe devant la méthode d'acceptation-rejet ? La [vidéo sur le sujet](#) fournit une illustration graphique qui devrait vous aider à mieux comprendre la procédure. Visionnez-la avant d'aller plus loin.

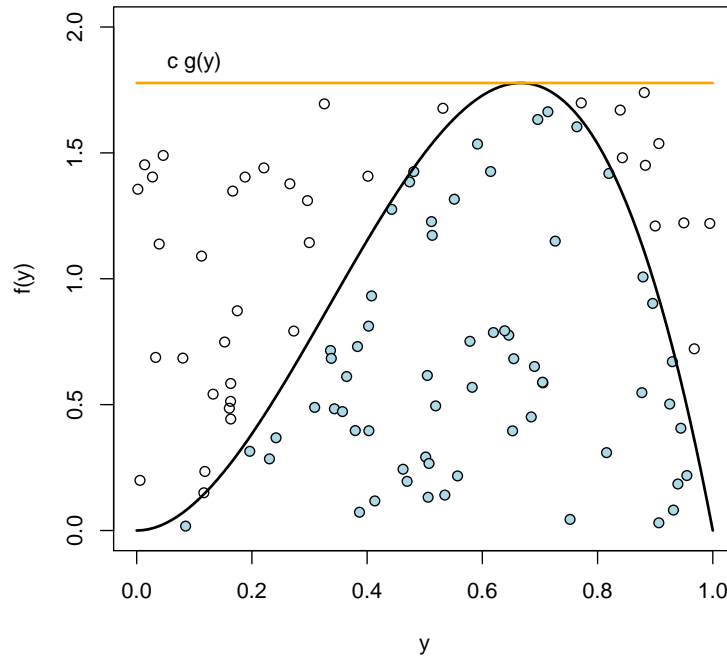


FIG. 2.5 – Interprétation alternative de la méthode acceptation-rejet. Chaque point représente un couple $(y, ucg_Y(y))$. On accepte les points pleins et on rejette les points vides.

Exemple 2.4. Soit $X \sim \text{Bêta}(3, 2)$. La densité de cette distribution est :

$$\begin{aligned} f_X(x) &= \frac{\Gamma(5)}{\Gamma(3)\Gamma(2)} x^{3-1}(1-x)^{2-1} \\ &= 12x^2(1-x), \quad 0 < x < 1. \end{aligned}$$

Elle est représentée aux figures 2.4 et 2.5. Nous pouvons facilement inscrire celle-ci dans un rectangle. Le mode de la densité se trouvant en $x = 2/3$, la hauteur du rectangle est $f(2/3) = 48/27 = 16/9$. Par conséquent,

$$cg_Y(x) = \frac{16}{9}, \quad 0 < x < 1.$$

Nous déduisons que $c = 16/9$ et que la densité g est une uniforme sur $(0, 1)$. Nous avons donc l'algorithme suivant :

1. Simuler deux nombres u_1 et u_2 d'une $U(0, 1)$.
2. Poser $y = u_1$.
3. Si

$$u_2 \leq \frac{f_X(y)}{16/9},$$

alors poser $x = y$. Sinon, retourner à l'étape 1.

L'aire du rectangle étant de $16/9$, nous pouvons nous attendre à rejeter une proportion de

$$\frac{16/9 - 1}{16/9} = \frac{7}{16} \approx 44 \%$$

des nombres u_1 simulés à l'étape 1 de l'algorithme ci-dessus. Si l'enveloppe était plus « serrée » autour de la densité, l'efficacité de l'algorithme en serait augmentée. \square

Exemple 2.5. Reprenons l'exemple 2.4 en tentant d'améliorer l'efficacité de l'algorithme. Il s'avère que nous pouvons inscrire la densité de la loi Bêta(3, 2) dans un triangle aux caractéristiques suivantes (voir la figure 2.6) :

1. l'arête gauche passe par $(0, 0)$, donc est de la forme $y = mx$. Cette droite étant tangente à $f(x)$, la pente m est telle que l'équation $mx = 12x^2(1 - x)$ a une seule racine autre que 0, d'où $y = 3x$;
2. l'arête droite passe par $(1, 0)$, donc est de la forme $y = mx + b$ avec $m + b = 0$. Comme la pente de cette droite est égale à la pente de $f(x)$ en $x = 1$, nous trouvons que $y = 12 - 12x$.

Ainsi,

$$cg_Y(x) = \begin{cases} 3x, & 0 < x < 0,8 \\ 12 - 12x, & 0,8 < x < 1. \end{cases}$$

Or, l'aire du triangle est

$$\frac{(1)cg_Y(0,8)}{2} = \frac{(1)(2,4)}{2} = 1,2,$$

d'où $c = 1,2$. Par conséquent,

$$g_Y(x) = \begin{cases} 2,5x, & 0 < x < 0,8 \\ 10 - 10x, & 0,8 < x < 1. \end{cases}$$

Pour simuler des observations de cette densité par la méthode de l'inverse, nous calculons la fonction de répartition correspondante,

$$G_Y(x) = \begin{cases} 1,25x^2, & 0 < x < 0,8 \\ -5x^2 + 10x - 4, & 0,8 < x < 1, \end{cases}$$

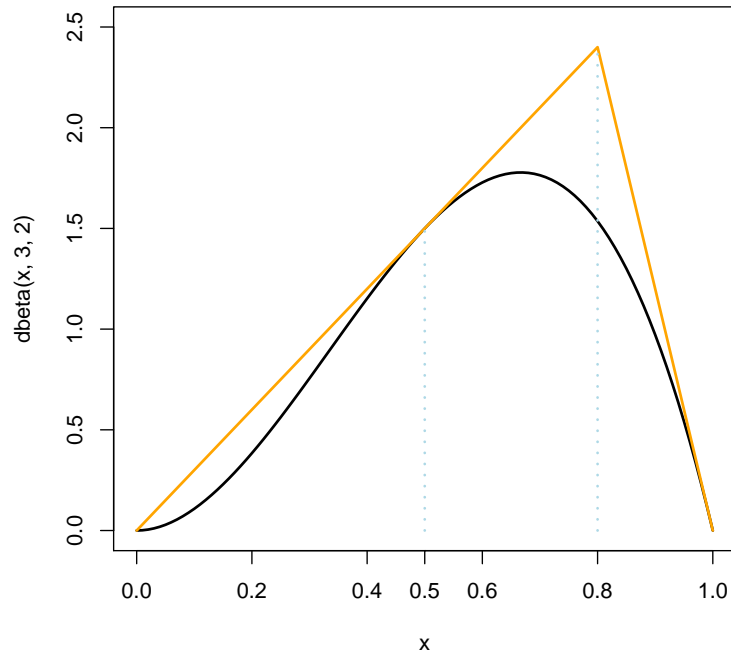


FIG. 2.6 - Fonction de densité de probabilité d'une loi Bêta(3, 2) enveloppée d'un triangle

d'où

$$G_Y^{-1}(y) = \begin{cases} \sqrt{0,8y}, & 0 < y < 0,8 \\ 1 - \sqrt{0,2 - 0,2y}, & 0,8 < y < 1. \end{cases}$$

Au final, nous obtenons l'algorithme suivant :

1. Simuler deux nombres u_1 et u_2 d'une $U(0, 1)$.
2. Poser $y = G_Y^{-1}(u_1)$.
3. Si

$$u_2 \leq \frac{f_X(y)}{1,2g_Y(y)} \Leftrightarrow 1,2g_Y(y)u_2 \leq f_X(y),$$

alors poser $x = y$. Sinon, retourner à l'étape 1.

L'aire du triangle étant de 1,2, nous pouvons maintenant nous attendre à ne rejeter que $0,2/1,2 \approx 17\%$ des nombres simulés, une amélioration importante par rapport à l'exemple 2.4.



Les lignes 40–78 du fichier de script `simulation.R` reproduit à la section 2.9 contiennent une mise en œuvre en R de l'algorithme ci-dessus. Étudiez ce code attentivement.

□

2.5 Indice pour le problème

L'interprétation géométrique de la méthode d'acceptation-rejet illustrée à la figure 2.5 nous met sur la piste d'une manière simple de générer des points distribués uniformément sur un disque.

En effet, il suffit de simuler des points uniformément sur un carré comme nous l'avons déjà fait au chapitre 1, puis de rejeter ceux qui ne se trouvent pas à l'intérieur du disque. La figure 2.7 illustre cette idée : nous accepterions les points pleins et nous rejeterions les points vides.

2.6 Fonctions de simulation dans Excel et R

Il est important de savoir simuler des valeurs d'une variable aléatoire quelconque à partir de la méthode de l'inverse ou d'un autre algorithme, surtout si la distribution de la variable aléatoire est peu usitée. Néanmoins, les différents outils statistiques fournissent en général la majorité des fonctions de simulation de variables aléatoires dont nous pouvons avoir besoin pour un usage quotidien.

2.6.1 Excel

Tel que mentionné à la section 2.2, il faut généralement utiliser la méthode de l'inverse pour simuler des observations de lois de probabilité dans Excel. Cette procédure est facilitée par le fait qu'il existe des fonctions Excel pour calculer la fonction de répartition inverse (ou fonction de quantile) des lois les plus courantes.

Ainsi, Excel fournit la fonction de densité de probabilité (lois continues) ou la fonction de masse de probabilité (lois discrètes) ou la fonction de répartition et, dans certains cas seulement, la fonction de quantile des lois de

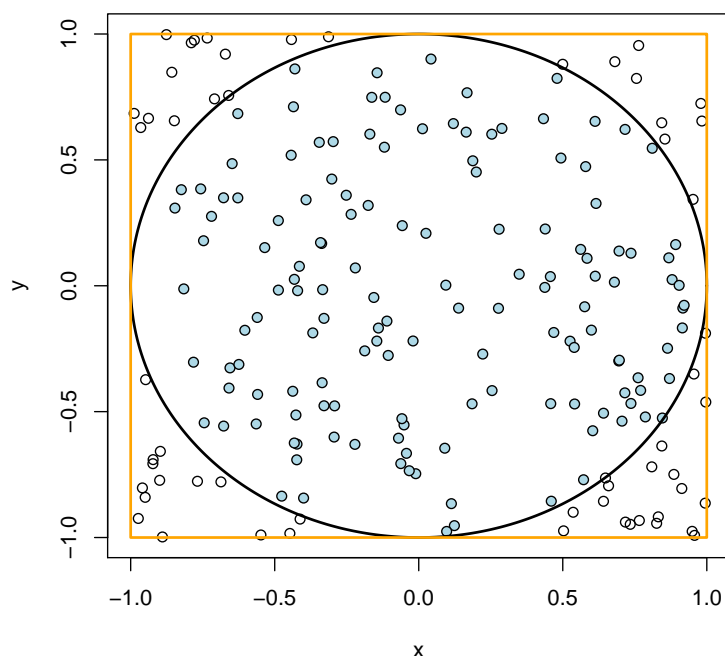


FIG. 2.7 – Simulation de points sur un disque par acceptation-rejet

probabilité présentées au [tableau 2.1](#). Les noms anglais des fonctions ont été modifiés pour être standardisés dans Excel 2010. Vous remarquerez que les traducteurs français ont omis de faire de même.

2.6.2 R

Un large éventail de fonctions donne directement accès aux caractéristiques de plusieurs lois de probabilité dans R. Pour chaque racine $\langle loi \rangle$, il existe quatre fonctions différentes :

1. $d\langle loi \rangle$ calcule la fonction de densité de probabilité (loi continue) ou la fonction de masse de probabilité (loi discrète);
2. $p\langle loi \rangle$ calcule la fonction de répartition;
3. $q\langle loi \rangle$ calcule la fonction de quantile;
4. $r\langle loi \rangle$ simule des observations de cette loi.

Loi	Fonctions Excel (nom anglais)	Fonctions Excel (nom français)
Bêta	BETA.DIST	LOI.BETA.N
	BETA.INV	BETA.INVERSE.N
Binomiale	BINOM.DIST	LOI.BINOMALE.N
	BINOM.INV	LOI.BINOMIALE.INVERSE
Binomiale nég.	NEGBINOM.DIST	LOI.BINOMIALE.NEG.N
Exponentielle	EXPON.DIST	LOI.EXPONENTIELLE.N
<i>F</i> (Fisher)	F.DIST	LOI.F.N
	F.INV	INVERSE.LOI.F.N
Gamma	GAMMA.DIST	LOI.GAMMA.N
	GAMMA.INV	LOI.GAMMA.INVERSE.N
Hypergéom.	HYPGEOM.DIST	LOI.HYPERGEOMETRIQUE.N
Khi carré	CHISQ.DIST	LOI.KHIDEUX
	CHISQ.INV	LOI.KHIDEUX.INVERSE
Log-normale	LOGNORM.DIST	LOI.LOGNORMALE.N
	LOGNORM.INV	LOI.LOGNORMALE.INVERSE.N
Normale	NORM.DIST	LOI.NORMALE.N
	NORM.INV	LOI.NORMALE.INVERSE.N
	NORM.S.DIST	LOI.NORMALE.STANDARD.N
	NORM.S.INV	LOI.NORMALE.STANDARD.INVERSE.N
Poisson	POISSON.DIST	LOI.POISSON.N
<i>t</i> (Student)	T.DIST	LOI.STUDENT.N
	T.INV	LOI.STUDENT.INVERSE.N
Weibull	WEIBULL.DIST	LOI.WEIBULL.N

TAB. 2.1 – Liste des fonctions relatives à des lois de probabilité depuis Excel 2010.

Les différentes lois de probabilité disponibles dans le système R de base, leur racine et le nom de leurs paramètres sont rassemblés au [tableau 2.2](#). Des paquetages fournissent des fonctions pour d'autres lois dont, entre autres, **actuar** (Dutang et collab., 2008) et **SuppDists** (Wheeler, 2013).

Toutes les fonctions du [tableau 2.2](#) sont vectorielles, c'est-à-dire qu'elles acceptent en argument un vecteur de points où la fonction (de densité, de répartition ou de quantile) doit être évaluée et même un vecteur de paramètres. Par exemple, l'expression ci-dessous retourne la probabilité que des lois de Poisson de paramètre 1, 4 et 10 prennent les valeurs 3, 0 et 8, dans

Loi de probabilité	Racine dans R	Noms des paramètres
Bêta	beta	shape1, shape2
Binomiale	binom	size, prob
Binomiale négative	nbinom	size, prob ou mu
Cauchy	cauchy	location, scale
Exponentielle	exp	rate
F (Fisher)	f	df1, df2
Gamma	gamma	shape, rate ou scale
Géométrique	geom	prob
Hypergéométrique	hyper	m, n, k
Khi carré	chisq	df
Logistique	logis	location, scale
Log-normale	lnorm	meanlog, sdlog
Normale	norm	mean, sd
Poisson	pois	lambda
t (Student)	t	df
Uniforme	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

TAB. 2.2 – Loïs de probabilité pour lesquelles il existe des fonctions dans le système R de base

l'ordre.

```
> dpois(c(3, 0, 8), lambda = c(1, 4, 10))
[1] 0.06131324 0.01831564 0.11259903
```

Le premier argument de toutes les fonctions de simulation est la quantité de nombres aléatoires désirée. Ainsi, l'expression suivante retourne trois nombres aléatoires issus de distributions de Poisson de paramètre 1, 4 et 10, respectivement.

```
> rpois(3, lambda = c(1, 4, 10))
[1] 1 1 9
```

Évidemment, spécifier un vecteur comme premier argument d'une fonction de simulation n'a pas tellement de sens, mais, si c'est fait, R retournera une quantité de nombres aléatoires égale à la *longueur* du vecteur (sans égard aux valeurs contenues dans le vecteur).

Autre joueur important dans l'équipe de simulation de R, la fonction `sample` permet de générer des nombres d'une distribution discrète quelconque. Sa syntaxe est

```
sample(x, size, replace = FALSE, prob = NULL)},
```

- ▶ `x` est un vecteur des valeurs possibles de l'échantillon à simuler (le support de la distribution);
- ▶ `size` est la quantité de nombres à simuler;
- ▶ `prob` est un vecteur de probabilités associées à chaque valeur de `x` (par défaut $1/\text{length}(x)$);
- ▶ `replace` détermine si l'échantillonnage s'effectue avec ou sans remise.

En analyse de données, la fonction `sample` est particulièrement utile pour choisir des éléments d'un vecteur ou des lignes d'un tableau au hasard. Le support de la « distribution » dans de tels cas est alors $1, 2, \dots, n$, où n est la longueur du vecteur ou le nombre de lignes du tableau de données.



Le code des lignes 81-165 du fichier de script `simulation.R` reproduit à la section 2.9 contient des exemples plus détaillés d'utilisation des fonctions mentionnées ci-dessus.

2.7 Modèles actuariels

Cette section traite de techniques de simulation pour des distributions couramment utilisées dans la modélisation des risques en sciences actuarielles.

2.7.1 Mélanges discrets

Les *mélanges discrets* (ou finis; *mixture* en anglais) sont utiles pour décrire des phénomènes considérés comme provenant de la combinaison de plusieurs phénomènes qui surviennent chacun avec une probabilité connue d'avance. Par exemple, si nous estimons qu'il y a 90 % de chances que la distribution du nombre de personnes dans une file d'attente soit une Poisson de paramètre 2 et 10 % de chances qu'il s'agisse plutôt d'une Poisson de paramètre 10, alors la fonction de masse de probabilité de la distribution du nombre de personnes dans la file d'attente en tout temps est :

$$\Pr[X = x] = 0,9 \times \frac{2^x e^{-2}}{x!} + 0,1 \times \frac{10^x e^{-10}}{x!}.$$

Cette fonction de masse de probabilité est un mélange discret de deux lois de Poisson.

Définition 2.1. La variable aléatoire Y est un mélange discret à k points des variables aléatoires X_1, X_2, \dots, X_k si sa fonction de répartition est

$$F_Y(x) = p_1 F_{X_1}(x) + p_2 F_{X_2}(x) + \dots + p_k F_{X_k}(x),$$

où $p_j \in [0, 1]$ pour tout $j = 1, \dots, k$ et $\sum_{j=1}^k p_j = 1$.

Dans la suite, nous ne considérerons que les mélanges de $k = 2$ distributions. Lorsque celles-ci sont continues avec fonctions de densité de probabilité f_1 et f_2 , la densité du mélange peut s'écrire

$$f(x) = p f_1(x) + (1 - p) f_2(x).$$

Exemple 2.6. Les mélanges discrets sont très souvent utilisés pour créer de nouvelles distributions aux caractéristiques particulières que l'on ne retrouve pas chez les distributions d'usage courant.

La [figure 2.8](#) montre un mélange de deux distributions log-normales qui résulte en une fonction de densité de probabilité bimodale ayant une queue similaire à celle d'une log-normale. \square

La conception d'un algorithme de simulation d'un mélange discret à deux distributions nécessite de bien comprendre que chaque observation provient de l'une *ou* de l'autre des distributions. Reste, à chaque fois, à déterminer à partir de quelle distribution simuler.

Algorithme 2.3 (Mélange discret de deux distributions). Soit $F(x) = p F_1(x) + (1 - p) F_2(x)$ la fonction de répartition d'un mélange discret des distributions avec fonctions de répartition F_1 et F_2 . Les étapes ci-dessous permettent de générer un nombre aléatoire x de la distribution F .

1. Obtenir un nombre u d'une loi $U(0, 1)$.
2. Si $u \leq p$, obtenir x de la distribution F_1 , sinon obtenir x de F_2 .

Vous aurez reconnu dans le test $u \leq p$, où u provient d'une loi $U(0, 1)$, le cœur de la procédure de simulation d'une loi de Bernoulli mentionnée à l'[exemple 2.3](#). L'algorithme 2.3 revient donc à simuler un nombre de la distribution F_1 si $u \leq p$ et aucun nombre de la distribution F_2 ou, au contraire, aucun nombre de F_1 si $u > p$ et un nombre de F_2 . Cette observation joue un rôle important pour concevoir une procédure de simulation de $n > 1$ valeurs d'un mélange.

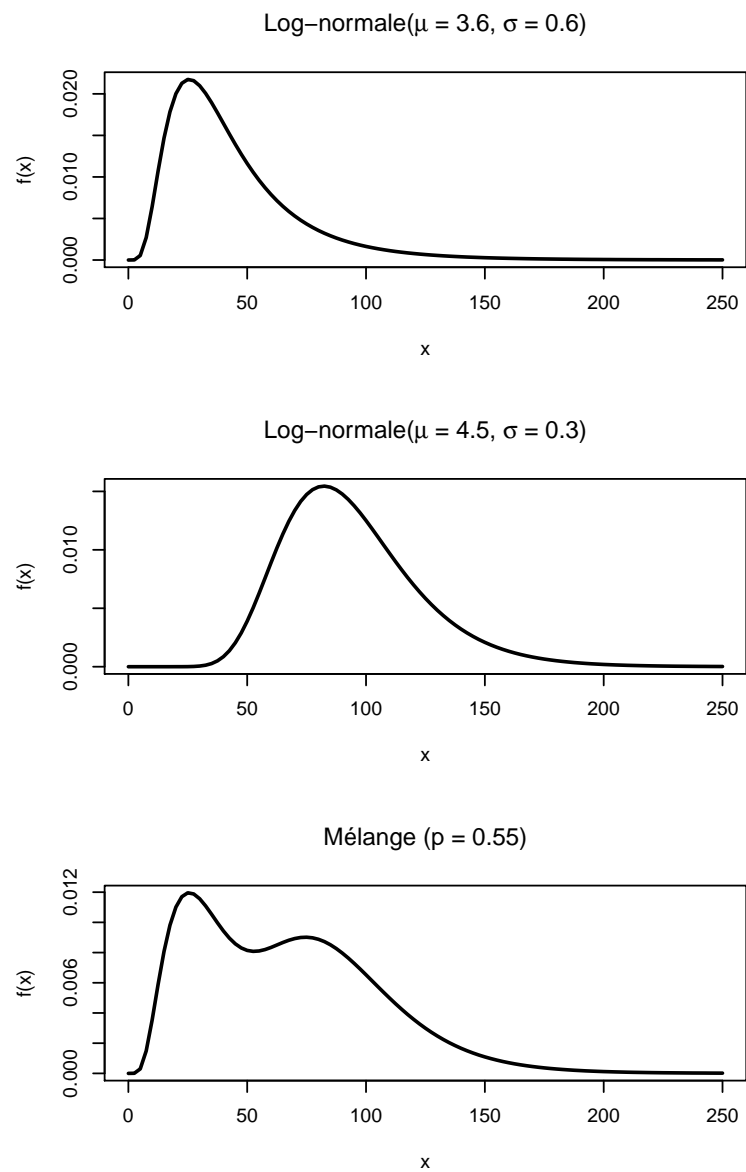


FIG. 2.8 - Fonctions de densité de probabilité de deux log-normales et de leur mélange discret

La fonction `rmixture` du paquetage **actuar** offre une interface conviviale pour obtenir des nombres aléatoires de mélanges discrets.



La vidéo sur la [simulation de mélanges discrets](#) propose une représentation visuelle de l'[algorithme 2.3](#), en plus d'expliquer une erreur commise très fréquemment.

2.7.2 Mélanges continus et modèles hiérarchiques

Nous pouvons étendre le concept de mélange discret à une quantité non dénombrable de distributions. Il en résulte ce que l'on nomme un *mélange continu*. La fonction de densité de probabilité $u(\theta)$ dans la définition ci-dessous joue le rôle des probabilités discrètes p_j de la [définition 2.1](#).

Définition 2.2. Soit X une variable aléatoire dont la distribution dépend d'un paramètre θ . On considère ce paramètre comme une réalisation d'une variable aléatoire Θ avec fonction de densité $u(\theta)$. La fonction de densité de la variable aléatoire $X|\Theta = \theta$ est $f(x|\theta)$. Par la loi des probabilités totales, la fonction de densité marginale (ou non conditionnelle) de la variable aléatoire X est

$$f_X(x) = \int_{\Omega} f(x|\theta)u(\theta) d\theta.$$

Cette distribution est un mélange continu des distributions de $X|\Theta$ et Θ .

Les mélanges continus sont beaucoup utilisés en analyse bayésienne et dans certaines disciplines de l'actuariat. Plusieurs lois de probabilité sont uniquement définies en tant que mélanges.

L'algorithme ci-dessous, très simple, permet d'obtenir des observations aléatoires d'un mélange de distributions.

Algorithme 2.4 (Mélange continu de deux distributions). Soit $f(x)$ la fonction de densité de probabilité du mélange continu des distributions avec densités $f(x|\theta)$ et $u(\theta)$. Les étapes ci-dessous permettent de générer un nombre aléatoire x de la densité f_X .

1. Obtenir un nombre θ issu de la distribution avec densité $u(\theta)$.
2. Obtenir x de la distribution avec densité $f(x|\theta)$.

Pour simuler plusieurs observations d'un mélange, il faut répéter les deux étapes de l'algorithme 2.4 pour *chaque* observation. Le paramètre de mélange (θ) change donc à chaque fois. C'est essentiel, autrement il n'y aurait pas de mélange et nous obtiendrions simplement un échantillon de la distribution $f(x|\theta)$ pour une valeur de θ quelconque.

La simulation de mélanges continus s'avère très simple à réaliser en R grâce à la nature vectorielle des fonctions de simulation.

Exemple 2.7. Soit X une variable aléatoire issue du mélange $X|\Theta = \theta \sim \text{Poisson}(\theta)$ et $\Theta \sim \text{Gamma}(5, 4)$. Les deux expressions R ci-dessous permettent de générer 1 000 observations de X .

```
> theta <- rgamma(1000, 5, 4)
> x <- rpois(1000, theta)
```

Il est tout à fait raisonnable, ici, d'écrire le tout en une seule expression. Cela fait d'ailleurs bien ressortir le fait que le paramètre de la loi de Poisson est lui-même issu d'une simulation.

```
> x <- rpois(1000, rgamma(1000, 5, 4))
```

□

Il est possible de répéter la procédure de mélange plus d'une fois, ce qui donne lieu à ce que nous nommerons des *modèles hiérarchiques*¹. La généralisation de l'[algorithme 2.4](#) pour les modèles hiérarchiques devrait aller de soi.

Exemple 2.8. Soit le modèle hiérarchique suivant où les variables aléatoires Θ et Λ représentent les paramètres de mélange :

$$\begin{aligned} X|\Lambda, \Theta &\sim \text{Poisson}(\Lambda) \\ \Lambda|\Theta &\sim \text{Gamma}(3, \Theta) \\ \Theta &\sim \text{Gamma}(2, 2). \end{aligned}$$

L'expression R suivante permet de générer n observations de la variable aléatoire X .

```
> rpois(n, rgamma(n, 3, rgamma(n, 2, 2)))
```

□



Étudiez les lignes 168-278 du fichier de script `simulation.R` reproduit à la [section 2.9](#). Vous y trouverez une procédure de simulation d'un mélange discret, des exemples d'utilisation de la fonction `rmixture`, ainsi que des exemples de mélanges continus.

1. En sciences actuarielles, cette terminologie est surtout employée dans les contextes de structures de classification des risques sous forme d'arbre. Dans de tels cas, le nombre de valeurs à simuler pourrait différer à chaque niveau de l'arbre.

2.7.3 Distributions composées

Les distributions composées jouent un rôle central dans la modélisation des risques en actuariat puisqu'elles permettent de représenter séparément l'effet de la fréquence et de la sévérité des sinistres dans un portefeuille d'assurance.

Définition 2.3. Soit N une variable aléatoire discrète avec fonction de répartition $P_N(x)$ et X_1, X_2, \dots des variables aléatoires mutuellement indépendantes, identiquement distribuées avec fonction de répartition $F_X(x)$ et indépendantes de la variable aléatoire N . La variable aléatoire $S = X_1 + \dots + X_N$ — avec, par convention, $S = 0$ lorsque $N = 0$ — a une distribution composée. Sa fonction de répartition est

$$F_S(x) = \sum_{n=0}^{\infty} F_X^{*n}(x) p_N(n),$$

où p_N est la fonction de masse de probabilité de N et F_X^{*n} est la fonction de répartition de la n^{e} convolution de X .

Dans l'appellation de la distribution de la variable aléatoire S , l'adjectif « composée » est accolé au nom de la distribution de la variable aléatoire N . Par exemple, si la distribution de N est une loi de Poisson, alors la distribution de S est une Poisson composée. C'est d'ailleurs là le cas le plus fréquent — et de loin — en modélisation des risques d'assurance.

L'algorithme suivant décrit la procédure de simulation d'observations d'une distribution composée.

Algorithme 2.5 (Distribution composée). Soit $S = X_1 + \dots + X_N$ une distribution composée. Les étapes ci-dessous permettent de générer une observation s de la variable aléatoire S .

1. Obtenir un nombre n de la distribution avec fonction de répartition P_N .
2. Obtenir des nombres x_1, \dots, x_n de la distribution avec fonction de répartition F_X .
3. Poser $s = x_1 + \dots + x_n$.

Contrairement à la simulation des mélanges continus, il n'est pas possible, en R, de vectoriser la procédure de simulation de l'algorithme 2.5. Il faut avoir recours à une boucle ou, mieux, à une fonction d'application.

La fonction `rcompound` du paquetage **actuar** permet de générer facilement et intuitivement des nombres aléatoires d'une distribution composée

quelconque. La fonction `rcomppois` offre une interface simplifiée pour le cas le plus courant de la Poisson composée. Enfin, bien que le sujet se trouve hors de la portée du présent document, mentionnons que la très générale fonction `rcomphierarc` permet de générer des observations de modèles hiérarchiques composés.



Le code du fichier de script `simulation.R` reproduit à la [section 2.9](#) fournit, aux lignes 280-345, deux mises en œuvre de l'[algorithme 2.5](#) ainsi que des exemples d'utilisation des fonctions `rcomppois` et `rcompound`.



L'[annexe A](#) propose quelques trucs pour bien planifier une étude de simulation en R.



2.8 Solution du problème

Nous adoptons la stratégie expliquée à l'astuce de la [page 28](#). La fonction R de la [figure 2.9](#) génère des points distribués uniformément sur un disque par la méthode d'acceptation-rejet en simulant d'abord des points sur un carré.

Avec en mains cette fonction, il devient simple de vérifier par simulation la probabilité $\frac{35}{12\pi^2} \approx 0,29552$ en procédant comme au [chapitre 1](#) :

```
[1] 0.29624
```

Une autre stratégie de simulation se révèle toutefois plus efficace. Elle repose sur la simulation des coordonnées polaires (rayon et angle) des points sur le disque, puis de leur transformation en coordonnées cartésiennes. Si un point du disque unité centré à l'origine se trouve à un rayon $r < 1$ du centre et à un angle θ de l'horizontale, alors ses coordonnées cartésiennes sont

$$\begin{aligned}x &= \sqrt{r} \cos \theta \\y &= \sqrt{r} \sin \theta.\end{aligned}$$

La [figure 2.10](#) propose une mise en œuvre de cette méthode en R. La démonstration de la validité de cette méthode fait quant à elle l'objet de l'[exercice 2.8](#).

Parce qu'elle ne requiert aucune boucle, cette seconde méthode est *beaucoup* plus rapide que la méthode d'acceptation-rejet :


```
sim.disque <- function(n)
{
  ## La fonction retourne les coordonnées des points
  ## dans une matrice de deux colonnes. Nous créons
  ## d'abord un contenant.
  X <- matrix(NA, n, 2)

  ## Remplissage de la matrice.
  i <- 1
  repeat
  {
    ## Simulation de coordonnées dans un carré
    ## 2 x 2 centré à l'origine.
    x <- runif(2, -1, 1)

    ## Si les coordonnées sont dans le disque...
    if (sum(x^2) < 1)
    {
      ## ... la paire est ajoutée à la matrice 'X'.
      X[i, ] <- x

      ## Il faut cesser après avoir accepté 'n' paires.
      if (n < (i <- i + 1))
        break
    }
  }
  X
}
```

FIG. 2.9 – Code d’une fonction pour simuler par la méthode d’acceptation-rejet des nombres uniformément sur un disque de rayon 1 centré à l’origine

```
> system.time(sim.disque(1E5))
  user  system elapsed
0.281   0.042   0.322
> system.time(sim.disque2(1E5))
  user  system elapsed
0.009   0.001   0.010
```

```

sim.disque2 <- function(n)
{
  ## Simulation des coordonnées polaires
  r <- runif(n)                # rayon
  angle <- runif(n, 0, 2 * pi) # angle

  ## Transformation en coordonnées cartésiennes
  sqrt(r) * cbind(cos(angle), sin(angle))
}

```

FIG. 2.10 – Code d'une fonction pour simuler des nombres uniformément sur un disque de rayon 1 centré à l'origine en passant par les coordonnées polaires

2.9 Code informatique

✓ Fichier d'accompagnement `simulation.R`

```

12 ###
13 ### MÉTHODE DE L'INVERSE
14 ###
15
16 ## Simulation d'un échantillon aléatoire d'une distribution
17 ## exponentielle par la méthode de l'inverse.
18 lambda <- 5
19 x <- -log(runif(1000))/lambda
20
21 ## Pour faire une petite vérification, nous allons tracer
22 ## l'histogramme de l'échantillon et y superposer la véritable
23 ## densité d'une exponentielle de paramètre lambda. Les deux
24 ## graphiques devraient concorder.
25
26 ## Tracé de l'histogramme. Il faut spécifier l'option 'prob =
27 ## TRUE' pour que l'axe des ordonnées soit gradué en
28 ## probabilités plutôt qu'en nombre de données. Sinon, le
29 ## graphique de la densité que nous allons ajouter dans un
30 ## moment n'apparaîtra pas sur le graphique.
31 hist(x, prob = TRUE) # histogramme gradué en probabilités
32
33 ## Ajoutons maintenant la densité à l'aide de la très utile
34 ## fonction 'curve' qui permet de tracer une fonction f(x)
35 ## quelconque. Avec l'option 'add = TRUE', le graphique est

```

```
36 ## ajouté au graphique existant.
37 curve(dexp(x, rate = lambda), add = TRUE)
38
39 ###
40 ### MÉTHODE ACCEPTATION-REJET
41 ###
42
43 ## Nous effectuons une mise en oeuvre de l'algorithme
44 ## d'acceptation-rejet pour simuler des observations d'une
45 ## distribution Bêta(3, 2). La procédure est intrinsèquement
46 ## itérative, alors nous devons utiliser une boucle. Il y a
47 ## diverses manières de faire, j'ai ici utilisé une boucle
48 ## 'repeat'; une autre mise en oeuvre est présentée dans les
49 ## exercices.
50 ##
51 ## Remarquez que le vecteur contenant les résultats est
52 ## initialisé au début de la fonction pour éviter le Syndrome
53 ## de la plaque à biscuits expliqué dans «Programmer avec R»
54 ## du même auteur.
55 rbeta.ar <- function(n)
56 {
57     x <- numeric(n)      # initialisation du contenant
58     g <- function(x)     # fonction enveloppante
59         ifelse(x < 0.8, 2.5 * x, 10 - 10 * x)
60     Ginv <- function(x)   # l'inverse de son intégrale
61         ifelse(x < 0.8, sqrt(0.8 * x), 1 - sqrt(0.2 - 0.2 * x))
62
63     i <- 0                # initialisation du compteur
64     repeat
65     {
66         y <- Ginv(runif(1))
67         if (1.2 * g(y) * runif(1) <= dbeta(y, 3, 2))
68             x[i <- i + 1] <- y # assignation et incrément
69         if (i > n)
70             break          # sortir de la boucle repeat
71     }
72     x                    # retourner x
73 }
74
75 ## Vérification empirique pour voir si ça fonctionne.
76 x <- rbeta.ar(1000)
77 hist(x, prob = TRUE)
78 curve(dbeta(x, 3, 2), add = TRUE)
79
80 ###
```

```

81 ### FONCTIONS DE SIMULATION DANS R
82 ###
83
84 ## La fonction de base pour simuler des nombres uniformes est
85 ## 'runif'.
86 runif(10)                # sur (0, 1) par défaut
87 runif(10, 2, 5)          # sur un autre intervalle
88 2 + 3 * runif(10)        # équivalent, moins lisible
89
90 ## R est livré avec plusieurs générateurs de nombres
91 ## aléatoires. Nous pouvons en changer avec la fonction
92 ## 'RNGkind'.
93 RNGkind("Wichmann-Hill") # générateur de Excel
94 runif(10)                # rien de particulier à voir
95 RNGkind("default")       # retour au générateur par défaut
96
97 ## La fonction 'set.seed' est très utile pour spécifier
98 ## l'amorce d'un générateur. Si deux simulations sont
99 ## effectuées avec la même amorce, nous obtiendrons exactement
100 ## les mêmes nombres aléatoires et, donc, les mêmes résultats.
101 ## Très utile pour répéter une simulation à l'identique.
102 set.seed(1)              # valeur sans importance
103 runif(5)                 # 5 nombres aléatoires
104 runif(5)                 # 5 autres nombres
105 set.seed(1)              # réinitialisation de l'amorce
106 runif(5)                 # les mêmes 5 nombres que ci-dessus
107
108 ## Plutôt que de devoir utiliser la méthode de l'inverse ou un
109 ## autre algorithme de simulation pour obtenir des nombres
110 ## aléatoires d'une loi de probabilité non uniforme, R fournit
111 ## des fonctions de simulation pour bon nombre de lois. Toutes
112 ## ces fonctions sont vectorielles. Ci-dessous, P == Poisson
113 ## et G == Gamma pour économiser sur la notation.
114 n <- 10                  # taille des échantillons
115 rbinom(n, 5, 0.3)        # Binomiale(5, 0,3)
116 rbinom(n, 1, 0.3)        # Bernoulli(0,3)
117 rnorm(n)                 # Normale(0, 1)
118 rnorm(n, 2, 5)           # Normale(2, 25)
119 rpois(n, c(2, 5))        # P(2), P(5), P(2), ..., P(5)
120 rgamma(n, 3, 2:11)       # G(3, 2), G(3, 3), ..., G(3, 11)
121 rgamma(n, 11:2, 2:11)    # G(11, 2), G(10, 3), ..., G(2, 11)
122
123 ## La fonction 'sample' sert pour simuler d'une distribution
124 ## discrète quelconque. Le premier argument est le support de
125 ## la distribution et le second, la taille de l'échantillon

```

```
126 ## désirée. Par défaut, l'échantillonnage se fait sans remise
127 ## et avec des probabilités égales sur tout le support.
128 sample(1:49, 7)           # numéros pour le 7/49
129 sample(1:10, 10)          # permutation des nombres de 1 à 10
130 sample(1:10)              # idem, plus simple
131 sample(10)                # idem, encore plus simple!
132
133 ## Échantillonnage avec remise.
134 sample(1:10, 10, replace = TRUE)
135
136 ## Distribution de probabilités non uniforme.
137 x <- sample(c(0, 2, 5), 1000, replace = TRUE,
138             prob = c(0.2, 0.5, 0.3))
139 table(x)                  # tableau de fréquences
140
141 ## Insérons des données manquantes de manière aléatoire dans
142 ## le vecteur précédent. Pour ce faire, nous allons tirer
143 ## des positions dans le vecteur avec 'sample' pour ensuite
144 ## remplacer les valeurs à ces positions par 'NA'. Le support
145 ## de la «distribution» dans 'sample' est 1, 2, ..., 1000, une
146 ## suite que l'on génère facilement avec 'seq_along'.
147 x[sample(seq_along(x), 50)] <- NA # 50 données manquantes
148 sum(is.na(x))               # vérification
149 table(x)                   # comparer avec ci-dessus
150
151 ## Le vecteur 'state.name' fourni avec R contient les noms des
152 ## cinquante États américains.
153 state.name
154
155 ## Sélection de 10 États au hasard en combinant indiciage et
156 ## fonction 'sample'.
157 state.name[sample(seq_along(state.name), 10)]
158
159 ## Le tableau de données 'quakes' fourni avec R contient les
160 ## positions de 1000 secousses sismiques de magnitude
161 ## supérieure à 4 autour des Fidji.
162 quakes
163
164 ## Sélection de 10 évènements au hasard.
165 quakes[sample(seq.int(nrow(quakes)), 10), ]
166
167 ###
168 ### MODÈLES ACTUARIELS
169 ###
```

```
170
171 ## Le paquetage actuar contient plusieurs fonctions de
172 ## simulation de modèles actuariels. Nous allons fournir des
173 ## exemples d'utilisation de ces fonctions dans la suite.
174 library(actuar)           # charger le paquetage
175
176 ## MÉLANGES DISCRETS
177
178 ## La clé pour simuler facilement d'un mélange discret en R
179 ## consiste à réaliser que l'ordre des observations est sans
180 ## importance et, donc, que l'on peut simuler toutes les
181 ## observations de la première loi, puis toutes celles de la
182 ## seconde loi.
183 ##
184 ## Reste à déterminer le nombre d'observations qui provient de
185 ## chaque loi. Pour chaque observation, la probabilité qu'elle
186 ## provienne de la première loi est p. Le nombre
187 ## d'observations provenant de la première loi suit donc une
188 ## distribution binomiale de paramètres n et p, où n est le
189 ## nombre total d'observations à simuler.
190 ##
191 ## Voici un exemple de simulation d'observations du mélange
192 ## discret de deux lois log-normales présenté dans le
193 ## chapitre. Les paramètres de la première loi sont 3,6 et
194 ## 0,6; ceux de la seconde loi sont 4,5 et 0,3. Le paramètre de
195 ## mélange est p = 0,55.
196 n <- 10000                # taille de l'échantillon
197 n1 <- rbinom(1, n, 0.55)   # quantité provenant de la loi 1
198 x <- c(rlnorm(n1, 3.6, 0.6), # observations de la loi 1
199       rlnorm(n - n1, 4.5, 0.3)) # observations de la loi 2
200 hist(x, prob = TRUE)      # histogramme
201 curve(0.55 * dlnorm(x, 3.6, 0.6) +
202       0.45 * dlnorm(x, 4.5, 0.3),
203       add = TRUE, lwd = 2, col = "red3") # densité théorique
204
205 ## La fonction 'rmixture' du paquetage actuar offre une
206 ## interface conviviale pour simuler des observations de
207 ## mélanges discrets (avec un nombre quelconque de
208 ## distributions).
209 ##
210 ## La fonction compte trois arguments:
211 ##
212 ## 'n': nombre d'observations à simuler;
213 ## 'probs': vecteur de poids relatif de chaque modèle dans le
214 ## mélange (normalisé pour sommer à 1);
```

```
215 ## 'models': vecteur d'expressions contenant les modèles
216 ##           formant le mélange.
217 ##
218 ## La méthode de formulation des modèles est commune à toutes
219 ## les fonctions de simulation de actuar. Il s'agit de
220 ## fournir, sous forme d'expression non évaluée, des appels à
221 ## des fonctions de simulation en omettant le nombre de
222 ## valeurs à simuler.
223 ##
224 ## Pour illustrer, reprenons l'exemple ci-dessus avec
225 ## 'rmixture'.
226 x <- rmixture(10000, probs = c(0.55, 0.45),
227               models = expression(rlnorm(3.6, 0.6),
228                                   rlnorm(4.5, 0.3)))
229
230 ## Vérifions la validité de la fonction en superposant à
231 ## l'histogramme de l'échantillon la densité théorique du
232 ## mélange.
233 hist(x, prob = TRUE) # histogramme
234 curve(0.55 * dlnorm(x, 3.6, 0.6) +
235        0.45 * dlnorm(x, 4.5, 0.3),
236        add = TRUE, lwd = 2, col = "red3") # densité théorique
237
238 ## Simulation d'un mélange de deux exponentielles (de moyennes
239 ## 1/3 et 1/7) avec poids égal. Le vecteur de poids est
240 ## recyclé automatiquement par 'rmixture'.
241 rmixture(10, 0.5, expression(rexp(3), rexp(7)))
242
243 ## MÉLANGES CONTINUS
244
245 ## La simulation des mélanges continus est simple à faire en R
246 ## puisque les fonctions r<loi> sont vectorielles. Il suffit
247 ## de simuler autant de paramètres de mélange que nous
248 ## souhaitons obtenir d'observations de la distribution
249 ## marginale.
250 ##
251 ## Reprenons ici l'exemple du texte, soit:
252 ##
253 ##   X|Theta = theta ~ Poisson(theta)
254 ##           Theta ~ Gamma(5, 4)
255 ##
256 ## D'abord en deux étapes.
257 theta <- rgamma(1000, 5, 4) # 1000 paramètres de mélange...
258 x <- rpois(1000, theta)     # ... pour 1000 lois de Poisson
259
```

```

260 ## Nous pouvons écrire le tout en une seule expression.
261 x <- rpois(1000, rgamma(1000, 5, 4))
262
263 ## On peut démontrer (faites-le en exercice!) que la
264 ## distribution non conditionnelle de X est une binomiale
265 ## négative de paramètres 5 et  $4/(4 + 1) = 0,8$ .
266 ##
267 ## Vérifions ce résultat empiriquement en calculant d'abord le
268 ## tableau de fréquences des observations de l'échantillon
269 ## avec la fonction 'table', puis en traçant le graphique des
270 ## résultats. Il existe une méthode de 'plot' pour les
271 ## tableaux de fréquences.
272 (p <- table(x)) # tableau de fréquences
273 plot(p/length(x)) # graphique (fréquences relatives)
274
275 ## Ajoutons au graphique les masses de probabilités
276 ## théoriques.
277 (xu <- unique(x)) # valeurs distinctes de x
278 points(xu, dnbinom(xu, 5, 0.8), pch = 21, bg = "red3")
279
280 ## DISTRIBUTIONS COMPOSÉES
281
282 ## La simulation des distributions composées  $S = X_1 + \dots +$ 
283 ##  $X_N$  est une procédure intrinsèquement itérative puisque
284 ## pour chaque observation de la variable aléatoire S, il faut
285 ## d'abord connaître le nombre d'observations à simuler de la
286 ## variable aléatoire X, nombre obtenu de la distribution de
287 ## la variable aléatoire N.
288 ##
289 ## Nous présentons deux techniques «manuelles» avant
290 ## d'illustrer les fonctions 'rcomppois' et 'rcompound' du
291 ## paquetage actuar.
292 ##
293 ## Considérons le cas d'une Poisson composée. Le paramètre de
294 ## Poisson vaut 2 et la distribution de la variable aléatoire
295 ## X est une Gamma(2, 1).
296 ##
297 ## Une première approche repose sur la fonction 'sapply'. Elle
298 ## applique pour chaque observation de la loi de Poisson une
299 ## fonction anonyme qui, elle, génère les observations  $x_1,$ 
300 ## ...,  $x_n$  et calcule la somme  $s = x_1 + \dots + x_n$ .
301 sapply(rpois(10, 2), function(n) sum(rgamma(n, 2, 1)))
302
303 ## Une approche alternative consiste à générer les
304 ## observations de la loi de Poisson et toutes les

```



```
305 ## observations de la loi gamma, à séparer ces dernières en
306 ## catégories à l'aide de facteurs et à effectuer les sommes
307 ## par catégorie. Il faut explicitement tenir compte, dans
308 ## cette procédure, des cas où N = 0.
309 s <- numeric(10)           # contenant avec des 0
310 N <- rpois(10, 2)           # échantillon loi de Poisson
311 x <- rgamma(sum(N), 2, 1)    # échantillon loi gamma
312 f <- rep.int(seq_len(10), N) # facteurs
313 s[which(N != 0)] <- tapply(x, f, sum) # sommes par catégorie
314 s                           # résultats
315
316 ## La seconde approche est moins intuitive que la première,
317 ## mais elle s'avère plus simple à programmer de manière
318 ## générale et, de plus, elle est un peu plus rapide.
319 n <- 1e6
320 system.time(sapply(rpois(n, 2), function(n) sum(rgamma(n, 2, 1))))
321 system.time({
322     s <- numeric(n)
323     N <- rpois(n, 2)
324     x <- rgamma(sum(N), 2, 1)
325     f <- rep.int(seq_len(n), N)
326     s[which(N != 0)] <- tapply(x, f, sum);
327     s})
328
329 ## C'est l'approche utilisée par les fonctions 'rcomppois' et
330 ## 'rcompound' de actuar.
331 rcomppois
332
333 ## Nous pouvons simuler une Poisson composée de manière plus
334 ## intuitive avec la fonction 'rcomppois'. Nous spécifions le
335 ## modèle de variable aléatoire X comme dans 'rmixture'.
336 ## Puisqu'il n'y a qu'un modèle à spécifier, il n'est pas
337 ## nécessaire de le placer dans 'expression'.
338 rcomppois(10, lambda = 2, model.sev = rgamma(2, 1))
339
340 ## La fonction 'rcompound' permet de simuler des distributions
341 ## composées générales où la distribution de fréquence est
342 ## autre que la Poisson. Voici un exemple de binomiale
343 ## négative composée.
344 rcompound(10, model.freq = rnbinom(2, 0.2),
345           model.sev = rgamma(2, 1))
```

2.10 Exercices

- 2.1** La transformation de Box-Muller est populaire pour simuler des nombres normaux à partir de nombres uniformes. Soit $U_1 \sim U(0, 1)$ et $U_2 \sim U(0, 1)$ deux variables aléatoires indépendantes et

$$X_1 = (-2 \log U_1)^{1/2} \cos(2\pi U_2)$$

$$X_2 = (-2 \log U_1)^{1/2} \sin(2\pi U_2).$$

- a) Vérifier de manière heuristique que la transformation ci-dessus est bijective de $\{(u_1, u_2); 0 < u_1 < 1, 0 < u_2 < 1\}$ à $\{(x_1, x_2); -\infty < x_1 < \infty, -\infty < x_2 < \infty\}$, c'est-à-dire qu'elle associe à un point (u_1, u_2) un et un seul point (x_1, x_2) .
- b) Démontrer que la transformation inverse est

$$U_1 = e^{-(X_1^2 + X_2^2)/2}$$

$$U_2 = \frac{1}{2\pi} \arctan \frac{X_2}{X_1}.$$

- c) Démontrer que X_1 et X_2 sont deux variables aléatoires indépendantes chacune distribuée selon une $N(0, 1)$.
- d) Vérifier empiriquement la validité de ces formules à l'aide de Excel ou de R. Dans R, il est possible de transformer les nombres uniformes obtenus avec la fonction `runif` en nombres normaux sans même utiliser de boucles grâce à la fonction `outer` et deux fonctions anonymes.
- 2.2** La distribution de Laplace, ou double exponentielle, est définie comme la différence entre deux distributions exponentielles identiques et indépendantes. Sa fonction de densité de probabilité est

$$f(x) = \frac{\lambda}{2} e^{-\lambda|x|}, \quad -\infty < x < \infty.$$

Proposer une ou plusieurs façons de simuler des nombres issus de cette distribution.

- 2.3** En ouverture de chapitre, nous mentionnons qu'une façon de simuler des nombres issus d'une loi géométrique consiste à extraire la partie entière de nombres aléatoires provenant d'une loi exponentielle. Démontrer que si la loi de la variable aléatoire X est une exponentielle, alors la loi de $\lfloor X \rfloor$ est une géométrique. Déterminer le paramètre de la loi géométrique en fonction du paramètre de la loi exponentielle.

2.4 Réaliser une mise en œuvre informatique de l'algorithme de simulation suivant. Il s'agit d'un algorithme pour simuler des observations d'une loi $\text{Gamma}(\alpha, 1)$, où $\alpha > 1$.

1. Générer u_1 et u_2 indépendamment d'une loi $U(0, 1)$ et poser

$$v = \frac{(\alpha - \frac{1}{6\alpha})u_1}{(\alpha - 1)u_2}.$$

2. Si

$$\frac{2(u_2 - 1)}{\alpha - 1} + v + \frac{1}{v} \leq 2,$$

alors retourner le nombre $x = (\alpha - 1)v$. Sinon, si

$$\frac{2 \log u_2}{\alpha - 1} - \log v + v \leq 1,$$

alors retourner le nombre $x = (\alpha - 1)v$.

3. Répéter au besoin la procédure depuis l'étape 1.

Faire les vérifications empiriques usuelles de la validité de l'algorithme.

2.5 En utilisant le résultat de l'exercice précédent, quelle procédure pourriez-vous suivre pour simuler des nombres d'une loi $\text{Gamma}(\alpha, \lambda)$ où $\alpha > 1$?

2.6 a) Démontrer que si $X|\Theta \sim \text{Exponentielle}(\Theta)$ et $\Theta \sim \text{Gamma}(\alpha, \lambda)$, alors $X \sim \text{Pareto}(\alpha, \lambda)$. La fonction de densité de probabilité d'une loi de Pareto est

$$f(x) = \frac{\alpha \lambda^\alpha}{(x + \lambda)^{\alpha+1}}, \quad x > 0.$$

b) Utiliser le résultat ci-dessus pour proposer un algorithme de simulation de nombres issus d'une loi de Pareto. Faire la mise en œuvre informatique de cet algorithme et les vérifications d'usage de sa validité.

2.7 La fonction de densité de probabilité de la loi de Pareto translatée est

$$f(x) = \frac{\alpha \lambda^\alpha}{x^{\alpha+1}}, \quad x > \lambda.$$

Simuler trois valeurs d'une telle distribution avec $\alpha = 2$ et $\lambda = 1\,000$ à l'aide de la méthode de l'inverse et du générateur congruentiel linéaire suivant :

$$x_n = (65x_{n-1} + 1) \bmod 2\,048.$$

Utiliser une amorce de 12.

- 2.8** Soit U_1 et U_2 deux variables aléatoires indépendantes uniformément distribuées sur l'intervalle $(0, 1)$ et soit la transformation

$$\begin{aligned} X_1 &= \sqrt{U_1} \cos(2\pi U_2) \\ X_2 &= \sqrt{U_1} \sin(2\pi U_2). \end{aligned}$$

Démontrer que la distribution conjointe de X_1 et X_2 est uniforme sur le disque de rayon de 1 centré en $(x_1, x_2) = (0, 0)$. À quoi ce résultat peut-il servir ?

- 2.9** a) Soit $Y_1 \sim \text{Gamma}(\alpha, 1)$ et $Y_2 \sim \text{Gamma}(\beta, 1)$ deux variables aléatoires indépendantes. Démontrer que

$$X = \frac{Y_1}{Y_1 + Y_2} \sim \text{Bêta}(\alpha, \beta).$$

- b) Utiliser le résultat en a) pour proposer un algorithme de simulation d'observations d'une loi Bêta(α, λ).
- c) Faire la mise en œuvre informatique de l'algorithme en b) ainsi que les vérifications d'usage.

- 2.10** a) Dans la méthode d'acceptation-rejet, un nombre y tiré d'une variable aléatoire Y avec fonction de densité de probabilité $g_Y(\cdot)$ est accepté comme réalisation d'une variable aléatoire X avec fonction de densité de probabilité $f_X(\cdot)$ si

$$U \leq \frac{f_X(y)}{c g_Y(y)},$$

où $U \sim U(0, 1)$. Calculer la probabilité d'accepter une valeur lors de toute itération de la méthode d'acceptation-rejet, c'est-à-dire

$$\Pr \left[U \leq \frac{f_X(Y)}{c g_Y(Y)} \right].$$

Astuce : utiliser la loi des probabilités totales en conditionnant sur $Y = y$.

- b) Déterminer la distribution du nombre d'essais avant d'accepter un nombre y dans la méthode d'acceptation-rejet.
- c) Déterminer le nombre moyen d'essais avant d'accepter un nombre y dans la méthode d'acceptation-rejet.

2.11 Soit X une variable aléatoire continue définie sur l'intervalle (a, b) , où a et b sont des nombres réels. Pour simuler des observations de cette variable aléatoire par la méthode d'acceptation-rejet, il est toujours possible d'inscrire la fonction de densité de probabilité de X dans un rectangle de hauteur M , où M est la valeur de la densité au mode de X .

- Énoncer l'algorithme d'acceptation-rejet découlant d'une telle procédure.
- Calculer l'*efficacité* de l'algorithme en a), soit la probabilité d'accepter une valeur lors d'une itération de l'algorithme.

2.12 Effectuer les calculs ci-dessous dans le contexte de la simulation d'observations d'une loi Bêta(3, 2) à l'aide de la méthode d'acceptation-rejet.

- Calculer l'efficacité de l'algorithme développé à l'exercice 2.11 et à l'exemple 2.4 pour le cas présent.
- Calculer l'efficacité de l'algorithme développé dans l'exemple 2.5, où nous avons déterminé que

$$f_X(x) \leq \begin{cases} 3x, & 0 < x < 0,8 \\ 12 - 12x, & 0,8 \leq x < 1. \end{cases}$$

- Faire la mise en œuvre informatique de l'algorithme le plus efficace entre celui de la partie a) et celui de la partie b). Vérifier la fonction en superposant l'histogramme d'un grand échantillon obtenu avec cette fonction et la vraie fonction de densité de la loi bêta.

2.13 La fonction `R` de la figure 2.11 permet de simuler des observations de la distribution Bêta(α, β).

- Identifier le type d'algorithme utilisé dans cette fonction.
- Vous disposez des valeurs suivantes, obtenues dans `R`.

```
> set.seed(12345)
> runif(10)
[1] 0.72 0.88 0.76 0.89 0.46 0.17 0.33 0.51 0.73
[10] 0.99
```

Évaluer le résultat des expressions ci-dessous.

```
> set.seed(12345)
> simul(2, alpha = 2, beta = 3)
```

```

simul <- function(n, alpha, beta)
{
  xmax <- (alpha - 1)/(alpha + beta - 2)
  M <- dbeta(xmax, alpha, beta)
  x <- numeric(n)
  i <- 0
  repeat
  {
    u <- runif(1)
    if (M * runif(1) <= dbeta(u, alpha, beta))
      x[i <- i + 1] <- u
    if (i == n)
      break
  }
  x
}

```

FIG. 2.11 – Fonction de simulation d'une loi Bêta(α, β) pour l'exercice 2.13

2.14 a) Démontrer que, si $0 < \alpha < 1$,

$$x^{\alpha-1}e^{-x} \leq \begin{cases} x^{\alpha-1}, & 0 \leq x \leq 1 \\ e^{-x}, & x > 1. \end{cases}$$

b) Développer un algorithme d'acceptation-rejet pour simuler des observations d'une loi Gamma($\alpha, 1$), $0 < \alpha < 1$ à partir du résultat en a).

2.15 Soit l'inégalité suivante, valide pour $\alpha \geq 1$:

$$x^{\alpha-1}e^{-x} \leq \alpha^{\alpha-1}e^{-x/\alpha+1-\alpha}, \quad x > 0.$$

Utiliser cette inégalité pour justifier l'algorithme d'acceptation-rejet ci-dessous servant à simuler des observations d'une loi Gamma($\alpha, 1$) avec $\alpha \geq 1$.

1. Simuler deux observations indépendantes v_1 et v_2 d'une loi exponentielle de moyenne 1.
2. Si $v_2 < (\alpha - 1)(v_1 - \ln v_1 - 1)$, poser $x = \alpha v_1$. Sinon, retourner à l'étape 1.

2.16 Dans la mise en œuvre de l’algorithme 2.3 de simulation d’un mélange discret présentée à la section 2.9, nous générons un nombre d’une distribution binomiale pour déterminer la quantité de nombres à simuler de la première distribution du mélange (et, conséquemment, de celle de la seconde distribution). Généraliser le concept pour simuler des observations d’un mélange discret de $k > 2$ distributions. *Astuce* : consulter le code source de la fonction `rmixture`.

2.17 Écrire des expressions R pour simuler des observations des modèles ci-dessous d’abord sans, puis avec, les fonctions dédiées du paquetage **actuar**, le cas échéant.

- a) Variable aléatoire dont la fonction de densité est $f(x) = 0,3f_1(x) + 0,7f_2(x)$, où f_1 est la densité d’une distribution exponentielle de moyenne $1/2$ et f_2 est la densité d’une distribution gamma de paramètre $\alpha = 2$ et de moyenne 6 .
- b) Mélange $f(x) = 0,5647I\{x = 0\} + 0,4353(2^x e^{-2}/x!)$, où $I\{\mathcal{A}\}$ est une fonction indicatrice qui vaut 1 lorsque la condition \mathcal{A} est vraie et 0 sinon.
- c) Variable aléatoire X définie ainsi : $X|\Theta = \theta \sim \text{Poisson}(\theta)$, $\Theta \sim \text{Inverse gaussienne}(2, 1)$. La fonction de densité de probabilité de la distribution inverse gaussienne de moyenne μ et de dispersion ϕ est

$$f(x) = \left(\frac{1}{2\pi\phi x^3} \right)^{1/2} \exp \left(-\frac{(x - \mu)^2}{2\mu^2\phi x} \right), \quad x \geq 0, \mu > 0, \phi > 0.$$

Astuce : la fonction `rinvgauss` du paquetage **actuar** permet de simuler des observations de la distribution inverse gaussienne.

- d) Distribution composée $S = X_1 + \dots + X_N$ avec $N \sim \text{Poisson}(2)$ et $X \sim \text{Log-normale}(7, 1)$.
- e) Binomiale négative composée où les paramètres de la binomiale négative sont $r = 2$ et $p = 0,3$, et la distribution de la variable aléatoire X est une log-normale de paramètres $(7, 1)$.

Réponses

2.3 $p = 1 - e^{-\lambda}$

2.7 1 271, 2 172, 1 137

2.10 a) $1/c$ b) Géométrique($1/c$) commençant à 1 c) c

2.11 a) $1/(M(b-a))$

2.12 a) $9/16$ b) $4/5$

2.13 b) $(0,46, 0,33)$

3 Intégration Monte Carlo

Objectifs du chapitre

- ▶ Définir le principe du calcul de la valeur d'une intégrale définie à l'aide de nombres aléatoires.
- ▶ Calculer la valeur d'une intégrale définie par la méthode Monte Carlo.

On appelle simulation Monte Carlo (ou méthode de Monte Carlo) toute méthode consistant à résoudre une expression mathématique à l'aide de nombres aléatoires. Par extension, l'appellation est souvent utilisée pour référer à toute utilisation de la simulation.

L'une des utilisations de la simulation Monte Carlo la plus répandue est le calcul d'intégrales, principalement pour des dimensions supérieures à un. Nous ne présentons ici que l'idée générale.



3.1 Énoncé du problème

L'intégrale de Gauss

$$\int e^{-x^2} dx$$

intervient dans plusieurs disciplines des mathématiques. En théorie des probabilités, on la retrouve à la base de la définition de la loi normale, ou loi gaussienne. Sans primitive, elle sert souvent, en calcul intégral, comme exemple d'intégrale définie qu'il faut résoudre par la transformation du domaine en coordonnées polaires.

Nous savons que

$$G = \int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}.$$

Comment pourrions-nous vérifier ce résultat à l'aide de la simulation ?

3.2 Contexte

Nous souhaitons calculer l'intégrale

$$\theta = \int_a^b h(x) dx, \quad (3.1)$$

mais celle-ci est trop compliquée pour être évaluée explicitement. Nous pourrions d'abord faire deux approximations de l'aire sous la fonction h à l'aide des sommes de Riemann

$$R_n^- = \sum_{k=0}^{n-1} h(a + k\Delta x) \Delta x$$

et

$$R_n^+ = \sum_{k=1}^n h(a + k\Delta x) \Delta x,$$

où $\Delta x = (b - a)/n$ et n est grand. Voir la [figure 3.1](#) pour une illustration de ces deux aires.

Par la suite, une estimation numérique de l'intégrale serait la moyenne des deux sommes de Riemann :

$$\hat{\theta}_n = \frac{R_n^- + R_n^+}{2}.$$

Cette procédure devient toutefois rapidement compliquée pour les intégrales multiples à plusieurs dimensions. Il y a éventuellement beaucoup trop de points à évaluer.

3.3 Méthode Monte Carlo

L'idée de l'intégration Monte Carlo consiste à évaluer la fonction à intégrer en des points choisis aléatoirement, puis à s'en remettre à la Loi des grands nombres pour estimer l'intégrale.

Nous exprimons tout d'abord la fonction $h(x)$ sous la forme

$$h(x) = g(x)f(x),$$

où $f(x)$ est une densité sur (a, b) . Ainsi, par définition de l'espérance :

$$\begin{aligned} \theta &= \int_a^b g(x)f(x) dx \\ &= E[g(X)], \end{aligned}$$

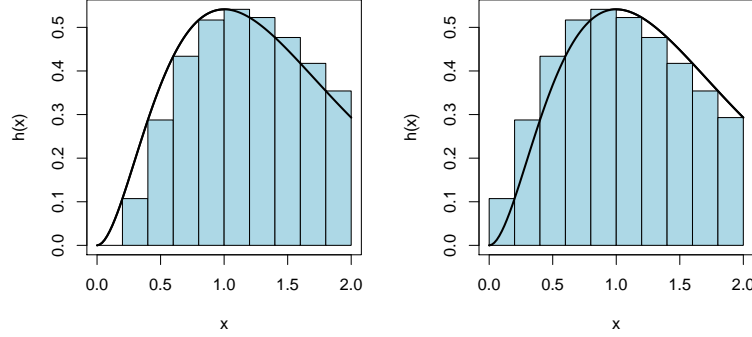


FIG. 3.1 – Approximation de l'aire sous une fonction par les deux sommes de Riemann R_n^- (gauche) et R_n^+ (droite)

où X est la variable aléatoire avec fonction de densité de probabilité $f(x)$. Par la suite, si x_1, \dots, x_n sont des observations simulées de la densité $f(x)$, alors

$$\hat{\theta}_n = \frac{1}{n} \sum_{i=1}^n g(x_i) \quad (3.2)$$

constitue une estimation de l'intégrale θ . Par la Loi des grands nombres, $\hat{\theta}_n \xrightarrow{n \rightarrow \infty} \theta$.

Par souci de simplicité et parce qu'il est facile d'obtenir un échantillon aléatoire d'une loi uniforme, nous posons en général $X \sim U(a, b)$, soit

$$f(x) = \frac{1}{b-a}, \quad a < x < b.$$

Dans ce cas, $g(x) = (b-a)h(x)$ et l'intégrale de départ (3.1) se réécrit

$$\begin{aligned} \theta &= (b-a) \int_a^b h(x) \frac{1}{b-a} dx \\ &= (b-a)E[h(X)]. \end{aligned} \quad (3.3)$$

L'estimation (3.2) de l'intégrale devient alors

$$\hat{\theta}_n = \frac{b-a}{n} \sum_{i=1}^n h(x_i),$$

où x_1, \dots, x_n est un échantillon aléatoire d'une loi $U(a, b)$.

Vous remarquerez qu'il est équivalent d'effectuer le changement de variable

$$\begin{aligned} u &= \frac{x-a}{b-a} & \Leftrightarrow & & x &= a + (b-a)u \\ du &= \frac{dx}{b-a} & & & dx &= (b-a) du \end{aligned}$$

dans l'intégrale (3.3). On obtient alors

$$\begin{aligned} \theta &= (b-a) \int_0^1 h(a + (b-a)u) (1) du \\ &= (b-a) E[h(a + (b-a)U)], \end{aligned}$$

où $U \sim U(0, 1)$. Une estimation de l'intégrale est donc

$$\hat{\theta}_n = \frac{b-a}{n} \sum_{i=1}^n h(a + (b-a)u_i),$$

où u_1, \dots, u_n est un échantillon aléatoire d'une loi $U(0, 1)$. Nous devons utiliser la technique du changement de variable lorsque le domaine est infini.

Exemple 3.1. Soit l'intégrale

$$\theta = \int_2^5 x^{11/5} e^{-x/10} dx.$$

Parce que l'exposant de x n'est pas un entier, nous ne pouvons évaluer cette intégrale par parties. Cependant, nous remarquons que la fonction à intégrer est, à une constante près, la densité d'une loi gamma :

$$\begin{aligned} \theta &= \int_2^5 x^{11/5} e^{-x/10} dx \\ &= \frac{\Gamma\left(\frac{16}{5}\right)}{\left(\frac{1}{10}\right)^{16/5}} \int_2^5 \frac{\left(\frac{1}{10}\right)^{16/5}}{\Gamma\left(\frac{16}{5}\right)} x^{16/5-1} e^{-x/10} dx \\ &= \frac{\Gamma\left(\frac{16}{5}\right)}{\left(\frac{1}{10}\right)^{16/5}} \left[G\left(5; \frac{16}{5}, \frac{1}{10}\right) - G\left(2; \frac{16}{5}, \frac{1}{10}\right) \right], \end{aligned}$$

où $G(x; \alpha, \lambda)$ est la fonction de répartition de la loi $\text{Gamma}(\alpha, \lambda)$. R nous fournit la valeur exacte de l'intégrale.

```
> gamma(3.2) * diff(pgamma(c(2, 5), 3.2, 0.1)) / 0.1^3.2
[1] 34.51998
```

Pour utiliser la méthode Monte Carlo, nous posons

$$\theta = 3 \int_2^5 x^{11/5} e^{-x/10} \left(\frac{1}{3}\right) dx.$$

Si $\{x_1, \dots, x_n\}$ est un échantillon aléatoire d'une loi $U(2, 5)$, alors

$$\hat{\theta}_n = \frac{3}{n} \sum_{i=1}^n x_i^{11/5} e^{-x_i/10}$$

est une estimation de θ . Nous obtenons les résultats suivants avec R (voir aussi la [figure 3.2](#)).

```
> f <- function(x) x^2.2 * exp(-x/10)
> x <- runif(1e2, 2, 5)
> 3 * mean(f(x))
[1] 35.12134
> x <- runif(1e3, 2, 5)
> 3 * mean(f(x))
[1] 34.00328
> x <- runif(1e4, 2, 5)
> 3 * mean(f(x))
[1] 34.83437
> x <- runif(1e5, 2, 5)
> 3 * mean(f(x))
[1] 34.57984
> x <- runif(1e6, 2, 5)
> 3 * mean(f(x))
[1] 34.51065
```



Étudiez le code R pour effectuer les calculs et les graphiques ci-dessus aux lignes 13-47 du fichier de script `montecarlo.R` reproduit à la [section 3.7](#).

□

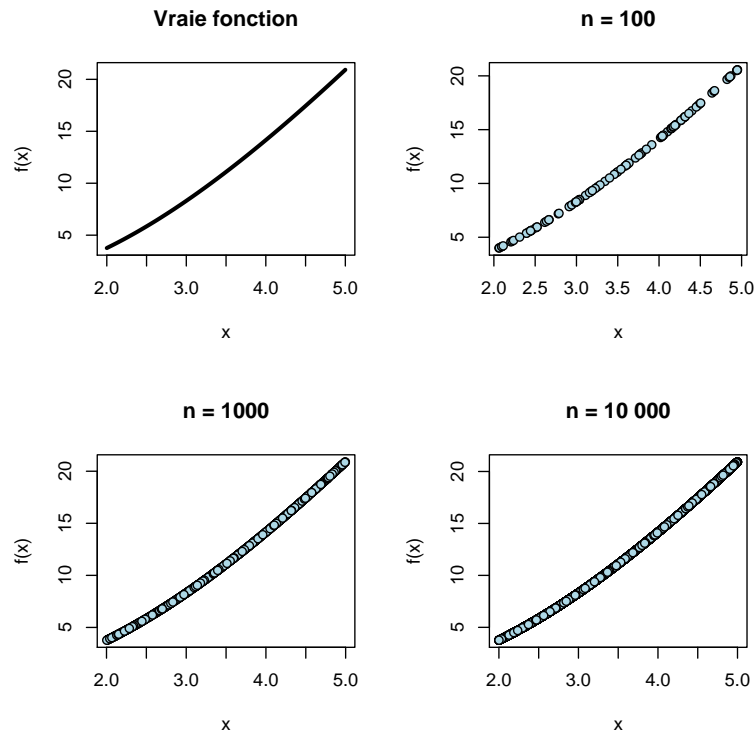


FIG. 3.2 – Fonction $h(x)$ de l'exemple 3.1 et points choisis aléatoirement où la fonction est évaluée pour l'intégration Monte Carlo



3.4 Indice pour le problème

La présentation ci-dessus repose sur un domaine d'intégration fini. Or, celui de l'intégrale à calculer est infini. Pour utiliser des nombres aléatoires uniformes dans la méthode Monte-Carlo, il faut ramener le domaine d'intégration infini à un domaine fini par le biais d'un changement de variable.

Tout d'abord, par symétrie de l'intégrande,

$$\begin{aligned}
 G &= \int_{-\infty}^{\infty} e^{-x^2} dx \\
 &= 2 \int_0^{\infty} e^{-x^2} dx \\
 &= 2 \left(\int_0^1 e^{-x^2} dx + \int_1^{\infty} e^{-x^2} dx \right).
 \end{aligned}$$

Ensuite, plusieurs changements de variable sont possibles pour transformer le domaine de la seconde intégrale de droite. Nous choisissons le changement de variable $u = x^{-1}$. Le domaine d'intégration subit donc la transformation $(1, \infty) \mapsto (1, 0)$. Ainsi,

$$\begin{aligned} G &= 2 \left(\int_0^1 e^{-x^2} dx + \int_1^0 e^{-u^{-2}} (-u^{-2}) du \right) \\ &= 2 \left(\int_0^1 e^{-x^2} dx + \int_0^1 e^{-u^{-2}} u^{-2} du \right) \\ &= 2 \int_0^1 (e^{-x^2} + e^{-x^{-2}} x^{-2}) dx. \end{aligned}$$

Exprimée ainsi, l'intégrale se prête à la méthode d'intégration Monte Carlo.

3.5 Intégrales multiples

La procédure à suivre avec les intégrales multiples est la même qu'avec les intégrales simples, sauf que nous tirons des points uniformément dans autant de dimensions que nécessaire.

Exemple 3.2. Soit l'intégrale

$$\theta = \int_0^{5/4} \int_0^{5/4} \sqrt{4 - x^2 - y^2} dy dx.$$

Posons :

$$\begin{aligned} \theta &= \int_0^{5/4} \int_0^{5/4} \sqrt{4 - x^2 - y^2} dy dx \\ &= \frac{25}{16} \int_0^{5/4} \int_0^{5/4} \sqrt{4 - x^2 - y^2} \frac{1}{\left(\frac{5}{4}\right) \left(\frac{5}{4}\right)} dy dx \\ &= \frac{25}{16} E[\sqrt{4 - X^2 - Y^2}], \end{aligned}$$

où X et Y sont des variables aléatoires indépendantes distribuées uniformément sur l'intervalle $(0, \frac{5}{4})$. Autrement dit, la densité conjointe de X et Y est uniforme sur $(0, \frac{5}{4}) \times (0, \frac{5}{4})$. Par conséquent, une estimation de θ calculée à partir d'un échantillon $\{(x_1, y_1), \dots, (x_n, y_n)\}$ tiré de cette loi conjointe est

$$\hat{\theta}_n = \frac{25}{16n} \sum_{i=1}^n \sqrt{4 - x_i^2 - y_i^2}.$$

Voici quelques résultats obtenus avec R (voir aussi la [figure 3.3](#)).

```

> u <- runif(1e2, 0, 1.25)
> v <- runif(1e2, 0, 1.25)
> mean(sqrt(4 - u^2 - v^2)) * 1.25^2
[1] 2.649276
> u <- runif(1e3, 0, 1.25)
> v <- runif(1e3, 0, 1.25)
> mean(sqrt(4 - u^2 - v^2)) * 1.25^2
[1] 2.682893
> u <- runif(1e4, 0, 1.25)
> v <- runif(1e4, 0, 1.25)
> mean(sqrt(4 - u^2 - v^2)) * 1.25^2
[1] 2.668216

```



Étudiez le code R pour effectuer les calculs et les graphiques ci-dessus aux lignes [50-110](#) du fichier de script `montecarlo.R` reproduit à la [section 3.7](#).

□

⚡ 3.6 Solution du problème

En posant

$$\begin{aligned}\theta &= 2 \int_0^1 (e^{-x^2} + e^{-x^{-2}} x^{-2}) dx \\ &= 2E[e^{-U^2} + e^{-U^{-2}} U^{-2}],\end{aligned}$$

où $U \sim U(0, 1)$, nous avons réécrit l'intégrale de Gauss comme l'espérance d'une transformation d'une variable aléatoire uniforme. Par la Loi des grands nombres, une bonne estimation de θ est

$$\hat{\theta}_n = \frac{2}{n} \sum_{i=1}^n (e^{-u_i^2} + e^{-u_i^{-2}} u_i^{-2})$$

pour n grand et où u_1, \dots, u_n sont des nombres aléatoires distribués uniformément sur l'intervalle $(0, 1)$.

Le calcul est ensuite très simple à faire avec R.

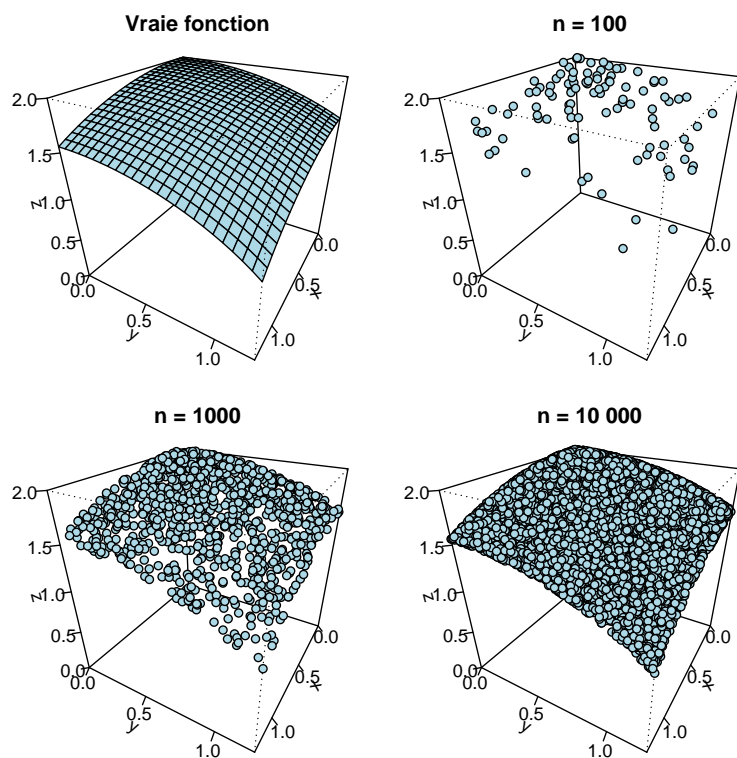


FIG. 3.3 – Fonction $h(x)$ de l'exemple 3.2 et points choisis aléatoirement où la fonction est évaluée pour l'intégration Monte Carlo

```
> u <- runif(1E5)
> 2 * (mean(exp(-u^2) + exp(-u^(-2))) * u^(-2)))
[1] 1.771907
```

Ce résultat est suffisamment près de $\sqrt{\pi} = 1,772454$ pour juger l'expérience concluante.

3.7 Code informatique

📄 Fichier d'accompagnement montecarlo.R

```
12 ###
13 ### INTÉGRALE SIMPLE
14 ###
```

```

15
16 ## Estimations successives de l'intégrale par la méthode Monte
17 ## Carlo avec des échantillons de plus en plus grands.
18 f <- function(x) x^2.2 * exp(-x/10)
19 x <- runif(1e2, 2, 5)
20 3 * mean(f(x))
21 x <- runif(1e3, 2, 5)
22 3 * mean(f(x))
23 x <- runif(1e4, 2, 5)
24 3 * mean(f(x))
25 x <- runif(1e5, 2, 5)
26 3 * mean(f(x))
27 x <- runif(1e6, 2, 5)
28 3 * mean(f(x))
29
30 ## Traçons des graphiques de la vraie fonction à intégrer et
31 ## de trois évaluations par la méthode Monte Carlo.
32 op <- par(mfrow = c(2, 2)) # 4 graphiques en grille 2 x 2
33 curve(f(x), xlim = c(2, 5), lwd = 2, main = "Vraie fonction")
34
35 x <- runif(1e2, 2, 5)
36 plot(x, f(x), main = "n = 100",
37       pch = 21, bg = "lightblue")
38
39 x <- runif(1e3, 2, 5)
40 plot(x, f(x), main = "n = 1000",
41       pch = 21, bg = "lightblue")
42
43 x <- runif(1e4, 2, 5)
44 plot(x, f(x), main = "n = 10 000",
45       pch = 21, bg = "lightblue")
46
47 par(op) # revenir aux paramètres par défaut
48
49 ###
50 ### INTÉGRALE DOUBLE
51 ###
52
53 ## Estimations successives de l'intégrale double par la
54 ## méthode Monte Carlo avec des échantillons de plus en plus
55 ## grands.
56 u <- runif(1e2, 0, 1.25)
57 v <- runif(1e2, 0, 1.25)
58 mean(sqrt(4 - u^2 - v^2)) * 1.25^2
59 u <- runif(1e3, 0, 1.25)

```

```
60 v <- runif(1e3, 0, 1.25)
61 mean(sqrt(4 - u^2 - v^2)) * 1.25^2
62 u <- runif(1e4, 0, 1.25)
63 v <- runif(1e4, 0, 1.25)
64 mean(sqrt(4 - u^2 - v^2)) * 1.25^2
65
66 ## Graphiques de la vraie fonction et des points où celle-ci
67 ## est évaluée avec la méthode Monte Carlo. Pour faire un
68 ## graphique en trois dimensions, nous pouvons utiliser la
69 ## fonction 'persp'.
70 op <- par(mfrow = c(2, 2), mar = c(1, 1, 2, 1))
71 f <- function(x, y) sqrt(4 - x^2 - y^2)
72 x <- seq(0, 1.25, length = 25)
73 y <- seq(0, 1.25, length = 25)
74 persp(x, y, outer(x, y, f), main = "Vraie fonction",
75       zlim = c(0, 2), theta = 120, phi = 30, col = "lightblue",
76       zlab = "z", ticktype = "detailed")
77
78 ## Pour faire les trois autres graphiques, nous allons d'abord
79 ## créer des graphiques en trois dimensions vides, puis y
80 ## ajouter des points avec la fonction 'points'. La fonction
81 ## 'trans3d' sert à convertir les coordonnées des points dans
82 ## la projection utilisée par 'persp'.
83 u <- runif(1e2, 0, 1.25)
84 v <- runif(1e2, 0, 1.25)
85 res <- persp(x, y, matrix(NA, length(x), length(y)),
86             main = "n = 100",
87             zlim = c(0, 2), theta = 120, phi = 30,
88             zlab = "z", ticktype = "detailed")
89 points(trans3d(u, v, f(u, v), pm = res),
90       pch = 21, bg = "lightblue")
91
92 u <- runif(1e3, 0, 1.25)
93 v <- runif(1e3, 0, 1.25)
94 res <- persp(x, y, matrix(NA, length(x), length(y)),
95             main = "n = 1000",
96             zlim = c(0, 2), theta = 120, phi = 30,
97             zlab = "z", ticktype = "detailed")
98 points(trans3d(u, v, f(u, v), pm = res),
99       pch = 21, bg = "lightblue")
100
101 u <- runif(1e4, 0, 1.25)
102 v <- runif(1e4, 0, 1.25)
103 res <- persp(x, y, matrix(NA, length(x), length(y)),
104             main = "n = 10 000",
```

```

105         zlim = c(0, 2), theta = 120, phi = 30,
106         zlab = "z", ticktype = "detailed")
107 points(trans3d(u, v, f(u, v), pm = res),
108        pch = 21, bg = "lightblue")
109
110 par(op)                                # revenir aux paramètres par défaut

```

3.8 Exercices

3.1 Évaluer l'intégrale

$$\int_0^1 \ln(5x + 4) dx$$

exactement ainsi qu'à l'aide de l'intégration Monte Carlo. Comparer les réponses.

3.2 Évaluer l'intégrale

$$\int_0^1 \int_0^1 e^{2xy} \ln(3x + y^2) dx dy$$

à l'aide de l'intégration Monte Carlo. Comparer la réponse obtenue avec la vraie valeur, 1,203758, obtenue à l'aide de Maple.

3.3 Soit l'intégrale

$$\theta = \int_0^\infty x^2 \sin(\pi x) e^{-x/2} dx.$$

- Évaluer cette intégrale par Monte Carlo en effectuant un changement de variable.
- Évaluer cette intégrale par Monte Carlo par échantillonnage direct d'une loi de probabilité appropriée.

3.4 Soit X_1, \dots, X_{25} un échantillon aléatoire de taille 25 d'une distribution $N(0, 1)$ et soit $X_{(1)} \leq \dots \leq X_{(25)}$ les statistiques d'ordre de cet échantillon, c'est-à-dire les données de l'échantillon triées en ordre croissant. Estimer $E[X_{(5)}]$ par intégration Monte Carlo. Comparer la réponse obtenue avec la vraie espérance, $-0,90501$.

Réponses

3.1 Valeur exacte : 1,845969

3.3 Valeur exacte : $-0,055292$

A Planification d'une simulation avec R

Il existe de multiples façons de réaliser la mise en œuvre informatique d'une simulation, mais certaines sont plus efficaces que d'autres. Cette annexe passe en revue diverses façons de faire des simulations avec R à l'aide d'un exemple simple de nature statistique.

A.1 Contexte

Soit X_1, \dots, X_n un échantillon aléatoire tiré d'une population distribuée selon une loi uniforme sur l'intervalle $(\theta - \frac{1}{2}, \theta + \frac{1}{2})$ et trois estimateurs sans biais du paramètre inconnu θ :

1. la moyenne arithmétique

$$\hat{\theta}_1 = \frac{1}{n} \sum_{i=1}^n X_i;$$

2. la médiane empirique

$$\hat{\theta}_2 = \begin{cases} X_{(\frac{n+1}{2})}, & n \text{ impair} \\ \frac{1}{2}(X_{(\frac{n}{2})} + X_{(\frac{n}{2}+1)}), & n \text{ pair,} \end{cases}$$

où $X_{(k)}$ est la k^e statistique d'ordre de l'échantillon aléatoire;

3. la mi-étendue

$$\hat{\theta}_3 = \frac{X_{(1)} + X_{(n)}}{2}.$$

À l'aide de la simulation, nous voulons, d'une part, vérifier si les trois estimateurs sont bel et bien sans biais et, d'autre part, déterminer lequel a la plus faible variance.

Pour ce faire, nous devons d'abord simuler un grand nombre N d'échantillons aléatoires de taille n d'une distribution $U(\theta - \frac{1}{2}, \theta + \frac{1}{2})$ pour une

valeur de θ choisie. Pour chaque échantillon, nous calculerons ensuite les trois estimateurs ci-dessus, puis la moyenne et la variance, par type d'estimateur, de tous les estimateurs obtenus. Si la moyenne des N estimateurs $\hat{\theta}_i$, $i = 1, 2, 3$ est près de θ , alors nous pourrions conclure que $\hat{\theta}_i$ est sans biais. De même, nous déterminerons lequel des trois estimateurs a la plus faible variance selon le classement des variances empiriques.

A.2 Première approche : avec une boucle

La façon la plus intuitive de mettre en œuvre cette étude de simulation en R consiste à utiliser une boucle `for`. Avec cette approche, il est nécessaire d'initialiser une matrice de 3 lignes et N colonnes (ou l'inverse) dans laquelle seront stockées les valeurs des trois estimateurs pour chaque simulation. Une fois la matrice remplie dans la boucle, il ne reste plus qu'à calculer la moyenne et la variance par ligne pour obtenir les résultats souhaités.

La [figure A.1](#) présente un exemple de code adéquat pour réaliser la simulation à l'aide d'une boucle.

Si l'on souhaite pouvoir exécuter le code de la [figure A.1](#) facilement à l'aide d'une seule expression, il suffit de placer l'ensemble du code dans une fonction. La fonction `simul1` de la [figure A.2](#) reprend le code de la [figure A.1](#), sans les commentaires.

```
> simul1(10000, 100, 0)
$biais
      Moyenne      Mediane  Mi-etendue
0.0002020893 0.0008945559 -0.0000498471

$variances
      Moyenne      Mediane  Mi-etendue
8.444864e-04 2.479267e-03 4.725045e-05
```

A.3 Seconde approche : avec `sapply`

Dans R, il est plus élégant — et parfois plus efficace — de masquer les calculs itératifs avec les fonctions d'application `lapply` et `sapply` (voir chapitre 6). Rappelons que la syntaxe de ces fonctions est :

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

```
## Bonne habitude à prendre: stocker les constantes dans
## des variables faciles à modifier au lieu de les écrire
## explicitement dans le code.
size <- 100                # taille de chaque échantillon
nsimul <- 10000            # nombre de simulations
theta <- 0                 # la valeur du paramètre

## Les lignes ci-dessous éviteront de faire deux additions
## 'nsimul' fois.
a <- theta - 0.5           # borne inférieure de l'uniforme
b <- theta + 0.5           # borne supérieure de l'uniforme

## Initialisation de la matrice dans laquelle seront
## stockées les valeurs des estimateurs. Les noms donnés
## aux lignes de la matrice permettent de facilement
## identifier les estimateurs.
x <- matrix(0, nrow = 3, ncol = nsimul)
rownames(x) <- c("Moyenne", "Mediane", "Mi-etendue")

## Simulation comme telle.
for (i in 1:nsimul)
{
  u <- runif(size, a, b)
  x[, i] <- c(mean(u),      # moyenne
              median(u)    # médiane
              mean(range(u))) # mi-étendue
}

## Nous pouvons maintenant calculer la moyenne et la
## variance par ligne.
rowMeans(x) - theta        # vérification du biais
apply(x, 1, var)           # comparaison des variances
```

FIG. A.1 - Code pour la simulation utilisant une boucle for

```

simul1 <- function(nsimul, size, theta)
{
  a <- theta - 0.5
  b <- theta + 0.5

  x <- matrix(0, nrow = 3, ncol = nsimul)
  rownames(x) <- c("Moyenne", "Mediane", "Mi-etendue")

  for (i in 1:nsimul)
  {
    u <- runif(size, a, b)
    x[, i] <- c(mean(u), median(u), mean(range(u)))
  }

  list(biais = rowMeans(x) - theta,
       variances = apply(x, 1, var))
}

```

FIG. A.2 - Définition de la fonction simul1

Les fonctions appliquent la fonction FUN à tous les éléments de la liste ou du vecteur x et retournent les résultats sous forme de liste (`lapply`) ou, lorsque c'est possible, de vecteur ou de matrice (`sapply`). Il est important de noter que les valeurs successives de x seront passées comme *premier* argument à la fonction FUN. Le cas échéant, les autres arguments de FUN sont spécifiés dans le champ '...'.

Pour pouvoir utiliser ces fonctions dans le cadre d'une simulation comme celle dont il est question ici, il s'agit de définir une fonction qui fera tous les calculs pour une simulation, puis de la passer à `sapply` pour obtenir les résultats de N simulations. La [figure A.3](#) présente une première version d'une telle fonction. Vous remarquerez que l'argument i ne joue aucun rôle dans la fonction. L'expression ci-dessous fournit un exemple d'utilisation pour un petit (4) nombre de simulations.

```

> sapply(1:4, fun1, size = 10, a = -0.5, b = 0.5)
      [,1]      [,2]      [,3]
Moyenne -0.03574319 -0.05860282 -0.1797460
Mediane  -0.08186902 -0.03895313 -0.2786489
Mi-etendue -0.02746296 -0.07059704 -0.1256943

```



```

fun1 <- function(i, size, a, b)
{
  u <- runif(size, a, b)
  c(Moyenne = mean(u),
    Mediane = median(u),
    "Mi-etendue" = mean(range(u)))
}

simul2 <- function(nsimul, size, theta)
{
  a <- theta - 0.5
  b <- theta + 0.5

  x <- sapply(1:nsimul, fun1, size, a, b)

  list(biais = rowMeans(x) - theta,
       variances = apply(x, 1, var))
}

```

FIG. A.3 - Définitions des fonction `fun1` et `simul2`

```

      [,4]
Moyenne  0.04418360
Mediane  0.13568992
Mi-etendue 0.04913279

```

Vous remarquerez que les résultats de chaque simulation se trouvent dans les colonnes de la matrice obtenue avec `sapply`.

Pour compléter l'analyse, nous englobons le tout dans une fonction `simul2`, dont le code se trouve à la [figure A.3](#).

```

> simul2(10000, 100, 0)
$biais
      Moyenne      Mediane  Mi-etendue
3.711994e-04 5.276894e-04 1.993503e-05

$variances
      Moyenne      Mediane  Mi-etendue
8.372538e-04 2.471342e-03 4.771435e-05

```

Il est généralement plus facile de déboguer le code avec cette approche puisque nous pouvons rapidement circonscrire un éventuel problème à `fun1` ou `simul2`.

A.4 Variante de la seconde approche

Une chose manque d'élégance dans la seconde approche : l'obligation d'inclure un argument factice dans la fonction `fun1`. La fonction `replicate` permet toutefois de passer outre cette contrainte. En effet, cette fonction exécute un nombre donné de fois une expression quelconque.

Les fonctions `fun2` et `simul3` de la [figure A.4](#) sont des versions légèrement modifiées de `fun1` et `simul2` pour utilisation avec `replicate`.

```
> simul3(10000, 100, 0)
$biais
      Moyenne      Mediane  Mi-etendue
6.196253e-04 8.312999e-04 5.203412e-05

$variances
      Moyenne      Mediane  Mi-etendue
8.562545e-04 2.483878e-03 4.827905e-05
```

A.5 Gestion des fichiers

Pour un petit projet comme celui utilisé en exemple ici, il est simple et pratique de placer tout le code informatique dans un seul fichier de script. Pour un plus gros projet, cependant, il vaut souvent mieux avoir recours à plusieurs fichiers différents. Le présent auteur utilise pour sa part un fichier par fonction.

À des fins d'illustration, supposons que nous utilisons l'approche de la [section A.4](#) avec la fonction `replicate` et que le code des fonctions `fun2` et `simul3` est sauvegardé dans des fichiers `fun2.R` et `simul3.R`, dans l'ordre. Si nous créons un autre fichier, disons `go.R`, ne contenant que des expressions source pour lire les autres fichiers, il est alors possible de démarrer des simulations en exécutant ce seul fichier. Dans notre exemple, le fichier `go.R` contiendrait les lignes suivantes :

```
fun2 <- function(size, a, b)
{
  u <- runif(size, a, b)
  c(Moyenne = mean(u),
    Mediane = median(u),
    "Mi-etendue" = mean(range(u)))
}

simul3 <- function(nsimul, size, theta)
{
  a <- theta - 0.5
  b <- theta + 0.5

  x <- replicate(nsimul, fun2(size, a, b))

  list(biais = rowMeans(x) - theta,
       variances = apply(x, 1, var))
}
```

FIG. A.4 - Définitions des fonction fun2 et simul3

```
source("fun2.R")
source("simul3.R")
simul3(10000, 100, 0)
```

La simple commande ci-dessous exécutera alors une simulation complète.

```
> source("go.R")
```

A.6 Exécution en lot

Les utilisateurs plus avancés pourront vouloir exécuter leur simulation R en lot (*batch*) pour en accélérer le traitement. Dans ce mode, aucune interface graphique n'est démarrée et tous les résultats sont redirigés vers un fichier pour consultation ultérieure. Pour les simulations demandant un long temps de calcul, c'est très pratique.

On exécute R en lot depuis la ligne de commande (Invite de commande sous Windows, Terminal sous macOS ou Linux). Une fois placé dans le ré-

pertoire contenant les fichiers de script, il suffit d'entrer à la ligne de commande :

```
$ R CMD BATCH go.R
```

La sortie de cette commande (et donc tous les résultats des expressions R du fichier `go.R`) seront placés par défaut dans le fichier `go.Rout`. Sous Windows, le dossier d'installation de R peut ne pas se trouver dans la variable d'environnement `%PATH%`, auquel cas il faut spécifier le chemin d'accès complet du programme à la ligne de commande :

```
$ "c:\Program Files\R\R-(x.y.z)\bin\R" CMD BATCH go.R
```

A.7 Conclusion

Le nombre de simulations, N , et la taille de l'échantillon, n , ont tous deux un impact sur la qualité des résultats, mais de manière différente. Quand n augmente, la précision des estimateurs augmente. Ainsi, dans l'exemple ci-dessus, le biais et la variance des estimateurs de θ seront plus faibles. D'autre part, l'augmentation du nombre de simulations diminue l'impact des échantillons aléatoires individuels et, de ce fait, améliore la fiabilité des conclusions de l'étude.

D'ailleurs, les conclusions de l'étude de simulation sur le biais et la variance des trois estimateurs de la moyenne d'une loi uniforme sont les suivantes : les trois estimateurs sont sans biais et la mi-étendue a la plus faible variance. En effet, on peut démontrer mathématiquement que, pour n impair,

$$\begin{aligned}\text{Var}[\hat{\theta}_1] &= \frac{1}{12n} \\ \text{Var}[\hat{\theta}_2] &= \frac{1}{4n+2} \\ \text{Var}[\hat{\theta}_3] &= \frac{1}{2(n+1)(n+2)}\end{aligned}$$

et donc

$$\text{Var}[\hat{\theta}_3] \leq \text{Var}[\hat{\theta}_1] \leq \text{Var}[\hat{\theta}_2]$$

pour tout $n \geq 2$.

B Transformations de variables aléatoires

Cette annexe porte sur les transformations, ou fonctions, de variables aléatoires. En termes mathématiques, étant donné la distribution conjointe des variables aléatoires X_1, \dots, X_n , nous cherchons à déterminer la fonction de probabilité ou de densité de la variable aléatoire $Y = u(X_1, \dots, X_n)$.

Voici quelques exemples de transformations fréquemment rencontrées :

$$Y = X^2$$

$$Y = \frac{X - E[X]}{\sqrt{\text{Var}[X]}}$$

$$Y = \frac{X_1 + \dots + X_n}{n}$$

$$Y = F_X(X).$$

Il existe trois techniques principales pour déterminer la distribution de la transformation Y :

1. la technique de la fonction de répartition;
2. la technique du changement de variable;
3. la technique de la fonction génératrice des moments.

B.1 Technique de la fonction de répartition

C'est la technique la plus simple, mais pas toujours la plus facile d'emploi. En effet, la fonction de répartition de certaines lois de probabilité est compliquée, voire n'existe pas sous forme explicite (penser ici aux lois normale et gamma, par exemple).

L'idée consiste simplement à calculer la fonction de répartition de la transformation avec

$$\begin{aligned} F_Y(y) &= \Pr[Y \leq y] \\ &= \Pr[u(X_1, \dots, X_n) \leq y], \end{aligned}$$

puis à calculer la densité (ou la fonction de probabilité) par différenciation :

$$f_Y(y) = F'_Y(y).$$



Le domaine de définition de la transformation n'est pas nécessairement le même que celui des variables aléatoires de départ.

Exemple B.1. Soit

$$f_X(x) = \begin{cases} 6x(1-x), & 0 < x < 1 \\ 0, & \text{ailleurs,} \end{cases}$$

c'est-à-dire $X \sim \text{Bêta}(2, 2)$. Déterminons la densité de $Y = X^3$. Nous avons :

$$\begin{aligned} F_Y(y) &= \Pr[X^3 \leq y] \\ &= \Pr[X \leq y^{1/3}] \\ &= F_X(y^{1/3}) \\ &= \int_0^{y^{1/3}} 6x(1-x) dx \\ &= 3y^{2/3} - 2y, \quad 0 < y < 1, \end{aligned}$$

et donc

$$\begin{aligned} f_Y(y) &= F'_Y(y) \\ &= \begin{cases} 2y^{-1/3} - 2, & 0 < y < 1 \\ 0, & \text{ailleurs.} \end{cases} \end{aligned}$$

□

Exemple B.2. Soit X une variable aléatoire continue quelconque avec fonction de répartition $F_X(x)$ et $Y = aX + b$, où a et b sont des constantes réelles. Nous avons :

$$\begin{aligned} F_Y(y) &= \Pr[aX + b \leq y] \\ &= F_X\left(\frac{y-b}{a}\right) \end{aligned}$$

et, par conséquent,

$$f_Y(y) = \frac{1}{a} f_X\left(\frac{y-b}{a}\right).$$

La transformation $Y = X + b$ représente une *translation* de X , vers la droite si $b > 0$ et vers la gauche si $b < 0$. En assurance, b pourra être interprété comme une franchise.

La transformation $Y = aX$ n'est quant à elle qu'un *changement d'échelle* — par exemple un changement d'unité monétaire. Si $a > 1$ on a une dilatation, alors que le cas où $0 < a < 1$ est une contraction. \square

Exemple B.3. Soit X une variable aléatoire continue quelconque avec densité $f_X(x)$. Nous voulons déterminer la densité de $Y = |X|$. Premièrement, il convient de remarquer que Y est définie au plus sur les réels positifs, même si X est définie sur tout l'axe des réels. Par la technique de la fonction de répartition,

$$\begin{aligned} F_Y(y) &= \Pr[|X| \leq y] \\ &= \Pr[-y < X < y] \\ &= F_X(y) - F_X(-y) \end{aligned}$$

et

$$f_Y(y) = \begin{cases} f_X(y) + f_X(-y), & y > 0 \\ 0, & \text{ailleurs.} \end{cases}$$

Par exemple, soit $X \sim N(0, 1)$, c'est-à-dire

$$\begin{aligned} f_X(x) &= \phi(x) \\ &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}. \end{aligned}$$

La densité de $Y = |X|$ est alors

$$\begin{aligned} f_Y(y) &= \phi(y) + \phi(-y) \\ &= 2\phi(y) \\ &= \frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2}, \quad y > 0. \end{aligned}$$

\square

Exemple B.4. Soit $Y = X_1 + X_2$, où X_1 et X_2 sont deux variables aléatoires indépendantes chacune distribuée uniformément sur l'intervalle $(0, 1)$. Nous avons donc

$$f_{X_1 X_2}(x_1, x_2) = f_{X_1}(x_1) = f_{X_2}(x_2) = 1, \quad 0 < x_1 < 1, 0 < x_2 < 1.$$

Le domaine de définition de Y sera l'intervalle $(0, 2)$. Le domaine d'intégration étant un carré, il faut distinguer quatre cas.

1. Si $y \leq 0$, alors clairement $F_Y(y) = 0$.
2. Si $0 < y < 1$, alors

$$\begin{aligned} F_Y(y) &= \int_0^y \int_0^{y-x_2} dx_1 dx_2 \\ &= \frac{1}{2} y^2. \end{aligned}$$

3. Si $1 < y < 2$, alors

$$\begin{aligned} F_Y(y) &= (y-1)(1) + \int_{y-1}^1 \int_0^{y-x_2} dx_1 dx_2 \\ &= 1 - \frac{1}{2} (2-y)^2. \end{aligned}$$

4. Enfin, si $y \geq 2$, clairement $F_Y(y) = 1$.

Par conséquent, nous avons

$$F_Y(y) = \begin{cases} 0, & y \leq 0 \\ \frac{1}{2} y^2, & 0 < y \leq 1 \\ 1 - \frac{1}{2} (2-y)^2, & 1 < y < 2 \\ 1, & y \geq 2 \end{cases}$$

et

$$f_Y(y) = \begin{cases} 0, & y \leq 0 \\ y, & 0 < y \leq 1 \\ 2-y, & 1 < y < 2 \\ 0, & y \geq 2. \end{cases}$$

□

B.2 Technique du changement de variable univariée

Cette technique est étroitement liée au changement de variable en intégration. Il convient toutefois de faire une distinction entre les cas discret et continu.

B.2.1 Cas discret

Soit la transformation $Y = u(X)$. Dans le cas discret, il suffit généralement de faire la substitution :

$$\begin{aligned}\Pr[Y = y] &= \Pr[u(X) = y] \\ &= \Pr[X = u^{-1}(y)].\end{aligned}$$

Les probabilités ne changent donc pas, elles ne sont qu'affectées à d'autres valeurs.

Exemple B.5. Soit la variable aléatoire X avec fonction de probabilité

$$\Pr[X = x] = \begin{cases} 1/16, & x = 0 \\ 4/16, & x = 1 \\ 6/16, & x = 2 \\ 4/16, & x = 3 \\ 1/16, & x = 4. \end{cases}$$

Nous voulons déterminer la fonction de probabilité de $Y = (X - 2)^2$. Les valeurs possibles de Y sont $y = 0, 1$ et 4 . Nous avons :

$$\begin{aligned}\Pr[Y = y] &= \Pr[(X - 2)^2 = y] \\ &= \Pr[X = \pm\sqrt{y} + 2]\end{aligned}$$

et donc

$$\begin{aligned}\Pr[Y = 0] &= \Pr[X = 2] = \frac{6}{16} \\ \Pr[Y = 1] &= \Pr[X = 1] + \Pr[X = 3] = \frac{8}{16} \\ \Pr[Y = 4] &= \Pr[X = 0] + \Pr[X = 4] = \frac{2}{16}.\end{aligned}$$

□

B.2.2 Cas continu

Le cas continu est beaucoup plus délicat. Nous étudions toujours la transformation $Y = u(X)$ avec les hypothèses suivantes :

- la fonction $u(\cdot)$ est différentiable;
- la fonction $u(\cdot)$ est soit croissante, soit décroissante sur tout le domaine de $f_X(x)$.

Ainsi, l'inverse $u^{-1}(\cdot) = w(\cdot)$ de la fonction u existe et est différentiable.

Théorème B.1. Soit $f_X(x)$ la fonction de densité de probabilité en x d'une variable aléatoire X et $y = u(x)$ une fonction satisfaisant les hypothèses ci-dessus. Alors la densité de la variable aléatoire $Y = u(X)$ est

$$\begin{aligned} f_Y(y) &= f_X(u^{-1}(y)) |(u^{-1}(y))'| \\ &= f_X(w(y)) |w'(y)|, \end{aligned}$$

où $w(y) = u^{-1}(y)$ et en supposant $u'(x) \neq 0$.

Démonstration. Considérons le cas où u est une fonction croissante. Alors

$$\begin{aligned} \Pr[a < Y < b] &= \Pr[u^{-1}(a) < X < u^{-1}(b)] \\ &= \int_{w(a)}^{w(b)} f_X(x) dx \end{aligned}$$

puis, avec le changement de variable $y = u(x) \Leftrightarrow x = w(y)$ et donc $dx = w'(y) dy$

$$\Pr[a < Y < b] = \int_a^b f_X(w(y)) w'(y) dy,$$

d'où $f_Y(y) = f_X(w(y)) w'(y)$. Si u est décroissante, alors

$$\begin{aligned} f_Y(y) &= -f_X(w(y)) w'(y) \\ &= f_X(w(y)) |w'(y)| \end{aligned}$$

car $w'(y) < 0$. □

Exemple B.6. Soit $Y = -2 \ln(X)$ où $X \sim U(0, 1)$. En premier lieu, il importe de spécifier qu'au domaine $(0, 1)$ de la variable aléatoire X correspond le domaine $(0, \infty)$ pour la transformation Y . Nous avons $u(x) = -2 \ln(x)$,

d'où $w(y) = u^{-1}(y) = e^{-y/2}$ et $w'(y) = -\frac{1}{2}e^{-y/2}$. Par le [théorème B.1](#), nous obtenons

$$\begin{aligned} f_Y(y) &= f_X(e^{-y/2}) \left| -\frac{1}{2}e^{-y/2} \right| \\ &= \frac{1}{2} e^{-y/2} \\ &= \frac{1/2}{\Gamma(1)} y^{1-1} e^{-y/2}, \quad y > 0, \end{aligned}$$

soit $Y \sim \text{Gamma}(1, 1/2) \equiv \chi^2(2)$. □

Exemple B.7. Nous voulons déterminer la distribution de $Y = Z^2$, où $Z \sim N(0, 1)$. La variable aléatoire Z étant définie sur \mathbb{R} , la transformation $y = z^2$ n'est pas bijective. Le truc consiste ici à d'abord définir la transformation $X = |Z|$, de sorte que X soit définie sur les réels positifs seulement. De l'[exemple B.3](#), nous savons que

$$f_X(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}.$$

Par la suite, nous définissons $Y = X^2 = |Z|^2 = Z^2$, une transformation bijective de X dont le domaine de définition est \mathbb{R}^+ . Nous avons alors $u(x) = x^2$ pour $x > 0$, soit $w(y) = \sqrt{y}$ et $w'(y) = \frac{1}{2}y^{-1/2}$, d'où

$$\begin{aligned} f_Y(y) &= f_X(\sqrt{y}) \left| \frac{y^{-1/2}}{2} \right| \\ &= \frac{y^{-1/2}}{\sqrt{2\pi}} e^{-y/2} \\ &= \frac{(\frac{1}{2})^{1/2}}{\Gamma(\frac{1}{2})} y^{-1/2} e^{-y/2}, \quad y > 0, \end{aligned}$$

soit $Y \sim \chi^2(1)$. □

B.3 Technique du changement de variable multivariée

Il s'agit simplement ici de généraliser les concepts étudiés à la section précédente à des transformations impliquant plusieurs variables aléatoires.

L'idée reste la même sinon qu'il faut s'assurer que la transformation compte autant de nouvelles variables que d'anciennes. Ainsi, si nous parlons de la distribution conjointe de deux variables aléatoires X_1 et X_2 , il faudra trouver la distribution conjointe de deux nouvelles variables aléatoires

$Y_1 = u_1(X_1, X_2)$ et $Y_2 = u_2(X_1, X_2)$. Plus souvent qu'autrement, seule la distribution de la variable Y_1 est d'intérêt. Il suffit alors de définir Y_2 comme une variable muette, par exemple $Y_2 = X_2$ (les textes anglais utilisent généralement l'expression *dummy variable*).

B.3.1 Cas discret

Si la transformation $Y_1 = u_1(X_1, X_2)$ et $Y_2 = u_2(X_1, X_2)$ est bijective, alors simplement

$$\begin{aligned}\Pr[Y_1 = y_1, Y_2 = y_2] &= \Pr[u_1(X_1, X_2) = y_1, u_2(X_1, X_2) = y_2] \\ &= \Pr[X_1 = w_1(y_1, y_2), X_2 = w_2(y_1, y_2)],\end{aligned}$$

où

$$\begin{aligned}w_1(y_1, y_2) &= u_1^{-1}(x_1, x_2) \\ w_2(y_1, y_2) &= u_2^{-1}(x_1, x_2).\end{aligned}$$

Les fonctions de probabilité marginales sont alors obtenues en sommant :

$$\begin{aligned}\Pr[Y_1 = y_1] &= \sum_{y_2=-\infty}^{\infty} \Pr[Y_1 = y_1, Y_2 = y_2] \\ \Pr[Y_2 = y_2] &= \sum_{y_1=-\infty}^{\infty} \Pr[Y_1 = y_1, Y_2 = y_2].\end{aligned}$$

Exemple B.8. Soit $X_1 \sim \text{Poisson}(\lambda_1)$, $X_2 \sim \text{Poisson}(\lambda_2)$ et X_1 et X_2 sont indépendantes. Posons $Y = X_1 + X_2$.

Le calcul de la distribution de Y requiert une variable muette. Définissons $Y_2 = X_2$. Nous avons alors :

$$\begin{aligned}Y_1 &= X_1 + X_2 & \Leftrightarrow & & X_1 &= Y_1 - Y_2 \\ Y_2 &= X_2 & & & X_2 &= Y_2.\end{aligned}$$

Le domaine de Y_1 est donc $0, 1, 2, \dots$, alors que celui de Y_2 est $0, 1, \dots, Y_1$. Ainsi,

$$\begin{aligned}\Pr[Y_1 = y_1, Y_2 = y_2] &= \Pr[X_1 + X_2 = y_1, X_2 = y_2] \\ &= \Pr[X_1 = y_1 - y_2, X_2 = y_2] \\ &= \Pr[X_1 = y_1 - y_2] \Pr[X_2 = y_2] \\ &= \frac{\lambda_1^{y_1 - y_2} e^{-\lambda_1}}{(y_1 - y_2)!} \frac{\lambda_2^{y_2} e^{-\lambda_2}}{y_2!}\end{aligned}$$

et donc

$$\begin{aligned}
 \Pr[Y_1 = y_1] &= \sum_{y_2=0}^{y_1} \frac{\lambda_1^{y_1-y_2} \lambda_2^{y_2} e^{-(\lambda_1+\lambda_2)}}{(y_1 - y_2)! y_2!} \\
 &= \frac{e^{-(\lambda_1+\lambda_2)}}{y_1!} \sum_{y_2=0}^{y_1} \frac{y_1!}{(y_1 - y_2)! y_2!} \lambda_1^{y_1-y_2} \lambda_2^{y_2} \\
 &= \frac{e^{-(\lambda_1+\lambda_2)}}{y_1!} \sum_{y_2=0}^{y_1} \binom{y_1}{y_2} \lambda_1^{y_1-y_2} \lambda_2^{y_2} \\
 &= \frac{e^{-(\lambda_1+\lambda_2)}}{y_1!} (\lambda_1 + \lambda_2)^{y_1},
 \end{aligned}$$

soit $Y \sim \text{Poisson}(\lambda_1 + \lambda_2)$. □

B.3.2 Cas continu

Généralisons le [théorème B.1](#) afin de trouver la distribution conjointe (et éventuellement les distributions marginales) de $Y_1 = u_1(X_1, X_2)$ et $Y_2 = u_2(X_1, X_2)$. Nous supposons que :

- toutes les premières dérivées partielles de u_1 et u_2 existent sur le domaine de X_1 et X_2 ;
- la transformation est bijective.

Ces hypothèses garantissent que les fonctions inverses $w_1 = u_1^{-1}$ et $w_2 = u_2^{-1}$ existent.

Théorème B.2. Soit $f_{X_1 X_2}(x_1, x_2)$ la fonction de densité de probabilité conjointe en (x_1, x_2) des variables aléatoires X_1 et X_2 et $y_1 = u_1(x_1, x_2)$, $y_2 = u_2(x_1, x_2)$ des fonctions satisfaisant les hypothèses ci-dessus. Alors la densité conjointe des variables aléatoires $Y_1 = u_1(X_1, X_2)$ et $Y_2 = u_2(X_1, X_2)$ est

$$f_{Y_1 Y_2}(y_1, y_2) = f_{X_1 X_2}(w_1(y_1, y_2), w_2(y_1, y_2)) |J|,$$

où

$$J = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix}$$

est appelé le Jacobien de la transformation.

Exemple B.9. Soit les variables aléatoires stochastiquement indépendantes $X_1 \sim \text{Gamma}(\alpha, 1)$ et $X_2 \sim \text{Gamma}(\eta, 1)$. Nous allons démontrer que les variables aléatoires $Y_1 = X_1 + X_2$ et $Y_2 = X_1 / (X_1 + X_2)$ sont indépendantes et trouver leur densité marginale respective.

Tout d'abord, nous avons

$$f_{X_1 X_2}(x_1, x_2) = \frac{1}{\Gamma(\alpha)\Gamma(\eta)} x_1^{\alpha-1} x_2^{\eta-1} e^{-x_1-x_2}, \quad x_1 > 0, x_2 > 0.$$

De plus,

$$\begin{aligned} Y_1 &= X_1 + X_2 \\ Y_2 &= \frac{X_1}{X_1 + X_2} \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} X_1 &= Y_1 Y_2 \\ X_2 &= Y_1 (1 - Y_2) \end{aligned}$$

et donc

$$\begin{aligned} \frac{\partial x_1}{\partial y_1} &= y_2 & \frac{\partial x_1}{\partial y_2} &= y_1 \\ \frac{\partial x_2}{\partial y_1} &= 1 - y_2 & \frac{\partial x_2}{\partial y_2} &= -y_1, \end{aligned}$$

d'où le Jacobien de la transformation est

$$J = \begin{vmatrix} y_2 & y_1 \\ 1 - y_2 & -y_1 \end{vmatrix} = -y_1.$$

Le domaine de Y_1 est \mathbb{R}^+ alors que celui de Y_2 est limité à l'intervalle $(0, 1)$. Par le [théorème B.2](#),

$$\begin{aligned} f_{Y_1 Y_2}(y_1, y_2) &= f_{X_1 X_2}(y_1 y_2, y_1 (1 - y_2)) | -y_1 | \\ &= \frac{1}{\Gamma(\alpha)\Gamma(\eta)} y_1 (y_1 y_2)^{\alpha-1} (y_1 (1 - y_2))^{\eta-1} e^{-y_1 y_2 - y_1 (1 - y_2)} \\ &= \frac{1}{\Gamma(\alpha)\Gamma(\eta)} y_1^{\alpha+\eta-1} y_2^{\alpha-1} (1 - y_2)^{\eta-1} e^{-y_1} \\ &= g(y_1) h(y_2) \end{aligned}$$

où $g(\cdot)$ et $h(\cdot)$ sont des fonctions quelconques. Ceci démontre que Y_1 et Y_2 sont indépendantes. De plus,

$$\begin{aligned} f_{Y_1}(y_1) &= \int_0^1 f_{Y_1 Y_2}(y_1, y_2) dy_2 \\ &= y_1^{\alpha+\eta-1} e^{-y_1} \int_0^1 \frac{1}{\Gamma(\alpha)\Gamma(\eta)} y_2^{\alpha-1} (1 - y_2)^{\eta-1} dy_2 \\ &= \frac{1}{\Gamma(\alpha + \eta)} y_1^{\alpha+\eta-1} e^{-y_1}, \quad y_1 > 0 \end{aligned}$$

et

$$\begin{aligned} f_{Y_2}(y_1, y_2) &= \frac{f_{Y_1 Y_2}(y_1, y_2)}{f_{Y_1}(y_1, y_2)} \\ &= \frac{\Gamma(\alpha + \eta)}{\Gamma(\alpha)\Gamma(\eta)} y_2^{\alpha-1} (1 - y_2)^{\eta-1}, \quad 0 < y_2 < 1. \end{aligned}$$

Nous avons donc $Y_1 = X_1 / (X_1 + X_2) \sim \text{Gamma}(\alpha + \eta, 1)$ et $Y_2 = X_2 / (X_1 + X_2) \sim \text{Bêta}(\alpha, \eta)$. \square

B.4 Technique de la fonction génératrice des moments

Cette technique s'avère tout spécialement puissante pour déterminer la distribution (marginale) d'une combinaison linéaire de variables aléatoires indépendantes. La technique repose sur le théorème suivant.

Théorème B.3. *Soit X_1, \dots, X_n des variables aléatoires indépendantes et $Y = X_1 + \dots + X_n$. Alors*

$$M_Y(t) = \prod_{i=1}^n M_{X_i}(t).$$

Il est laissé en exercice de refaire les exemples B.8 et B.9 (distribution de Y_1 seulement) à l'aide de la technique de la fonction génératrice des moments.

C Solutions des exercices

Chapitre 1

1.1 Dans tous les cas, le générateur de nombres aléatoires est

$$\begin{aligned}x_i &= (ax_{i-1} + c) \bmod m \\ &= (ax_{i-1} + c) - \left\lfloor \frac{ax_{i-1} + c}{m} \right\rfloor m\end{aligned}$$

où $m = 64$ et $x_0 = 19$. Les suites ont été générées avec la fonction `rand` de la [figure C.1](#).

a) `> rand(n = 5, a = 29, c = 17, m = 64, seed = 19)`
`[1] 56 41 54 47 36`

b) `> rand(n = 5, a = 9, c = 1, m = 64, seed = 19)`
`[1] 44 13 54 39 32`

```
rand <- function(n, a, c, m, seed)
{
  x <- numeric(n + 1)
  x[1] <- seed
  for (i in seq(n))
    x[i + 1] <- (a * x[i] + c) %% m
  x[-1]
}
```

FIG. C.1 – Code de la fonction `rand`

```
c) > rand(n = 5, a = 13, c = 0, m = 64, seed = 19)
[1] 55 11 15 3 39
```

```
d) > rand(n = 5, a = 11, c = 0, m = 64, seed = 19)
[1] 17 59 9 35 1
```

- 1.2 a) Utilisons de nouveau la fonction `rand` de la [figure C.1](#). Le graphique de la [figure C.2](#) a été créé avec les commandes ci-dessous. On compte 17 lignes dans le graphique.

```
> x <- rand(n = 500, a = 17, c = 0, m = 2^13 - 1,
+          seed = 19)
> plot(x[-length(x)], x[-1], xlab = expression(x[i]),
+      ylab = expression(x[i + 1]), pch = 19)
```

- b) Similaire à la partie a), sauf que les nombres sont générés avec l'expression suivante.

```
> x <- rand(n = 500, a = 85, c = 0, m = 2^13 - 1,
+          seed = 19)
```

Ce générateur semble préférable à celui en a) puisque les points sont plus uniformément disposés sur le graphique (voir [figure C.3](#)).

- 1.3 a) On obtient environ 200 points alignés sur 10 lignes.

```
> u <- rand(n = 20002, a = 65539, c = 0, m = 2^31,
+          seed = 19)/2^31
> mat <- matrix(c(u[1:20000], u[2:20001], u[3:20002]),
+             ncol = 3)
> mat <- mat[(0.5 <= mat[,2]) & (mat[,2] <= 0.51),
+           c(1, 3)]
> plot(mat, xlab = expression(u[i]),
+      ylab = expression(u[i + 2]))
```

```
b) > library(rgl)
> u <- rand(n = 1002, a = 65539, c = 0, m = 2^31,
+          seed = 19)/2^31
> plot3d(u[1:1000], u[2:1001], u[3:1002])
```

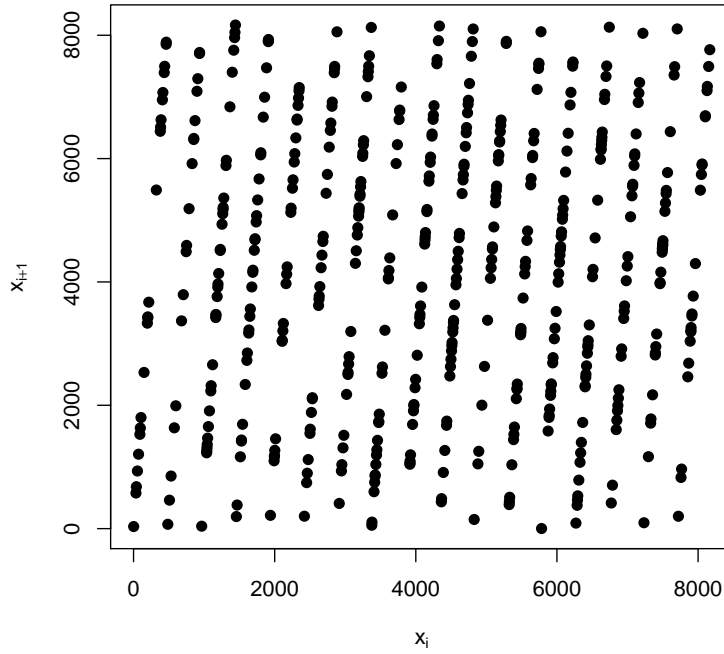


FIG. C.2 - Paires de valeurs du générateur congruentiel multiplicatif avec $m = 2^{31} - 1$ et $a = 17$

Chapitre 2

2.1 a) Tout d'abord, on voit que

$$\cos(2\pi U_2) \in (-1, 1)$$

$$\sin(2\pi U_2) \in (-1, 1)$$

et

$$(-2 \log U_1)^{1/2} \in (0, \infty).$$

Par conséquent, $X_1 \in (-\infty, \infty)$ et $X_2 \in (-\infty, \infty)$. On vérifie la bijectivité de façon heuristique avec quelques valeurs de u_1 et u_2 .

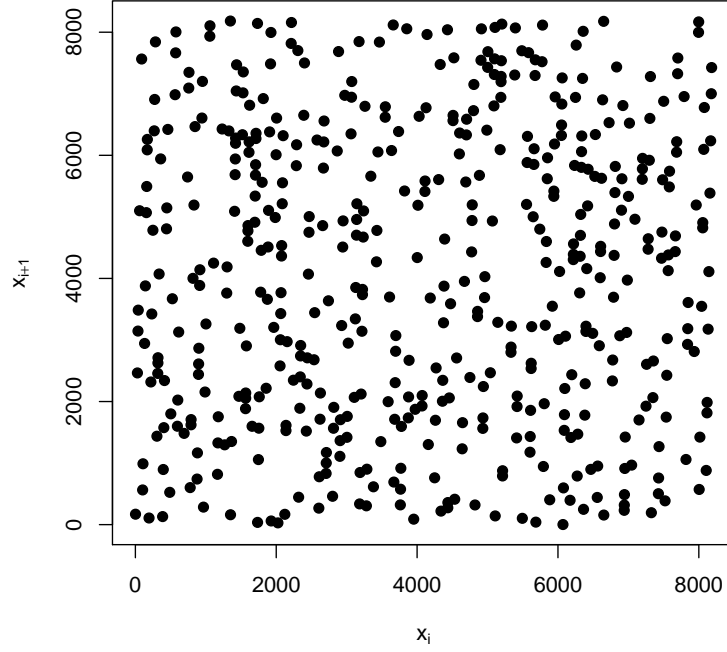


FIG. C.3 - Paires de valeurs du générateur congruentiel multiplicatif avec $m = 2^{31} - 1$ et $a = 85$

b) On a

$$X_1^2 = (-2 \log U_1) \cos^2(2\pi U_2)$$

$$X_2^2 = (-2 \log U_1) \sin^2(2\pi U_2).$$

Or, puisque $\sin^2(x) + \cos^2(x) = 1$, $X_1^2 + X_2^2 = -2 \log U_1$, d'où $U_1 = e^{-(X_1^2 + X_2^2)/2}$. D'autre part, $\sin(x)/\cos(x) = \tan(x)$, donc $\tan(2\pi U_2) = X_2/X_1$ ou, de manière équivalente, $U_2 = (2\pi)^{-1} \arctan X_2/X_1$.

c) Soit les fonctions

$$x_1(u_1, u_2) = (-2 \log u_1)^{1/2} \cos(2\pi u_2) \quad u_1(x_1, x_2) = e^{-(x_1^2 + x_2^2)/2}$$

$$x_2(u_1, u_2) = (-2 \log u_1)^{1/2} \sin(2\pi u_2) \quad u_2(x_1, x_2) = \frac{1}{2\pi} \arctan \frac{x_2}{x_1}.$$

Les variables aléatoires U_1 et U_2 sont indépendantes, donc leur fonction de densité de probabilité conjointe est le produit des densités marginales :

$$f_{U_1, U_2}(u_1, u_2) = 1, \quad 0 < u_1 < 1, 0 < u_2 < 1.$$

La densité conjointe de X_1 et X_2 est, par définition d'une transformation,

$$f_{X_1, X_2}(x_1, x_2) = f_{U_1, U_2}(x_1(u_1, u_2), x_2(u_1, u_2)) |\det(J)|,$$

où

$$J = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -x_1 e^{-(x_1^2 + x_2^2)/2} & -x_2 e^{-(x_1^2 + x_2^2)/2} \\ -\frac{1}{2\pi} \frac{x_2}{x_1^2 + x_2^2} & \frac{1}{2\pi} \frac{x_1}{x_1^2 + x_2^2} \end{bmatrix}.$$

Or,

$$\begin{aligned} |\det(J)| &= \frac{1}{2\pi} e^{-(x_1^2 + x_2^2)/2} \\ &= \frac{1}{\sqrt{2\pi}} e^{-x_1^2/2} \cdot \frac{1}{\sqrt{2\pi}} e^{-x_2^2/2}, \end{aligned}$$

d'où

$$f_{X_1, X_2}(x_1, x_2) = \frac{1}{\sqrt{2\pi}} e^{-x_1^2/2} \cdot \frac{1}{\sqrt{2\pi}} e^{-x_2^2/2}.$$

Par conséquent, X_1 et X_2 sont deux variables aléatoires $N(0, 1)$ indépendantes.

```
d) > u1 <- runif(500)
> u2 <- runif(500)
> x1 <- outer(u1, u2, function(x, y)
+           sqrt((-2 * log(x))) * cos(2 * pi * y))
> x2 <- outer(u1, u2, function(x, y)
+           sqrt((-2 * log(x))) * sin(2 * pi * y))
> hist(x1, prob = TRUE)
> curve(dnorm(x), add = TRUE)
```

La [figure C.4](#) illustre d'une autre façon que la transformation de Box-Muller fonctionne bel et bien. Le graphique de gauche contient plusieurs couples de points (u_1, u_2) , où chaque composante provient d'une distribution uniforme sur l'intervalle $(0, 1)$.

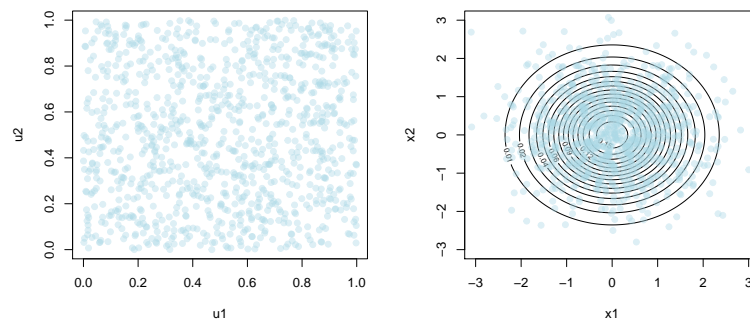


FIG. C.4 - Démonstration graphique du fonctionnement de la transformation de Box-Muller

Chacun de ces points a été transformé en un point (x_1, x_2) selon la transformation de Box-Muller, puis placé dans le graphique de droite. Nous avons superposé le nuage de points ainsi obtenu aux lignes de niveau d'une distribution normale bivariée (avec paramètre $\rho = 0$). Vous observerez que la répartition et la densité du nuage de points correspond effectivement aux lignes de niveau.

2.2 Deux suggestions :

1. Simuler deux nombres indépendants x_1 et x_2 d'une loi exponentielle de paramètre λ et poser $y = x_1 - x_2$.
2. La fonction de répartition de la distribution de Laplace est

$$F_Y(y) = \begin{cases} \frac{1}{2}e^{\lambda y}, & y < 0 \\ 1 - \frac{1}{2}e^{-\lambda y}, & y \geq 0, \end{cases}$$

d'où

$$F_Y^{-1}(u) = \begin{cases} \frac{1}{\lambda} \ln(2u) & u < 0,5 \\ -\frac{1}{\lambda} \ln(2(1-u)) & u \geq 0,5. \end{cases}$$

Il est donc aussi possible d'utiliser la méthode de l'inverse.

2.3 Soit $X \sim \text{Exponentielle}(\lambda)$ et $K = \lfloor X \rfloor$. Par conséquent :

$$\begin{aligned}\Pr[K = 0] &= \Pr[\lfloor X \rfloor = 0] \\ &= \Pr[0 \leq X < 1] \\ &= F_X(1) \\ &= 1 - e^{-\lambda}, \\ \Pr[K = 1] &= \Pr[1 \leq X < 2] \\ &= F_X(2) - F_X(1) \\ &= (1 - e^{-2\lambda}) - (1 - e^{-\lambda}) \\ &= (e^{-\lambda} - e^{-2\lambda}) \\ &= (1 - e^{-\lambda})e^{-\lambda},\end{aligned}$$

soit, de manière générale,

$$\begin{aligned}\Pr[K = k] &= \Pr[k \leq X < k + 1] \\ &= (1 - e^{-\lambda})(e^{-\lambda})^k, \quad k = 0, 1, 2, \dots\end{aligned}$$

La forme de cette fonction de masse de probabilité est celle d'une loi géométrique de paramètre $p = 1 - e^{-\lambda}$.

2.4 Voir la fonction R de la [figure C.5](#). Les expressions suivantes permettent de tracer les graphiques pour vérifier la validité de l'algorithme.

```
> hist(rgamma2(1000, 5), prob = TRUE)
> curve(dgamma(x, 5, 1), add = TRUE)
```

2.5 Deux suggestions.

1. Si $X \sim \text{Gamma}(\alpha, 1)$, alors $Y = X/\lambda \sim \text{Gamma}(\alpha, \lambda)$. Nous pouvons donc générer un nombre x d'une loi $\text{Gamma}(\alpha, 1)$ avec l'algorithme de l'[exercice 2.4](#), puis poser $y = x/\lambda$.
2. Si α est entier, nous pouvons également générer α nombres x_1, \dots, x_α d'une distribution $\text{Exponentielle}(\lambda)$ et poser $y = \sum_{i=1}^{\alpha} x_i$.

2.6 a) Vous aurez reconnu dans cette combinaison de variables aléatoires un mélange d'une exponentielle et d'une gamma. Par la loi des probabilités totales,

$$\begin{aligned}f_X(x) &= \int_0^\infty f(x|\theta)u(\theta) d\theta \\ &= \frac{\lambda^\alpha}{\Gamma(\alpha)} \int_0^\infty \theta^{\alpha+1-1} e^{-(\lambda+x)\theta} d\theta\end{aligned}$$

```

rgamma2 <- function(nsim, alpha)
{
  x <- numeric(nsim)
  i <- 0
  while (i < nsim)
  {
    u <- runif(2)
    v <- (alpha - 1/(6 * alpha)) * u[1] /
          ((alpha - 1) * u[2])

    if ((2 * (u[2] - 1)/(alpha - 1) +
          v + 1/v <= 2) |
        (2 * log(u[2])/(alpha - 1) -
          log(v) + v <= 1))
      x[i <- i + 1] <- (alpha - 1) * v
  }
  x
}

```

FIG. C.5 – Fonction de simulation d'une distribution $\text{Gamma}(\alpha, 1)$ avec $\alpha > 1$

$$\begin{aligned}
 &= \frac{\lambda^\alpha}{\Gamma(\alpha)} \frac{\Gamma(\alpha + 1)}{(\lambda + x)^{\alpha+1}} \\
 &= \frac{\alpha \lambda^\alpha}{(\lambda + x)^{\alpha+1}}.
 \end{aligned}$$

- b) Pour générer un nombre d'une distribution de Pareto de paramètres α et λ avec le résultat en a), il faut d'abord obtenir un nombre θ d'une distribution gamma de mêmes paramètres, puis obtenir un nombre x d'une distribution exponentielle de paramètre θ .

```
> rexp(1, rgamma(1, alpha, lambda))
```

2.7 La fonction de répartition de la Pareto translatée(α, λ) est

$$\begin{aligned}
 F(x) &= \int_{\lambda}^x \frac{\alpha \lambda^\alpha}{y^{\alpha+1}} dy \\
 &= \begin{cases} 0, & x \leq \lambda \\ 1 - \left(\frac{\lambda}{x}\right)^\alpha, & x > \lambda \end{cases}
 \end{aligned}$$

et son inverse est

$$F^{-1}(y) = \begin{cases} \lambda, & y = 0 \\ \frac{\lambda}{(1-y)^{1/\alpha}}, & 0 < y < 1. \end{cases}$$

Par conséquent, si $U \sim U(0, 1)$, alors

$$X = \frac{\lambda}{(1-U)^{1/\alpha}} \sim \text{Pareto translatée}(\alpha, \lambda).$$

Les trois premières valeurs retournées par le générateur

$$x_n = (65x_{n-1} + 1) \bmod 2048$$

avec une amorce de 12 sont 781, 1 614 et 463. La division de ces nombres par 2 048 fournit des nombres dans l'intervalle $(0, 1)$:

$$0,3813 \quad 0,7881 \quad 0,2261.$$

Finalement, les observations de la Pareto(2, 1 000) sont les suivantes.

```
> 1000/sqrt(1 - c(0.3813, 0.7881, 0.2261))
[1] 1271.335 2172.374 1136.731
```

Remarque : puisque $1 - U \sim U(0, 1)$ si $U \sim U(0, 1)$, les nombres issus de la transformation $\lambda(1 - U)^{-1/\alpha}$ seraient tout aussi distribués selon une Pareto translatée. Les réponses seraient toutefois différentes.

2.8 Nous avons la transformation

$$\begin{aligned} X_1 &= \sqrt{U_1} \cos(2\pi U_2) & U_1 &= X_1^2 + X_2^2 \\ X_2 &= \sqrt{U_1} \sin(2\pi U_2) & \Leftrightarrow U_2 &= \frac{1}{2\pi} \arctan \frac{X_2}{X_1}. \end{aligned}$$

Cette transformation associe les points de l'espace $\{(u_1, u_2) : 0 < u_1 < 1, 0 < u_2 < 1\}$ à ceux de l'espace $\{(x_1, x_2) : x_1^2 + x_2^2 < 1 \setminus (0, 0)\}$. Cela se vérifie aisément en examinant les limites de l'espace de départ :

$$\begin{aligned} u_1 > 0 & \Rightarrow x_1^2 + x_2^2 > 0 \\ u_1 < 1 & \Rightarrow x_1^2 + x_2^2 < 1 \\ u_2 > 0 & \Rightarrow \frac{x_2}{x_1} > 0 \\ u_2 < 1 & \Rightarrow \frac{x_2}{x_1} < 0. \end{aligned}$$

Les troisième et quatrième inégalités définissent les quadrants I et III, puis II et IV de \mathbb{R}^2 , respectivement. Vous remarquerez que le point $(0, 0)$, qui a probabilité zéro, ne se trouve pas dans l'espace image.

Le Jacobien de la transformation est

$$\begin{aligned} J &= \begin{vmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} \end{vmatrix} \\ &= \begin{vmatrix} 2x_1 & 2x_2 \\ -\frac{1}{2\pi} \frac{x_2}{x_1^2 + x_2^2} & \frac{1}{2\pi} \frac{x_1}{x_1^2 + x_2^2} \end{vmatrix} \\ &= \frac{1}{\pi}. \end{aligned}$$

La fonction de densité de probabilité conjointe de X_1 et X_2 est donc

$$\begin{aligned} f_{X_1, X_2}(x_1, x_2) &= f_{U_1, U_2}(u_1, u_2) |J| \\ &= \frac{1}{\pi}, \quad -1 < x_1 < 1, -\sqrt{1 - x_1^2} < x_2 < \sqrt{1 - x_1^2}, \end{aligned}$$

soit une distribution uniforme sur le disque unité.

Le résultat peut évidemment servir à simuler des points uniformément répartis sur un disque de rayon 1 centré en $(0, 0)$. La [figure C.6](#) illustre d'ailleurs cette transformation. Les points (u_1, u_2) dans le graphique de gauche sont tirés aléatoirement sur le carré $(0, 1) \times (0, 1)$. Le graphique de droite montre que suite à la transformation ci-dessus, les points (x_1, x_2) sont distribués uniformément sur un disque de rayon 1 centré en $(0, 0)$.

2.9 a) Nous avons

$$\begin{aligned} f_{Y_1}(y_1) &= \frac{1}{\Gamma(\alpha)} y_1^{\alpha-1} e^{-y_1}, \quad y_1 > 0, \\ f_{Y_2}(y_2) &= \frac{1}{\Gamma(\beta)} y_2^{\beta-1} e^{-y_2}, \quad y_2 > 0 \end{aligned}$$

et

$$f_{Y_1, Y_2}(y_1, y_2) = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} y_1^{\alpha-1} y_2^{\beta-1} e^{-(y_1+y_2)}, \quad y_1 > 0, y_2 > 0.$$

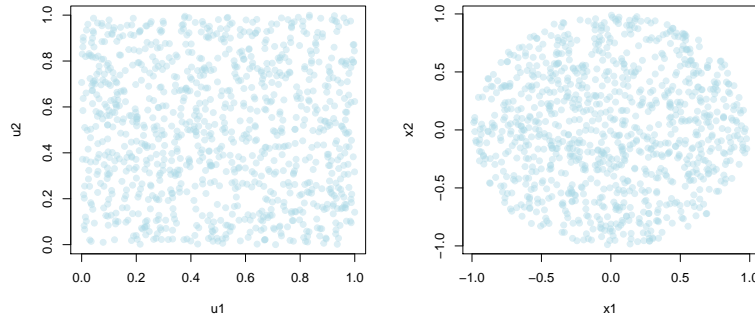


FIG. C.6 - Démonstration graphique du fonctionnement de la transformation de l'exercice 2.8

Soit $X_1 = Y_1/(Y_1 + Y_2)$ et $X_2 = Y_1 + Y_2$ (le choix de X_2 étant justifié par l'exposant de la distribution conjointe de Y_1 et Y_2). Nous cherchons la distribution conjointe de X_1 et X_2 , $f_{X_1, X_2}(x_1, x_2)$ à partir de la transformation

$$\begin{aligned} x_1 &= \frac{y_1}{y_1 + y_2} & \Leftrightarrow & \quad y_1 = x_1 x_2 \\ x_2 &= y_1 + y_2 & & \quad y_2 = x_2 - x_1 x_2. \end{aligned}$$

Cette transformation associe de manière évidente les points de l'espace $\{(y_1, y_2) : y_1 > 0, y_2 > 0\}$ à ceux de l'espace $\{(x_1, x_2) : 0 < x_1 < 1, x_2 > 0\}$.

Le Jacobien de la transformation est

$$\begin{aligned} J &= \begin{vmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{vmatrix} \\ &= \begin{vmatrix} x_2 & x_1 \\ -x_2 & 1 - x_1 \end{vmatrix} \\ &= x_2. \end{aligned}$$

La fonction de densité de probabilité conjointe de X_1 et X_2 est donc

$$\begin{aligned} f_{X_1, X_2}(x_1, x_2) &= f_{Y_1, Y_2}(y_1, y_2) |J| \\ &= \frac{1}{\Gamma(\alpha)\Gamma(\beta)} x_1^{\alpha-1} (1-x_1)^{\beta-1} x_2^{\alpha+\beta-1} e^{-x_2} \\ &= \left[\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x_1^{\alpha-1} (1-x_1)^{\beta-1} \right] \left[\frac{1}{\Gamma(\alpha+\beta)} x_2^{\alpha+\beta-1} e^{-x_2} \right], \end{aligned}$$

pour $0 < x_1 < 1$, $x_2 > 0$, d'où X_1 et X_2 sont indépendantes, $X_1 \sim \text{Bêta}(\alpha, \beta)$ et $X_2 \sim \text{Gamma}(\alpha + \beta)$ (un résultat connu).

b) La conversion du résultat en un algorithme est très simple :

1. Générer y_1 d'une distribution $\text{Gamma}(\alpha, 1)$.
2. Générer y_2 d'une distribution $\text{Gamma}(\beta, 1)$.
3. Poser $x = y_1 / (y_1 + y_2)$.

Cet algorithme suppose évidemment qu'une source de nombres provenant d'une loi gamma est disponible.

La [figure C.7](#) illustre le fonctionnement de cette transformation. Dans le graphique de gauche, nous avons un nuage de points (y_1, y_2) tirés de manière indépendante de deux distributions gamma de paramètre d'échelle égal à 1. Nous avons superposé ce nuage de points aux courbes de niveaux de la distribution conjointe des deux lois gamma.

Dans le graphique de droite, les points $x = y_1 / (y_1 + y_2)$ résultant de la transformation sont placés en abscisse. Vous constaterez que la répartition et la densité de ces points correspond à la densité de la loi bêta également représentée sur le graphique.

c) En R :

```
> (y <- rgamma(1, alpha, 1))/(y + rgamma(1, beta, 1))
```

2.10 a) Nous avons

$$\begin{aligned} \Pr \left[U \leq \frac{f_X(y)}{c g_Y(y)} \right] &= \int_{-\infty}^{\infty} \Pr \left[U \leq \frac{f_X(y)}{c g_Y(y)} \middle| Y = y \right] g_Y(y) dy \\ &= \int_{-\infty}^{\infty} \frac{f_X(y)}{c g_Y(y)} g_Y(y) dy \\ &= \frac{1}{c} \int_{-\infty}^{\infty} f_X(y) dy \\ &= \frac{1}{c}. \end{aligned}$$

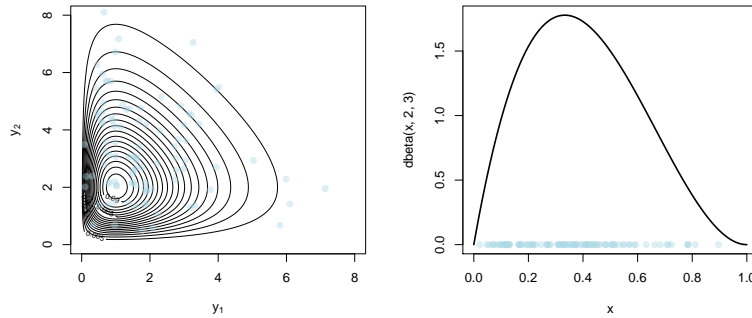


FIG. C.7 – Démonstration graphique du fonctionnement de la transformation de l'exercice 2.9

- b) Les essais étant indépendants, la distribution du nombre d'essais avant d'obtenir un succès (accepter un nombre y) est géométrique de paramètre $1/c$, c'est-à-dire que

$$\Pr[Z = z] = \left(\frac{1}{c}\right) \left(1 - \frac{1}{c}\right)^{z-1}, \quad z = 1, 2, \dots,$$

où Z représente le nombre d'essais avant d'accepter un nombre.

- c) Nous savons que $E[Z] = 1/(1/c) = c$.

2.11 a) Posons

$$cg_Y(x) = M, \quad a < x < b,$$

soit $Y \sim U(a, b)$ et $c = M(b - a)$. L'algorithme d'acceptation-rejet est donc le suivant :

1. Simuler deux nombres indépendants u_1 et u_2 d'une loi $U(0, 1)$.
2. Poser $y = a + (b - a)u_1$.
3. Si $u_2 \leq f_X(y)/M$, poser $x = y$. Sinon, retourner à l'étape 1.

- b) L'efficacité est

$$\frac{1}{c} = \frac{1}{M(b - a)}.$$

2.12 a) Il est facile de démontrer que le mode M d'une distribution bêta de paramètres α et β se trouve en

$$x = \frac{\alpha - 1}{\alpha + \beta - 2}.$$

```

rbeta.ar2 <- function(n)
{
  g <- function(x)
    ifelse(x < 0.8, 2.5 * x, 10 - 10 * x)
  Ginv <- function(y)
    ifelse(y < 0.8, sqrt(0.8 * y),
           1 - sqrt(0.2 - 0.2 * y))
  x <- numeric(n)
  i <- 0
  while(i < n)
  {
    y <- Ginv(runif(1))
    if(1.2 * g(y) * runif(1) <=
       dbeta(y, shape1 = 3, shape2 = 2))
      x[i <- i + 1] <- y
  }
  x
}

```

FIG. C.8 - Code R de la fonction `rbeta.ar2`

Par conséquent, l'efficacité de l'algorithme d'acceptation-rejet décrit à l'[exercice 2.11](#) et consistant à borner la densité par un rectangle de hauteur M est

$$\begin{aligned}
 \frac{1}{M} &= \frac{1}{f((\alpha - 1)/(\alpha + \beta - 2))} \\
 &= \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)} \left(\frac{\alpha - 1}{\alpha + \beta - 2} \right)^{1-\alpha} \left(\frac{\beta - 1}{\alpha + \beta - 2} \right)^{1-\beta}.
 \end{aligned}$$

Avec $\alpha = 3$ et $\beta = 2$, l'efficacité est $9/16$.

- b) Nous avons obtenu $c = 1,2$ dans l'[exemple 2.5](#), soit une efficacité de $1/c = 5/6$. Cet algorithme est évidemment plus efficace puisque la surface de l'enveloppe de la densité bêta est nettement plus petite.
- c) Utilisons l'algorithme développé à l'[exemple 2.5](#). Une première mise en œuvre de l'algorithme en R est fournie dans le code informatique de la [section 2.9](#). La [figure C.8](#) en propose une autre. Vous pouvez vérifier l'exactitude la fonction `rbeta.ar2` avec les expressions suivantes.

```
> x <- rbeta.ar2(10000)
> hist(x, prob = TRUE)
> curve(dbeta(x, 3, 2), add = TRUE)
```

- 2.13** a) Vous reconnaitrez l'algorithme d'acceptation-rejet de l'exercice 2.11.
 b) Vous devez simuler deux observations d'une loi Bêta(2, 3) dont la fonction de densité de probabilité est

$$f(x) = 12x(1-x)^2, \quad 0 < x < 1.$$

Le mode de cette densité se trouve en $x = 1/3$ (voir la solution de l'exercice 2.12) et la valeur de ce mode est $M = f(1/3) = 16/9$. Pour obtenir le résultat de l'appel de la fonction `simul`, il faut s'assurer d'utiliser les nombres uniformes dans le bon ordre. Quatre itérations de la boucle `repeat` seront nécessaires. Voici leurs résultats.

1. On a $u = 0,72$, puis $(16/9)(0,88) > f(0,72)$, donc u est rejeté.
2. On a $u = 0,76$, puis $(16/9)(0,89) > f(0,76)$, donc u est rejeté.
3. On a $u = 0,46$, puis $(16/9)(0,17) < f(0,46)$, donc u est accepté : $x_1 = 0,46$.
4. On a $u = 0,33$, puis $(16/9)(0,51) < f(0,33)$, donc u est accepté : $x_2 = 0,33$.

Le résultat est donc le vecteur $\mathbf{x} = (0,46, 0,33)$.

- 2.14** a) Si $0 \leq x \leq 1$, $e^{-1} < e^{-x} < 1$, d'où $x^{\alpha-1}e^{-x} \leq x^{\alpha-1}$. De même, puisque $0 < \alpha < 1$, $x^{\alpha-1} < 1$ pour $x > 1$, d'où $x^{\alpha-1}e^{-x} \leq e^{-x}$ pour $x > 1$.
 b) Il faut borner la densité $f_X(x) = x^{\alpha-1}e^{-x}/\Gamma(\alpha)$, $x > 0$ et $0 < \alpha < 1$. Du résultat en a) :

$$f_X(x) \leq \begin{cases} x^{\alpha-1}/\Gamma(\alpha), & 0 \leq x \leq 1 \\ e^{-x}/\Gamma(\alpha), & x > 1. \end{cases}$$

Posons

$$cg_Y(x) = \begin{cases} x^{\alpha-1}/\Gamma(\alpha), & 0 \leq x \leq 1 \\ e^{-x}/\Gamma(\alpha), & x > 1. \end{cases}$$

L'aire totale sous la fonction $cg_Y(x)$ est

$$\int_0^1 \frac{x^{\alpha-1}}{\Gamma(\alpha)} dx + \int_1^\infty \frac{e^{-x}}{\Gamma(\alpha)} dx = \frac{1}{\Gamma(\alpha)} \left(\frac{1}{\alpha} + \frac{1}{e} \right),$$

d'où

$$g_Y(x) = \begin{cases} \frac{x^{\alpha-1}}{(1/\alpha) + (1/e)}, & 0 \leq x \leq 1 \\ \frac{e^{-x}}{(1/\alpha) + (1/e)}, & x > 1, \end{cases}$$

$$G_Y(x) = \begin{cases} \frac{e}{\alpha + e} x^\alpha, & 0 \leq x \leq 1 \\ 1 - \frac{e^{-x}}{(1/\alpha) + (1/e)}, & x > 1, \end{cases}$$

et

$$G_Y^{-1}(x) = \begin{cases} \left(\frac{\alpha + e}{e} x \right)^{1/\alpha}, & 0 \leq x \leq e/(\alpha + e) \\ -\ln[(1/\alpha) + (1/e)(1 - x)], & e/(\alpha + e) < x \leq 1. \end{cases}$$

Or,

$$\frac{f_X(x)}{c g_Y(x)} = \begin{cases} e^{-x}, & 0 \leq x \leq 1 \\ x^{\alpha-1}, & x > 1. \end{cases}$$

Il découle de ce précède l'algorithme de simulation suivant.

1. Simuler deux nombres u_1 et u_2 d'une $U(0, 1)$.
2. Poser $y = G_Y^{-1}(u_1)$.
3. Si

$$u_2 \leq \begin{cases} e^{-y}, & 0 \leq y \leq 1 \\ y^{\alpha-1}, & y > 1, \end{cases}$$

alors poser $x = y$. Sinon, retourner à l'étape 1.

2.15 Vous devez simuler des observations de la fonction de densité de probabilité $f_X(x) = x^{\alpha-1} e^{-x} / \Gamma(\alpha)$ avec $\alpha \geq 1$. Or, vous savez que

$$f_X(x) \leq \frac{\alpha^\alpha}{\Gamma(\alpha)} e^{1-\alpha} \frac{1}{\alpha} e^{-x/\alpha}, \quad x > 0,$$

d'où $f_X(x) \leq c g_Y(x)$ avec

$$c = \frac{\alpha^\alpha}{\Gamma(\alpha)} e^{1-\alpha}$$

et

$$g_Y(x) = \frac{1}{\alpha} e^{-x/\alpha}.$$

Ainsi, $Y \sim \text{Exponentielle}(1/\alpha)$. Soit y une observation de la variable aléatoire Y et u une observation d'une loi $U(0, 1)$. Selon l'algorithme d'acceptation-rejet, on accepte la valeur y comme observation d'une loi $\text{Gamma}(\alpha, 1)$ avec $\alpha \geq 1$ si

$$\begin{aligned}
 u &\leq \frac{f_X(y)}{c g_Y(y)} = y^{\alpha-1} \frac{e^{-y(1-1/\alpha)}}{\alpha^{\alpha-1} e^{-(\alpha-1)}} \\
 &\Leftrightarrow \\
 u^{1/(\alpha-1)} &\leq \left(\frac{y}{\alpha}\right) \frac{e^{-y/\alpha}}{e^{-1}} \\
 &\Leftrightarrow \\
 \ln u &\leq (\alpha-1) \left[\ln\left(\frac{y}{\alpha}\right) - \frac{y}{\alpha} + 1 \right] \\
 &\Leftrightarrow \\
 -\ln u &> (\alpha-1) \left[\frac{y}{\alpha} - \ln\left(\frac{y}{\alpha}\right) - 1 \right].
 \end{aligned}$$

Or, tant la distribution de $-\ln U$ que celle de Y/α est une exponentielle de moyenne 1, d'où l'algorithme donné dans l'énoncé.

2.16 Il faut déterminer le nombre d'observations provenant de chacune des k distributions F_1, \dots, F_k sachant que la probabilité qu'une observation provienne de la distribution j est p_j . La distribution conjointe du nombre d'observations de chaque distribution est une multinomiale de paramètres p_1, \dots, p_k dont la fonction de masse de probabilité est :

$$\Pr[X_1 = x_1, \dots, X_k = x_k] = \frac{n!}{x_1! \dots x_k!} \prod_{j=1, \dots, k} p_j^{x_j},$$

où $n = x_1 + \dots + x_k$. La fonction `rmultinom` de R permet de simuler des valeurs x_1, \dots, x_k pour n et p_1, \dots, p_k données. C'est la stratégie employée par la fonction `rmixture` de **actuar**.

2.17 Dans chacun des cas ci-dessous, nous considérons que l'objet `n` contient le nombre de valeurs à simuler.

a) Il s'agit de simuler des observations d'un mélange discret d'une Exponentielle(2) et d'une Gamma(2, 1/3). Nous pouvons procéder en suivant la procédure explicitée à la [section 2.10](#) ou avec la fonction `rmixture` du paquetage **actuar**.

```
> n1 <- rbinom(1, n, 0.3)
> c(rexp(n1, 2), rgamma(n - n1, 2, scale = 3))

> rmixture(n, c(0.3, 0.7),
+           expression(rexp(2), rgamma(2, scale = 3)))
```

- b) Le mélange discret est, ici, celui d'une distribution dégénérée en $x = 0$ et d'une Poisson de paramètre $\lambda = 2$. La simulation « manuelle » est en tous points similaire à ce que nous avons fait en a). Cependant, à moins de créer une fonction de « simulation » d'une distribution dégénérée, nous ne pouvons utiliser directement la fonction `rmixture`.

```
> n1 <- rbinom(1, n, 0.5647)
> c(rep(0, n1), rpois(n - n1, 2))
```

Le paquetage **actuar** fournit néanmoins une fonction pour simuler des observations d'un tel mélange. En effet, celui-ci est simplement une distribution de Poisson avec une probabilité en $x = 0$ de $p_0 = 0,5647 + 0,4353e^{-2} = 0,6236$, une distribution autrement connue sous le nom de Poisson zéro modifiée. Les fonctions avec racine `zmpois` de **actuar** permettent de travailler avec cette distribution.

```
> rzmpois(n, 2, p0 = 0.6236)
```

- c) Nous devons simuler des observations de la distribution Poisson-inverse gaussienne qui, par ailleurs, est définie par le mélange continu présenté dans la question. Nous pouvons procéder « manuellement » à l'aide de la fonction `rinvgauss` du paquetage **actuar**, ou encore plus directement avec la fonction `rpoisinvgauss` du même paquetage.

```
> rpois(n, rinvgauss(n, 2, 1))
```

```
> rpoisinvgauss(n, 2, 1)
```

- d) Nous devons simuler des observations d'une distribution Poisson composée. Le paquetage **actuar** fournit la fonction `rcomppois` pour ce faire.

```
> sapply(rpois(n, 2),
+       function(n) sum(rlnorm(n, 7, 1)))
```

```
> rcomppois(n, 2, rlnorm(7, 1))
```

- e) Il faut cette fois simuler des observations d'une distribution binomiale négative composée. Le paquetage **actuar** fournit la fonction `rcompound` pour ce cas général.

```
> sapply(rnbinom(n, 2, 0.3),
+        function(n) sum(rlnorm(n, 7, 1)))
```

```
> rcompound(n, rnbinom(2, 0.3), rlnorm(7, 1))
```

Chapitre 3

3.1 On a

$$\begin{aligned}\theta &= \int_0^1 \ln(5u + 4) dx \\ &= E[\ln(5U + 4)],\end{aligned}$$

où $U \sim U(0, 1)$, ou encore simplement

$$\theta = E[\ln(X)],$$

où $X \sim U(4, 9)$. Une approximation de θ est

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n \ln(x_i),$$

où x_1, \dots, x_n est un échantillon aléatoire d'une distribution $U(4, 9)$. Voici le résultat d'une évaluation avec R.

```
> mean(log(runif(1e6, 4, 9)))
[1] 1.846088
```

3.2 Posons

$$\begin{aligned}\theta &= \int_0^1 \int_0^1 e^{2xy} \ln(3x + y^2) dx dy \\ &= E[e^{2XY} \ln(3X + Y^2)],\end{aligned}$$

où $X \sim U(0, 1)$, $Y \sim U(0, 1)$ et X et Y sont indépendantes. Ainsi, leur densité conjointe est uniforme sur $(0, 1) \times (0, 1)$. Une approximation de θ est

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n e^{2x_i y_i} \ln(3x_i + y_i^2)$$

où x_1, \dots, x_n et y_1, \dots, y_n sont deux échantillons aléatoires indépendants d'une distribution $U(0, 1)$. Voici le résultat d'une évaluation avec R.

```
> x <- runif(1e6)
> y <- runif(1e6)
> mean(exp(2 * x * y) * log(3 * x + y^2))
[1] 1.203413
```

- 3.3 a) Soit le changement de variable $u = e^{-x/2} \Leftrightarrow x = -\ln u^2$, d'où $-2du = e^{-x/2}dx$. On a donc

$$\begin{aligned}\theta &= \int_0^1 2(-\ln u^2)^2 \sin(-\pi \ln u^2) du \\ &= 2E[(-\ln U^2)^2 \sin(-\pi \ln U^2)],\end{aligned}$$

où $U \sim U(0, 1)$. Une estimation de θ est

$$\hat{\theta} = \frac{2}{n} \sum_{i=1}^n (-\ln u_i^2)^2 \sin(-\pi \ln u_i^2),$$

où u_1, \dots, u_n est un échantillon aléatoire d'une distribution $U(0, 1)$. Voici le résultat d'une évaluation avec R.

```
> u <- runif(1e6)
> 2 * mean((-log(u^2))^2 * sin(pi * (-log(u^2))))
[1] -0.02215899
```

- b) On remarque que la fonction à intégrer contient, à une constante près, la fonction de densité de probabilité d'une loi Gamma(3, 1/2). Ainsi,

$$\begin{aligned}\theta &= 16 \int_0^\infty \sin(\pi x) \frac{(1/2)^3}{\Gamma(3)} x^2 e^{-x/2} dx \\ &= 16E[\sin(\pi X)],\end{aligned}$$

où $X \sim \text{Gamma}(3, 1/2)$. Une estimation de θ est donc

$$\hat{\theta} = \frac{16}{n} \sum_{i=1}^n \sin(\pi x_i)$$

où x_1, \dots, x_n est un échantillon aléatoire d'une Gamma(3, 1/2). Voici le résultat d'une évaluation avec R.

```
> 16 * mean(sin(pi * rgamma(1e6, 3, 0.5)))  
[1] -0.04380848
```

3.4 Vous pouvez effectuer l'estimation avec R simplement avec les expressions suivantes.

```
> x <- replicate(10000, rnorm(25))  
> mean(apply(x, 2, function(x) sort(x)[5]))  
[1] -0.9082946
```


Bibliographie

- Devroye, L. 1986, *Non-Uniform Random Variate Generation*, Springer-Verlag, ISBN 0-38796305-7. URL <http://luc.devroye.org/rnbookindex.html>. Cité à la page 15.
- Dutang, C., V. Goulet et M. Pigeon. 2008, «actuar: An R package for actuarial science», *Journal of Statistical Software*, vol. 25, n° 7. URL <http://www.jstatsoft.org/v25/i07>. Cité à la page 30.
- Knuth, D. E. 1997, *The Art of Computer Programming*, vol. 2, Seminumerical Algorithms, Addison-Wesley, ISBN 0-20189684-2. Cité à la page 5.
- McCullough, B. D. et D. A. Heiser. 2008a, «Microsoft Excel's 'Not The Wichmann-Hill' random number generators», *Computational statistics and data analysis*, vol. 52, n° 10, p. 4587-4593. Cité à la page 8.
- McCullough, B. D. et D. A. Heiser. 2008b, «On the accuracy of statistical procedures in Microsoft Excel 2007», *Computational statistics and data analysis*, vol. 52, n° 10, p. 4570-4578. Cité à la page 8.
- Microsoft Office Blog. 2009, «Function improvements in Excel 2010», URL <http://blogs.office.com/b/microsoft-excel/archive/2009/09/10/function-improvements-in-excel-2010.aspx>. Cité à la page 9.
- Sylvester, J. J. 1865, «On a special class of questions on the theory of probabilities», *Birmingham British Assoc. Rept.*, p. 8-9. Cité à la page 1.
- Weisstein, E. W. «Sylvester's four-point problem», From MathWorld - A Wolfram Web Resource. URL <http://mathworld.wolfram.com/SylvestersFour-PointProblem.html>. Cité à la page 1.
- Wheeler, B. 2013, *SuppDists: Supplementary Distributions*. URL <https://cran.r-project.org/package=SuppDists>, R package version 1.1-9. Cité à la page 30.

Ce document a été produit avec le système de mise en page \LaTeX . Le texte principal est composé en Lucida Bright OT 11 points, les mathématiques en Lucida Bright Math OT, le code informatique en Lucida Grande Mono DK et les titres en Fira Sans. Des icônes proviennent de la police Font Awesome. Les graphiques ont été réalisés avec R.

