ACT 2002
Méthodes numériques en actuariat

Partie II
Simulation stochastique

UNIVERSITÉ LAVAL
Faculté des sciences et de génie
École d'actuariat

ACT 2002
Méthodes numériques en actuariat

Partie II
Simulation stochastique

Vincent Goulet
Professeur titulaire | École d'actuariat | Université Laval

Avec la collaboration de
Laurent Caron

Notes de cours | Exercices — édition 2016

Code source

Code informatique des sections d'exemples — Télécharger

Code source du document — Accéder au dépôt

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. On n'a qu'à penser 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 collision virtuels, ou encore aux prévisions météo qui ne en fait 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 7.

En actuariat, nous avons habituellement besoin de nombres aléatoires provenant d'une loi de probabilité non uniforme. Le chapitre 8 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 de Excel à ce chapitre.

Enfin, cette partie du cours se termine au chapitre 9 par une application à première vue inusitée de la simulation, soit le calcul d'intégrales définies par la méthode dite Monte Carlo.

Chaque chapitre propose un problème à résoudre au fil du texte. L'énoncé du problème, les indications en cours de chapitre et la solution complète se présentent dans des encadrés de couleur contrastante et marqués des symboles ⚙, ϟ et 💡.

L'étude de ce document implique quelques allers-retours entre le texte et les sections de code informatique présentes dans chaque chapitre. Les sauts vers ces sections sont clairement indiqués dans le texte par des mentions mises en évidence par le symbole ⏭.

Tous les chapitres comportent des exercices. Les réponses de ceux-ci se retrouvent à la fin de chacun des chapitres et les solutions complètes, en annexe. En consultation électronique, le numéro d'un exercice est un hyperlien vers sa solution, et vice versa.

On trouvera également en annexe un bref exposé sur la planification d'une simulation en R et des rappels sur la transformation de variables aléatoires.

Les fichiers de code informatique des sections d'exemples sont disponibles dans le Portail libre de l'École d'actuariat. On peut y accéder facilement en suivant le lien fourni à la page des notices de copyright.

Un symbole de lecture vidéo dans la marge indique qu'une capsule vidéo est disponible dans la chaîne YouTube du cours sur le sujet en hyperlien.

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. Je remercie également Mmes Marie-Pier Laliberté et Véronique Tardif pour l'infographie des pages couvertures.

Table des matières

Introduction v

7 Génération de nombres aléatoires uniformes 1
- 7.1 Pourquoi faire de la simulation ? 2
- 7.2 Générateurs de nombres aléatoires 4
- 7.3 Congruence et modulo 5
- 7.4 Générateurs congruentiels linéaires 6
- 7.5 Générateurs utilisés dans Excel, VBA et R 8
- 7.6 Code informatique 12
- 7.7 Exercices 14

8 Simulation de nombres aléatoires non uniformes 17
- 8.1 Méthode de l'inverse 18
- 8.2 Méthode acceptation-rejet 25
- 8.3 Fonctions de simulation de variables aléatoires dans Excel, VBA et R 30
- 8.4 Autres techniques de simulation 34
- 8.5 Code informatique 37
- 8.6 Exercices 44

9 Intégration Monte Carlo 51
- 9.1 Contexte 52
- 9.2 Méthode Monte Carlo 52
- 9.3 Code informatique 60
- 9.4 Exercices 62

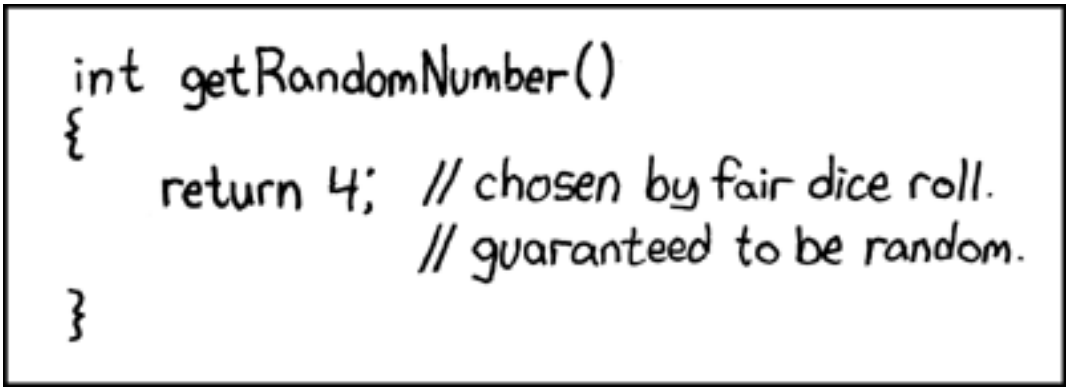
A Planification d'une simulation en R 65
- A.1 Contexte 65
- A.2 Première approche : avec une boucle 66
- A.3 Seconde approche : avec `sapply` 66

A.4 Variante de la seconde approche 70
A.5 Gestion des fichiers 70
A.6 Exécution en lot 72
A.7 Conclusion 72

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 7 87
Chapitre 8 89
Chapitre 9 103

Bibliographie 107



Tiré de XKCD.com

7 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, VBA 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.

Énoncé du problème

L'enveloppe convexe (*convex hull*) d'un ensemble de points (dans le plan, pour simplifier) est l'ensemble des points qui, lorsque reliés entre eux, forment une structure convexe contenant tous les autres points. La figure 7.1 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).

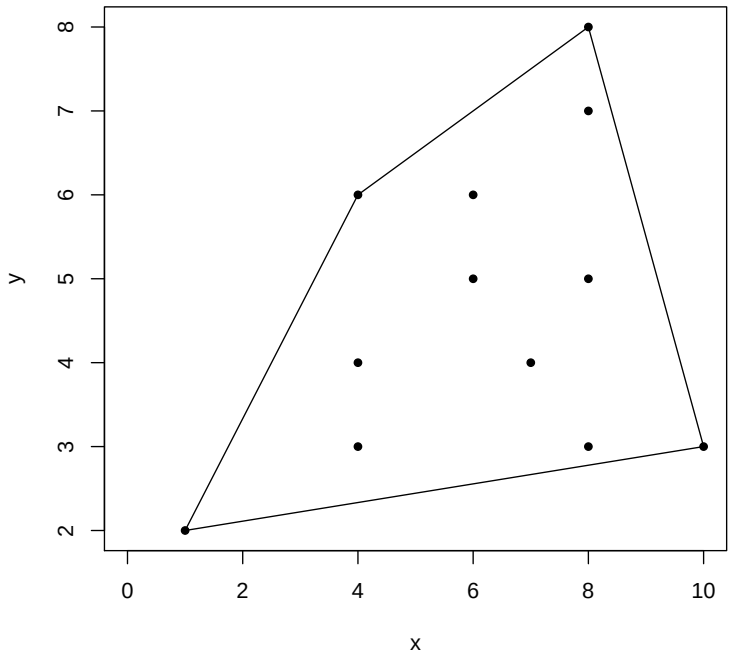


FIG. 7.1 – Exemple d'enveloppe convexe d'un ensemble de points

Énoncé du problème (suite)

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

7.1 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 stochastiques 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 primodiale 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.

Astuce

Nous pourrons faire la vérification demandée par 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 répètant cette procédure un grand nombre de fois, nous pourrons 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

Astuce (suite)

des points faisant partie de l'enveloppe. Par exemple, les coordonnées cartésiennes des points de la figure 7.1 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 :

```
> chull(x, y)
[1] 12  2  9  1
```

7.2 Générateurs de nombres aléatoires

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

1. On peut 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 (tablaux RAND) ;
   - les intervalles de temps dans un processus de décroissance radioactive sont considérés parfaitement aléatoires ; le site HotBits [1] 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 si 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

1. http://www.fourmilab.ch/hotbits/

les quatre chiffres du milieu, et ainsi de suite. Par exemple :

$$8653^2 = 74\mathbf{8744}09$$
$$8744^2 = 76\mathbf{4575}36$$
$$4575^2 = 20\mathbf{9306}25$$

3. On peut 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 *reproduisibles*. 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, on exige 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 reproduisible à partir d'un point de départ donné, mais qu'il soit autrement impossible à prédire.

7.3 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 7.1. 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,

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

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 7.1. On a

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

- R : `%%` ;
- Excel : `MOD()` ;
- VBA : `%`.

7.4 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*»).

Lorsque l'incrément $c = 0$, on parle d'un générateur *multiplicatif* ; dans le cas contraire on a un générateur *mixte*.

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

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

Astuce

Ce 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.

Remarques.

1. La méthode de génération des nombres est entièrement déterministe, c'est pourquoi il convient mieux de parler de nombres *pseudo*-aléatoires.
2. 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 on cherche donc à avoir la période la plus longue possible, tout en obtenant des suites en apparence aléatoires.
3. Pour les générateurs multiplicatifs ($c = 0$), on atteint la période maximale $m - 1$ 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 7.2. Soit un générateur congruentiel multiplicatif avec $a = 7$ et $m = 31$. Les quatre premiers nombres pseudo-aléatoires avec l'amorce $x_0 = 19$ sont :

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

□

Exemple 7.3. Cet exemple illustre l'effet des différents paramètres d'un générateur congruentiel sur la qualité des nombres pseudo-aléatoires produits.

⏭ Exécuter le code informatique R de la section 7.6 correspondant à cet exemple.

□

Exemple 7.4. Un générateur apparemment de qualité en une dimension peut rapidement se révéler médiocre dans les dimensions supérieures. Le présent exemple en fait la démonstration en deux dimensions avec un générateur tout simple, alors que l'exercice 7.3 reprend les mêmes idées en trois dimensions avec un générateur longtemps considéré standard.

⏭ Exécuter le code informatique R de la section 7.6 correspondant à cet exemple.

□

7.5 Générateurs utilisés dans Excel, VBA et R

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

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

7.5.1 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 aléatoire 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) RNG [2], 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 dé-

2. *Random Number Generator*

tails qui permettraient de dissiper tout doute sur la qualité des nombres aléatoires fournis pas Excel.

7.5.2 Générateur de VBA

La fonction `RND()` de VBA génère un nombre aléatoire. Selon l'article 231847 de la base de connaissances Microsoft (http://support.microsoft.com/kb/231847), le générateur de nombres aléatoires utilisé par la fonction `RND()` est un simple générateur congruentiel linéaire.

Étant donné l'avancement actuel des connaissances dans le domaine des générateurs de nombres pseudo-aléatoires, un tel générateur 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, on doit donc à tout prix éviter la fonction `RND()` et de lui préférer un appel à la fonction `RAND()` de Excel.

7.5.3 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 on veut 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 défaut, R utilise le générateur Marsenne–Twister, considéré comme le plus avancé en ce moment. La période de ce générateur est $2^{19\,937} - 1$ (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`.

Astuce

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

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. On a :

```
> 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, on aura recours, tel qu'expliqué à l'annexe A, à la fonction `replicate`. Puisque chaque expression exécutée par `replicate` retourne `TRUE` ou `FALSE`, le résultat de

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

sera un vecteur booléen. Comme on le sait, ces valeurs booléennes sont converties en valeurs numériques 0 et 1 pour les calculs algébriques. La valeur moyenne du vecteur booléen correspond donc à la proportion d'enveloppes convexes triangulaires au fil des 100 000 simulations.
Au final, on obtient :

```
> mean(replicate(1E5,
+      is.triangle(x = runif(4), y = runif(4))))
[1] 0.30665
```

soit une valeur suffisamment proche de $\frac{11}{36} \approx 0{,}30556$ pour juger l'expérience concluante.

7.6 Code informatique

```
###
### EXEMPLE 7.3
###

## On définit tout d'abord une petite fonction pour calculer
## les valeurs successives d'un générateur congruentiel
## linéaire général.
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]
}

## On peut toujours obtenir une période maximale de m avec un
## générateur congruentiel en posant a = c = 1, mais l'aspect
## aléatoire de la suite de nombres obtenus en prend alors
## pour son rhume...
rand(17, a = 1, c = 1, m = 16, seed = 0)

## Avec un meilleur choix de multiplicateur et d'incrément, la
## période est tout aussi longue, mais la suite davantage
## aléatoire.
rand(17, a = 5, c = 1, m = 16, seed = 0)

## Un tout petit changement et la période est beaucoup plus
## courte. Même la valeur de l'amorce a une influence sur la
## période.
rand(17, a = 5, c = 4, m = 16, seed = 0)
rand(17, a = 5, c = 4, m = 16, seed = 1)
rand(17, a = 5, c = 4, m = 16, seed = 2)

## Le générateur multiplicatif de l'exemple 7.2 ne satisfait
## pas les conditions pour que la période soit maximale (7
## n'est pas une racine primitive de 31).
rand(32, a = 7, c = 0, m = 31, seed = 19)
length(unique(rand(32, a = 7, c = 0, m = 31, seed = 19)))

## Avec a = 3, on atteint la période de 30 car 3 est une
## racine primitive de 31.
```

```
length(unique(rand(32, a = 3, c = 0, m = 31, seed = 19)))

###
### EXEMPLE 7.4
###

## Un générateur avec une période de 30. Les valeurs obtenues
## semblent assez aléatoires.
(x <- rand(30, a = 3, c = 0, m = 31, seed = 19))

## Un graphique des valeurs successives ne montre pas de
## structure particulière.
par(pch = 19)
plot(x)

## Par contre, si l'on fait un graphique des paires de valeurs
## successives (c'est-à-dire x[1] en fonction de x[2], x[2] en
## fonction de x[3], etc.), un portrait dérangeant apparaît.
plot(head(x, -1), tail(x, -1))

## Augmenter la valeur du multiplicateur améliore la
## situation. Si l'on utilise plutôt a = 12 (une racine
## primitive), alors la période est toujours de 30 et donc les
## valeurs de la suite sont les mêmes que ci-dessus, dans un
## ordre différent. Cependant, les valeurs sont mieux
## distribuées.
x <- rand(30, a = 12, c = 0, m = 31, seed = 9)
plot(head(x, -1), tail(x, -1))

## On illustre par six autres graphiques comment le
## choix des paramètres d'un générateur congruentiel
## peut avoir une importance majeure sur la qualité des
## nombres générés.
par(mfrow = c(3, 2)) # 6 graphiques sur 3 lignes, 2 colonnes

x <- rand(2048, a = 65, c = 1, m = 2048, seed = 0)
plot(tail(x, -1), head(x, -1), main = "a = 65, c = 1")

x <- rand(2048, a = 1365, c = 1, m = 2048, seed = 0)
plot(tail(x, -1), head(x, -1), main = "a = 1365, c = 1")

x <- rand(2048, a = 1229, c = 1, m = 2048, seed = 0)
plot(tail(x, -1), head(x, -1), main = "a = 1229, c = 1")

x <- rand(2048, a = 157, c = 1, m = 2048, seed = 0)
```

```
plot(tail(x, -1), head(x, -1), main = "a = 157, c = 1")

x <- rand(2048, a = 45, c = 0, m = 2048, seed = 1234)
plot(tail(x, -1), head(x, -1), main = "a = 45, c = 0")

x <- rand(2048, a = 43, c = 0, m = 2048, seed = 1234)
plot(tail(x, -1), head(x, -1), main = "a = 43, c = 0")
```

7.7 Exercices

L'exercice 7.3 requiert d'installer le package R `rgl`, disponible sur CRAN[3]. Tel qu'expliqué à l'annexe B des notes de cours de la partie I, on installe le package dans R avec la commande

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

Pour utiliser les fonctions du package, il faut ensuite charger le package en mémoire avec

```
> library(rgl)
```

L'utilisation de `rgl` sous Mac OS X, requiert l'application `X11.app`, qui n'est plus livrée avec le système depuis la version 10.8 (Mountain Lion). Un serveur et des bibliothèques client X11 compatibles avec les versions de OS X 10.6 et suivantes sont disponibles à partir du projet XQuartz. Il suffit de télécharger l'application depuis http://xquartz.macosforge.org/.

7.1 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) $a = 11, c = 0$

7.2 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 ?

3. http://cran.r-project.org

b) Répéter la partie a) avec $a = 85$.

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

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

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. On se contente 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 package `rgl`, puis faire pivoter le graphique pour voir les quinze plans sur lesquels se trouvent répartis les points.

8 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, VBA 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 $F_X(x)$. Ceci 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. D'autres méthodes populaires en actuariat sont mentionnées à la section 8.4.

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.

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 $\frac{35}{12\pi^2} \approx 0{,}29552$.

8.1 Méthode de l'inverse

La méthode de l'inverse repose sur une idée toute simple, soit que l'on peut 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 8.1. *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*.

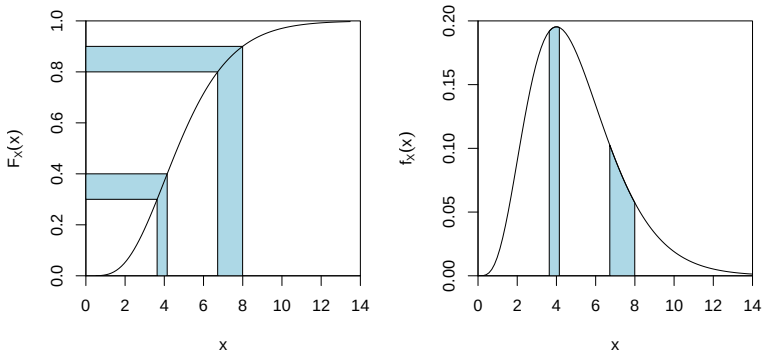


FIG. 8.1 – Représentation graphique de la méthode de l'inverse. À des intervalles égaux en ordonnée correspondent des intervalles différents en abscisse selon la forme de la distribution. La fonction de répartition à gauche correspond à la densité de droite.

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 8.1 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.

Remarque. Dans Excel (et VBA), on doit nécessairement utiliser la méthode de l'inverse. Plusieurs fonctions de quantiles sont disponibles (section 8.3).

8.1.1 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 8.1. On veut 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 $U \sim U(0,1) \Leftrightarrow 1 - U \sim U(0,1)$, on peut se 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 $x = -\lambda^{-1}\ln u$.

Exécuter le code informatique R de la section 8.5 correspondant à cet exemple pour une illustration de l'algorithme ci-dessus.

□

8.1.2 Distributions discrètes

On peut 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 $a < u < b$, on pose $x = x_0$. Ainsi, x_0 sera simulé dans une proportion $b - a$ du temps, ce qui correspond à $\Pr[X = x_0]$. Voir la figure 8.2 pour une illustration.

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

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

On procède ainsi parce que plusieurs générateurs produisent des nombres uniformes sur $[0, 1)$.

Exemple 8.2. Soit X une variable aléatoire avec fonction de densité de probabilité

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

On a une variable aléatoire *mixte* (en partie continue et en partie continue) dont la fonction de répartition est

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

(Voir la figure 8.3.)

Par conséquent, un algorithme pour simuler des nombres aléatoires de cette distribution est :

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

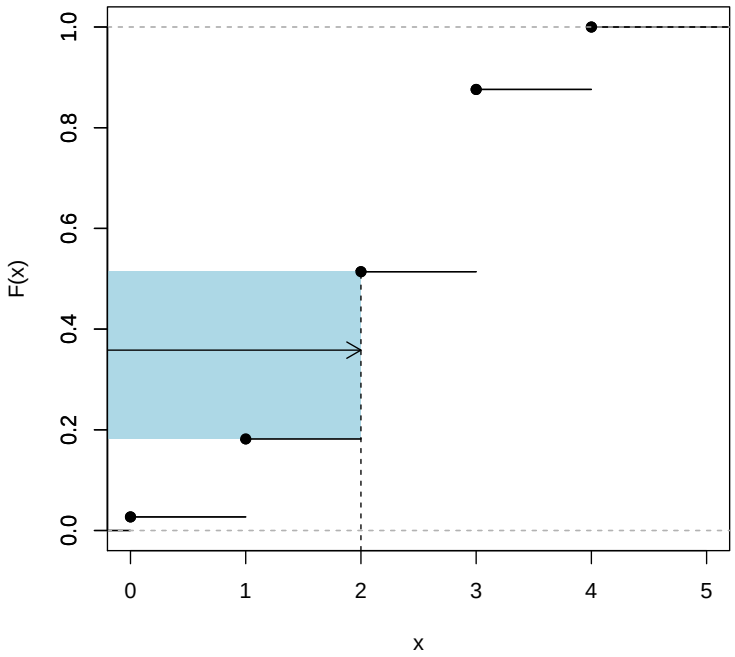


FIG. 8.2 – Illustration de la méthode de l'inverse pour une distribution discrète. Tout nombre uniforme tiré dans la zone colorée est converti en la même valeur, ce qui correspond à la probabilité d'obtenir cette valeur (la hauteur du saut).

2. Poser

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

⏭ Une mise en œuvre en R de l'algorithme ci-dessus est présentée dans le code informatique de la section 8.5.

□

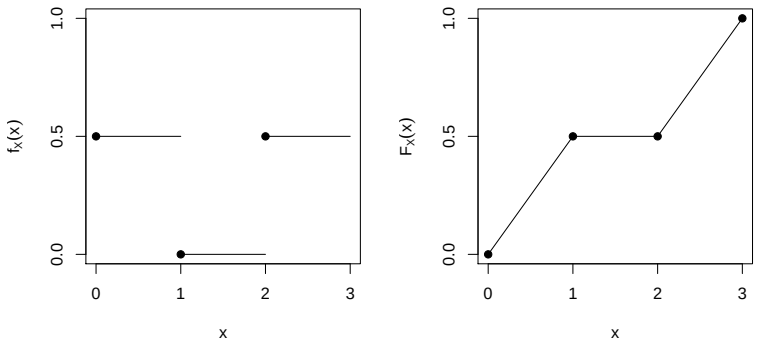


FIG. 8.3 – Fonction de densité de probabilité (gauche) et fonction de répartition (droite) de l'exemple 8.2

Exemple 8.3. On veut simuler des observations d'une distribution binomiale de paramètres n et θ. On pourrait, 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 :

1. Simuler n nombres uniformes indépendants $u_1, \ldots, u_n$ 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

□

Remarque. 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()`.

Astuce

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

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

▶

Astuce (suite)

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.

8.2 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 on peut 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 on a l'algorithme d'acceptation-rejet suivant :

1. Générer une réalisation y de la variable aléatoire Y avec fonction de densité de probabilité $g_Y(\cdot)$.
2. Générer 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.

Remarque. Puisque, par définition, l'aire sous les densités f et g vaut 1 dans les deux cas on ne peut avoir $cg_Y(x) \geq f_X(x)$ 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. Dans l'illustration de la figure 8.4, 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). On constate aisément que certaines valeurs y seront acceptées plus souvent que d'autres en conformité avec la forme de la densité.

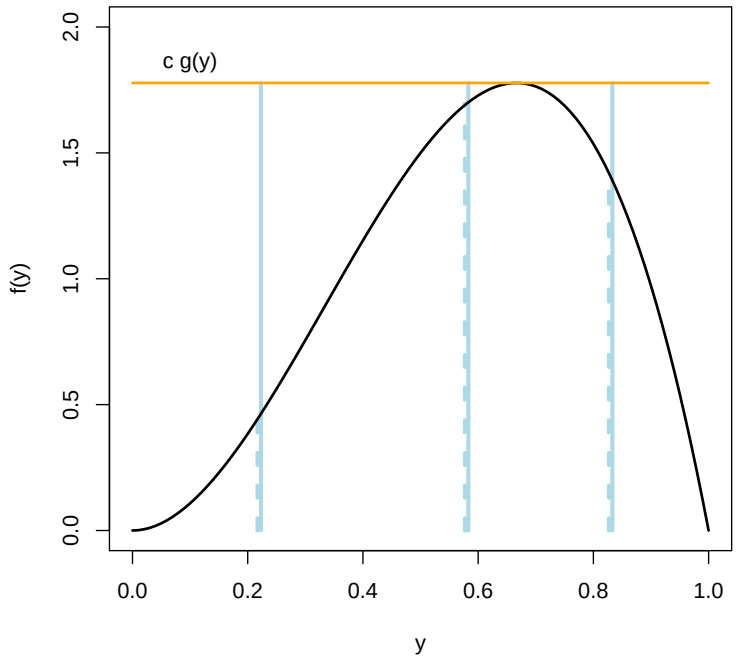


FIG. 8.4 – Illustration de la méthode acceptation-rejet

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

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

où u est un nombre issu d'une loi $U(0,1)$. Graphiquement, cela signifie que l'on accepte la valeur y si le point $(y, ucg_Y(y))$ se trouve sous la courbe f en y et qu'on le rejette s'il se trouve entre f et l'enveloppe. La figure 8.5 illustre cette interprétation.

Remarques.

1. La principale difficulté avec la méthode d'acceptation-rejet consiste à trouver la fonction enveloppante $cg_Y(x)$.
2. Évidemment, plus l'enveloppe est près de $f_X(x)$, plus l'algorithme est performant puisque l'on rejette alors moins de valeurs.

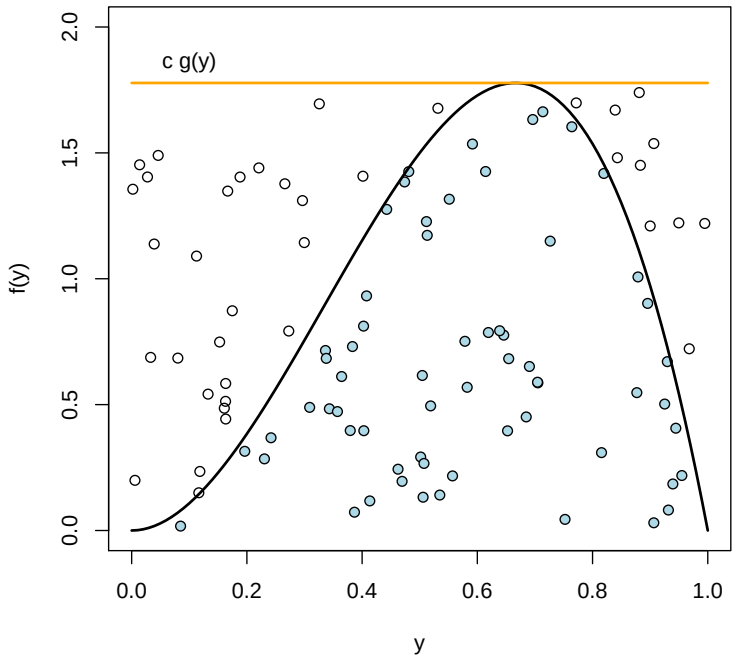


FIG. 8.5 – Interprétation alternative de la méthode acceptation-rejet. Chaque point représente un couple $(y, ucg_Y(y))$. On accepte les points pleins et on rejette les points vides.

3. Il est aussi possible d'envelopper une densité à support infini… par une autre densité à support fini ; voir les exercices.

Exemple 8.4. Soit $X \sim \text{Bêta}(3,2)$. On a

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

C'est la densité représentée aux figures 8.4 et 8.5. On l'inscrit facilement 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,

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

On déduit que $c = 16/9$ et que la densité g est une uniforme sur $(0,1)$. On a 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 $16/9$, on peut s'attendre à rejeter

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

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. □

Exemple 8.5. On reprend l'exemple 8.4 en tentant d'améliorer l'efficacité de l'algorithme. Il s'avère que l'on peut inscrire la densité de la loi Bêta$(3,2)$ dans un triangle aux caractéristiques suivantes (voir la figure 8.6) :

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 $x = 1$, on trouve 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,

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

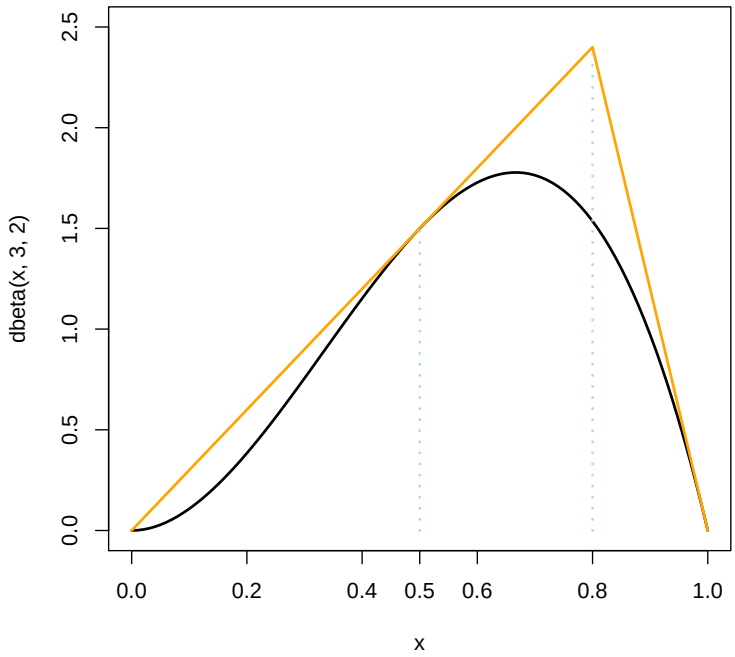


FIG. 8.6 – Fonction de densité de probabilité d'une loi Bêta$(3,2)$ enveloppée d'un triangle

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

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

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, on a 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
$$u_2 \leq \frac{f_X(y)}{1{,}2 g_Y(y)} \Leftrightarrow 1{,}2 g_Y(y) u_2 \leq f_X(y),$$
alors poser $x = y$. Sinon, retourner à l'étape 1.

L'aire du triangle étant de 1,2, on peut maintenant s'attendre à ne rejeter que 20 % des nombres simulés, une amélioration importante par rapport à l'exemple 8.4. □

Astuce

L'interprétation géométrique de la méthode d'acceptation-rejet illustrée à la figure 8.5 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 7, puis de rejeter ceux qui ne se trouvent pas à l'intérieur du disque. La figure 8.7 illustre cette idée : nous accepterions les points pleins et nous rejetterions les points vides.

8.3 Fonctions de simulation de variables aléatoires dans Excel, VBA 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 on peut avoir besoin pour un usage quotidien.

8.3.1 Excel et VBA

Tel que mentionné à la section 3.1, 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.

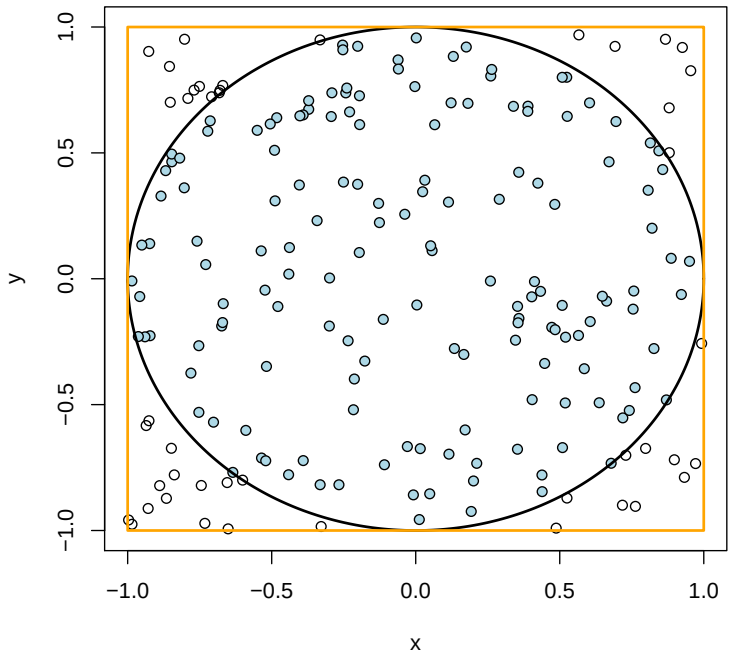


FIG. 8.7 – Simulation de points sur un disque par une méthode d'acceptation-rejet

Ainsi, on trouvera dans Excel 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 probabilité présentées au tableau 8.1. Les noms anglais des fonctions ont été modifiés pour être standardisés dans Excel 2010. On remarquera que les traducteurs français ont omis de faire de même.

Aucune fonction statistique n'existe dans VBA. On fera donc appel aux fonctions de Excel en préfixant les noms de fonctions *anglais* du tableau 8.1 de la propriété `WorksheetFunction`. En utilisant une structure

```
With WorksheetFunction
    ...
End With
```

Loi de probabilité	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 négative	NEGBINOM.DIST	LOI.BINOMIALE.NEG.N
Exponentielle	EXPON.DIST	LOI.EXPONENTIELLE.N
F (Fisher)	F.DIST	LOI.F.N
	F.INV	INVERSE.LOI.F.N
Gamma	GAMMA.DIST	LOI.GAMMA.N
	GAMMA.INV	LOI.GAMMA.INVERSE.N
Hypergéométrique	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
t (Student)	T.DIST	LOI.STUDENT.N
	T.INV	LOI.STUDENT.INVERSE.N
Weibull	WEIBULL.DIST	LOI.WEIBULL.N

TAB. 8.1 – Liste des fonctions relatives à des lois de probabilité depuis Excel 2010.

dans une routine VBA, il est également possible accéder directement aux fonctions Excel en les préfixant seulement d'un point.

8.3.2 R

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

1. d*loi* calcule la fonction de densité de probabilité (loi continue) ou la fonction de masse de probabilité (loi discrète) ;

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. 8.2 – Lois de probabilité pour lesquelles il existe des fonctions dans le système R de base

2. `p`*`loi`* calcule la fonction de répartition ;
3. `q`*`loi`* calcule la fonction de quantile ;
4. `r`*`loi`* simule des observations de cette loi.

Les différentes lois de probabilité disponibles dans le système R de base, leur racine et le nom de leurs paramètres sont rassemblés au tableau 8.2. Des packages fournissent des fonctions pour d'autres lois dont, entre autres, **actuar** (Dutang et collab., 2008) et **SuppDists** (Wheeler, 2013).

Toutes les fonctions du tableau 8.2 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,

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

retourne la probabilité que des lois de Poisson de paramètre 1, 4 et 10 prennent les valeurs 3, 0 et 8, dans l'ordre.

Le premier argument de toutes les fonctions de simulation est la quantité de nombres aléatoires désirée. Ainsi,

```
> rpois(3, lambda = c(1, 4, 10))
[1]  4  4 12
```

retourne trois nombres aléatoires issus de distributions de Poisson de paramètre 1, 4 et 10, respectivement. Évidemment, passer un vecteur comme premier argument n'a pas tellement de sens, mais, si c'est fait, R retournera une quantité de nombres aléatoires égale à la *longueur* du vecteur (sans égard aux valeurs contenues dans le vecteur).

La fonction `sample` permet de simuler des nombres d'une distribution discrète quelconque. Sa syntaxe est

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

où `x` est un vecteur des valeurs possibles de l'échantillon à simuler (le support de la distribution), `size` est la quantité de nombres à simuler et `prob` est un vecteur de probabilités associées à chaque valeur de `x` (`1/length(x)` par défaut). Enfin, si `replace` est `TRUE`, l'échantillonnage se fait avec remise.

Le code informatique R de la section 8.5 contient des exemples plus détaillés d'utilisation des fonctions mentionnées ci-dessus.

8.4 Autres techniques de simulation

Il existe plusieurs autres algorithmes de simulation de loi non uniformes, certains beaucoup plus élaborés que d'autres. Nous n'irons pas plus loin dans ce sujet dans ce cours, sinon pour mentionner deux autres techniques simples qui sont souvent employées en actuariat :

1. les mélanges de distributions continus et discrets ;
2. la convolution de variables aléatoires.

Ces techniques sont présentées et illustrées à même le code informatique R de la section 8.5.

```
sim.disque <- function(n)
{
    ## La fonction retourne les coordonnées des points
    ## dans une matrice de deux colonnes. On crée
    ## 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)
        {
            ## ... on ajoute la paire à la matrice 'X'.
            X[i, ] <- x

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

FIG. 8.8 – Code d'une fonction pour simuler des nombres uniformément sur un disque de rayon 1 centré à l'origine par la méthode d'acceptation-rejet

En terminant, mentionnons que l'annexe A propose quelques trucs pour bien planifier une étude de simulation en R.

Solution du problème

Nous adoptons la stratégie expliquée à l'astuce de la page 30. La fonction R de la figure 8.8 genère 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 7 :

```
> mean(replicate(1E5, 3 == length(chull(sim.disque(4)))))
[1] 0.29453
```

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

$$x = \sqrt{r}\cos\theta$$
$$y = \sqrt{r}\sin\theta.$$

La figure 8.9 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 8.7.
Parce qu'elle ne requiert aucune boucle, cette seconde méthode est *beaucoup* plus rapide que la méthode d'acceptation-rejet :

```
> system.time(sim.disque(1E5))
   user  system elapsed
  0.719   0.009   0.738
> system.time(sim.disque2(1E5))
   user  system elapsed
  0.016   0.001   0.017
```

```
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. 8.9 – 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

8.5 Code informatique

```
###
### EXEMPLE 8.1
###

## Simulation un échantillon aléatoire d'une distribution
## exponentielle par la méthode de l'inverse.
lambda <- 5
x <- -log(runif(1000))/lambda

## Pour faire une petite vérification, on va tracer
## l'histogramme de l'échantillon et y superposer la véritable
## densité d'une exponentielle de paramètre lambda. Les deux
## graphiques devraient concorder.

## Tracé de l'histogramme. Il faut spécifier l'option 'prob =
## TRUE' pour que l'axe des ordonnées soit gradué en
## probabilités plutôt qu'en nombre de données. Sinon, le
## graphique de la densité que l'on va ajouter dans un moment
## n'apparaîtra pas sur le graphique.
hist(x, prob = TRUE) # histogramme gradué en probabilités

## Pour ajouter la densité, on a la très utile fonction
## curve() pour tracer une fonction f(x) quelconque. Avec
```

```
## l'option 'add = TRUE', le graphique est ajouté au graphique
## existant.
curve(dexp(x, rate = lambda), add = TRUE)

###
### EXEMPLE 8.4
###

## On trouvera ci-dessous une mise en oeuvre de l'algorithme
## d'acceptation-rejet pour simuler des observations d'une
## distribution Bêta(3, 2). La procédure est intrinsèquement
## itérative, alors nous devons utiliser une boucle. Il y a
## diverses manières de faire, j'ai ici utilisé une boucle
## 'repeat'; une autre mise en oeuvre est présentée dans les
## exercices.
##
## On remarque que le vecteur contenant les résultats est
## initialisé au début de la fonction pour éviter l'écueil de
## la «boîte à biscuits» expliqué à la section 4.5 du document
## de référence de la partie I.
rbeta.ar <- function(n)
{
    x <- numeric(n)         # initialisation du contenant
    g <- function(x)        # fonction enveloppante
        ifelse(x < 0.8, 2.5 * x, 10 - 10 * x)
    Ginv <- function(x)     # l'inverse de son intégrale
        ifelse(x < 0.8, sqrt(0.8 * x), 1 - sqrt(0.2 - 0.2 * x))

    i <- 0                  # initialisation du compteur
    repeat
    {
        y <- Ginv(runif(1))
        if (1.2 * g(y) * runif(1) <= dbeta(y, 3, 2))
            x[i <- i + 1] <- y # assignation et incrément
        if (i > n)
            break           # sortir de la boucle repeat
    }
    x                       # retourner x
}

## Vérification empirique pour voir si ça marche.
x <- rbeta.ar(1000)
hist(x, prob = TRUE)
curve(dbeta(x, 3, 2), add = TRUE)
```

```
###
### FONCTIONS DE SIMULATION DANS R
###

## La fonction de base pour simuler des nombres uniformes est
## 'runif'.
runif(10)                   # sur (0, 1) par défaut
runif(10, 2, 5)             # sur un autre intervalle
2 + 3 * runif(10)           # équivalent, moins lisible

## R est livré avec plusieurs générateurs de nombres
## aléatoires. On peut en changer avec la fonction 'RNGkind'.
RNGkind("Wichmann-Hill")    # générateur de Excel
runif(10)                   # rien de particulier à voir
RNGkind("default")          # retour au générateur par défaut

## La fonction 'set.seed' est très utile pour spécifier
## l'amorce d'un générateur. Si deux simulations sont
## effectuées avec la même amorce, on obtiendra exactement les
## mêmes nombres aléatoires et, donc, les mêmes résultats.
## Très utile pour répéter une simulation à l'identique.
set.seed(1)                 # valeur sans importance
runif(5)                    # 5 nombres aléatoires
runif(5)                    # 5 autres nombres
set.seed(1)                 # réinitialisation de l'amorce
runif(5)                    # les mêmes 5 nombres que ci-dessus

## Plutôt que de devoir utiliser la méthode de l'inverse ou un
## autre algorithme de simulation pour obtenir des nombres
## aléatoires d'une loi de probabilité non uniforme, R fournit
## des fonctions de simulation pour bon nombre de lois. Toutes
## ces fonctions sont vectorielles. Ci-dessous, P == Poisson
## et G == Gamma pour économiser sur la notation.
n <- 10                     # taille des échantillons
rbinom(n, 5, 0.3)           # Binomiale(5, 0,3)
rbinom(n, 1, 0.3)           # Bernoulli(0,3)
rnorm(n)                    # Normale(0, 1)
rnorm(n, 2, 5)              # Normale(2, 25)
rpois(n, c(2, 5))           # P(2), P(5), P(2), ..., P(5)
rgamma(n, 3, 2:11)          # G(3, 2), G(3, 3), ..., G(3, 11)
rgamma(n, 11:2, 2:11)       # G(11, 2), G(10, 3), ..., G(2, 11)

## La fonction 'sample' sert pour simuler d'une distribution
## discrète quelconque. Le premier argument est le support de
## la distribution et le second, la taille de l'échantillon
```

```
## désirée. Par défaut, l'échantillonnage se fait avec remise
## et avec des probabilités égales sur tout le support.
sample(1:49, 7)            # numéros pour le 7/49
sample(1:10, 10)           # mélange des nombres de 1 à 10

## On peut échantillonner avec remise.
sample(1:10, 10, replace = TRUE)

## On peut aussi spécifier une distribution de probabilités
## non uniforme.
x <- sample(c(0, 2, 5), 1000, replace = TRUE,
            prob = c(0.2, 0.5, 0.3))
table(x)                   # tableau de fréquences

###
### AUTRES TECHNIQUES DE SIMULATION
###

## MÉLANGES CONTINUS

## Un mélange continu de deux variables aléatoires est créé
## lorsque l'on suppose qu'un paramètre d'une distribution f
## est une réalisation d'une autre variable aléatoire avec
## densité u, comme ceci:
##
## X|Theta = theta ~ f(x|theta)
##           Theta ~ u(theta)
##
## Ce genre de modèle est fréquent en analyse bayesienne et
## souvent utilisé en actuariat. Certaines lois de probabilité
## sont aussi uniquement définies en tant que mélanges.
##
## L'intérêt, ici, est d'obtenir des observations de la
## variable aléatoire non conditionnelle X. L'algorithme de
## simulation est simple:
##
## 1. Simuler un nombre theta de la distribution de Theta.
## 2. Simuler une valeur x de la distribution de
##    X|Theta = theta.
##
## Ce qu'il importe de remarquer dans l'algorithme ci-dessus,
## c'est que le paramètre de mélange (theta) change pour
## chaque observation simulée. Autrement il n'y a juste pas de
## mélange, on obtient simplement un échantillon de la
## distribution f(x|theta).
```

```
##
## Les mélanges continus sont simples à faire en R puisque les
## fonctions de simulation sont vectorielles. Par exemple,
## simulons 1000 observations du mélange
##
## X|Theta = theta ~ Poisson(theta)
##           Theta ~ Gamma(5, 4)
theta <- rgamma(1000, 5, 4) # 1000 paramètres de mélange...
x <- rpois(1000, theta)     # ... pour 1000 Poisson différentes

## On peut écrire le tout en une seule expression.
x <- rpois(1000, rgamma(1000, 5, 4))

## On peut démontrer que la distribution non conditionnelle de
## X est une binomiale négative de paramètres 5 et 4/(4 + 1) =
## 0,8. Faisons une vérification empirique. On calcule d'abord
## le tableau de fréquences des observations de l'échantillon
## avec la fonction 'table'. Il existe une méthode de 'plot'
## pour les tableaux de fréquences.
(p <- table(x))             # tableau de fréquences
plot(p/length(x))           # graphique

## On ajoute au graphique les masses de probabilités théoriques.
(xu <- unique(x))           # valeurs distinctes de x
points(xu, dnbinom(xu, 5, 0.8), pch = 21, bg = "red3")

## MÉLANGES DISCRETS

## Le mélange discret est un cas limite du mélange continu
## lorsque la distribution de Theta est une Bernoulli de
## paramètre p, c'est-à-dire que Pr[Theta = 1] = p. Le
## résultat du mélange est une distribution avec densité de la
## forme
##
## f(x) = p f1(x) + (1 - p) f2(x),
##
## où f1 et f2 sont deux densités quelconques. 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.
##
## Par exemple, le mélange suivant de deux distributions
## log-normales résulte en une fonction de densité de
## probabilité bimodale, mais ayant néanmoins une queue
## similaire à celle d'une log-normale. Graphiquement:
```

```
op <- par(mfrow = c(3, 1)) # trois graphiques superposés
curve(dlnorm(x, 3.575, 0.637),
      xlim = c(0, 250),  ylab = "f(x)",
      main = expression(paste("Log-normale(",
          mu, " = 3,575, ", sigma, " = 0,637)")))
curve(dlnorm(x, 4.555, 0.265),
      xlim = c(0, 250), ylab = "f(x)",
      main = expression(paste("Log-normale(",
          mu, " = 4,555, ", sigma, " = 0,265)")))
curve(0.554 * dlnorm(x, 3.575, 0.637) +
      0.446 * dlnorm(x, 4.555, 0.265),
      xlim = c(0, 250),  ylab = "f(x)",
      main = expression(paste("Mélange (p = 0,554)")))
par(op)                    # revenir aux paramètres par défaut

## L'algorithme de simulation d'un mélange discret est
##
## 1. Simuler un nombre u uniforme sur (0, 1).
## 2. Si u <= p, simuler un nombre de f1(x), sinon simuler de
##    f2(x).
##
## La clé pour simuler facilement d'un mélange discret en R:
## réaliser que l'ordre des observations est sans importance
## et, donc, que l'on peut simuler toutes les observations de
## la première loi, puis toutes celles de la seconde loi. La
## seule chose dont il reste à tenir compte: le nombre
## d'observations qui provient de chaque loi; pour chaque
## observation, la probabilité qu'elle provienne de la
## première loi est p.
##
## Ici, on veut simuler des observations d'un mélange discret
## de deux lois log-normales, l'une de paramètres 3,575 et
## 0,637, l'autre de paramètres 4,555 et 0,265. Le paramètre
## de mélange est p = 0,554.
n <- 1000                  # taille de l'échantillon
n1 <- rbinom(1, n, 0.554)  # quantité provenant de la loi 1
x <- c(rlnorm(n1, 3.575, 0.637),     # observations de loi 1
       rlnorm(n - n1, 4.555, 0.265)) # observations de loi 2
hist(x, prob = TRUE)                 # histogramme
curve(0.554 * dlnorm(x, 3.575, 0.637) +
      0.446 * dlnorm(x, 4.555, 0.265),
      add = TRUE, lwd = 2, col = "red3") # densité théorique

## CONVOLUTIONS
```

```
## Une convolution est la somme de deux variables aléatoires
## indépendantes. De manière générale, une convolution est
## très compliquée à évaluer, même numériquement. Certaines
## convolutions sont toutefois bien connues:
##
## 1. une somme de Bernoulli est une binomiale;
## 2. une somme de géométriques est une binomiale négative;
## 3. une somme d'exponentielles est une gamma;
## 4. une somme de Poisson est une Poisson;
## 5. une somme de normales est une normale, etc.
##
## On peut utiliser ces résultats pour simuler des
## observations de certaines lois.
##
## Par exemple, pour simuler une observation d'une
## distribution Gamma(alpha, lambda), on peut sommer alpha
## observations d'une Exponentielle(lambda). Ces dernières
## sont obtenues par la méthode de l'inverse.
alpha <- 5
lambda <- 2
- sum(log(runif(alpha)))/lambda # une observation de la gamma

## Pour simuler un échantillon de taille n de la gamma, il
## faut simuler n * alpha observations de l'exponentielle. Il
## existe des algorithmes plus efficaces pour simuler d'une
## loi gamma.
n <- 1000                   # taille de l'échantillon
x <- - rowSums(matrix(log(runif(n * alpha)),
                      nrow = n))/lambda
hist(x, prob = TRUE)        # histogramme
curve(dgamma(x, alpha, lambda),
      add = TRUE, lwd = 2, col = "red3") # densité théorique

## La simulation peut aussi servir à estimer des
## caractéristiques de la distribution d'une convolution
## difficiles à calculer explicitement. Par exemple, supposons
## que l'on a
##
## X ~ Gamma(3, 4)
## Y ~ Gamma(5, 2)
##
## et que l'on souhaite calculer le 95e centile de X + Y. Il
## n'y a pas de solution explicite pour la distribution de X +
## Y puisque les deux lois gamma n'ont pas le même paramètre
## d'échelle (lambda). Procédons donc par simulation pour
```

```
## obtenir une approximation du résultat.
x <- rgamma(10000, 3, 4)   # échantillon de la première loi
y <- rgamma(10000, 5, 2)   # échantillon de la deuxième loi
quantile(x + y, 0.95)      # 95e centile de la convolution

## Il est laissé en exercice de faire le même calcul avec Excel.
```

8.6 Exercices

Les étudiants jugent souvent que les exercices de ce chapitre sont plus difficiles que ceux des autres chapitres. Les exercices font beaucoup appel à des notions de transformations de variables aléatoires qui ont pourtant déjà fait l'objet d'autres cours. Pour un rappel des principaux résultats dans ce domaine, consulter l'annexe B.

8.1 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 R. En R, on peut 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.

8.2 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.

8.3 Faire la mise en oeuvre informatique (dans le langage de votre choix) de l'algorithme de simulation suivant. Il s'agit d'un algorithme pour simuler des observations d'une loi Gamma$(\alpha, 1)$, où $\alpha > 1$.

1. Générer u_1 et u_2 indépendemment 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.

8.4 En utilisant le résultat de l'exercice précédent, quelle procédure pourrait-on suivre pour simuler des nombres d'une loi Gamma(α, λ) où $\alpha > 1$?

8.5 a) Démontrer que si $X|\Theta \sim$ Exponentielle(Θ) et $\Theta \sim$ Gamma(α, λ), alors $X \sim$ Pareto(α, λ). 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 oeuvre informatique de cet algorithme et les vérifications d'usage de sa validité.

8.6 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.

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

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

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 ?

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

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

b) Utiliser le résultat en a) pour proposer un algorithme de simulation d'observations d'une loi $\text{Bêta}(\alpha, \lambda)$.

c) Faire la mise en oeuvre informatique de l'algorithme en b) ainsi que les vérifications d'usage.

8.9 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.

8.10 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, on peut toujours inscrire la fonction de densité de probabilité de X dans un rectangle de hauteur M, ou M est la valeur de la densité au mode de X.

a) Énoncer l'algorithme d'acceptation-rejet découlant d'une telle procédure.

b) Calculer l'*efficacité* de l'algorithme en a), soit la probabilité d'accepter une valeur lors d'une itération de l'algorithme.

8.11 Considérer le problème de simulation d'observations d'une loi Bêta$(3, 2)$ à l'aide de la méthode d'acceptation-rejet.

a) Calculer l'efficacité de l'algorithme développé à l'exercice 8.10 et à l'exemple 8.4 pour le cas présent.

b) Calculer l'efficacité de l'algorithme développé dans l'exemple 8.5, où l'on a déterminé que

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

c) Faire la mise en oeuvre 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.

8.12 La fonction R de la figure 8.10 permet de simuler des observations de la distribution Bêta(α, β).

a) Identifier le type d'algorithme utilisé dans cette fonction.

b) On vous donne également les 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
```

```
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. 8.10 – Fonction de simulation d'une loi Bêta(α, β) pour l'exercice 8.12

Évaluer le résultat des expressions suivantes :

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

8.13 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).

8.14 On vous donne 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 suivant pour 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.

Réponses

8.6 1 271, 2 172, 1 137

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

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

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

8.12 b) 0.46, 0.33

9 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. On ne présente ici que l'idée générale.

Énoncé du problème

L'intégrale de Gauss

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

intervient dans plusieurs disciplines des mathématiques. En probabilités, elle est notamment à la base de la définition de la loi normale, ou loi gaussienne. Sans primitive, on l'utilise souvent, en calcul intégral, comme un exemple d'intégrale définie qu'il faut résoudre par la transformation du domaine en coordonnées polaires.
On sait que

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

Comment pourrait-on vérifier ce résultat à l'aide de la simulation ?

9.1 Contexte

Supposons que l'on souhaite calculer l'intégrale

$$\theta = \int_a^b h(x)\,dx, \tag{9.1}$$

mais que celle-ci est trop compliquée pour être évaluée explicitement. On pourrait 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 9.1 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.

9.2 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.

On exprime 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 :

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

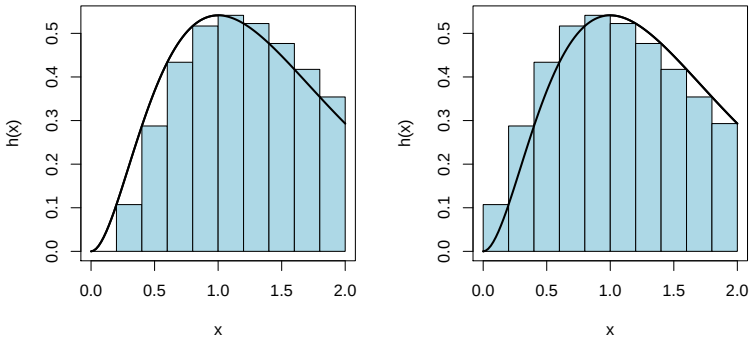


FIG. 9.1 – Approximation de l'aire sous une fonction par les deux sommes de Riemann R_n^- (gauche) et R_n^+ (droite)

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

$$\hat{\theta}_n = \frac{1}{n} \sum_{i=1}^{n} g(x_i) \tag{9.2}$$

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

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

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

Dans ce cas, on a $g(x) = (b-a)h(x)$ et l'intégrale de départ (9.1) se réécrit

$$\begin{aligned} \theta &= (b-a) \int_a^b h(x) \frac{1}{b-a} \, dx \\ &= (b-a) E[h(X)]. \end{aligned} \tag{9.3}$$

L'estimation (9.2) de l'intégrale devient alors

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

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

On remarquera qu'il est équivalent de faire le changement de variable

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

dans l'intégrale (9.3). 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 $u_1, \ldots, u_n$ est un échantillon aléatoire d'une loi $U(0,1)$. On doit utiliser la technique du changement de variable lorsque le domaine est infini.

Astuce

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 ▶

Astuce (suite)

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.

Exemple 9.1. 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, on ne peut évaluer cette intégrale par parties. Cependant, on remarque 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 Gamma(α,λ). Avec R, on obtient la valeur exacte

```
> 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, on pose

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

Si $\{x_1, \ldots, x_n\}$ 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. On a obtenu les résultats suivants avec R (voir aussi la figure 9.2) :

```
> f <- function(x) x^2.2 * exp(-x/10)
> x <- runif(1e2, 2, 5)
> 3 * mean(f(x))
[1] 34.20083
> x <- runif(1e3, 2, 5)
> 3 * mean(f(x))
[1] 34.35818
> x <- runif(1e4, 2, 5)
> 3 * mean(f(x))
[1] 34.77787
> x <- runif(1e5, 2, 5)
> 3 * mean(f(x))
[1] 34.48
> x <- runif(1e6, 2, 5)
> 3 * mean(f(x))
[1] 34.51308
```

Le code R pour effectuer les calculs et les graphiques ci-dessus est fourni à la section 9.3.

□

Exemple 9.2. Soit l'intégrale

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

La procédure à suivre avec les intégrales multiples est la même qu'avec les intégrales simples, sauf que l'on tire des points uniformément dans autant

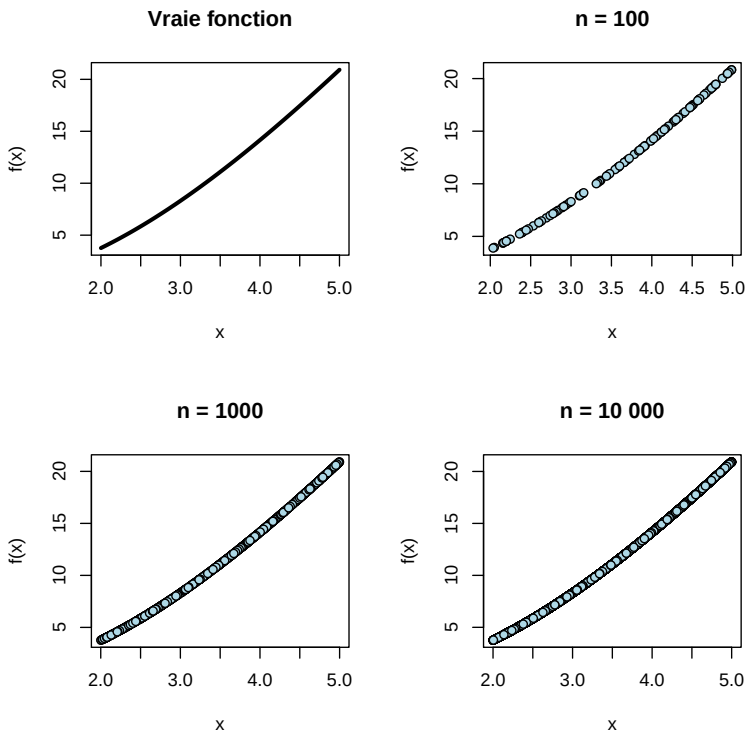


FIG. 9.2 – Fonction $h(x)$ de l'exemple 9.1 et points choisis aléatoirement où la fonction est évaluée pour l'intégration Monte Carlo

de dimensions que nécessaire. Ainsi, dans le cas présent, on pose

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

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 $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ tiré de cette loi conjointe est

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

On a obtenu les résultats suivants avec R (voir aussi la figure 9.3) :

```
> u <- runif(1e2, 0, 1.25)
> v <- runif(1e2, 0, 1.25)
> mean(sqrt(4 - u^2 - v^2)) * 1.25^2
[1] 2.676732
> u <- runif(1e3, 0, 1.25)
> v <- runif(1e3, 0, 1.25)
> mean(sqrt(4 - u^2 - v^2)) * 1.25^2
[1] 2.653669
> u <- runif(1e4, 0, 1.25)
> v <- runif(1e4, 0, 1.25)
> mean(sqrt(4 - u^2 - v^2)) * 1.25^2
[1] 2.673615
```

Le code R pour effectuer les calculs et les graphiques ci-dessus est fourni à la section 9.3.

□

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

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

▶

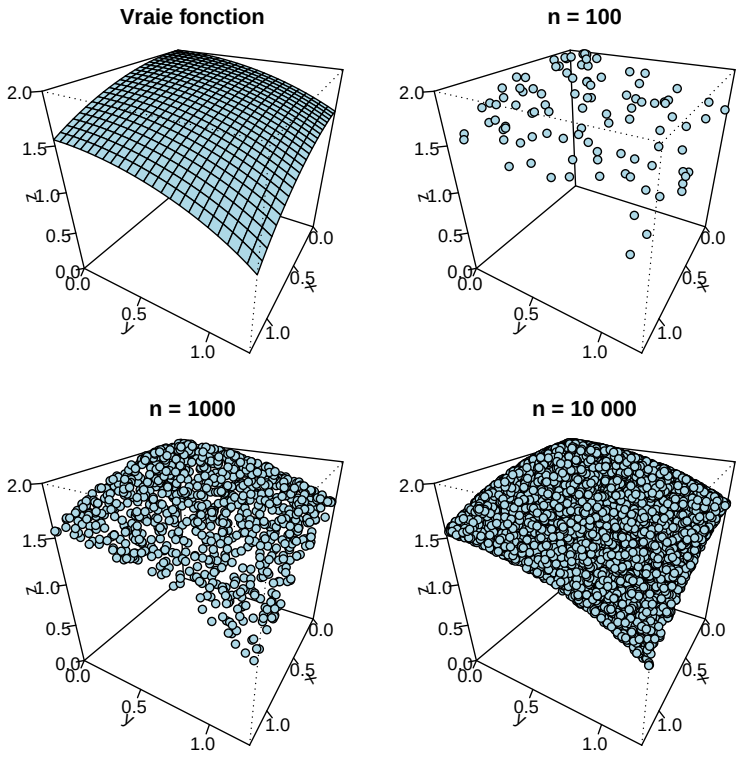


FIG. 9.3 – Fonction $h(x)$ de l'exemple 9.2 et points choisis aléatoirement où la fonction est évaluée pour l'intégration Monte Carlo

Solution du problème (suite)

pour n grand et où $u_1, \ldots, u_n$ sont des nombres aléatoires distribués uniformément sur l'intervalle $(0, 1)$.
Le calcul est ensuite très simple à faire avec R :

```
> u <- runif(1E5)
> 2 * (mean(exp(-u^2) + exp(-u^(-2)) * u^(-2)))
[1] 1.772589
```

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

9.3 Code informatique

```
###
### EXEMPLE 9.1
###

## Estimations successives de l'intégrale par la méthode Monte
## Carlo avec des échantillons de plus en plus grands.
f <- function(x) x^2.2 * exp(-x/10)
x <- runif(1e2, 2, 5)
3 * mean(f(x))
x <- runif(1e3, 2, 5)
3 * mean(f(x))
x <- runif(1e4, 2, 5)
3 * mean(f(x))
x <- runif(1e5, 2, 5)
3 * mean(f(x))
x <- runif(1e6, 2, 5)
3 * mean(f(x))

## On trace des graphiques de la vraie fonction à intégrer et
## de trois évaluations par la méthode Monte Carlo.
op <- par(mfrow = c(2, 2)) # 4 graphiques en grille 2 x 2
curve(f(x), xlim = c(2, 5), lwd = 2, main = "Vraie fonction")

x <- runif(1e2, 2, 5)
plot(x, f(x), main = "n = 100",
     pch = 21, bg = "lightblue")

x <- runif(1e3, 2, 5)
plot(x, f(x), main = "n = 1000",
     pch = 21, bg = "lightblue")

x <- runif(1e4, 2, 5)
plot(x, f(x), main = "n = 10 000",
     pch = 21, bg = "lightblue")

par(op)                    # revenir aux paramètres par défaut

###
### EXEMPLE 9.2
###

## Estimations successives de l'intégrale double par la
```

```
## méthode Monte Carlo avec des échantillons de plus en plus
## grands.
u <- runif(1e2, 0, 1.25)
v <- runif(1e2, 0, 1.25)
mean(sqrt(4 - u^2 - v^2)) * 1.25^2
u <- runif(1e3, 0, 1.25)
v <- runif(1e3, 0, 1.25)
mean(sqrt(4 - u^2 - v^2)) * 1.25^2
u <- runif(1e4, 0, 1.25)
v <- runif(1e4, 0, 1.25)
mean(sqrt(4 - u^2 - v^2)) * 1.25^2

## Graphiques de la vraie fonction et des points où celle-ci
## est évaluée avec la méthode Monte Carlo. Pour faire un
## graphique en trois dimensions, on peut utiliser la fonction
## 'persp'.
op <- par(mfrow = c(2, 2), mar = c(1, 1, 2, 1))
f <- function(x, y) sqrt(4 - x^2 - y^2)
x <- seq(0, 1.25, length = 25)
y <- seq(0, 1.25, length = 25)
persp(x, y, outer(x, y, f), main = "Vraie fonction",
      zlim = c(0, 2), theta = 120, phi = 30, col = "lightblue",
      zlab = "z", ticktype = "detailed")

## Pour faire les trois autres graphiques, nous allons d'abord
## créer des graphiques en trois dimensions vides, puis y
## ajouter des points avec la fonction 'points'. La fonction
## 'trans3d' sert à convertir les coordonnées des points dans
## la projection utilisée par 'persp'.
u <- runif(1e2, 0, 1.25)
v <- runif(1e2, 0, 1.25)
res <- persp(x, y, matrix(NA, length(x), length(y)),
             main = "n = 100",
             zlim = c(0, 2), theta = 120, phi = 30,
             zlab = "z", ticktype = "detailed")
points(trans3d(u, v, f(u, v), pm = res),
       pch = 21, bg = "lightblue")

u <- runif(1e3, 0, 1.25)
v <- runif(1e3, 0, 1.25)
res <- persp(x, y, matrix(NA, length(x), length(y)),
             main = "n = 1000",
             zlim = c(0, 2), theta = 120, phi = 30,
             zlab = "z", ticktype = "detailed")
points(trans3d(u, v, f(u, v), pm = res),
```

```
       pch = 21, bg = "lightblue")

u <- runif(1e4, 0, 1.25)
v <- runif(1e4, 0, 1.25)
res <- persp(x, y, matrix(NA, length(x), length(y)),
             main = "n = 10 000",
             zlim = c(0, 2), theta = 120, phi = 30,
             zlab = "z", ticktype = "detailed")
points(trans3d(u, v, f(u, v), pm = res),
       pch = 21, bg = "lightblue")

par(op)                    # revenir aux paramètres par défaut
```

9.4 Exercices

9.1 Évaluer l'intégrale

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

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

9.2 Évaluer l'intégrale

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

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

9.3 Soit l'intégrale

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

a) Évaluer cette intégrale par Monte Carlo en effectuant un changement de variable.

b) Évaluer cette intégrale par Monte Carlo par échantillonnage direct d'une loi de probabilité appropriée.

9.4 Soit $X_1, \ldots, X_{25}$ un échantillon aléatoire de taille 25 d'une distribution $N(0,1)$ et soit $X_{(1)} \leq \ldots \leq X_{(25)}$ 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

9.1 Valeur exacte : $1{,}845969$

9.3 Valeur exacte : $-0{,}055292$

A Planification d'une simulation en 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 $X_1, \ldots, X_n$ un échantillon aléatoire tiré d'une population distribuée selon une loi uniforme sur l'intervalle $(\theta - \frac{1}{2}, \theta + \frac{1}{2})$. On considère 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 on veut, 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, on doit d'abord simuler un grand nombre N d'échantillons aléatoires de taille n d'une distribution $U(\theta - \frac{1}{2}, \theta + \frac{1}{2})$ pour une valeur de θ choisie. Pour chaque échantillon, on calculera ensuite les trois estimateurs ci-dessus, puis la moyenne et la variance, par type d'estimateur, de tous les estimateurs obtenus. Si la moyenne des N estimateurs $\hat{\theta}_i$, $i = 1, 2, 3$ est près de θ, alors on pourra conclure que $\hat{\theta}_i$ est sans biais. De même, on déterminera lequel des trois estimateurs a la plus faible variance selon le classement des variances empiriques.

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. On a alors :

```
> simul1(10000, 100, 0)
$biais
     Moyenne      Mediane   Mi-etendue
8.996343e-04 1.339788e-03 7.935145e-05

$variances
     Moyenne      Mediane   Mi-etendue
8.331893e-04 2.425861e-03 4.751204e-05
```

A.3 Seconde approche : avec `sapply`

On le sait, les boucles sont inefficaces en R. Il est en général plus efficace de déléguer les boucles aux fonctions `lapply` et `sapply` (section 6.3). On rappelle que la syntaxe de ces fonctions est

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

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

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

## Simulation comme 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
}

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

FIG. 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`

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

Ces 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. On remarquera que l'argument `i` ne joue aucun rôle dans la fonction. Voici un exemple d'utilisation pour un petit nombre de simulations (4) :

```
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`

```
> sapply(1:4, fun1, size = 10, a = -0.5, b = 0.5)
                    [,1]         [,2]        [,3]
Moyenne    -0.007968176  0.092944440 -0.05931761
Mediane     0.047313279  0.112486287 -0.07179958
Mi-etendue -0.013956836 -0.005761647 -0.10104435
                   [,4]
Moyenne    -0.09876820
Mediane    -0.08262927
Mi-etendue -0.07542318
```

On remarque donc que les résultats de chaque simulation se trouvent dans les colonnes de la matrice obtenue avec `sapply`.

Pour compléter l'analyse, on englobe le tout dans une fonction `simul2`, dont le code se trouve à la figure A.3 :

```
> simul2(10000, 100, 0)
$biais
      Moyenne       Mediane    Mi-etendue
 3.518943e-04  8.999696e-04 -8.920205e-06

$variances
     Moyenne      Mediane   Mi-etendue
8.243666e-04 2.404536e-03 4.807234e-05
```

Il est généralement plus facile de déboguer le code avec cette approche puisque l'on peut 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` (section 6.5) 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`. On a alors

```
> simul3(10000, 100, 0)
$biais
      Moyenne       Mediane    Mi-etendue
 1.734737e-04  8.891869e-05 -5.583918e-05

$variances
     Moyenne      Mediane   Mi-etendue
8.353117e-04 2.400002e-03 4.853127e-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.

```
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`

À des fins d'illustration, supposons que l'on utilise 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 l'on crée 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 :

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

Une simple commande

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

exécutera alors une simulation complète.

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 OS X ou Linux). Une fois placé dans le répertoire 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 de l'exécutable à la ligne de commande :

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

Remplacer `R-x.y.z` par le numéro de version courant de 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,

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

et donc

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

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 $X_1, \dots, X_n$, on cherche 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 :

$$
\begin{aligned}
Y &= X^2 \\
Y &= \frac{X - E[X]}{\sqrt{\mathrm{Var}[X]}} \\
Y &= \frac{X_1 + \cdots + X_n}{n} \\
Y &= F_X(X).
\end{aligned}
$$

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

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

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

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

Remarque. Il importe de noter que 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 $X \sim \text{Bêta}(2,2)$. On détermine la densité de $Y = X^3$. On a

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

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. On a

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

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 quand 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. □

Exemple B.3. Soit X une variable aléatoire continue quelconque avec densité $f_X(x)$. On cherche 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

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

□

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 $(0,1)$. On a donc

$$f_{X_1X_2}(x_1,x_2) = f_{X_1}(x_1) = f_{X_2}(x_2) = 1, \quad 0 < x_1 < 1, 0 < x_2 < 1.$$

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 $y \leq 0$, on a clairement $F_Y(y) = 0$.
2. Si $0 < y < 1$, on a
$$\begin{aligned} F_Y(y) &= \int_0^y \int_0^{y-x_2} dx_1\, dx_2 \\ &= \frac{1}{2}\, y^2. \end{aligned}$$
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, on a

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

et

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

□

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

On considère 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

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

On cherche la fonction de probabilité de $Y = (X-2)^2$. Les valeurs possibles de Y sont $y = 0, 1$ et 4. On a

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

et donc

$$\begin{aligned}\Pr[Y = 0] &= \Pr[X = 2] = \frac{6}{16} \\ \Pr[Y = 1] &= \Pr[X = 1] + \Pr[X = 3] = \frac{10}{16} \\ \Pr[Y = 4] &= \Pr[X = 0] + \Pr[X = 4] = \frac{2}{16}.\end{aligned}$$

□

B.2.2 Cas continu

Le cas continu est beaucoup plus délicat. On considère 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. On considère 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. On a $u(x) = -2\ln(x)$, d'où

$w(y) = u^{-1}(y) = e^{-y/2}$ et $w'(y) = -\frac{1}{2}e^{-y/2}$. Par le théorème B.1, on obtient

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

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

Exemple B.7. On cherche 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, on sait que

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

Par la suite, on définit $Y = X^2 = |Z|^2 = Z^2$, une transformation bijective de X dont le domaine de définition est $\mathbb{R}^+$. On a 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 l'on part 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. On pose $Y = X_1 + X_2$.

Le calcul de la distribution de Y requiert une variable muette. Définissons $Y_2 = X_2$. On a donc

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

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

On généralise 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 $Y_2 = u_2(X_1, X_2)$. On suppose 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 $f_{X_1X_2}(x_1, x_2)$ 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*

$$f_{Y_1Y_2}(y_1, y_2) = f_{X_1X_2}(w_1(y_1, y_2), w_2(y_1, y_2))\,|J|,$$

où

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

est appelé le Jacobien *de la transformation.*

Exemple B.9. Soit les variables aléatoires stochastiquement indépendantes $X_1 \sim \text{Gamma}(\alpha, 1)$ et $X_2 \sim \text{Gamma}(\eta, 1)$. On va 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, on a

$$f_{X_1X_2}(x_1, x_2) = \frac{1}{\Gamma(\alpha)\Gamma(\eta)} x_1^{\alpha-1} x_2^{\eta-1} e^{-x_1-x_2}, \quad x_1 > 0, x_2 > 0.$$

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,

$$\begin{aligned} f_{Y_1Y_2}(y_1, y_2) &= f_{X_1X_2}(y_1y_2, y_1(1 - y_2)) \, |-y_1| \\ &= \frac{1}{\Gamma(\alpha)\Gamma(\eta)} y_1 (y_1y_2)^{\alpha-1} (y_1(1 - y_2))^{\eta-1} e^{-y_1y_2 - y_1(1-y_2)} \\ &= \frac{1}{\Gamma(\alpha)\Gamma(\eta)} y_1^{\alpha+\eta-1} y_2^{\alpha-1} (1 - y_2)^{\eta-1} e^{-y_1} \\ &= g(y_1)\, h(y_2) \end{aligned}$$

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

$$\begin{aligned} f_{Y_1}(y_1) &= \int_0^1 f_{Y_1Y_2}(y_1, y_2)\, dy_2 \\ &= y_1^{\alpha+\eta-1} e^{-y_1} \int_0^1 \frac{1}{\Gamma(\alpha)\Gamma(\eta)} y_2^{\alpha-1} (1 - y_2)^{\eta-1} dy_2 \\ &= \frac{1}{\Gamma(\alpha + \eta)} y_1^{\alpha+\eta-1} e^{-y_1}, \quad y_1 > 0 \end{aligned}$$

et

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

On a donc $Y_1 = X_1 + X_2 \sim \text{Gamma}(\alpha+\eta, 1)$ et $Y_2 = X_1/(X_1+X_2) \sim$ Bêta(α, η). □

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 $X_1, \ldots, X_n$ des variables aléatoires indépendantes et $Y = X_1 + \cdots + X_n$. 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 7

7.1 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
```

7.2 a) On utilise de nouveau la fonction `rand` de la figure C.1. Le graphique de la figure C.2 a été créé avec les commandes

```
> 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)
```

On compte 17 lignes dans le graphique.

b) Similaire à la partie a), sauf que les nombres sont générés avec

```
> 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).

7.3 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])
```

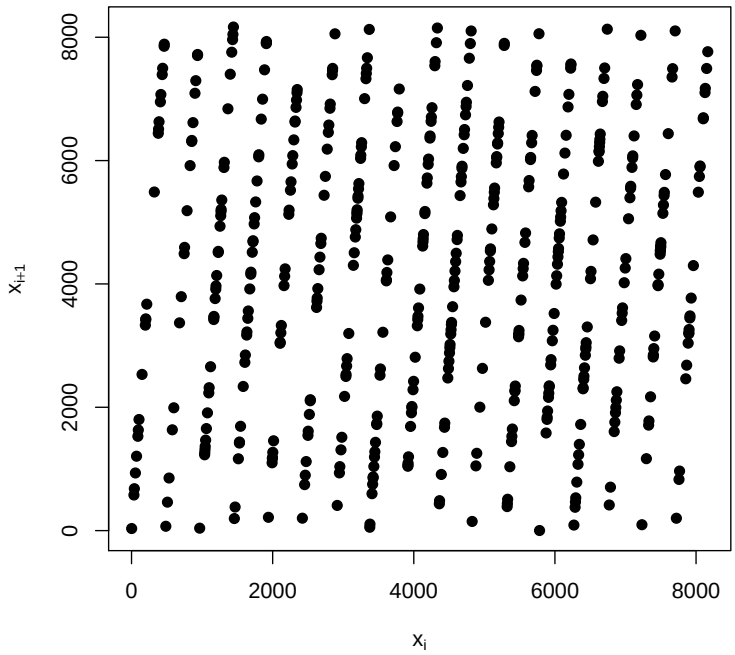


FIG. C.2 – Paires de valeurs du générateur congruentiel multiplicatif avec $m = 2^{31} - 1$ et $a = 17$

Chapitre 8

8.1 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.

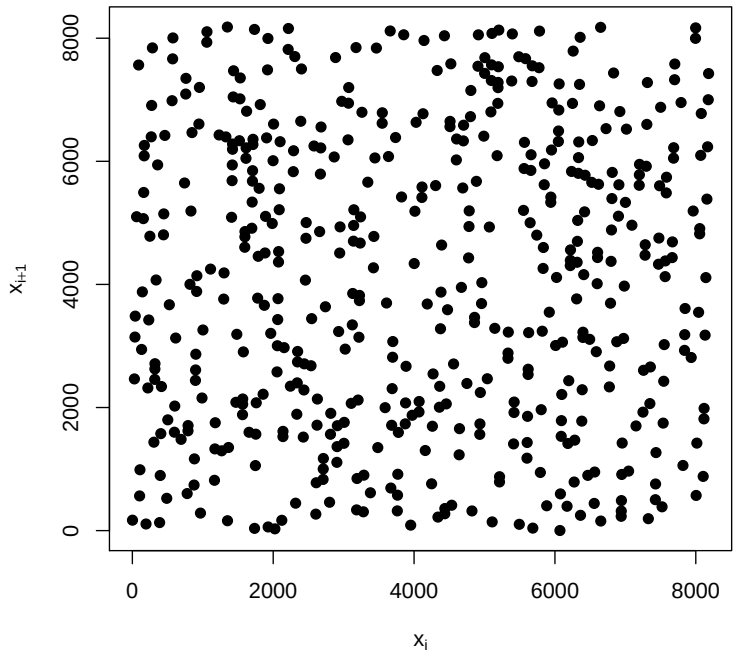


FIG. C.3 – Paires de valeurs du générateur congruentiel multiplicatif avec $m = 2^{31} - 1$ et $a = 85$

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

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

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. Dans le graphique de gauche, on a plusieurs couples de points (u_1,u_2) où chaque composante provient d'une distribution uniforme sur l'intervalle $(0,1)$.

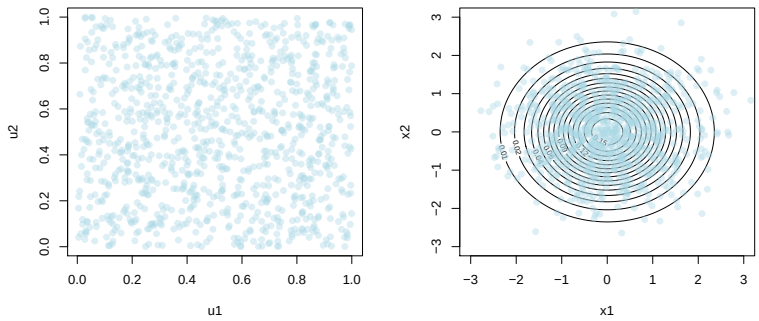


FIG. C.4 – Démonstration graphique du fonctionnement de la transformation de Box–Muller

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. On a superposé le nuage de points ainsi obtenu aux lignes de niveau d'une distribution normale bivariée (avec paramètre $\rho = 0$). On observe que la répartition et la densité du nuage de points correspond effectivement aux lignes de niveau.

8.2 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

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

d'où

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

On peut donc utiliser la méthode inverse.

8.3 Voir la fonction R de la figure C.5. On vérifie graphiquement la validité de l'algorithme :

```
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 Gamma$(\alpha, 1)$, $\alpha > 1$

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

8.4 Deux suggestions.

1. Si $X \sim \text{Gamma}(\alpha, 1)$, alors $Y = X/\lambda \sim \text{Gamma}(\alpha, \lambda)$. On peut donc générer un nombre x d'une loi Gamma$(\alpha, 1)$ avec l'algorithme de l'exercice 8.3, puis poser $y = x/\lambda$.
2. Si α est entier, on peut générer α nombres (indépendants) $x_1, \ldots, x_\alpha$ d'une distribution Exponentielle(λ) et poser $y = \sum_{i=1}^{\alpha} x_i$.

8.5 a) On a $X|\Theta \sim \text{Exponentielle}(\Theta)$ et $\Theta \sim \text{Gamma}(\alpha, \lambda)$. Par la loi des probabilités totales,

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

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

b) Pour générer un nombre d'une distribution de Pareto de paramètres α et λ avec le résultat en a), on génère d'abord un nombre θ d'une distribution gamma de mêmes paramètres, puis on génère un nombre x d'une distribution exponentielle de paramètre θ. En R :

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

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

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

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

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

avec une amorce de 12 sont 781, 1 614 et 463. En divisant ces nombres par 2 048, on obtient 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

```
> 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.

8.7 On a la transformation

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

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 $\{(x_1, x_2) : x_1^2 + x_2^2 < 1 \backslash (0,0)\}$. Cela se vérifie aisément en examinant les limites de l'espace de départ :

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

Les troisième et quatrième inégalités définissent les quadrants I et III, puis II et IV de $\mathbb{R}^2$, respectivement. On remarque également 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

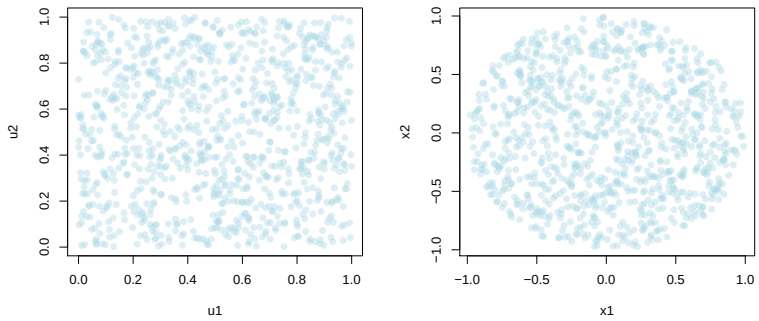


FIG. C.6 – Démonstration graphique du fonctionnement de la transformation de l'exercice 8.7

gauche sont tirés aléatoirement sur le carré $(0,1)\times(0,1)$. Le graphique de droite montre que suite à la tranformation ci-dessus, on obtient des points (x_1,x_2) distribués uniformément sur un disque de rayon 1 centré en $(0,0)$.

8.8 a) On a

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

et

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

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). On cherche la distribution conjointe de X_1 et X_2, $f_{X_1,X_2}(x_1,x_2)$. On a la transformation

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

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, $X_1 \sim$ Bêta(α,β) et $X_2 \sim$ Gamma$(\alpha+\beta)$ (un résultat connu).

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

1. Générer y_1 d'une distribution Gamma$(\alpha,1)$.
2. Générer y_2 d'une distribution 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, on a un nuage de points (y_1, y_2) tirés indépendemment de deux distributions gamma de paramètre d'échelle égal à 1. On a superposé ce nuage de points aux courbes de niveaux de la distribution conjointe des deux lois gamma.

Dans le graphique de droite, on a placé en abscisse les points $x = y_1/(y_1+y_2)$ résultant de la transformation. On voit 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 :

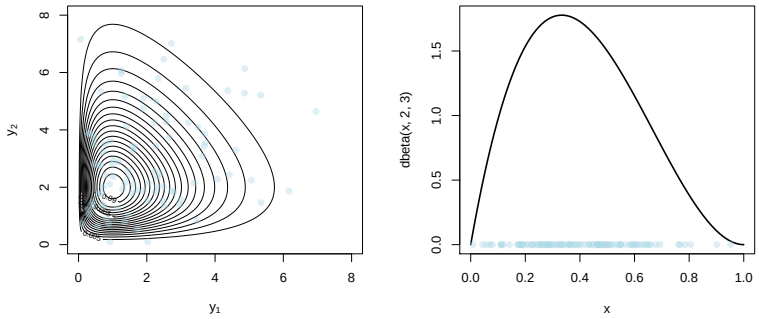


FIG. C.7 – Démonstration graphique du fonctionnement de la transformation de l'exercice 8.8

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

8.9 a) On a

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

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

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

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

c) On a $E[Z] = 1/(1/c) = c$.

8.10 a) On pose

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

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

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

8.11 a) On démontre facilement que le mode M d'une distribution bêta de paramètres α et β se trouve en

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

Par conséquent, l'efficacité de l'algorithme d'acceptation-rejet décrit à l'exercice 8.10 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 $\beta = 2$, on obtient une efficacité de $9/16$.

b) On a trouvé $c = 1{,}2$ dans l'exemple 8.5, d'où 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) On utilise l'algorithme développé à l'exemple 8.5. Une première mise en œuvre de l'algorithme en R est fournie dans le code informatique de la section 8.5. La figure C.8 en propose une autre. On propose aussi une mise en œuvre VBA à la figure C.9. On peut vérifier l'exactitude la fonction `rbeta.ar2` avec

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

8.12 a) On reconnaît l'algorithme d'acceptation-rejet de l'exercice 8.10.

b) On doit simuler deux observations d'une loi Bêta$(2,3)$ dont la fonction de densité de probabilité est

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

```
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`

Le mode de cette densité se trouve en $x = 1/3$ (voir la solution de l'exercice 8.11) 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 résultas.

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 $\mathbf{x} = (0{,}46, 0{,}33)$.

8.13 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 $x > 1$.

```
Private Function sqrt(x)
    sqrt = x ^ 0.5
End Function

Private Function g(x As Double)
    DensiteTriangle = IIf(x < 0.8, 2.5 * x,
                          10 - 10 * x)
End Function

Private Function Ginv(u As Double)
    InverseTriangle = IIf(u < 0.8, sqrt(0.8 * u),
                          1 - sqrt(0.2 - 0.2 * u))
End Function

Private Function dbeta(x As Double, shape1 As Double,
                       shape2 As Double)
    Dim cte As Double
    With WorksheetFunction
        cte = Exp(.GammaLn(shape1 + shape2) -
                  .GammaLn(shape1) -
                  .GammaLn(shape2))
        dbeta = cte * x ^ (shape1 - 1) *
                (1 - x) ^ (shape2 - 1)
    End With
End Function

Function betasim()
    Dim u1 As Double, u2 As Double, y As Double

    Do
        u1 = Rnd
        u2 = Rnd
        y = Ginv(u1)
    Loop Until u2 <= dbeta(y, 3, 2) / (1.2 * g(y))

    SimuleBeta = y
End Function
```

FIG. C.9 – Code VBA de la fonction `betasim`

b) On veut borner la densité $f_X(x) = x^{\alpha-1}e^{-x}/\Gamma(\alpha)$, $x > 0$ et $0 < \alpha < 1$. Du résultat en a), on 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

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

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

et

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

On remarque que

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

On a donc 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.

8.14 On veut 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, on nous donne

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

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, $Y \sim$ 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 Gamma$(\alpha, 1)$ avec $\alpha \geq 1$ si

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

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.

Chapitre 9

9.1 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 $x_1, \ldots, x_n$ est un échantillon aléatoire d'une distribution $U(4,9)$. Une évaluation avec R donne

```
> mean(log(runif(1e6, 4, 9)))
[1] 1.846422
```

9.2 On pose

$$\begin{aligned}\theta &= \int_0^1 \int_0^1 e^{2xy} \ln(3x + y^2)\, dx dy \\ &= E[e^{2XY} \ln(3X + Y^2)],\end{aligned}$$

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 $x_1, \ldots, x_n$ et $y_1, \ldots, y_n$ sont deux échantillons aléatoires indépendants d'une distribution $U(0,1)$. Une évaluation avec R donne

```
> x <- runif(1e6)
> y <- runif(1e6)
> mean(exp(2 * x * y) * log(3 * x + y^2))
[1] 1.2032
```

9.3 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 $u_1, \ldots, u_n$ est un échantillon aléatoire d'une distribution $U(0,1)$. Une évaluation avec R donne

```
> u <- runif(1e6)
> 2 * mean((-log(u^2))^2 * sin(pi * (-log(u^2))))
[1] -0.07235232
```

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 $X \sim$ Gamma$(3, 1/2)$. Une estimation de θ est donc

$$\hat{\theta} = \frac{16}{n} \sum_{i=1}^{n} \sin(\pi x_i)$$

où $x_1, \ldots, x_n$ est un échantillon aléatoire d'une Gamma$(3, 1/2)$. Une évaluation avec R donne

```
> 16 * mean(sin(pi * rgamma(1e6, 3, 0.5)))
[1] -0.04599569
```

9.4 On peut faire l'estimation en R simplement avec les expressions suivantes :

```
> x <- replicate(10000, rnorm(25))
> mean(apply(x, 2, function(x) sort(x)[5]))
[1] -0.9056586
```

Bibliographie

Devroye, L. 1986, *Non-Uniform Random Variate Generation*, Springer-Verlag, ISBN 0-38796305-7. URL http://luc.devroye.org/rnbookindex.html.

Dutang, C., V. Goulet et M. Pigeon. 2008, «**actuar**: An R package for actuarial science», *Journal of Statistical Software*, vol. 25, nº 7. URL http://www.jstatsoft.org/v25/i07.

Knuth, D. E. 1997, *The Art of Computer Programming*, vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, MA.

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, nº 10, p. 4587–4593.

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, nº 10, p. 4570–4578.

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.

Sylvester, J. J. 1865, «On a special class of questions on the theory of probabilities», *Birmingham British Assoc. Rept.*, p. 8–9.

Weisstein, E. W. «Sylvester's four-point problem», From MathWorld – A Wolfram Web Resource. URL http://mathworld.wolfram.com/SylvestersFour-PointProblem.html.

Wheeler, B. 2013, ***SuppDists**: Supplementary Distributions*. URL http://cran.r-project.org/package=SuppDists, R package version 1.1-9.

Ce document a été produit avec le système de mise en page XeLaTeX. Le texte principal est 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 Adobe Myriad Pro. Des icônes proviennent de la police Font Awesome. Les graphiques ont été réalisés avec R.