Méthode de Monte-Carlo & Application aux Processus Aléatoires

Rémi Peyre

printemps 2015

Chapitre 1

Quelques outils

1.1 Notions de convergence en théorie des probabilités

1.1.1 Contexte probabiliste

Cette sous-section se penche sur quelques considérations hautement techniques, que vous pouvez zapper sans dommage, mais qu'il est impératif de mentionner pour que les théorèmes énoncés dans ce cours soient formellement corrects :

Définition 1.1 (Espace polonais)**.** Un espace topologique[*] $\mathcal{X}$ est dit *séparable*[†] s'il en existe un sous-ensemble dénombrable $\mathcal{X}'$ qui y soit dense.

Un espace métrique $(\mathcal{X}, dist)$ est dit *complet* s'il ne peut pas être considéré comme une partie dense d'un espace strictement plus grand.

Un espace topologique $\mathcal{X}$ est dit *polonais* s'il est séparable et qu'il peut être muni d'une métrique *dist* (compatible avec sa topologie) qui le rende complet.

☛ *Pour que la théorie des probabilités donne des résultats intéressants sur les variables aléatoires et leurs lois, il faut supposer que les v.a. qu'on considère sont à valeurs dans des espaces polonais. Dans tout ce cours, cette hypothèse sera faite de façon implicite. Ces espaces polonais seront en outre implicitement munis de leurs tribus boréliennes.*

Remarque 1.2. En pratique, la condition ci-dessus est "transparente", car presque tous les espaces qu'on rencontre "dans la vraie vie" sont polonais. Notamment :

Proposition 1.3.

— *Les ensembles dénombrables (munis de la topologie de la convergence simple) sont polonais.*

[*]. Un *espace topologique* est un ensemble de points sur lequel on a défini une notion de partie ouverte qui vérifie les propriétés élémentaires des ouverts et des fermés des espaces métriques ; dès lors, on peut définir les notions d'ensembles fermés, de compacts, de convergence, de continuité, etc., afférentes.

[†]. Ne pas confondre avec la notion d'espace séparable avec celle d'espace *séparé*, qui n'a rien à voir...

— *Les espaces $\mathbf{R}^d$ (munis de leur topologie naturelle) sont polonais.*
— *Pour F un espace de Banach séparable, l'espace $\mathcal{C}^k(\mathbf{R}^d, F)$ des fonctions k fois continument dérivables de $\mathbf{R}^d$ dans F, muni de la topologie de la convergence uniforme sur les compacts pour les dérivées d'ordre 0 à k, est polonais.*
— *Pour F un espace de Banach séparable, pour $p \in [1, \infty)$, l'espace de Sobolev $W^{m,p}(\mathbf{R}^d, F)$* [‡] *est polonais.*
— *Un produit d'un nombre dénombrable d'espaces polonais (muni de la topologie de la convergence simple pour chaque coordonnées) est polonais.*
— *Un sous-espace fermé d'un espace polonais est polonais.*
— *Une partie ouverte d'un espace polonais est polonaise.*

Définition 1.4. Un espace mesurable $(\Omega, \mathcal{B})$ est dit *standard* s'il est isomorphe (en tant qu'espace mesurable) à un espace polonais muni de sa tribu borélienne.

Lemme 1.5. *Tout espace standard est isomorphe à l'espace $[0, 1]$ muni d'une sous-tribu de sa tribu borélienne.*

☛ *Lorsqu'on ne considère que des variables aléatoires à valeurs dans des espaces polonais, munis de leurs tribus boréliennes, on peut toujours supposer que l'espace probabilisé sur lequel on travaille est standard ; nous ferons donc toujours implicitement cette hypothèse dans la suite.*

1.1.2 Convergence de variables aléatoires

☛ *Dans toute cette section, les variables aléatoires sont considérées à équivalence presque-sure près.*

Définition 1.6 (Convergence en probabilité)**.** Pour $(\Omega, \mathbb{P})$ un espace probabilisé, soient $(X_n)_{n\in\mathbf{N}}$ et X_∞ des variables aléatoires sur Ω à valeurs dans un espace $\mathcal{X}$. On dit que X_n *converge en probabilité* vers X_∞ quand

$$\forall U \subset \mathcal{X} \times \mathcal{X} \text{ ouvert} \qquad \mathbb{P}\big(\{\omega \mid (X_n(\omega), X_\infty(\omega)) \in U\}\big) \overset{n\to\infty}{\to} 1.$$

Remarque 1.7. Dans le cas où $\mathcal{X}$ est muni d'une métrique *dist*, cette condition s'exprime plus simplement par :

$$\forall \varepsilon > 0 \qquad \mathbb{P}\big(\{\omega \mid dist(X_n(\omega), X_\infty(\omega)) \leqslant \varepsilon\}\big) \overset{n\to\infty}{\to} 1.$$

Définition 1.8 (Convergence presque-sure)**.** Pour $(\Omega, \mathbb{P})$ un espace probabilisé, soient $(X_n)_{n\in\mathbf{N}}$ et X_∞ des variables aléatoires sur Ω à valeurs dans un espace $\mathcal{X}$. On dit que X_n *converge presque-surement* vers X_∞ quand

$$\mathbb{P}(\{\omega \mid X_n(\omega) \overset{n\to\infty}{\to} X_\infty(\omega)\}) = 1.$$

Lemme 1.9 (Convergence presque-sure et convergence en probabilité)**.** *Si X_n converge presque-surement vers X_∞, alors la convergence a aussi lieu en probabilité.*

[‡]. Défini formellement, pour $m \in \mathbf{N}$, par $\|f\|^p_{W^{m,p}(\mathbf{R}^d,F)} := \sum_{i=0}^m \|D^i f\|^p_{L^p(\mathbf{R}^d, F^{d^i})}$.

Lemme 1.10 (Convergence en probabilité et mesure-image)**.** *Soit $\Phi_n \colon \mathcal{X} \mapsto \mathcal{Y}$ des fonction continues telles que Φ_n converge vers Φ_∞ en vérifiant la condition de convergence localement uniforme suivante :*

$$\forall x_0 \in \mathcal{X} \quad \forall V \ni \Phi_\infty(x_0) \text{ ouvert} \quad \exists U \ni x_0 \text{ ouvert} \quad \exists n_0 \in \mathbf{N} \quad \forall n \geqslant n_0 \quad x \in U \Rightarrow \Phi_n(x) \in V \text{ [§]}.$$

Alors, si $(X_n)_n$ sont des variables aléatoires sur $\mathcal{X}$ convergeant en probabilité vers X_∞, $\Phi(X_n)$ converge en probabilité vers $\Phi(X_\infty)$.

Lemme 1.11. *Si $(X_n^{(0)})_{n\in\mathbf{N}}, (X_n^{(1)})_{n\in\mathbf{N}}, \ldots$ sont des suites de variables aléatoires resp. à valeurs dans $\mathcal{X}^{(0)}, \mathcal{X}^{(1)}, \ldots$, convergeant en probabilité respectivement vers $X_\infty^{(0)}, X_\infty^{(1)}, \ldots$, alors, pour $k \in \bar{\mathbf{N}}$, le k-uplet $(X_n^{(i)})_{i<k}$, qui est une variable aléatoire à valeurs dans l'espace-produit $\prod_{i<k} \mathcal{X}^{(i)}$, converge en probabilité vers $(X_\infty^{(i)})_{i<k}$ lorsque $n \to \infty$.*

Remarque 1.12. Noter que ce théorème n'impose aucune condition d'indépendance entre les $X_n^{(i)}$.

Corolaire 1.13. *Sous les hypothèses du théorème précédent, si $\Phi \colon \prod_{i<k} \mathcal{X}^{(i)} \to \mathcal{Y}$ est une fonction continue, alors $\Phi\big((X_n^{(i)})_{i<k}\big)$ converge en probabilité vers $\Phi\big((X_\infty^{(i)})_{i<k}\big)$ lorsque $n \to \infty$.*

1.1.3 Convergence de lois

Définition 1.14 (Convergence de lois)**.** Soient $(\mu_n)_{n\in\mathbf{N}}$ et μ_∞ des lois (càd. des mesures de probabilité) sur un espace $\mathcal{X}$. On dit que la lois μ_n converge (on précise parfois « converge en loi », ou « converge faiblement ») vers μ_∞ quand, pour toute fonction f continue et bornée sur $\mathcal{X}$, $\int_\mathcal{X} f(x)\, d\mu_n(x) \to \int_\mathcal{X} f(x)\, d\mu_\infty(x)$ quand $n \to \infty$.

Remarque 1.15. En pratique, on dit souvent « la variable aléatoire X_n converge vers X_∞ » pour signifier « la loi de X_n converge (faiblement) vers la loi de X_∞ », ce qui peut donner l'impression que la « convergence en loi » est un mode de convergence des *variables aléatoires*. Mais cette impression est fallacieuse : en effet, deux variables aléatoires peuvent être totalement différentes tout en ayant la même loi, comme le sont p.ex. une v.a. X suivant la loi de Rademacher d'une part, et la v.a. $(-X)$ d'autre part.

Lemme 1.16 (Convergence en loi et convergence des évènements)**.** *Soient $(\mu_n)_{n\in\bar{\mathbf{N}}}$ des mesures de probabilité sur un même espace $\mathcal{X}$, telles que μ_n converge vers μ_∞. Alors, si B désigne un borélien de $\mathcal{X}$ tel que $\mu_\infty(\partial X) = 0$,*

$$\mu_n(B) \to \mu_\infty(B) \quad \text{quand } n \to \infty.$$

Corolaire 1.17. *En particulier, si $(\mu_n)_{n\in\bar{\mathbf{N}}}$ sont des mesures de probabilité sur $\mathbf{R}$ telles que $\mu_n \overset{n\to\infty}{\to} \mu_\infty$ et que μ_∞ soit sans atome, pour tout intervalle $I \subset \mathbf{R}$, on a $\mu_n(I) \to \mu_\infty(I)$.*

[§]. Cette condition est en particulier vérifiée si la suite $(\Phi_n)_n$ est constante égale à Φ_∞.

Lemme 1.18 (Fonction de répartition et convergence en loi)**.** *Soient $(\mu_n)_{n\in\bar{\mathbf{N}}}$ des mesures de probabilité sur $\mathbf{R}$, μ_∞ étant supposée sans atome ; notons $(F_n)_{n\in\bar{\mathbf{N}}}$ les fonctions de répartitions respectives des μ_n. Alors μ_n converge vers μ_∞ si et seulement si la fonction F_n converge simplement vers F sur $\mathbf{R}$.*

Lemme 1.19 (Convergence en probabilité et convergence en loi)**.** *Soient $(X_n)_{n\in\mathbf{N}}$ et X_∞ des variables aléatoires définies sur un même espace probabilisé $(\Omega, \mathbb{P})$, à valeurs dans un espace $\mathcal{X}$; notons $(\mu_n)_n$ et μ_∞ les lois respectives de ces variables aléatoires. Si la variable aléatoire X_n converge en probabilité vers X_∞ quand $n \to \infty$, alors la loi μ_n converge vers μ_∞.*

Lemme 1.20 (Convergence en loi et convergence en probabilité)**.** *Soient $(\mu_n)_{n\in\mathbf{N}}$ et μ_∞ des lois sur un même espace $\mathcal{X}$, telles que μ_n converge vers μ_∞ quand $n \to \infty$. Alors il existe un espace probabilisé $(\Omega, \mathbb{P})$ et des variables aléatoires $(X_n)_{n\in\mathbf{N}\cup\{\infty\}}$ sur cet espace, tels que $X_n \overset{pba}{\to} X_\infty$ quand $n \to \infty$.*

Lemme 1.21 (Convergence vers une constante)**.** *Soient $(X_n)_{n\in\mathbf{N}}$ des variables aléatoires définies sur un même espace probabilisé $(\Omega, \mathbb{P})$, à valeurs dans un même espace X ; notons $(\mu_n)_n$ les lois respectives de ces variables aléatoires ; et soit $x_\infty \in \mathcal{X}$. Alors les affirmations suivantes sont équivalentes :*

1. *X_n converge en probabilité vers la v.a. constante x_∞ ;*
2. *μ_n converge vers $\boldsymbol{\delta}_{x_\infty}$.*

Lemme 1.22 (Convergence en loi et mesure-image)**.** *Soient $(\Phi_n)_{n\in\mathbf{N}\cup\{\infty\}}$ des fonctions continues de $\mathcal{X}$ dans $\mathcal{Y}$ vérifiant l'hypothèse du lemme 1.10.*

Alors, si $(\mu_n)_{n\in\bar{\mathbf{N}}}$ sont des mesures sur $\mathcal{X}$ telles que $\mu_n \overset{pba}{\to} \mu_\infty$, on a convergence pour les mesures-images : $\Phi_n * \mu_n \overset{pba}{\to} \Phi_\infty * \mu_\infty$.

Lemme 1.23. *Si $(X_n)_{n\in\mathbf{N}}$ est une suite de variables aléatoires sur un même espace $\mathcal{X}$ convergeant en loi vers X_∞, que $(\Lambda_n)_{n\in\mathbf{N}}$ est une suite de variables aléatoires sur $\mathcal{L}$ convergeant en probabilité vers la constante λ_∞, et que $\Phi\colon \mathcal{L} \times \mathcal{X} \to \mathcal{Y}$ est une application continue sur un voisinage de $\{\lambda_\infty\} \times \mathcal{X}$, alors $\Phi(X_n, \Lambda_n)$ converge en loi vers $\Phi(X_\infty, \lambda_\infty)$.*

Remarque 1.24. Notez qu'aucune hypothèse n'est faite sur la loi jointe de Λ_n et x_n.

Remarque 1.25. Attention ; il est indispensable dans l'énoncé de ce lemme que λ_∞ soit une *constante* !

1.2 Théorèmes-limites

1.2.1 Loi des grands nombres

Lemme 1.26 (Loi des grands nombres, version fréquentielle)**.** *Soient, sur un espace probabilisé $(\Omega, \mathbb{P})$, des variables aléaoires de Bernoulli $(X_i)_{i\in\mathbf{N}}$ indépendantes et de même paramètre p. Alors la variable aléatoire $\bar{X}_n := n^{-1} \sum_{i=0}^{n-1} X_i$ converge presque-surement vers la constante p.*

Lemme 1.27 (Loi des grands nombres)**.** *Soient, sur un espace probabilisé* $(\Omega, \mathbb{P})$*, des variables aléaoires réelles* $(X_n)_{n\in\mathbf{N}^*}$ *indépendantes et de même loi. Supposons que cette loi soit telle que* X_n *soit intégrable*[¶]*, et notons* $\mathbb{E}(X_n) =: m$[|||]*. Alors la variable aléatoire* $\bar{X}_N := N^{-1}\sum_{n=1}^N X_N$ *converge presque-surement, lorsque* N *tend vers l'infini, vers la constante* m*.*

Remarque 1.28. Ce théorème est encore valable si les variables X_n sont à valeurs dans un espace $\mathbf{R}^d$, ou plus généralement dans un espace de Banach.

1.2.2 Théorème-limite central

Lemme 1.29 (Théorème-limite central)**.** *Soient, sur un espace probabilisé* $(\Omega, \mathbb{P})$*, des variables aléatoires réelles* $(X_n)_{n\in\mathbf{N}}$ *indépendantes et de même loi; et supposons que cette loi soit telle que* X_n *soit* L^2*, centrée, non constante. Alors, notant* $\mathbb{E}(X_n) =: m$*,*

$$\frac{\frac{1}{N}\big(\sum_{n=1}^N X_n\big) - m}{\mathrm{Var}(X_n)^{1/2}N^{-1/2}} \xrightarrow{loi} \mathcal{N}(0,1) \quad \textit{quand } N \to \infty.$$

Remarque 1.30. La théorème-limite central est souvent appelé en français « théorème central-limite ». Il s'agit là d'une traduction impropre de l'angais « central limit theorem » (lui-même traduction de l'allemand « zentraler Grenzwertsatz ») : en fait, « théorème-limite central » signifie « théorème à propos d'une limite, dont l'importance est centrale en théorie des probabilités ». La formulation « théorème de la limite centrale » qu'on trouve aussi régulièrement, dans le sens de « théorème à propos de la limite de ce qui se passe au centre de la distribution de probabilités », est ingénieuse mais relève d'une étymologie erronée.

Lemme 1.31 (Inégalité de Berry-Esseen)**.** *Reprenons les hypothèses du théorème 1.29, et supposons en outre que* X *ait un moment d'ordre* 3 *tel que :*

$$\mathbb{E}(|X-m|^3) \leqslant M_3\,\mathrm{Var}(X)^{3/2}.$$

Alors pour tout $N \in \mathbf{N}^*$*, pour tout intervalle* $I \subset \mathbf{R}$ *:*

$$\left|\mathbb{P}\left(\frac{\frac{1}{N}\big(\sum_{n=1}^N X_n\big) - m}{\mathrm{Var}(X_n)^{1/2}N^{-1/2}} \in I\right) - \mathbb{P}\big(\mathcal{N}(0,1)\in I\big)\right| \leqslant M_3 N^{1/2}.$$

Remarque 1.32. L'inégalité de Berry-Esseen n'est en général pas présentée exactement sous cette forme ; mais le lemme ci-dessus en est un corollaire immédiat (pour une certaine version explicite de l'inégalité).

Lemme 1.33 (Théorème-limite central multidimensionnel)**.** *Soient, sur un espace probabilisé* $(\Omega, \mathbb{P})$*, des variables aléatoires* $(\vec{X}_n)_{n\in\mathbf{N}}$ *à valeurs dans* $\mathbf{R}^d$*, indépendantes et de même loi; et supposons que cette loi soit telle que* X_n *soit* L^2*. Alors, notant* $\mathbb{E}(\vec{X}_n) =: \vec{m}$*,*

$$\frac{\frac{1}{N}\big(\sum_{n=1}^N X_n\big) - m}{N^{-1/2}} \xrightarrow{loi} \mathcal{N}(0, \mathrm{Var}(X_n)) \quad \textit{quand } N \to \infty,$$

où $\mathrm{Var}(X_n)$ *désigne la matrice de covariance de* X_n*.*

[¶]. Comme tous les X_n ont la même loi, cette propriété ne dépend pas de la valeur de n

[|||]. À nouveau, cette valeur ne dépend pas de n.

Remarque 1.34. Ce résultat est encore vrai en supposant que les $\vec{X}_n$ prennent leurs valeurs dans un espace de Hilbert (en remplaçant la notion de matrice de covariance par celle de forme quadratique continue, et en définissant une notion convenable de loi gaussienne sur un espace de Hilbert).

1.3 Concepts statistiques

Définition 1.35 (Modèle statistique)**.** Un modèle statistique est la donnée d'un espace probabilisable Ω et d'une famille de lois $(\mathbb{P}_\theta)_{\theta\in\Theta}$ sur Ω. On appelle θ « le » *paramètre* ; et Θ l'*espace des paramètres*. Une *expérience statistique* est le tirage au sort d'une variable aléatoire X selon une loi $\mathbb{P}_\theta$ où le paramètre θ, quoique fixé, est *inconnu*. La valeur tirée de X, qu'on pourra noter $X(\omega)$ (où $\omega \in \Omega$ désigne l'éventualité effectivement réalisée au sein de l'espace des probabilités) est appelée *observation*.

Définition 1.36 (Paramètre)**.** « Un » *paramètre du modèle* est, en toute généralité, une fonction *déterministe* de θ.

Définition 1.37 (Estimateur)**.** Un *estimateur* est, en toute généralité, une fonction *déterministe* de l'observation $X(\omega)$.

Définition 1.38 (Estimateur convergent)**.** Une famille d'estimateurs $(T_n)_{n\in\mathbf{N}}$ étant donnée[**], on dit que $T_n(X)$ est une estimateur *convergent* d'un paramètre du modèle $\lambda(\theta)$ quand, *quelle que soit la valeur du paramètre* $\theta \in \Theta$, sous la loi $\mathbb{P}_\theta$, la variable aléatoire $T_n(X)$ converge en probabilité vers $\lambda(\theta)$ quand $n \to \infty$.

Définition 1.39 (Estimateur fortement convergent)**.** Dans la définition ci-dessus, quand on peut remplacer « en probabilité » par « presque-surement », on dit T_n est un estimateur *fortement convergent* de $\lambda(\theta)$.

Définition 1.40 (Intervalle de confiance)**.** Un intervalle $I(X)$ (c'est-à-dire une statistique qui a la forme d'un intervalle) est appelée *intervalle de confiance au niveau* $(1-\alpha)$ (ou *au risque* α) pour le paramètre $\lambda(\theta)$ si, pour tout θ,

$$\mathbb{P}_\theta\big(I(X) \not\ni \lambda(\theta)\big) \leqslant \alpha.$$

Définition 1.41 (Intervalle de confiance asymptotique)**.** Un intervalle $I(X)$ dépendant d'un entier n (c'est-à-dire une suite d'intervalles $I_n(X)$, mais souvent le paramètre n est sous-entendu) est appelé *intervalle de confiance asymptotique au niveau* $(1-\alpha)$ pour le paramètre $\lambda(\theta)$, si, pour tout θ,

$$\overline{\lim_{n\to\infty}}\ \mathbb{P}_\theta\big(I_n(X) \not\ni \lambda(\theta)\big) \leqslant \alpha.$$

[**]. En pratique, on parlera souvent de « l'estimateur T_n », la dépendance en n étant implicite.

1.4 Simulation des variables aléatoires

La méthode de Monte-Carlo repose de manière essentielle sur la capacité à générer des réalisations (indépendantes) d'une variable aléatoire de loi donnée. Quelques rappels méritent donc d'être faits sur la génération des variables aléatoires. Dans toute cette section, nous supposerons que nous disposons d'un outil à même de générer des (réalisations de) variables aléatoires (indépendantes) uniformément distribuées sur $[0, 1]$.

De manière générale, lorsqu'on a affaire à une variables aléatoire "compliquées" à simuler, celle-ci peut souvent s'écrire comme une fonction de plusieurs variables aléatoires "plus simples" indépendantes. Nous ne nous intéresserons donc ici qu'aux méthodes pour simuler ces variables aléatoires plus simples.

1.4.1 Génération de la loi normale à partir de la loi uniforme

Dans le cadre de ce cours, nous supposerons que, outre notre générateur de variables aléatoires de loi *Uniforme*$(0, 1)$, nous disposons aussi d'un générateur de variables aléatoires de loi normale standard $\mathcal{N}(0, 1)$. Si ce n'était pas le cas, il est bon de rappeler comment on peut procéder :

Lemme 1.42 (Méthode de Box-Muller)**.** *Si U_0 et U_1 sont des variables aléatoires indépendantes de loi Uniforme$(0, 1)$, alors le couple de variables aléatoires (G_0, G_1) défini par :*

$$(G_0, G_1) := 2|\log U_0|\big(\cos(2\pi U_1), \sin(2\pi U_1)\big)$$

est tel que G_0 et G_1 sont indépendantes et suivent chacune la loi $\mathcal{N}(0, 1)$. En particulier, G_0 suit la loi normale standard.

Remarque 1.43. On peut aussi générer la loi normale à partir de la loi uniforme en utilisant la méthode de la fonction de répartition (en particulier lorsque la fonction réciproque erf^{-1} de la fonction d'erreur est déjà implémentée dans le logiciel), ou la méthode de rejet [††].

1.4.2 Méthode de Cholesky

La méthode de Cholesky permet de simuler une variable aléatoire gausienne centrée vectorielle à partir de la donnée de sa matrice de covariance.

Lemme 1.44. *Soit $\boldsymbol{C}$ une matrice (réelle) symétrique positive $n \times n$. Alors, si $\boldsymbol{\Sigma} \in \mathbf{R}^{n \times p}$ est une matrice telle que $\boldsymbol{\Sigma}\boldsymbol{\Sigma}^{\mathsf{T}} = C$; si $\vec{X} \in \mathbf{R}^p$ est un vecteur gaussien standard (càd. que $X_1, X_2, \ldots, X_n$ sont des v.a. normales standard i.i.d.), le vecteur $\boldsymbol{\Sigma}\vec{X}$ soit la loi $\mathcal{N}(\vec{0}, \boldsymbol{C})$.*

Si ce lemme est utile en pratique, c'est parce, lorsque $\boldsymbol{C}$ est définie positive, il est facile de trouver $\boldsymbol{\Sigma} \in \mathbf{R}^{n \times n}$ vérifiant la condition du lemme : c'est ce qu'on appelle l'*algorithme de Cholesky* (qui retourne, en outre, une matrice triangulaire inférieure,

[††]. Lorsqu'on programme en bas niveau, ce sont ainsi des variantes de la méthode de rejet qui sont le plus souvent utilisées pour simuler la loi normale (notamment la « méthode de la Ziggourat »), car celles-ci sont alors les plus rapides.

alors déterminée de façon unique). Cet algorithme est par exemple implémenté dans Matlab : la fonction `chol` appliquée à une matrice réelle définie positive $\boldsymbol{C}$ renvoie l'unique matrice $\boldsymbol{\Sigma}$ de même taille et triangulaire supérieure telle que $\boldsymbol{\Sigma}^{\mathsf{T}}\boldsymbol{\Sigma} = \boldsymbol{C}$ [‡‡].

Remarque 1.45. L'algorithme de Cholesky étant inapte à traiter des matrices positives *non définies*, on peut alors être confronté à un souci pour simuler des vecteurs gaussiens *dégénérés*, càd. dont la matrice de covariance est de rang non plein... Il est alors utile de se rappeler le lemme suivant : si $(X_1, \dots, X_n)$ est un vecteur gaussien de matrice de covariance $\boldsymbol{C}$, notant S un ensemble d'indices tels que les vecteurs-colonnes de $\boldsymbol{C}$ d'indices $j \in S$ forment une base de $\operatorname{Im} \boldsymbol{C}$, alors le vecteur gaussien $(X_j)_{j\in S}$ est non dégénéré ; et pour tout $j' \notin S$, il existe des coefficients $(a_j)_{j\in S}$ tels que $\sum_{j\in S} a_j X_j = X_{j'}$, qu'on peut déterminer en résolvant le système $\operatorname{Cov}(\sum_{j\in S} a_j X_j, \vec{X}_S) = \vec{0}$.

1.4.3 Méthode de la fonction de répartition

Définition 1.46 (Fonction de répartition)**.** Rappelons que la *fonction de répartition* d'une loi $\mathbb{P}$ sur $\mathbf{R}$ est la fonction $F\colon \mathbf{R} \to [0,1]$ définie par $F(x) := \mathbb{P}(\bullet \leqslant x)$.

Définition 1.47 (Réciproque de la fonction de répartition)**.** Sous les hypothèses précédentes, on appelle *fonction réciproque de la fonction de répartion de* $\mathbb{P}$, la fonction de $[0,1] \to \bar{R}$ définie par

$$F^{-1}(p) := \inf\{x \in \mathbf{R}\colon F(x) \geqslant p\}.$$

Remarque 1.48. On a alors, pour $\mathbb{P}$-presque-tout $x \in \mathbf{R}$, que $F^{-1}(F(x)) = x$ (resp. $\bar{F}^{-1}(\bar{F}(x)) = x$). En outre, si $\mathbb{P}$ est sans atome, on a pour presque-tout $p \in [0,1]$ que $F(F^{-1}(p)) = p$, ce qui permet alors de déterminer (presque-partout [∗]) $F^{-1}(p)$ comme la solution x de l'équation $F(x) = p$.

Lemme 1.49 (Méthode de la fonction de répartition)**.** *Si U suit une loi uniforme sur $[0,1]$, alors $F_{\mathbb{P}}^{-1}(U)$ suit la loi $\mathbb{P}$.*

Remarque 1.50. Il arrive que la « fonction de répartition complémentaire » $\bar{F}(x) := \mathbb{P}(\bullet \geqslant x)$ ait une expression plus simple que x. Dans la mesure où $\bar{F}$ n'est autre que la fonction de répartition de $\mathbb{P}$ quand on a renversé l'ordre sur $\mathbf{R}$, toute cette sous-section peut être réécrite en remplaçant « F » par « $\bar{F}$ », mutatis mutandis. J'appellerai alors l'analogue du lemme 1.49 « méthode de la fonction de répartition inverse ».

[‡‡]. On remarquera que la convention n'est pas la même que celle évoquée juste avant. La raison en est que Matlab préfère manipuler des vecteurs-lignes que des vecteurs-colonnes, de sorte que tout doit être transposé ; ainsi, sous Matlab, pour générer un vecteur gaussien centré de covariance $\boldsymbol{C}$, on l'écrit sous la forme $\vec{X}\boldsymbol{\Sigma}$, où $\vec{X}$ est un vecteur-*ligne* gaussien standard, et $\boldsymbol{\Sigma}$ la matrice de Cholesky telle que $\boldsymbol{\Sigma}'\boldsymbol{\Sigma} = \boldsymbol{C}$. Bien entendu, si on préfère manipuler tout de même des vecteurs-colonnes, il suffit de transposer la matrice fournie par la fonction `chol` de Matlab pour récupérer la matrice de Cholesky avec la convention standard.

[∗]. Notez que ce « presque-partout » est sans importance pour appliquer la méthode de la fonction de répartition.

1.4.4 Méthode de rejet

Lemme 1.51 (Méthode de rejet)**.** *Soit P une loi sur $\mathcal{X}$ et soit Q une loi sur x telle que Q soit à densité par rapport à P et que la densité dQ/dP soit uniformément bornée par une constante explicite M ; et supposons en outre que cette densité soit calculable explicitement*[†]*. Alors, si $(x_i, u_i)_{i\in\mathbf{N}}$ est une suite de couples i.i.d. de variables aléatoires, telles que x_i suive la loi P et que u_i soit indépendante de x_i et suive la loi Uniforme$(0,1)$, notant*

$$i_* := \min\{i \in \mathbf{N} \colon \big(dQ/dP\big)(x_i) \geqslant K u_i\},$$

la variable aléatoire x_{i_*} suit la loi Q.

Remarque 1.52. Pour appliquer la méthode de rejet, il faut en pratique que la densité dQ/dP soit explicitement calculable, et qu'on sache en déterminer un majorant explicite. En général, les situations où on saura calculer explicitement dQ/dP sera lorsque P et Q auront chacune une densité explicite par rapport à une même mesure de référence λ (typiquement, la mesure de Lebesgue), la densité de P étant en outre non nulle ; auquel cas on aura $\big(dQ/dP\big)(x) = \big(dQ/d\lambda\big)(x) \,/\, \big(dP/d\lambda\big)(x)$.

1.4.5 Nombres pseudo-aléatoires

L'implémentation informatique des méthodes de Monte-Carlo faisant appel à la génération pseudo-aléatoire, il convient d'en parler un peu ici.

Caractère *pseudo*-aléatoire des fonctions de type `rand`

Dans les utilisations informatiques, les fonctions aléatoires utilisées (p.ex. `rand` et `randn` sous Matlab) sont généralement, en fait, des fonctions *pseudo-aléatoires* : en réalité, les calculs qu'elles produisent sont tout ce qu'il y a de plus déterministes ! L'utilisation des ces nombres comme « aléatoires » repose donc sur un "acte de foi" :

Principe 1.53. Nous admettrons que les nombres pseudo-aléatoires fournis par nos logiciels se comportent *comme* des réalisations de variables aléatoires indépendantes (ayant la loi qu'elles sont censées avoir).

Remarque 1.54. « Se comporter comme des réalisations de variables aléatoires » signifie par exemple que, si $u_1, \dots, u_N$ sont censées être des réalisations de lois *Uniforme*$(0,1)$ indépendantes, alors pour tout $p \in [0,1]$, le nombre de u_i inférieures ou égales à p vaut pN, à quelques $N^{1/2}$ près. Plus généralement, cela signifie en substance *justement* que les calculs de Monte-Carlo donneront des résultats corrects (au risque statistique près).

Cet acte de foi est-il justifié ? La réponse est « oui » ! En effet, du fait justement de l'importance des méthodes de Monte-Carlo dans la recherche appliquée et dans l'industrie, la qualité des générateurs pseudo-aléatoires a fait depuis quelques décennies l'objet d'investigations poussées. Pour les générateurs pseudo-aléatoires contemporains (par exemple le « Mersenne twister »), on sait qu'il est pratiquement impossible de

[†]. En général, ce sera le cas

mettre en défaut ces générateurs, à moins de faire des tests extraordinairement poussés ou d'utiliser spécifiquement l'information sur la façon de fonctionner du générateur — ce qui ne correspond pas aux situations pratiques. Dans les situations que vous rencontrerez, vous pourrez ainsi considérer comme exclu que la nature *pseudo*-aléatoire du générateur vous pose souci — ce souci est bien moins à craindre, par exemple, que celui des erreurs d'arrondis...

Remarque 1.55. Notez qu'il existe des solutions technologiquement matures qui permettent aujourd'hui d'obtenir des nombres "véritablement" aléatoires, notamment en utilisant les fluctuations thermiques ou quantiques d'un système physique, ou en s'appuyant sur les interactions de l'utilisateur [‡] avec la machine (après un « hachage » convenable). Ces solutions ne sont que rarement utilisées, parce qu'elles sont sensiblement plus lentes que les générateurs pseudo-aléatoires (et requièrent parfois du matériel supplémentaire), alors que comme nous l'avons vu ceux-ci sont quasiment infaillibles (du moins concernant les plus modernes) vis-à-vis des méthodes de Monte-Carlo. L'aléa "vrai" est néanmoins essentiel dans les applications en cryptographie, où l'enjeu n'est pas tant que les suites de simulations se comportent comme des variables aléatoires, que le fait qu'il soit impossible de *deviner* les nombres fournis par la machine — ce qui un objectif qu'on ne pourrait pas atteindre à partir d'un algorithme déterministe, pour des raisons théoriques fondamentales.

Remarque 1.56. Cela dit, quel que soit le mode de production retenu pour simuler des variables aléatoires, il est stricto sensu impossible de prouver qu'une réalisation censée provenir d'une variable aléatoire [§] en provient effectivement ! Ainsi, si je lance une pièce de monnaie 1 000 fois et qu'elle retombe 1 000 fois sur « face », cela ne constitue pas une *preuve* pour autant que ma pièce soit truquée, puisqu'il existe une possibilité non nulle (quoique infime) que ce résultat se produise avec une pièce non truquée... Tout ce que vous aurez est une *présomption* (en l'occurrence extrêmement forte, et qu'il serait complètement *déraisonnable* de ne pas suivre). Le problème inverse se pose aussi : quand on dit que le bruit thermique d'une résistance (par exemple) est aléatoire, qu'entend-on par là ? Celui-ci provient pourtant d'un processus physique très compliqué, mais in fine déterministe [¶]... Tout ce que nous voulons dire par « aléatoire vrai » est alors que nous *croyons* qu'aucun moyen pratique ne pourra nous permettre de distinguer les résultats produits par ce processus de l'aléatoire théorique — au sens de la présomption évoqué ci-dessus — ; mais, l'ensemble des ces moyens pratiques envisageables étant très grand, il s'agit là aussi d'une sorte d'« acte de foi »... Même pour les phénomènes quantiques, dont la théorie nous dit qu'ils sont *parfaitement* aléatoires, pouvons-nous en être surs ?... Après tout, les physiciens les *modélisent* par des phénomènes parfaitement aléatoires, mais peut-être cela est-il faux, bien que le moyen de mettre le contraire en évidence soit extrêmement délicat... Bref ; quand on parle d'une *réalisation*, dire que celle-ci est aléatoire n'a pas de sens précis, car tenter de définir rigoureusement ce qu'est une réalisation aléatoire donnerait lieu très vite à des paradoxes philosophiques

[‡]. Qui est, au fond, lui-même un système physique ! ☺

[§]. « Une » réalisation d'une variable aléatoire peut ici désigner en fait *plusieurs* réalisations indépendantes, en considérant celles-ci comme la réalisation d'une unique variable aléatoire à valeurs dans l'espace produit.

[¶]. On suppose dans cet argument que la physique est entièrement « classique ».

insoutenables... Ce n'est pas par hasard que la théorie mathématique des probabilités renonce à dire ce qu'est *un* phénomène aléatoire et ne sait parler que de la *probabilité* de telle ou telle observation [|||]...

La question de la graine

Dans le détail, tous les générateurs pseudo-aléatoires fonctionnent sur le même principe. Tout d'abord, lorsqu'il y a besoin de générer une réalisation pseudo-aléatoire de loi déterminée, le logiciel s'appuie sur une suite de bits peuso-aléatoires i.i.d. équidistribués sur $\{0,1\}$ [**] ; et lit le nombre nécessaire de ces bits pour générer sa variable aléatoire en fonction de ceux-ci [††] — cette fonction étant déterminée de sorte que, si les bits étaient *vraiment* i.i.d. uniformes, la loi de l'objet qu'on construit à partir d'eux serait vraiment celle demandée [‡‡].

Ces bits pseudo-aléatoires sont générés par paquets de k, à mesure que le logiciel en a besoin, de la façon suivante. Il existe dans les registres du logiciel une variable réservée que nous noterons ici g, à valeurs dans un ensemble G fini mais gigantesque [*], initialisée à une certaine valeur g_0 ; le logiciel dispose en outre d'une certain fonction judicieusement choisie $f\colon G \to G$ et d'une autre fonction $p\colon G \to R$, où R désigne l'ensemble $\{0,1,2,\dots,2^k-1\}$. À chaque fois que le logiciel a besoin d'un nouveau paquet de k bits aléatoires, il procède de la façon suivante :

1. g est remplacé par $f(g)$;
2. On calcule le nombre $p(g)$; on l'exprime en numération binaire — avec k bits, quitte à ajouter des zéros non significatifs au début — ; et ces k bits sont renvoyés comme étant pseudo-aléatoires.

Si le générateur pseudo-aléatoire est bien conçu, l'itération de f produira un processus très irrégulier ayant la périodicité maximale $|G|$ (G étant gigantesque, c'est donc comme si la suite des itérées de f était apériodique) ; et les valeurs $p(g)$ successivement calculées

[|||]. À vrai dire, je simplifie un peu. Lorsqu'on est face à une quantité *infinie* d'information, par exemple une suite bits censés être i.i.d. uniformes, la *théorie de la complexité de Kolmogorov* permet de donner un sens précis (et tout-à-fait satisfaisant) à la question de savoir si *une* suite donnée est aléatoire ou non... Mais, ce sens n'étant défini qu'*asymptotiquement*, lorsque le nombre de bits observés tend vers l'infini, cela ne saurait s'appliquer aux situations pratiques où on n'observe jamais qu'une quantité d'information *finie*.

[**]. Du point de vue théorique, tout ce qui compte est le caractère i.i.d. et le fait que la loi soit connue. Pour ce qui est du choix d'utiliser des bits uniformes, on pourrait en revanche faire le tirage parmi un autre ensemble que $\{0,1\}$ ou prendre une distribution non équidistibuée ; quoique ce ne soit jamais le choix retenu en pratique.

[††]. Soulignons ici que, même lorsque l'ordinateur prétend tirer des nombres *réels*, ceux-ci sont bien entendu *représentés* dans la machine sous une forme *tronquée* (la mémoire de l'ordinateur n'est pas infinie !) (typiquement un nombre « réel » est décrit par 64 bits), de sorte que le tirage d'un « réel » aléatoire se fait en fait au sein d'un nombre *fini* de possibilités et n'exige par conséquent de recourir qu'à un nombre fini de bits.

[‡‡]. Par exemple, si mon logiciel utilise une représentation binaire en virgule fixe avec 53 bits après la virgule, pour tirer une variable aléatoire de loi *Uniforme*$(0,1)$, le logiciel tirera 53 bits (pseudo)-aléatoires $b_1, b_2, \dots, b_{53}$ en renverra la valeur (en binaire) $0, b_1 b_2 \dots b_{k2}$, qui suit bien (aux questions d'arrondis près) la loi uniforme sur $[0,1]$ lorsque les b_i soient i.i.d. uniformes.

[*]. Par exemple, pour le « Mersenne twister », $|G| = 2^{19\,937-1}$ (et $k = 32$).

seront pseudo-aléatoires i.i.d. uniformes sur R, ce qui correspond bien à des bits i.i.d. uniformes après conversion en binaire.

Une observation frappe alors immédiatement : si l'initialisation de la graine est identique d'une exécution à l'autre du logiciel, alors les bits pseudo-aléatoires, et donc les variables pseudo-aléatoires générées à partir de ces bits, seront exactement les mêmes à chaque exécution [†] ! Cette situation présente à la fois un inconvénient et un avantage :

Inconvénient : Les variables pseudo-aléatoires obtenues lors de deux éxécutions successives du logiciel, ou sur deux machines différentes [‡], ne seront donc pas indépendantes (en l'absence de précaution supplémentaire), ce qui interdit de distribuer le calcul entre plusieurs exécutions du logiciel (cf. §**??** et **??**) sous peine de résultats gravement erronés.

Avantage : Cela donne des résultats « aléatoires »... *reproductibles* ! Par exemple, si vous prétendez que tel résultat résulte de l'exécution de tel code, une personne exécutant le même code de son côté trouvera bien le même résultat ; de sorte qu'aucune suspicion ne pourra être retenue contre vous d'avoir "truqué" votre aléa pour obtenir des résultats qui vous arrangent [§]... Un autre exemple d'utilisation de la reproductibilité est si vous faites une simulation qui génère un flux énorme de données, trop gros pour être transmis, voire pour être stocké ; et que voulez transmettre cette simulation à quelqu'un en vue de lui en faire observer certains phénomènes : alors que v en outre, si votre but est par exemple de générer une simulation aléatoire dont les données sont monstrueusement volumineuses, vous pouvez vous contenter d'envoyer votre code-source à un tiers ; il observera exactement les mêmes phénomènes !

Concernant le premier point évoqué ci-dessus, si votre générateur de nombres pseudo-aléatoires est de bonne qualité (nous admettrons que c'est le cas), une solution efficace pour obtenir deux suites de nombres pseudo-aléatoires *indépendantes* d'une exécution du logiciel à l'autre est d'imposer soi-même une graine différente d'une exécution à l'autre. On a en effet, pour un bon générateur aléatoire, le phénomène suivant :

Principe 1.57. On admettra comme acte de foi que, si on réalise deux (ou plus) déroulements du générateur pseudo-aléatoires avec deux initialisations différentes de la graine, à moins de le faire vraiment (vraiment !) exprès [¶], les deux (ou plus) suites de bits obtenues (et donc les valeurs aléatoires données par les fonctions appelées) de comporteront comme si elles étaient *indépendantes*.

[†]. Si la façon dont la suite de bits pseudo-aléatoires est utilisée fournir les variables « aléatoires » diffère d'une exécution à l'autre (parce qu'on n'aura pas demandé les mêmes calculs à la machine), ces variables pseudo-aléatoires ne seront alors bien entendu pas *exactement* identiques ; mais elles seront tout de même fortement corrélées, ce qui présente le même inconvénient qu'évoqué ci-dessous.

[‡]. Ou même, dans le cas de deux logiciels différents mais utilisant la même générateur pseudo-aléatoire.

[§]. Ainsi, dans les exemples de ce polycopié, je réinitialise la graine avant de lancer chacun de mes programmes ; afin que vous puissiez vérifier que vous obtenez bien les mêmes résultats chez vous !

[¶]. Une telle façon de le « faire exprès » serait d'initialiser une suite avec la graine v_0, et l'autre suite avec la graine $f(v_0)$, auqeul cas les deux suites de bits pseudo-aléatoires seraient identiques à un décalage près... Mais évidemment, f est choisie en pratique de façon suffisamment subtile pour que vous n'ayez aucune chance de tomber sur un tel cas par hasard !

Exemple 1.58. En pratique, on pourra ainsi, si on veut s'assurer que chaque exécution du programme d'une fois à l'autre est indépendante, initialiser la graine aléatoire avec la date et l'heure du jour ; ou pour distribuer le calcul entre différentes machines, initialiser la graine de la première par 1, le graine de la seconde par 2, et ainsi de suite.

Remarque 1.59. Les erreurs d'arrondi sont en pratique plus à craindre que les faiblesses éventuelles du générateur pseudo-aléatoire...

Première partie

Méthode de Monte-Carlo pour le calcul d'une espérance

Chapitre 2

Principe de la méthode de Monte-Carlo

2.1 Généralités sur *les* méthodes Monte-Carlo

2.1.1 Qu'est-ce qu'« une » méthode Monte-Carlo ?

En fait, ce que nous appelons dans ce cours « la » méthode de Monte-Carlo devrait plutôt être appelé *méthode de Monte-Carlo pour le calcul des espérances*. Le véritable concept de méthode de Monte-Carlo est en réalité bien plus général :

Définition 2.1. De manière générale, ce qu'on appelle une *méthode de Monte-Carlo* est une technique visant à calculer une quantité *déterministe* par le biais d'un procédé *aléatoire*.

Exemple 2.2. Une des plus célèbres méthodes de Monte-Carlo est le *recuit simulé*, qui sert à trouver le minimum *global* d'une fonction *non convexe* de $\mathbf{R}^d$ dans $\mathbf{R}$, en particulier lorsque la dimension d est grande. L'idée consiste à suivre une méthode de descente de gradient perturbée par un bruit dont l'intensité diminue progressivement. En termes mathématiques, notant f la fonction à minimiser (supposée de classe $\mathcal{C}^1$), partant d'un point $X_0 \in \mathbf{R}^d$ arbitraire, on simule la solution de l'ÉDS

$$dX_t = -\nabla f(X_t)dt + \sigma(t)dW_t,$$

où $\sigma(t)$ est une fonction déterministe qui tend lentement vers 0.

On peut alors montrer que, sous certaines conditions assez générales (coercivité de f, décroissance suffisamment lente de $\sigma(t)$), quel que soit le point de départ choisi, $f(X_t)$ converge en loi vers la constante $\min f$ — ce qui implique aussi, si f atteint son minimum en un unique point x_*, que X_t converge en loi vers x_*.

Remarque 2.3. Il existe de nombreuses variantes de la méthode de recuit simulé : où on peut rendre la convergence presque-sure, où $\sigma(t)$ n'est pas forcément déterministe, où la fonction n'est pas forcément dérivable, où la fonction est définie sur un espace discret, ... Le recuit simulé constituerait un cours à lui tout seul !...

Remarque 2.4. Vous entendrez aussi peut-être parler de « méthode de Monte-Carlo par chaine de Markov » (ou « méthode MCMC »). Il ne s'agit pas d'une méthode de

Monte-Carlo à proprement parler, dans la mesure où le but n'est pas d'évaluer une quantité déterministe, mais plutôt de simuler une certaine loi aléatoire [*], lorsque celle-ci présente certaines formes qui la rendraient impossible à simuler par les méthodes classiques. Typiquement, il s'agira d'une loi sur $(\mathbf{R}^n)^d$, avec d très grand, définie par

$$\log\left(\frac{d\mathbb{P}}{dx}\right)\big(x=(x_1,\dots,x_d)\big)=\sum_{i=1}^{d}V_i(x_i)+\sum_{1\leqslant i<j\leqslant d}W_{i,j}(x_i,x_j)+\log(Z),$$

où les $V_i\colon \mathbf{R}^n\to\mathbf{R}$ et les $W_{i,j}\colon \mathbf{R}^n\times\mathbf{R}^n\to\mathbf{R}$ sont certaines fonctions déterministes, et où la constante Z est à déterminer de sorte que $\mathbb{P}$ soit une mesure de probabilité. Ce genre de lois, appelées *mesures de Gibbs*, apparaissent fréquemment lorsqu'on s'intéresse à la mesure d'équilibre d'un système physique à l'échelle atomique.

2.1.2 Généralités sur les méthodes de Monte-Carlo

Historiquement, c'est en 1949 que le physicien gréco-américain Nicholas Metropolis et le mathématicien américain d'origine polonaise Stanisław Ulam publient l'article fondateur de cette méthode de calcul et lui donnent son nom. Pour être plus exact, l'idée de procéder à des tirages aléatoires pour évaluer des intégrales compliquées était dans l'air du temps parmi la communauté des physiciens, mais l'apport majeur de Metropolis & Ulam fut de proposer la technique d'échantillonnage préférentiel (cf. § 3.2), qui améliore largement l'efficacité de la méthode... Pour l'anecdote, c'est dans le cadre des recherches du projet *Manhattan* sur le développement de la bombe atomique que ces chercheurs (avec quelques autres dont notamment John von Neumann) avaient commencé à développer leurs idées.

Des développements importants des méthodes de Monte-Carlo furent l'algorithme de Metropolis – Hastings pour la simulation de certaines variables aléatoires en physique statistique (travaux dus notamment à Marshall Rosenbluth en 1953 et à Keith Hastings en 1970), algorithme qui à son tour fut la base de la méthode du recuit simulé (1983) pour trouver des extrema globaux de fonctions définies sur des espaces de grande dimension. Plus récemment (2008), on a aussi parlé des méthodes de Monte-Carlo à l'occasion de leur utilisation dans des logiciels joueurs de go (très grossièrement, l'idée est que l'ordinateur évalue la qualité d'une position en imaginant que les joueurs terminent leur partie en jouant au hasard), où ces méthodes ont permis des progrès spectaculaires. [†]

La dénomination « Monte-Carlo » provient simplement de l'appel au hasard dans les algorithmes, par allusion au célèbre quartier de Monaco réputé pour son casino...

2.1.3 Un mot sur les méthodes quasi-Monte-Carlo

L'écueil principal des méthodes de Monte-Carlo est leur taux de convergence en $O(N^{-1/2})$, ce qui est plutôt lent... Il se trouve que, notamment dans le calcul d'intégrales de pas trop grande dimension, on peut sous certaines conditions faire sensiblement

[*]. Ce qui, bien entendu, peut ensuite servir pour calculer par la méthode de Monte-Carlo une espérance associée à cette loi...

[†]. Avec notamment le logiciel *MoGo*, développé au sein de laboratoires de recherche français !

mieux avec la méthode dite *quasi-Monte-Carlo*. Cette méthode consiste, comme la méthode de Monte-Carlo « standard », à estimer $\mathbb{E}(f)$ par une quantité de la forme

$$N^{-1} \sum_{i=1}^{N} f(X_i),$$

à ceci près que les X_i ne sont plus indépendants ! On peut en effet montrer que, sous réserve que les X_i soient judicieusement choisis (il s'agira en général d'une suite *déterministe*, mais *irrégulière*), la convergence de cette quantité est en $O(N^{-1})$, sous réserve de certaines propriétés de régularité de f.

Malgré leur vitesse de convergence très supérieure, les méthodes quasi-Monte-Carlo ne sont pas une panacée pour autant : les conditions de régularité à exiger sur f sont plus restrictives ; la précision de la méthode n'est pas facile à évaluer ; et la méthode n'est pas aussi robuste à la dimension que la méthode de Monte-Carlo... Quoi qu'il en soit, malgré la similarité des formules et du nom, les bases mathématiques de la méthode de Monte-Carlo et de la méthode quasi-Monte-Carlo n'ont en fait pas grand-chose à voir, cette dernière ne faisant essentiellement aucun appel au hasard...

2.2 Estimation d'une espérance par la méthode de Monte-Carlo

2.2.1 Estimation d'une probabilité par la méthode de rejet

Exemple 2.5 (Le championnat de basket). Un passionné de basketball souhaite évaluer la probabilité que son équipe préférée, Nancy, gagne la championnat national pour l'année à venir. Pour ce faire, notre passionné a mis au point une modélisation détaillée du déroulement du championnat, modélisation qui lui permet de calculer exactement la probabilité du résultat de chaque match en fonction des équipes en lice, sachant que le modèle du passionné affirme aussi que les résultats des différents matchs sont indépendants les uns des autres. Les détails techniques figurent dans l'annexe A.1.

Le problème est que, pour passer du calcul de la probabilité des résultats des matchs individuels à celui de la probabilité du résultat global du championnat, le très grand nombre de possibilités à prendre en compte rend l'approche théorique totalement inextricable... Que faire alors ? Notre passionné à une idée. Il se dit : « finalement ; à la base une probabilité que Nancy gagne le championnat, c'est la proportion de fois où Nancy gagnerait le championnat si des championnats indépendants avaient lieu dans un très très grand nombre d'univers parallèles ! Or, avec ma modélisation je suis capable de *simuler* le déroulement du championnat, et même d'en simuler un très grand nombre de réalisations indépendantes ! La proportion de fois où Nancy aura gagné lors de ces simulations devrait ainsi être une valeur approchée fiable de la véritable probabilité de victoire finale ».

Ainsi, notre passionné écrit le programme suivant (les fonctions auxiliaires appelées par ce programme figurent dans les annexes A.1.4 et A.1.5) :

```
% La fonction "basket_rejet" estime la probabilité de victoire de l'équipe
```

```
% de Nancy par la méthode de rejet.
function basket_rejet ()
% "Nsim" est le nombre de simulations à effectuer.
Nsim = 60000;
% "NANCY" est le numéro associé à l'équipe de Nancy.
NANCY = 3;
% bilan est le nombre de simulations où l'équipe de Nancy a été championne.
bilan = 0;
% On lance les simulations
for i = 0 : Nsim - 1
    % On lance la fonction "championne" qui simule la vainqueure du
    % championnat. S'il s'agit de Nancy, on ajoute 1 à "bilan" ; sinon on
    % lui ajoute 0.
    bilan = bilan + (championne () == NANCY);
end
% "proportion" est la proportion de victoires de Nancy sur cet ensemble de
% simulations.
proportion = bilan / Nsim;
% On affiche le résultat.
fprintf ('La probabilité estimée de victoire pour l''équipe de Nancy ');
fprintf ('est de %f %%.\n', 100 * proportion);
return
end
```

L'exécution lui donne :

```
>> basket_rejet
Probabilité de victoire évaluée pour l'équipe de Nancy :
    0.1086
```

En fait, l'idée que vient d'avoir notre passionné n'est rien d'autre que la méthode de Monte-Carlo, dans sa forme la plus simple, dite « méthode de rejet », pour estimer une probabilité. Nous pouvons formaliser cette méthode de la façon suivante :

Définition 2.6. Supposons qu'on cherche à évaluer la probabilité $\mathbb{P}(A) =: p$ d'un évènement A sous une loi $\mathbb{P}$ qu'on sait simuler. La *méthode de Monte-Carlo* consiste alors à réaliser un grand nombre N_{sim} de simulations indépendantes de la loi $\mathbb{P}$, à compter le nombre n_{succ} de ces simulations pour lesquelles A a eu lieu, et à estimer $\mathbb{P}(A)$ par la proportion $\hat{p}$ (que nous noterons aussi $\hat{p}_{N_{\text{sim}}}$ lorsque nous voudrons souligner la dépendance en N_{sim}) de telles simulations :

$$\hat{p} := n_{succ} \,/\, N_{\text{sim}}.$$

Remarque 2.7. À proprement parler, n_{succ} et $\hat{p}$ sont des *variables aléatoires*. Ce que notre programme calcule, par contre, est une *réalisation* de ces variables aléatoires, càd. les valeurs $n_{\text{succ}}(\omega)$ et $\hat{p}(\omega)$ pour l'éventualité ω simulée par nore programme. Dans la suite du cours, nous désignerons de la même façon les variables aléatoires et leurs réalisations, le contexte permettant de trancher ; il convient toutefois de rester attentif à cette nuance.

Théorème 2.8. *Quand N_{sim} tend vers l'infini, l'estimateur $\hat{p}_{N_{\text{sim}}}$ est convergent vers p.*

Démonstration. C'est simplement une reformulation de la version fréquentielle de la loi des grands nombres. □

Remarque 2.9. Dans le théorème 2.8, on peut même préciser que $\hat{p}_N$ converge *fortement*. Il y a aussi convergence L^p pour tout $p < \infty$.

2.2.2 Estimation d'une espérance

La méthode présentée dans la définition 2.6 est en fait un cas particulier de la méthode générale de Monte-Carlo, qui permet d'évaluer des *espérances* :

Définition 2.10. Supposons qu'on cherche à évaluer l'espérance $\mathbb{E}(f) =: m$ d'une variable aléatoire f de classe L^1 sous une loi $\mathbb{P}$ qu'on sait simuler. La *méthode de Monte-Carlo* consiste alors à réaliser une grand nombre N de simulations de la loi $\mathbb{P}$, à regarder la valeur $f(\omega_i)$ de f pour chacune de ces simulations, et à estimer $\mathbb{E}(f)$ par la moyenne empirique $\hat{m}$ de ces valeurs :

$$\hat{m} := N^{-1} \sum_{i=1}^{N} f(\omega_i).$$

Remarque 2.11. Dans le cas particulier où f est l'indicatrice $\mathbf{1}_{\{A\}}$ d'un évènement A, la définition 2.10 redonne la définition 2.6.

Théorème 2.12. *L'estimateur $\hat{m}_{N_{\text{sim}}}$ converge (fortement) vers $\mathbb{E}(f)$, et converge aussi dans L^1.*

Démonstration. C'est encore la loi (forte) des grands nombres, cette fois-ci dans sa version générale. □

Remarque 2.13. Attention, les théorèmes 2.8 et 2.8 ne sont valables en toute rigueur que si les simulations ω_i sont *exactement* i.i.d. $\mathbb{P}$. En pratique, les calculs informatiques utilisent généralement des nombres seulement *pseudo*-aléatoires, ce qui peut faire échouer la méthode de Monte-Carlo dans les cas les plus extrêmes. De même, les théorèmes ne fonctionnent en toute rigueur que si la probabilité simulée est *exactement* la loi $\mathbb{P}$...

Exemple 2.14 (Le Keno). La loterie nationale a décidé de lancer un nouveau jeu qu'elle appellera *Keno*. Dans ce jeu, un joueur doit cocher un certain nombre de numéros sur sa grille, puis un tirage national est organisé ; en fonction de l'adéquation des numéros cochés à ceux tirés, le joueur marque alors plus ou moins de points ; et au-delà d'un certain nombre de points, cela se traduit pour le joueur par un gain financier qui est une fonction directe du nombre de points marqués (voir les détails techniques dans l'annexe **??**).

La loterie nationale ayant déjà lancé l'impression de ses tickets de jeu sur lesquels figure l'échelle des gains, une question essentielle se pose alors : « à quel prix vendre le jeu ? ». En effet, pour que le jeu soit rentable pour l'organisatrice, il faut que le prix d'achat des ticket soit supérieur le montant des paiements... Le jeu étant destiné à être

joué un très grand nombre de fois, ce qui compte d'après la loi des grands nombres, c'est que le prix de vente d'un ticket soit supérieur à l'*espérance de gain* du joueur — laquelle ne dépend pas des numéros cochés. Mais, de même que pour le championnat de basket, calculer théoriquement cette espérance serait affreusement complexe ! En revanche, on peut la simuler par la méthode de Monte-Carlo, ce qui produit le code suivant (les fonctions auxiliaires se trouvent dans les annexes A.3.1 à A.3.3) :

```
% La fonction "keno_simple" estime l'espérance de gain d'un joueur de Keno
% par la méthode de Monte-Carlo simple.
function keno_simple ()
% "Ncoch" est le nombre de numéros à cocher.
Ncoch = 16;
% "Nsim" est le nombre de simulations effectuées.
Nsim = 1000000;
% "total" est le total des résultats obtenus au cours des diverses
% simulations.
total = 0;
% On lance la boucle de simulations.
for i = 1 : Nsim
    % On appelle "carton" la donnée des numéros cochés par le joueur,
    % "carton(p)" désignant le numéro choisi à p points. Puisque la loi du
    % gain du joueur est la même quels que soient les numéros cochés, on
    % utilisera toujours le même carton.
    carton = 1 : Ncoch;
    % À l'aide de la fonction "tirage" pour simuler le tirage, de la
    % fonction "score" pour calculer le score à partir de "carton" et
    % "tirage", et de la fonction "gain" qui associe le gain correspondant
    % au score, on détermine la quantité d'argent gagnée par le joueur,
    % qu'on ajoute à "total".
    total = total + gain (score (carton, tirage ()));
end
% "moyenne" est le gain moyen, qui estime l'espérance de gain.
moyenne = total / Nsim;
% On affiche le résultat.
fprintf ('L''espérance de gain du Keno est estimée à ');
fprintf ('%f €.\n', moyenne);
% On quitte le programme.
return
end
```

L'exécution donne :

```
>> keno_simple
L'espérance de gain du Keno est estimée à 3.003952 €.
```

Il semble donc que l'espérance de gain sera proche de 3 € : il faudra donc vendre la grille au-delà de ce prix, p. ex. à 4 €.

2.2.3 Application au calcul d'intégrales

La méthode de Monte-Carlo semble de prime abord assez naturelle : pour évaluer une espérance, objet de nature probabiliste exprimant une moyenne sur l'infinité d'éventualités posibles, quoi de plus logique que de recourir à la simulation aléatoire pour considérer la moyenne sur un grand nombre de réalisations ?... Pourtant, à bien y regarder, il y a une antinomie fondamentale entre l'estimateur $\hat{m}$ et la moyenne m : en effet, l'estimateur de Monte-Carlo est une *variable aléatoire*, dont la réalisation que nous obtenons est *indéterministe* (au sens où deux simulations identiques donneront à priori des résultats différents) ; alors que l'espérance à évaluer, bien que de nature « probabiliste », est tout bien pesé une quantité strictement *déterministe*... !

En effet, souvenez-vous comment est définie une espérance : en toute rigueur mathématique, ce n'est rien de plus qu'une *intégrale* de forme un peu particulière, à savoir

$$\mathbb{E}(f) := \int_\Omega f(\omega)\, d\mathbb{P}(\omega),$$

où $\mathbb{P}$ n'est à la base qu'une *mesure* de masse totale 1, que nous *interprétons* en lui donnant une signification en termes de « probabilités ». Au final, on voit que la méthode de Monte-Carlo est une méthode qui permet d'évaluer une *intégrale*, de nature *déterministe*, à l'aide d'un estimateur *aléatoire* !

Remarque 2.15. C'est cet aspect paradoxal consistant à estimer le déterministe par l'aléatoire qui fait la force de la méthode de Monte-Carlo. Ainsi, le hasard peut parfois être un atout plutôt qu'un handicap... C'est de cet aspect aléatoire inattendu que vient le nom « méthode de Monte-Carlo », par allusion au quartier de Monaco célèbre pour ses casinos ☺

On peut alors se demander si la méthode de Monte-Carlo ne peut pas être aussi utilisée pour estimer des intégrales "quelconques" ne s'écrivant pas naturellement sous la forme d'espérances, par exemple l'intégrale d'une fonction (intégrable) sur $\mathbf{R}^d$ par rapport à la mesure de Lebesgue. De fait, c'est le cas ; et c'est précisément l'objet de ce chapitre :

Proposition 2.16. *Pour f une fonction intégrable sur $\mathbf{R}^d$, et $\mathbb{P}$ une mesure de probabilité sur $\mathbf{R}^d$ ayant par rapport à la mesure de Lebesgue une densité $d\mathbb{P}/dx$ qui est non nulle partout où f est non nulle, on a :*

$$\int_{\mathbf{R}^d} f(x)\, dx = \mathbb{E}\big((d\mathbb{P}/dx)^{-1} f\big).$$

Cela permet alors d'utiliser l'estimateur de Monte-Carlo (2.10) *pour évaluer $\int f(x)\, dx$.*

Démonstration. Notons Ω l'ensemble des points où $d\mathbb{P}/dx$ est non nulle ; la mesure de Lebesgue a alors sur Ω une densité par rapport à $\mathbb{P}$ qui est l'inverse de la densité $d\mathbb{P}/dx$. D'autre part, puisque f est nulle en-dehors de Ω, on a $\int_{\mathbf{R}^d} f(x)\, dx = \int_\Omega f(x)\, dx$, d'où $\int_{\mathbf{R}^d} f(x)\, dx = \int_\Omega f(x)\big(dx/d\mathbb{P}\big)(x)\, d\mathbb{P}(x) = \int_\Omega \big(d\mathbb{P}/dx\big)^{-1}(x) f(x)\, d\mathbb{P}(x) = \mathbb{E}\big((dP/dx)^{-1} f\big)$, CQFD. □

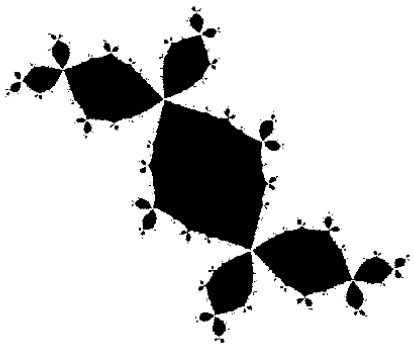

FIGURE 2.1 – Le lapin de Douady.

Remarque 2.17. Évidemment, il reste indispensable pour pouvoir appliquer la méthode de Monte-Carlo de se ramener in fine à une espérance... ! C'est pourquoi, à l'issue de ce chapitre, nous ne nous adresserons plus qu'au cas de l'évaluation par la méthode de Monte-Carlo d'une quantité écrite sous la forme d'une espérance. Ce qui est important, c'est de se rappeler que cela englobe aussi le cas des intégrales "ordinaires", une fois celles-ci réécrites sous la forme d'espérances à l'aide de la technique exposée dans la proposition 2.16.

Remarque 2.18. Il y a très nombreuses possibilités pour choisir la loi $\mathbb{P}$ destinée à appliquer la proposition 2.16, qui conduisent à autant d'estimateurs de Monte-Carlo *différents* pour la *même* intégrale ! Certains de ces estimateurs s'avèreront plus efficaces que d'autres, ce qui rendra le choix de $\mathbb{P}$ particulièrement important. Quand c'est pour évaluer une *espérance* qu'on utilise la méthode de Monte-Carlo, il y a évidemment un choix de $\mathbb{P}$ naturel, à savoir la loi sous laquelle l'espérance est prise ; cependant nous verrons qu'on peut quand même changer la loi d'échantillonnage, et ce parfois avec profit : cela correspond à ce qu'on appelle l'*échantillonnage préférentiel*, dont nous parlerons dans la § 3.2.

Exemple 2.19 (Le lapin de Douady). Le lapin de Douady est une figure du plan dont vous pouvez observer l'allure sur la figure 2.1. La définition de cette figure est bien précise ; et il est algorithmiquement facile de savoir si un point donné du plan appartient ou non au lapin de Douady (voir l'annexe A.4.1 pour les détails). En revanche, le bord du lapin de Douady est très irrégulier [‡], ce qui rend son aire délicate à calculer : ainsi, si on veut estimer cet aire en maillant le plan régulièrement et en regardant si chaque point du maillage tombe ou non dans le lapin, est-on certain qu'un tel maillage régulier sera bien adapté ?... C'est pourquoi ici nous allons plutôt procéder par la méthode de Monte-Carlo.

L'aire de cette figure, est définie comme $\int_{\mathbf{R}^2} \mathbf{1}_{\{x+yi \in lapin\}}\, dxdy$; c'est donc à priori une intégrale et pas du tout une espérance. Voyons comment nous pouvons la réécrire comme une espérance.

[‡]. Le bord du lapin de Douady est en fait un objet fractal, dont la dimension fractale est ici 1,393 4 environ. Pour la petite histoire, c'est un exemple de ce qu'on appelle un *ensemble de Julia*, du nom du mathématicien français Gaston JULIA qui a étudié ce genre de figures.

La définition du lapin de Douady nous assure que la figure est complètement contenue dans le carré $[-2,2] \times [-2,2]$. Nous pouvons donc réécrire l'aire comme

$$\int_{(x,y)\in[-2,2]\times[-2,2]} \mathbf{1}_{\{x+yi\in lapin\}}\, dxdy.$$

Soit maintenant $\mathbb{P}$ la probabilité uniforme sur la carré $[-2,2]\times[-2,2]$. Cette probabilité est facile à simuler : en effet, il suffit de tirer deux variables aléatoires U et V uniformes sur $[0,1]$ et indépendantes, et tirer alors $(x,y) = (-2+4U, -2+4V)$. En outre, la densité de $\mathbb{P}$ par rapport à la mesure de Lebsegue est immédiate à calculer, et vaut $1/16$ uniformément sur le carré.

Cela conduit au programme suivant pour estimer l'aire du lapin (voir les annexes A.4.3 et A.4.4 pour les fonctions auxiliaires) :

```
% La fonction "lapin" estime l'aire du lapin de Douady par la méthode de
% Monte-Carlo, en échantillonnant selon la loi uniforme sur le carré
% [-2,2]×[-2,2].
function lapin
% "Nsim" est le nombre de simulations.
Nsim = 15000000;
% "somme" est la somme des simulations effectuées.
somme = 0;
% Boucle de Monte-Carlo.
for i = 1 : Nsim
    % On tire z uniformément sur le carré [-2,2]×[-2,2].
    z = (-2 + rand () * 4) + (-2 + rand () * 4) * 1i;
    % Calcule la fonction dont on prendra l'espérance. La fonction
    % "enlapin" renvoie 1 si on est dans le lapin, et 0 sinon.
    somme = somme + enlapin (z) * 16;
end
% "moyenne" est la moyenne des simulations effectuées.
moyenne = somme / Nsim;
% Affichage du résultat.
fprintf ('L''aire du lapin de Douady est estimée à ');
fprintf ('%f.\n', moyenne);
end
```

J'ai obtenu à l'exécution :

```
>> lapin
L'aire du lapin de Douady est estimée à 1.307076.
```

Remarque 2.20. Lorsque la fonction à intégrer est à support compact, l'idée présentée dans cet exemple, consistant à choisir pour mesure d'échantillonnage la mesure uniforme sur un pavé contenant le support de la fonction, est la façon "de base" d'appliquer Monte-Carlo ; il faut donc être capable de transposer cet exemple immédiatement à d'autres situations similaires.

Exemple 2.21 (SOS analyse complexe). Voici un autre exemple d'application de la méthode de Monte-Carlo au calcul d'une intégrale. Dans cet exemple, un ingénieur a été amené à calculer à la main la valeur exacte de l'intégrale

$$I := \int_{-\infty}^{\infty} \frac{x^2+1}{x^2+x+1}\left(\frac{\sin x}{x}\right)^2 dx.$$

Il a trouvé un résultat de

$$\pi + \frac{\pi}{2\sqrt{3}} - \frac{\pi e^{-\sqrt{3}}\cos(1)}{4\sqrt{3}} - \frac{\pi e^{-\sqrt{3}}\sin(1)}{4} \simeq 3{,}888\ 221;$$

cependant il a un doute, le calcul étant fort compliqué... Il demande alors à sa petite sœur en licence de mathématiques de faire le calcul de son côté, et celle-ci trouve quant à elle un résultat de

$$\pi + \frac{\pi}{2\sqrt{3}} - \frac{\pi e^{-\sqrt{3}}\cos(1)}{2\sqrt{3}} - \frac{\pi e^{-\sqrt{3}}\sin(1)}{2} \simeq 3{,}727\ 950.$$

Pour trancher entre les deux options, l'idée vient alors à notre ingénieur de calculer l'intégrale numériquement. Ne sachant pas trop quelle est la meilleure façon de gérer la discrétisation et troncature pour le calcul de I par une méthode classique comme celle des rectangles, il décide — pourquoi pas ? — de passer plutôt par la méthode de Monte-Carlo.

Dans un premier temps, il faut choisir une loi sur $\mathbf{R}$ ayant une densité non nulle partout où l'intégrande est non nulle — càd. en l'occurrence sur tout $\mathbf{R}$ (à un ensemble négligeable près) —, qui soit facile à simuler et dont la densité soit facile à calculer. Notre ingénieur choisit la loi $\mathbb{P}$ de densité

$$d\mathbb{P}(x) = \frac{1}{(1+|x|)^2}dx,$$

dont la méthode de simulation est expliquée dans l'annexe A.5.1

On peut alors écrire

$$I = \mathbb{E}_{\mathbb{P}}\left((1+|x|)^2\frac{x^2+1}{x^2+x+1}\left(\frac{\sin x}{x}\right)^2\right),$$

ce qui permet d'utiliser la méthode de Monte-Carlo, avec le programme suivant (voir les fonctions auxiliaires en annexe) :

```
function SOS ()
N = 4000000;
total = 0;
for i = 1:N
x = echantillonnage_SOS ();
y = integrande_SOS (x) / densite_SOS (x);
total = total + y;
end
moyenne = total / N;
fprintf ('L''intégrale I est estimée à ');
fprintf ('%f.\n', moyenne);
end
```

À l'exécution :

```
>> SOS
L'intégrale I est estimée à 3.729438.
```

Les deux premières décimales coïncident avec la valeur prévue par la petite sœur ; il y a donc tout lieu de croire que c'est elle qui a raison [§].

2.3 Intervalles de confiance

2.3.1 Intervalle de confiance pour une espérance calculée par Monte-Carlo

Savoir que l'estimateur de Monte-Carlo converge est une bonne chose, mais tant qu'on n'a pas d'information sur la vitesse de sa convergence, l'interprétation du résultat est essentiellement impossible... Il est donc important d'être capable non seulement de fournir une estimation pour $\mathbb{E}(f)$, mais aussi d'indiquer quelle est sa *précision*.

Bien entendu, l'estimateur de Monte-Carlo est aléatoire, et de ce fait il peut *théoriquement* donner un résultat complètement aberrant : par exemple, si on cherche à évaluer la probabilité qu'une pièce non truquée tombe sur "pile", il y a une probabilité non nulle (quoique extrêmement faible) que tous les lancers faits pour évaluer la probabilité donnent "face" et donc qu'on évalue à tort la probabilité de "pile" comme étant nulle ! Nous ne pourrons donc pas donner un intervalle de *certitude* pour la valeur réelle de $\mathbb{E}(f)$, mais seulement un intervalle de *confiance*, c'est-à-dire un intervalle dont on sait qu'il contiendra effectivement $E(f)$ avec une très grande probabilité p fixée à l'avance (p.ex. $p = 95\ \%$).

C'est à la construction de tels intervalles de confiance que s'intéresse cette section.

Théorème 2.22. *Supposons qu'on cherche à évaluer $\mathbb{E}(f)$ par la méthode de Monte-Carlo avec f de classe L^2, et notons σ la variance (supposée non nulle) de f. Alors, notant $\hat{m}$ l'estimateur de Monte-Carlo, quand le nombre de simulations N_{sim} tend vers l'infini,*

$$\frac{\hat{m}_{N_{\text{sim}}} - E(f)}{\sigma N_{\text{sim}}^{1/2}} \xrightarrow{loi} \mathcal{N}(0,1).$$

Corolaire 2.23. *En particulier, sous les hypothèses du théorème précédent, pour tous $c_+ < c_-$,*

$$\lim_{N_{\text{sim}}\to\infty} [\hat{m}_{N_{\text{sim}}} + c_-\sigma N_{\text{sim}}^{-1/2}, \hat{m}_{N_{\text{sim}}} + c_+\sigma N_{\text{sim}}^{-1/2}] \ni \mathbb{E}(f) \to \mathbb{P}\big(\mathcal{N}(1) \in [c_-, c_+]\big)$$
$$= (2\pi)^{-1/2} \int_{c_-}^{c^+} e^{-x^2/2}\, dx,$$

ce qui signifie que $[\hat{m}_{N_{\text{sim}}} + c_-\sigma N_{\text{sim}}^{-1/2}, \hat{m}_{N_{\text{sim}}} + c_+\sigma N_{\text{sim}}^{-1/2}]$ est un intervalle de confiance pour $E(f)$ de niveau asymptotique $(2\pi)^{-1/2} \int_{c_-}^{c_+} e^{-x^2/2}\, dx$.

[§]. Moralité : Il faut toujours écouter sa petite soeur ! ☺

Démonstration. Le théorème 2.22 est une reformulation directe du théorème-limite central ; et le corolaire 2.23 en découle via le lemme 1.17 □

En pratique, σ n'est généralement pas connu. Il faut alors utiliser le théorème suivant :

Corolaire 2.24. *Supposons qu'on cherche à évaluer $\mathbb{E}(f)$ par la méthode de Monte-Carlo avec f de classe L^2. Notons $\hat{\sigma}$ la* variance empirique *de f à l'issue des N simulations, autrement dit la variance qu'aurait f sous la loi uniforme sur les ω_i* [¶] *:*

$$\hat{\sigma}_{N_{\text{sim}}} := \left(N_{\text{sim}}^{-1} \sum_{i=1}^{N_{\text{sim}}} f(\omega_i)^2 - \left(N_{\text{sim}}^{-1} \sum_{i=1}^{N_{\text{sim}}} f(\omega_i) \right)^2 \right)^{1/2}.$$

Alors, quand $N \to \infty$,

$$\frac{\hat{m}_{N_{\text{sim}}} - E(f)}{\hat{\sigma} N_{\text{sim}}^{1/2}} \overset{loi}{\to} \mathcal{N}(0,1).$$

De même, on a aussi l'équivalent du corolaire 2.23 en y remplaçant « σ » par « $\hat{\sigma}$ ».

Démonstration. Notons $\hat{m} := N_{\text{sim}}^{-1} \sum_{i=1}^{N_{\text{sim}}} f(\omega_i)$, resp. $\hat{m}_2 := N_{\text{sim}}^{-1} \sum_{i=1}^{N_{\text{sim}}} f(\omega_i)^2$. D'après le théorème 2.8, $\hat{m}$ est un estimateur convergent de $\mathbb{E}(f)$, resp. $\hat{m}_2$ est un estimateur convergent de $\mathbb{E}(f^2)$. En utilisant la continuité de la fonction $(x,y) \mapsto (y-x^2)^{1/2}$ [|||], on en déduit par le lemme 1.11 que $\hat{\sigma}$ converge (fortement) vers σ. Puis, en utilisant le lemme 1.23, vu la continuité de l'opération $(x,y) \mapsto x \,/\, (s/\sigma)$ en tous les points de la forme (x,σ), on déduit (2.24) de (2.22) et de la convergence de $\hat{\sigma}$ vers σ. □

Remarque 2.25. Attention, les théorèmes 2.23 et 2.24 exigent pour être valides *que f soit de classe L^2*, ce qui est strictement plus fort que la condition L^1 pour avoir la convergence de l'estimateur de Monte-Carlo. Dans la suite de ce cours, nous ne considèrerons plus que le cas où f est effectivement de classe L^2, vu que c'est ce qui arrive le plus souvent ; toutefois il y a aussi des situations où la condition L^2 n'est pas satisfaite, et alors il ne faut surtout pas utiliser le théorème 2.24 sous peine d'obtenir des résultats complètement aberrants !

Remarque 2.26. Dans les situations pratiques, il n'est toujours évident de prouver que f est effectivement L^2 ; et même lorsque c'est le cas cela ne suffit d'ailleurs pas toujours (voir remarque 2.27) ci-dessous. Une manière d'essayer de vérifier l'intégrabilité L^2 de f peut être de regarder si l'estimateur $\hat{\sigma}$ de la variance reste bien stable quand le nombre de simulations augmente : on peut en effet montrer que, dès lors que f n'est pas L^2, cet estimateur diverge presque-surement vers $+\infty$.

Remarque 2.27. Bien qu'on parle souvent simplement d'« intervalle de confiance au risque α », il convient de garder à l'esprit que les intervalles de confiance que donne le théorème 2.24 sont seulement *asymptotiques*, càd. que la probabilité de confiance $(1-\alpha)$ n'est valable que quand N tend vers l'infini... En pratique, on pose comme

[¶]. Il est sous-entendu que l'observation ω que l'on fait correspond à une collection d'observations indépendantes $(\omega_i)_{i \in \mathbf{N}^*}$

[|||]. On notera que $(\hat{m}, \hat{m}_2)$ et $(\mathbb{E}(f), \mathbb{E}(f^2))$ sont automatiquement dans le domaine de définition de $(x,y) \mapsto (y-x^2)^{1/2} \,/\, \sigma$, d'après l'inégalité de Jensen (discrète, resp. continue).

« acte de foi » que N a été choisi suffisamment grand pour que l'intervalle de confiance trouvé soit presque de probabilité $(1-\alpha)$, mais il peut y avoir des cas où cela n'est pas vrai... En fait, la raison qui peut rendre la convergence dans (2.24) particulièrement lente est lorsqu'une partie importante de la variance de f est due à une situation très improbable où f dévie beaucoup de ses valeurs habituelles — ce qu'on pourrait appeler une situation de "queues épaisses". Néanmoins, il n'est pas toujours possible de détecter une telle situation ; et on peut trouver des cas, notamment en situation de sous-échantillonnage (voir remarque 3.17), où tout semble se dérouler selon les hypothèses favorables alors pourtant non seulement le niveau de confiance n'est pas le bon, mais parfois même l'intervalle de confiance calculé est complètement faux [**] !... Il n'existe donc guère d'autre solution que de se fier à son intuition du modèle pour savoir si on se trouve, ou non, dans une situation de queues épaisses.

Remarque 2.28. Concernant la vitesse de convergence, en l'absence de queues épaisses, on pourra retenir la règle empirique suivante : si on souhaite obtenir un intervalle de confiance au risque α, il faut procéder de l'ordre de α^{-2} simulations pour qu'on puisse considérer que le niveau réel de l'intervalle de confiance a à peu près atteint sa valeur asymptotique. Plus précisément, on a le théorème suivant, qui est une variante de l'inégalité de Berry-Esseen (cf. théorème 1.31) :

Théorème 2.29. *Si f a un moment d'ordre* 6 *avec $\mathbb{E}(|f-\mathbb{E}(f)|^6) \leqslant C_6 \operatorname{Var}(f)^3$, alors l'intervalle de confiance au risque α* [††] *que fournira la méthode de Monte-Carlo pour N_{sim} simulations sera incorrect avec probabilité au plus $k\alpha$ (où $k>1$ est choisi à notre guise) dès lors que $N_{\text{sim}} \geqslant A\big(C_6 \,/\, (k-1)\alpha\big)^{-2}$, pour une certaine constante numérique explicite A* [‡‡].

Exemple 2.30 (Le Keno, bis). Reprenons le problème du Keno que nous avons traité tout-à-l'heure (exemple 2.14). Nous avons vu précédemment que les gains moyens par carton s'élevaient à environ 3 €; et dans notre histoire, la loterie avait décidé de vendre les cartons à 4 € pièce. Mais voilà qu'un commercial propose de promouvoir ce jeu en baissant son prix à 3 €... Les dirigeants de la loterie nationale rétorquent que cela reviendrait à vendre le jeu à perte, ce qu'ils ne peuvent pas se permettre, mais le commercial fait valoir que le prix estimé par Monte-Carlo est tellement proche de 3 € qu'on ne peut pas savoir à priori si la véritable valeur est située au-délà ou en deçà ! Il y a donc besoin de reprendre le programme précédent pour quantifier la précision de l'estimateur. L'enjeu financier d'une évaluation trop basse étant potentiellement très important, les dirigeants exigent que le risque de se tromper dans ce sens-là ne soit que de 1 ‰ ; dans l'autre sens, on se contentera d'un risque classique de 5 %.

Le programme est alors le suivant (avec les mêmes fonctions auxiliaires que précédemment) :

```
% La fonction "keno_IC" estime l'espérance de gain d'un joueur de Keno
```

[**]. De même, la méthode proposée à la fin de la remarque 2.26 ne marche pas toujours, car on peut avoir l'impression qu'il y a convergence alors qu'en réalité la divergence est très lente ou très tardive.

[††]. J'écris ici « l' »intervalle de confiance : le résultat est en fait valable (avec la même valeur de A) aussi bien pour l'intervalle symétrique que pour le risque à gauche ou à droite, et plus généralement pour tout intervalle de la forme du théorème 2.23.

[‡‡]. NdA : Énoncé à améliorer et à raffiner...

```
% par la méthode de Monte-Carlo, en donnant un intervalle de confiance.
% Nous ne commentons que les différences par rapport à "keno_simple".
function keno_IC ()
Ncoch = 16;
Nsim = 1000000;
total = 0;
% Pour calculer l'intervalle de confiance on a aussi besoin de calculer le
% total des carrés des quantités simulées.
total_carres = 0;
for i = 1 : Nsim
    carton = 1 : Ncoch;
    % "legain" est le gain obtenu par le joueur. Comme on va l'utiliser
    % deux fois, on doit le stocker dans une variable d'abord.
    legain = gain (score (carton, tirage ()))
    total = total + legain;
    % On met aussi à jour "total_carres".
    total_carres = total_carres + legain * legain;
end
moyenne = total / Nsim;
% On détermine aussi la variance (estimée) du gain.
variance_gain = total_carres / Nsim - moyenne * moyenne;
% On en déduit l'écart-type (estimé) de la moyenne empirique
ect_estimtr = sqrt (variance_gain / Nsim);
% On calcule les coefficients pour l'intervalle de confiance - noter qu'on
% peut aussi faire cette étape au préalable et juste entrer les résultats
% numériquement; en l'occurrence c0 vaut -1.65 et c1 vaut +3.10.
c0 = sqrt (2) * erfinv (-.90);
c1 = sqrt (2) * erfinv (+.998);
% On affiche le résultat.
fprintf ('Avec un risque de 1 ‰ pour la sous-estimation, )
fprintf ('resp. de 5 %% pour la surestimation,\n');
fprintf (L''espérance de gain du Keno se situe dans l'intervalle ');
fprintf ('[%f, %f] €', moyenne + c0 *ect_estimtr, ...
    moyenne + c1 * ect_estimtr);
return
end
```

J'ai obtenu le résultat suivant :

```
>> keno_IC;
Avec un risque de 1 ‰ pour la sous-estimation, resp. 5 % pour la surestimation,
l'espérance de gain du Keno se situe dans l'intervalle [2.981023, 3.047030] €.
```

Le commercial a donc vu juste : on ne peut pas raisonnablement exclure que le gain moyen par carton soit inférieur à 3 € ! Pour plus de précision, il faudrait pousser le temps de calcul ; en demandant 100 fois plus de simulations [*], j'ai obtenu :

[*]. Temps de calcul sur ma machine : une heure !

```
Avec un risque de 1 ‰ pour la sous-estimation, resp. 5 % pour la surestimation,
l'espérance de gain du Keno se situe dans l'intervalle [2.990735, 2.997387] €.
```

Cette fois-ci, le calcul poussé nous permet de conclure ; et nous constatons que le gain moyen par carton est, au-delà de tout doute raisonnable, inférieur à 3 € ! La proposition du commercial est don financièrement tenable [†].

Remarque 2.31. Comme nous l'avons fait remarquer, admettre que le niveau réel de l'intervalle de confiance correspond à peu près à son niveau asymptotique relève souvent de l'« acte de foi »... En l'occurrence, cet acte de choix était-il justifié ? D'abord, dans la mesure où les gains potentiels du joueur sont bornés, il est évident que la fonction dont nous estimons l'espérance est *à fortiori* L^2, et donc que l'intervalle de confiance donnée sera effectivement correct dans l'asymptotique $N_{\text{sim}} \to \infty$. Mais une valeur de $N_{\text{sim}} = 10^6$ est-elle effectivement suffisamment grande ? Cela n'a rien d'évident, dans la mesure où nous sommes dans une situation typique de queues épaisses, des évènements de très faibles probabilité (obtenir un très gros score, disons supérieur ou égal à 67) jouent pour une part importante dans la variance (vu que le gain sera alors de 31 € ou plus, très éloigné des fluctuations typiques du gain). En l'occurrence, il se trouve que la valeur $N_{\text{sim}} = 10^6$ est suffisante, mais de justesse. Une façon de nous rassurer à ce sujet serait de profiter de la grande simulation à 10^8 itérations pour regarder comment fluctue l'écart-type estimé du gain d'un bloc de 10^6 simulations à l'autre. On verrait alors que les etimations sont assez stables, ne variant jamais plus que de quelques pourcents autour de leur valeur typique, ce qui tend à indiquer que la convergence est suffisante à ce niveau [‡]. Un élément qui va dans le même sens est d'ailleurs que l'augmentation d'un facteur 100 du nombre de simulations a conduit a une réduction de la largeur de confiance d'un facteur 9,92, soit pratiquement le facteur $100^{1/2} = 10$ prévu en régime asymptotique.

2.3.2 Notion d'efficacité

Le théorème 2.23 nous dit que, sous réserve que f soit de classe L^2, la largeur de l'intervalle de confiance décroit comme $N^{-1/2}$ quand le nombre de simulations N augmente. Cela est dû au fait que l'estimateur $\hat{m}_N$ tombe à une distance d'ordre $O(N^{-1/2})$ de la véritable valeur $\mathbb{E}(f)$: en effet, la variance de $\hat{m}_N$ décroit en $O(N^{-1})$, puisque d'après la formule d'additivité des variances pour les variables indépendantes, $\text{Var}(\hat{m}_N) = (N^{-1})^2 \times N \,\text{Var}(f) = N^{-1} \,\text{Var}(f)$. De l'autre côté, le cout total du calcul (qui peut correspondre, selon les circonstances, au nombre de simulations, au temps de calcul, à la consommation de ressources de la ferme de calcul, voire au cout monétaire) croît en général linéairement en N. Cela suggère la définition suivante :

[†]. Bien sûr, dans la vraie vie, il y aurait aussi des frais de gestion non négligeables... Ceci n'est qu'un exemple simplifié ☺

[‡]. Il existe toutefois des situations très piégeuses où même cette vérification peut échouer à détecter une mauvaise convergence, typiquement lorsque les queues sont très épaisses, ou lorsqu'un évènement très très rare, qualitativement très différent des autres, est susceptible de changer complètement l'estimation. Mais l'intuition du modèle nous dit que cela n'est guère à craindre ici, car l'échelle des gains fonctionne avec une barême très régulier qui exclut de tels sauts qualitatifs.

Définition 2.32. On appelle *efficacité* d'un calcul de Monte-Carlo l'inverse de la variance (effective) de $\hat{m}$ divisée par le cout de calcul. L'efficacité est essentiellement indépendante du nombre de simulations, et correspond à l'inverse de la variance (effective) de la v.a. f dont on estime l'espérance, multipliée par le nombre de simulations effectuées par unité de coût.

À coût de calcul fixé, la précision du calcul d'une quantité par la méthode de Monte-Carlo sera donc d'autant meilleure que l'efficacité du calcul est grande.

Quand le cout de calcul considéré est le nombre de simulations effectuées, nous parlerons d'« *efficacité par simulation* ». Pour la méthode de Monte-Carlo basique (2.10), cette efficacité par simulation est simplement l'inverse de $\mathrm{Var}(f)$.

Remarque 2.33. Il faut que j'explique l'adjectif « effective » qui apparait entre parenthèses dans la définition de l'efficacité. Dans certaines situations, la quantité γ qu'on cherche à estimer n'est déduite qu'*indirectement* de l'estimateur de Monte-Carlo. Dans un tel cas, sous réserve que l'estimateur de Monte-Carlo sur lequel on s'appuie ait des fluctuations gaussiennes décroissant en $N_{\mathrm{sim}}^{-1/2}$, il en va de même pour l'estimateur $\hat{\gamma}$ de γ qui s'en déduit : il existe une constante $\sigma_\gamma^{\mathrm{eff}} < \infty$ telle que

$$\frac{\hat{\gamma}_{N_{\mathrm{sim}}} - \gamma}{\sigma_\gamma^{\mathrm{eff}} N_{\mathrm{sim}}^{-1/2}} \overset{\mathrm{loi}}{\to} \mathcal{N}(0,1).$$

Le cas échéant, c'est la quantité $\sigma_\gamma^{\mathrm{eff}} N_{\mathrm{sim}}^{-1/2}$ qui est appelée « variance effective » de $\hat{\gamma}_{N_{\mathrm{sim}}}$. Mais il se peut dans certains cas que cela ne corresponde pas du tout à la variance de $\hat{\gamma}$, en particulier lorsque celle-ci est infinie... !

Exemple 2.34. Un petit exemple aide à mieux comprendre la remarque ci-dessus. Imaginons que nous disposions d'une pièce de monnaie biaisée, et qu'on nous demande d'estimer, non pas la probabilité (notée p) de *pile*, mais le ratio $\gamma := p \,/\, (1-p)$ entre la probabilité de *pile* et la probabilité de *face*. En ce cas, si nous réalisons N lancer de la pièce, nous obtenons un estimateur empirique $\hat{p}$ de p pour lequel on a bien $(\hat{p} - p) \,/\, N^{-1/2}\sigma \overset{\mathrm{loi}}{\to} \mathcal{N}(0,1)$ avec $\sigma = \big(p(1-p)\big)^{1/2}$, où $\sigma N^{-1/2}$ est bien l'écart-type de $\hat{p}_N$. De l'estimateur $\hat{p}$ de p nous déduisons l'estimateur $\hat{\gamma}$ de γ, et on déduit de l'asymptotique de $\hat{p}_N$ que

$$\frac{\hat{\gamma}_N - \gamma}{\sigma' N^{-1/2}} \overset{\mathrm{loi}}{\to} \mathcal{N}(0,1),$$

où

$$\sigma' = D\big(p \mapsto p(1-p)\big)(p) \times \sigma = (1-2p)\big(p(1-p)\big)^{1/2}.$$

Pourtant, l'écart-type stricto sensu de $\hat{\gamma}_N$ ne correspond pas du tout à $\sigma' N^{-1/2}$, fût-ce symptotiquement : en effet, du fait que $\hat{\gamma}_N$ a une probabilité non nulle (quoique exponentiellement petite) d'être infini, la variance de $\hat{\gamma}_N$ est toujours infinie... ! Quand nous parlons de variance « effective », nous voulons donc dire que $\hat{\gamma}_N$, asymptotiquement, se comporte *en loi* comme une gaussienne de variance $\sigma' N^{-1/2}$.

Remarque 2.35. Si l'on souhaite évaluer empiriquement l'efficacité d'un calcul, on pourra estimer la variance par la variance empirique (comme nous l'avons fait pour trouver les intervalles de confiance), ce qui fournira un estimateur convergent de l'efficacité.

Remarque 2.36. L'efficacité est une grandeur *dimensionnée* : si, par exemple, la quantité à évaluer se mesure en € et que le cout de calcul considéré est le temps requis (qui se mesure en s), l'efficacité sera en $€^{-2} \cdot s^{-1}$.

2.4 Intérêt et limites de la méthode de Monte-Carlo

2.4.1 Robustesse

Remarque 2.37. On voit ainsi que l'erreur commise par la méthode de Monte-Carlo décroit comme $N^{-1/2}$ (sous réserve que f soit L^2). Il est remarquable que ce mode de convergence ne dépend pas de la structure précise du problème ! En particulier, au vu du paragraphe 2.2.3, cela signifie que grâce à la méthode de Monte-Carlo on pourra calculer (presque) n'importe quelle intégrale avec une précision en $O(N^{-1/2})$ quand N est le nombre de calculs effectués. En comparaison, les méthodes classiques de calcul numérique d'intégrales ont une façon de converger qui se dégrade quand la dimension de l'espace d'intégration augmente ou quand la régularité de la fonction diminue : par exemple, en dimension d la méthode des rectangles avec N points d'évaluation converge en $O(N^{-2/d})$ si la fonction à intégrer est lisse, et en $O(N^{-1/d})$ seulement si la fonction n'est que lipschitzienne... On en conclut que la méthode de Monte-Carlo est particulièrement bien adaptée au calcul d'intégrales :

— Quand la dimension de l'espace d'intégration est grande (typiquement à partir de la dimension 4 ou 5) ;
— Et d'autant plus que la fonction à intégrer est peu régulière.

2.4.2 Parallélisation

Un avantage important de la méthode de Monte-Carlo est sa capacité à la *parallélisation*. On dit qu'un calcul peut être mené de façon *parallèle* lorsqu'il peut être décomposé, pour l'essentiel, en une multitude de sous-calculs *indépendants* : informellement, un calcul est parallélisable lorsque cent unités de calcul (intelligemment coordonnées) peuvent l'effectuer plus rapidement qu'un seul. [§]

Par exemple, simuler l'évolution de la trajectoire d'une bille dans un potentiel est une tâche typiquement *non* parallélisable, puisqu'on est obligé de calculer successivement la position de la bille aux différents moments... À l'inverse, la méthode de Monte-Carlo, avec ses simulations indépendantes, se parallélise très bien. Supposons ainsi qu'on dispose de 100 unités de calculs pour effectuer 1 000 000 de simulations. Si chaque unité de calcul effectue indépendamment [¶] 10 000 simulations et calcule la moyenne

[§]. Attention : un calcul parallélisable peut être rendu très rapide si on dispose de suffisamment d'unités de calcul, mais cela n'est avantageux que si le facteur limitant est le temps ; à l'inverse, si le facteur limitant est la consommation de chaque unité de calcul, on ne gagne rien à diviser le calcul en cent sous-calculs, voire on y perd en fonction du type de découpage.

[¶]. Attention, il faut bien veiller à ce que chaque unité de calcul effectue les simulations avec des aléas *indépendants* ! En particulier, si la méthode de Monte-Carlo recourt à des nombres *pseudo*-aléatoires (comme c'est généralement le cas), il faut s'assurer que les différentes unités de calcul utilisent pour leurs générateurs pseudo-aléatoires des graines qui ne risquent pas d'interférer.

empirique obtenue pour ses propres simulations, il n'y a à la fin plus qu'à mettre les 100 moyennes partielles en commun pour obtenir la moyenne globale.

Remarque 2.38. Noter que le type de parallélisation concernant la méthode de Monte-Carlo est particulièrement commode, car il n'y a besoin d'essentiellement *aucune* coordination entre les unités de calcul : ce n'est qu'à la fin que les cerveaux mettront en commun leur différents résultats partiels ! D'autre types de parallélisation ne présentent pas cet avantage. Par exemple, si on souhaite modéliser l'évolution des vagues à la surface de la mer, on peut découper la mer en petites parcelles et attribuer à chauqe unité de calcul la simulation de l'évolution d'une parcelle (car deux parcelles éloignées n'interfèrent pas sur de courtes échelles de temps), mais il faudra régulièrement faire communiquer les différentes unités de calcul entre elles pour tenir compte de l'évolution des conditions aux bords due à l'interaction avec les parcelles voisines.

Remarque 2.39. Il existe un type particulier de parallélisation appelé *vectorisation*, où la parallélisation de fait au sein même d'une unité intégrée (par exemple un carte graphique), sous réserve que tous les calculs à faire séparément soient les mêmes (mais avec des données différentes bien sûr) et ne soient pas trop compliqués. Toutefois, nous n'étudierons pas cette méthode ici, car elle demande de tenir compte des spécificités de l'unité de parallélisation (avec codage dans un langage de programmation spécifique ou en assembleur).

À noter que, bien que des logiciels comme *Scilab* et MATLAB soient « spécialisés dans le traitement des calculs vectoriels », ils n'utilisent pas la vectorisation (tout au plus le pipeline du microprocesseur), sauf ajout d'un module complémentaire spécifique.

2.4.3 Raffinement

Si un premier calcul de Monte-Carlo a été effectué (disons avec 200 000 simuations) et que sa précision s'avère insuffisante (disons qu'il aurait fallu 500 000 simulations), la méthode de Monte-Carlo présente cet avantage considérable qu'on peut s'appuyer sur les calculs déjà effectués comme une première étape du calcul raffiné ! En effet, si on a stocké quelque part les résultats relatifs à nos 200 000 première simulations, il n'y aura plus que 300 000 simulations à faire indépendamment pour arriver au total désiré de 500 000.

En outre, il est particulièrement remarquable qu'on n'a pas réellement besoin de connaitre le résultat détaillé des 200 000 premières simulations, mais seulement d'un résultat intermédiaire très concis, à savoir le nombre de simulations effectuées, le total des résultats et le total des carrés ! Pour cette raison, il ne faut jamais hésiter à stocker ces résultats d'une méthode de Monte-Carlo, quand bien même on pense avoir peu de chances de souhaiter s'en resservir.

Remarque 2.40. D'une certaine manière, on peut dire que le raffinement est une forme de vectorisation *dans le temps*...

Remarque 2.41. Précisons toutefois qu'un élément qui permet cette forme de raffinement sans repartir de zéro est que les calculs en virgule flottant de notre ordinateur seront normalement de précision largement supérieure à celle que nous pensons obtenir au final avec notre méthode de Monte-Carlo. Si on souhaitait raffiner un premier calcul pour aller au-delà de la précision de la machine, alors cela ne marcherait pas, car les

résultats partiels étant obtenus avec une précision insuffisante, on ne pourrait pas les intégrer au calcul du résultat final ! De même, on ne peut pas raffiner un calcul de série fait pour obtenir 15 décimales de e en rajoutant des termes jusqu'à une précision de 40 décimales, si le premier calcul n'a pas été effectué avec une précision de 40 décimales (quand bien même on sait que les décimales calculées par le premier calcul au-delà de la 15^{e} ne sont pas celles de e, on a besoin de les connaitre pour la suite !).

Cette remarque vaut également pour la parallélisation.

2.4.4 Précision

À l'inverse, la méthode de Monte-Carlo présente également des handicaps non négligeables. Par exemple, si nous souhaitons utiliser la méthode de Monte-Carlo pour un calcul de π (ce qui est tout-à-fait possible), la précision du calcul pour un cout de calcul T sera en $T^{-1/2}$, soit $O(\log T)$ décimales exactes. Alors que les méthodes non probabilistes permettent d'obtenir assez facilement un nombre de décimales exactes en $O(T^{1/2})$ voire en $O(T^{1-\varepsilon})$... Autant dire que si nous nous étions limités à la méthode de Monte-Carlo, même au prix d'efforts démesurés nous ne connaitrions même pas une vingtaine de décimales exactes, à comparer aux milliards que nous avons aujourd'hui...

Cela dit, pour l'ingénieur il est très (très !) rare d'avoir besoin de descendre à des niveaux de précisions aussi exigeants ; cet écueil ne devrait donc guère vous réfréner d'employer la méthode de Monte-Carlo pour les applications industrielles. Il n'en reste pas moins que parfois la relative lenteur de la méthode de Monte-Carlo fait que d'autres méthodes de calcul concurrentes s'avèrent beaucoup plus appropriées. [|||]

2.4.5 Le caractère aléatoire, un problème ?

Le premier reproche qu'on a envie de faire à la méthode de Monte-Carlo est qu'elle ne permet pas d'avoir une certitude absolue sur la validité d'un résultat. Cette limitation ne doit pas vous effrayer, à moins que vous soyez mathématiciens théoriciens ! En effet, du point de vue *pratique*, une probabilité inférieure à 10^{-10} est totalement négligeable (c'est à peu près la probabilité que vous mouriez pendant cette séance de cours d'une rupture d'un anévrisme non détecté, et bien moins que la probabilité que votre patron vienne d'être arrêté pour meurtre sans que vous le sachiez !).

Remarque 2.42. On serait tenté de rétorquer que la convergence vers la loi normale est souvent trop lente pour qu'on puisse supposer que l'approximation par la gaussienne soit valable à 10^{-10} près (cf. la remarque 2.28). Cela n'empêche pas pour autant d'obtenir facilement des intervalles de confiance à une telle probabilité : en effet, si par exemple on lance 34 simulations indépendantes qui ont chacune au moins 50 % de chances d'être valides, il n'y aura qu'une chance sur 10^{10} que le résultat ne soit dans aucun des intervalles de confiance trouvés ! Certes, cela donnera (à cout de calcul égal) un intervalle

[|||]. D'ailleurs, n'oubliez pas que quand vous cherchez à estimer une valeur d'intérêt dans le cadre d'un modèle (par exemple le juste prix d'une option dans le cadre des marchés financiers, ce qui est une application classique de la méthode de Monte-Carlo), le modèle *lui-même* n'est généralement pas exact ou pas parfaitement calibré ! Il est donc absurde est dangereux de donner une précision supérieure à quelques décimales dans ce cas-là, en vertu du fameux adage « garbage in, garbage out ».

de confiance nettement moins fin que si on avait utilisé l'approximation gaussienne, mais au moins cela règle la question d'un niveau de confiance extrême.

Remarque 2.43. Bien entendu, cela ne règle pas le plus gros danger, à savoir que le niveau réel de l'intervalle de confiance n'est peut-être pas celui calculé d'après l'asymptotique. Nous avons évoqués dans la remarque 2.27 des méthodes pour essayer de distinguer les situations dangereuses, problème que nous allons rappeler dans la sous-section suivante.

2.4.6 Caractère asymptotique des intervalles de confiance

C'est sans doute le pire piège auquel vous serez confrontés pour appliquer la méthode de Monte-Carlo. On a tendance à oublier que les intervalles de confiance donnés n'ont le niveau de précision annoncé que *sous l'hypothèse que la fonction dont on évalue l'espérance soit* L^2, et que même sous cette hypothèse ils ne sont valables qu'*asymptotiquement* ! !

Or, dans bien des cas, on n'est pas en mesure d'être capable de prouver des intervalles de confiance non asymptotiques pour les données qu'on calcule (voire parfois on ne sait même pas s'assurer qu'elles sont L^2). Cela pose un vrai problème, et certains des exercices associés à ce cours montrent comment on peut être piégé et obtenir des intervalles de confiances faux sans qu'aucun signe évident nous en ait averti ! C'est donc *le* point auquel il convient d'être prudent.

Il existe des moyens d'obtenir des intervalles de confiance non asymptotiques, mais pour cela il faut être capable d'obtenir un contrôle *explicite* sur la loi de probabilité. Une méthode naïve, mais déjà relativement efficace, est d'utiliser la connaissance d'une borne sur un moment polynomial (cf. théorème 2.29) ; et on peut atteindre un résultat non asymptotique de même qualité que la gaussienne dès lors qu'on sait contrôler explicitement un moment exponentiel de la fonction (càd. qu'on a une borne sur $\mathbb{E}(e^{\gamma|X|})$ pour un certain $\gamma > 0$, où X est la v.a. réelle dont on estime l'espérance). Notamment, il existe des bornes simples et efficaces dès lors que la fonction que nous simulons est explicitement *bornée*. Nous n'aborderons pas plus en détail cette question ici, bien qu'elle tout-à-fait intéressante ; mais certains des exercices associés au cours donnent des idées sur le sujet.

2.5 Vers la réduction de la variance

Nous avons vu dans la § 2.3.2 que le cout d'une calcul de Monte-Carlo était à la précision souhaitée (en appelant « précision » l'inverse de la variance effective de l'estimateur), avec une constante de proportionnalité que nous avons appelée *efficacité*. En pratique, on recontre de nombreuses situations où les circonstances du problème font que le cout de calcul devient une véritable contrainte, et on cherche donc à limiter le cout de ce calcul. Mais comment ? Ce que nous allons voir dans le prochain chapitre, c'est que certaines manipulations algébriques permettent de ramener un calcul d'efficacité par méthode de Monte-Carlo à un *autre* calcul de Monte-Carlo, dont le *résultat* est le même, mais dont l'*efficacité* sera différente (et, nous l'espérons, meilleure dans les cas qui nous intéressant !).

Chapitre 3

Techniques de réduction de la variance

3.1 Généralités

3.1.1 Qu'est-ce que la réduction de la variance ?

Comme nous l'avons dit à la fin du chapitre précédent, les *techniques de réduction de la variance* se proposent, à l'aide de certaines manipulations algériques, de ramener un calcul par Monte-Carlo à un autre calcul d'efficacité meilleure. L'appellation de "réduction de la variance" vient de ce que, dans nombre des méthodes exposées ci-dessous, on essaye d'abord d'améliorer l'efficacité *par simulation*, laquelle est précisément égale à l'inverse de la variance de la quantité simulée : améliorer l'efficacité par simulation, c'est donc réduire la variance.

Remarque 3.1. Cela dit, il convient de garder à l'esprit que l'amélioration de l'efficacité *par simulation* ne se traduit pas nécessairement par l'amélioration de l'efficacité *tout court*, car la complexification de l'algorithme peut conduire à une augmentation du cout de calcul par simulation.

3.1.2 Le niveau zéro de la réduction de la variance : améliorer son code !

Les techniques de réduction de la variance qui vont être présentées dans ce chapitre fonctionnent au niveau *algorithmique* : pour améliorer l'efficacité, on change fondamentalement la *nature* du calcul effectué. Mais la première façon d'améliorer l'efficacité est d'abord d'optimiser son code de calcul ! En effet, pour faire calculer la même chose à votre ordinateur, l'ordre ou l'organisation des calculs peut lui demander beaucoup plus d'efforts...

Donc, utiliser des techniques de réduction de la variance, c'est très bien, mais si on ne conçoit pas bien son code, on ruine bêtement ses efforts ! Un code mal conçu peut faire perdre un facteur 2 voire 10 dans la vitesse d'exécution, et même, s'il est très mal conçu, rendre le programme totalement inopérant en pratique. Surtout, cela n'est pas valable que pour la méthode de Monte-Carlo, mais pour *toute* implémentation informatique !

Exemple 3.2. À titre d'exemple, voyons comment nous pourrions améliorer notre code du Keno. Une première remarque qui vient à l'esprit est que l'appel à la fonction `gain`, qui donne le gain financier obtenu à partir d'un certain score, n'est pas utilisée de façon très intelligente... En effet, à chaque fois que nous avons à calculer un gain, nous allons refaire toute la boucle de calcul afférente au calcul de ce gain, alors qu'il est bien évident que nous pourrions calculer une fois pour toutes les différentes valeurs de gains associées aux différents scores (qui ne sont qu'en nombre fini et petit), et nous contenter de *lire* ensuite ces résultats stockés dans un tableau ! On obtient ainsi le programme suivant (voir la fonction `calcul_gains.m` en annexe :

```
% La fonction "keno_v2" fait le même travail que "keno_IC", mais avec un
% code amélioré au niveau du calcul des gains, qu'on pré-réalise avant la
% boucle.
function keno_v2 ()
Nsim = 1000000;
Ncoch = 16;
total = 0;
total_carres = 0;
% "tableau_gains(s + 1)" sera la valeur du gain pour un score de s. On le
% calcule une fois pour toutes à l'aide de la fonction auxiliaire
% "calcul_gains".
tableau_gains = calcul_gains ();
for i = 1 : Nsim
    carton = 1 : Ncoch;
    % "legain" est le gain obtenu par le joueur. Comme on va l'utiliser
    % deux fois, on doit le stocker dans une variable d'abord.
    legain = tableau_gains(score (carton, tirage ()) + 1);
    total = total + legain;
    total_carres = total_carres + legain * legain;
end
moyenne = total / Nsim;
variance_gain = total_carres / Nsim - moyenne * moyenne;
ect_estimtr = sqrt (variance_gain / Nsim);
c0 = sqrt (2) * erfinv (-.90);
c1 = sqrt (2) * erfinv (+.998);
fprintf ('Avec un risque de 1 ‰ pour la sous-estimation, ')
fprintf ('resp. de 5 %% pour la surestimation,\n');
fprintf ('l''espérance de gain du Keno se situe dans l''intervalle ');
fprintf ('[%f, %f] €.\n', moyenne + c0 *ect_estimtr, ...
    moyenne + c1 * ect_estimtr);
return
end
```

Sur ma machine, l'exécution du programme passe d'environ 22,4 secondes à 16,6 secondes seulement. Bien entendu, le résultat est exactement le même (sous réserve d'avoir pris la même graine pour le générateur pseudo-aléatoire), vu que les calculs relatifs à la simulation sont identiques.

Mais on peut faire encore mieux ! Remarquons en effet que nous simulons l'ensemble des numéros tirés du carton, alors que tout ce qui nous intéresse est de savoir combien

de numéros sont tirés parmi ceux de '1' à '16' (rappelons en effet qu'on peut joueur toujours la même grille, en cochant '1' pour le numéro '1', etc.). On peut choisir de regarder successivement les numéros de '1' à '16' et voir, conditionnellement à ceux qui ont été tirés ou non parmi les précédents, s'ils vont être tirée ou non. Cela conduit à une fonction `tirage_reduit` qui renvoie un 16-vecteur valant '`true`' en sa case i quand le numéro coché pour i a été tiré et '`false`' sinon. Le programme obtenu étant alors :

```
% La fonction "keno_v3" fait le même travail que "keno_v2", mais avec un
% code encore amélioré, où on ne simule les numéros tirés que parmi ceux
% qui sont cochés.
function keno_v3 ()
Nsim = 1000000;
Ncoch = 16;
total = 0;
total_carres = 0;
tableau_gains = calcul_gains ();
for i = 1 : Nsim
    % La fonction "tirage_reduit" donne quels numéros ont été tirés parmi
    % ceux qu'on a cochés; on stocke son résultat dans "letirage".
    letirage = tirage_reduit ();
    % "lescore" est le score obtenu, qui se déduit facilement de letirage.
    lescore = sum ((1:Ncoch) .* letirage);
    legain = tableau_gains(lescore + 1);
    total = total + legain;
    total_carres = total_carres + legain * legain;
end
moyenne = total / Nsim;
variance_gain = total_carres / Nsim - moyenne * moyenne;
ect_estimtr = sqrt (variance_gain / Nsim);
c0 = sqrt (2) * erfinv (-.90);
c1 = sqrt (2) * erfinv (+.998);
fprintf ('Avec un risque de 1 ‰ pour la sous-estimation, ')
fprintf ('resp. de 5 %% pour la surestimation,\n');
fprintf ('l''espérance de gain du Keno se situe dans l''intervalle ');
fprintf ('[%f, %f] €.\n', moyenne + c0 *ect_estimtr, ...
    moyenne + c1 * ect_estimtr);
return
end
```

Le temps de calcul descend alors à 12,8 secondes... Noter que cette fois, les résultats obtenus ne sont pas strictement identiques, même en prenant une graine peudo-aléatoire identique, car la simulation utilise lestirages aléatoires de façon différente ; par contre, la précision du résultat est bien la même (l'intervalle de confiance a la même largeur), dans la mesure où les opérations effectuées sont identiques *en substance* (on simule des tirages et on regarde le gain moyen).

On pourrait encore faire quelques améliorations à la marge, mais je voudrais en montrer une dernière plus radicale : changer de langage de programmation ! Le langage de Matlab que j'ai utilisé jusque-là est en effet un langage de haut niveau (je ne contrôle

donc pas très bien les claculs effectivement faits par le processeur), interprété (le logiciel ne découvre les lignes du programme qu'au moment de l'exécution, et ne peut donc pas préparer d'optimisation en fonction de la structure globale du programme), et très faiblement type (à chaque fois que Matlab rencontre un objet, il doit savoir s'il va le traiter comme une matrice, un entier, un réel, un complexe, ...). Tout cela perd beaucoup de temps à l'exécution. Un langage de plus bas niveau, compilé, au langage moins ambigu devrait donc être susceptible de faire beaucoup mieux... Essayons donc avec le roi des langages de bas niveau, à savoir le langage C... Le temps de calcul est alors de... 0,5 secondes ! Autant dire que, si notre préoccupation était vraiment la rapidité, on aurait mieux fait de tout de suite commencer par là...

Remarque 3.3. Certaines optimisation de code peuvent avoir un résultat pratiquement nul, parce qu'on a amélioré une partie du code qui ne prenait qu'une toute partie du temps total. Il faut un peu d'expérience de la programmation pour savoir quelles sont les opérations les plus lourdes pour le processeur, et à quels endroits un code non optimisé risque donc de nous couter vraiment cher... Bien sûr, le bon sens aide aussi : il est ainsi évident qu'il y a énormément plus d'intérêt à optimiser une opération répétée un très grand nombre de fois à l'intérieur d'une boucle, plutôt qu'une opération effectuée une fois pour toutes au début du programme !

Remarque 3.4. Certains logiciels ou certains langages ont leurs spécificités qu'il est bon de connaitre pour accélérer l'exécution. Ainsi, sous Matlab, les opérations portant sur les matrices sont pré-compilées pour être particulièrement rapides à l'exécution, de sorte qu'il est en général beaucoup plus efficace d'utiliser une fonction Matlab toute faite plutôt que de passer par une boucle pour faire le même calcul case par case... Ces considérations sont surtout intéressantes lorsqu'on est spécialisé dans un langage donné. Dans la mesure où l'objectif de ce cours est de présenter les méthodes de Monte-Carlo de manière générale, sans mettre en avant un langage particulier, nous n'exploiterons dans la suite que les optimisations de code reposant sur des considérations informatiques générales, sans faire attention aux spécificités propres à Matlab.

3.1.3 Les techniques de réduction de la variance peuvent se combiner

La suite de ce chapitre va présenter les principales techniques de réduction de la variance indépendamment, chacune à part. Mais vous ne devez pas perdre de vue que rien n'interdit de combiner plusieurs techniques de réduction de la variance successivement pour une efficacité encore meilleure ! Nous en verrons des exemples dans les exercices.

3.1.4 La liste des techniques de réduction de la variance n'est pas close

Les techniques de réduction de la variance que nous allons présenter recouvrent la très grande majorité de celles que vous allez rencontrer en pratique. Mais gardez à l'esprit que ce n'est pas une liste close : n'importe quelle manipulation, si elle améliore l'efficacité, est une réduction de la variance, qu'elle fasse partie de ce cours ou non.

3.2 Échantillonnage préférentiel

3.2.1 Principe de l'échantillonnage préférentiel

Théorème 3.5. *Soit P une mesure de probabilité sur Ω et f une variable aléatoire réelle intégrable sous P. Soit Q une autre mesure de probabilité sur Ω telle que P soit à densité par rapport à Q ; alors :*

$$\mathbb{E}_P(f) = \mathbb{E}_Q\big((dP/dQ) \times f\big).$$

Démonstration. Cela résulte immédiatement de la définition de la densité. □

Remarque 3.6. En réalité, l'espérance de f n'est pas forcément une espérance sur Ω tout entier, mais seulement sur la partie où f est non nulle ; par conséquent, on peut dans le théorème 3.5 n'exiger que P n'ait une densité par rapport à Q que sur $\{\omega \in \Omega \mid f(\omega) \neq 0\}$.

Définition 3.7. Lorsque, pour estimer $\mathbb{E}_P(f)$ par la méthode de Monte-Carlo, on emploie le théorème 3.5 afin de se ramener à une méthode de Monte-Carlo où on fait les tirages selon la loi Q, on dit qu'on utilie la méthode d'*échantillonnage préférentiel*. La loi Q est alors appelée *loi d'échantillonnage*.

Remarque 3.8. Pour pouvoir appliquer la méthode d'échantillonnage préférentiel, il faut : (outre la nécessité de savoir calculer f),

— Qu'on sache simuler la loi Q ;
— Qu'on sache calculer la densité dP/dQ.

Remarque 3.9. Le théorème 3.5 ressemble beaucoup au théorème 2.16... En fait, c'est exactement le même ! La seule "différence" provient de ce que, lorsque nous cherchions à évaluer une intégrale de Monte-Carlo, nous n'avions pas de mesure de probabilité *canonique* pour écrire l'intégrale comme une espérance, de sorte que la question portait sur le *choix* d'une mesure de probabilité ; alors que dans l'énoncé ci-dessus, nous disposons *déjà* d'une mesure de probabilité naturelle, de sorte que la méthode d'échantillonnage préférentiel d'apparente plutôt à un *changement* de loi de probabilité.

Nous verrons dans la suite de cette section que l'échantillonnage préférentiel est pertinent lorsque la loi Q donne plus de poids aux ω qui contribuent le plus à l'espérance de f. Mais avant cela, voyons tout de suite deux exemples pour illustrer ce concept essentiel :

Exemple 3.10 (La centrale nucléaire). Une ingénieure doit certifier le risque qu'une centrale nucléaire subisse un accident majeur à l'occasion d'une certaine opération de maintenance quotidienne. Après modélisation, elle en arrive à l'observation que trois facteurs sont impliqués dans un tel risque, facteurs qu'elle aprelle respectivement X, Y et Z, et que ces trois vecteurs sont distribués selon la loi normale

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \sim \mathcal{N}(\vec{m}, \boldsymbol{C}).$$

(voir la §A.6.1 pour les détails techniques). L'accident se produit quand les trois facteurs X, Y et Z deviennent *tous les trois* positifs ; il faut donc évaluer $\mathbb{P}\big((X,Y,Z)^\mathsf{T} \in \mathbf{R}_+^3\big) =: p$. La norme de certification impose que p soit inférieur à 10^{-10}.

L'ingénieure décide d'utiliser la méthode de Monte-Carlo pour évaluer p. Dans un premier temps, elle écrit un programme "naïf" dont vous trouverez le code dans l'annexe A.6.2. Avec 10^7 simulations, elle obtient le résultat suivant :

```
>> centrale_naif (10000000)
Probabilité d'accident : 0 ± 0
```

À première vue, l'affaire semble entendue : il n'y aurait strictement aucun risque. « Mais en fait, réfléchit l'ingénieure, ce résultat est dégénéré : il signifie simplement que toutes les simulations ont été négatives... Or, si le risque véritable est de 10^{-8}, ce n'est pas en faisant seulement 10^7 simulations qu'on va pouvoir observer quelque chose !... ». L'ennui, c'est que l'ordinateur poussif de notre ingénieure ne permet guère de dépasser 10^7 simulations... D'où l'idée d'utiliser une technique d'échantillonnage préférentiel pour obtenir un résultat significatif.

L'ingénieure choisit d'utiliser comme loi d'échantillonnage préférentiel la loi $Q := \mathcal{N}(\vec{0}, \boldsymbol{C})$ correspondant à un vecteur gaussien de même variance, mais centré au point $(0,0,0)^\mathsf{T}$. La véritable loi P du vecteur gaussien a bien une densité par rapport à Q, qui est effectivement calculable, comme le montre la §A.6.1. On arrive alors au code suivant (voir les fonctions auxiliaires en annexe) :

```
% La fonction "centrale_pref" applique la méthode d'échantillonnage
% préférentiel pour calculer le risque d'accident.
function centrale_pref (Nsim)
global m C R Cinv;
m = [-3 -4 -5];
C = [1, .1, .2; .1, 1, .3; .2, .3, 1];
R = chol(C);
% "Cinv" est la matrice inverse de C, utilisée pour le calcul de la
% densité.
Cinv = inv (C);
% La variable dont on estime l'espérance n'étant plus une indicatrice, il
% faut maintenir séparément la somme et la somme des carrés.
somme_f = 0;
somme_f2 = 0;
for i = 1: Nsim
    % Cette fois-ci on échantillonne XYZ selon la loi Q.
    XYZ = tirer_Q ();
    % La fonction dont on évalue l'espérance doit tenir compte de la
    % densité.
    f = all (XYZ >= 0) * densitePQ (XYZ);
    somme_f = somme_f + f;
    somme_f2 = somme_f2 + f * f;
end
moy = somme_f / Nsim;
var_f = somme_f2 / Nsim - moy * moy;
```

```
ect_estr = sqrt (var_f / Nsim);
fprintf ('La probabilité d''accident est estimée à ');
fprintf ('%e ± %e.\n', moy, 1.96 * ect_estr);
fprintf ('(Note : Intervalle de confiance standard à 1,96 sigmas).\n');
return
end
```

L'exécution donne :

```
centrale_pref(5000000)
La probabilité d'accident est estimée à 4.586479e-11 ± 3.184524e-13.
(Note : Intervalle de confiance standard à 1,96 sigmas).
```

Cette fois-ci le résultat n'est plus dégénéré, ce qui nous rassure sur la fiabilité de l'intervalle de confiance. On trouve que les accidents arriveront avec une probabilité d'environ 5×10^{-11}, ce qui permet donc de certifier la centrale.

Remarque 3.11. On remarquera que la complexité accrue des calculs a rendu le temps de calcul simulation environ deux fois plus lent dans la version préférentielle que dans la version naïve. De ce fait, nous n'avons pu procéder qu'à deux fois moins de simulations ; mais cela n'empêche pas le résultat d'être incomparablement meilleur ☺

Remarque 3.12. Il est essentiel de bien comprendre que la méthode d'échantillonnage préférentiel est une méthode d'échantillonnage *sous* Q, *pour la fonction « fonction d'intérêt multipliée par densité »*. Ainsi, si nous notons f la fonction d'intérêt et ρ la densité dP/dQ, la variance qui doit intervenir dans le calcul de l'intervalle de confiance est $\mathrm{Var}_Q(\rho f)$, et non pas $\mathrm{Var}_Q(f)$ ou $\mathrm{Var}_P(f)$... !

Exemple 3.13. Le concept d'échantillonnage préférentiel étant particulièrement important, nous allons en donner un second exemple, cette fois-ci dans un cadre discret. — Lorsque les probabilités P et Q portent sur un espace discret, $\big(dP/dQ\big)(\omega)$ est simplement $P(\{\omega\}) \,/\, Q(\{\omega\})$. Nous considérons à nouveau la problème du championnat de basket, mais cette fois-ci nous supposons que l'équipe de Nancy est la n° 14 et a donc très peu de chances de remporter le championnat. Dans ces conditions, l'échantillonnage selon la loi de probabilité "naturelle" (notée P) n'est sans doute pas le plus adapté, car on tombe très rarement sur des victoires finales de Nancy... Nous proposons alors la loi d'échantillonnage préférentiel suivant, notée Q :

> Sous la probabilité Q, les matchs n'impliquant pas Nancy se déroulent normalement, mais les matchs impliquant Nancy ont leur résultat tiré au sort *comme si Nancy jouait avec le niveau de l'équipe n° 1.* (À part cela, les résultats des différents matchs sont toujours indépendants).

La densité de P par rapport à Q peut être, là encore, calculée explicitement : l'annexe A.1.6 montre comment.

On obtient alors le code suivant :

```
% La fonction "basket_pref" estime la probabilité de victoire de l'équipe
% de Nancy, qui porte ici le numéro 14, par la méthode de rejet (pour la
% loi d'échantillonnage préférentiel définie dans le polycopié). "Nsim"
```

```
% est le nombre de simulations demandées. Je ne commente que les
% différences avec le code de basket_14.
function basket_pref ()
Nsim = 10000;
NANCY = 14;
% La probabilité que nous cherchons ne s'écrivant plus comme l'espérance
% d'une indicatrice, nous devons maintenir séparément la somme et la
% somme des carrés.
somme_f = 0;
somme_f2 = 0;
for i = 1 : Nsim
    % La fonction "championne_Q" à laquelle nous allons faire appel
    % fonctionne comme "championne", mais a deux différences importantes
    % près: 1. La loi de simulation est la loi d'échantillonnage Q; 2. La
    % fonction renvoie non seulement qui est la championne, mais aussi
    % quelle est la densité dP/dQ au point simulé. On stocke
    % respectivement ces deux valeurs de retour dans "ch" et "densite".
    [ch, densite] = championne_Q ();
    % Nous appelons "f" la réalisation de la fonction dont on va prendre
    % l'espérance.
    f = (ch == NANCY) * densite;
    somme_f = somme_f + f;
    somme_f2 = somme_f2 + f * f;
end
moy = somme_f / Nsim;
var_f = somme_f2 / Nsim - moy * moy;
ect_estr = sqrt (var_f / Nsim);
fprintf ('La probabilité de victoire hde Nancy est ');
fprintf ('%f ± %f %%.\n', 100 * moy, 1.96 * 100 * ect_estr);
return
end
```

À l'exécution :

```
tic; basket_pref; toc
La probabilité de victoire de Nancy est 0.481892 ± 0.029634 %.
Elapsed time is 51.449873 seconds.
```

Si nous comparons avec le code naïf correspondant (voir §A.1.3) :

```
tic; basket_14; toc;
La probabilité de victoire de Nancy est 0.510000 ± 0.056997 %.
Elapsed time is 67.326017 seconds.
```

Là encore, si le nombre de simulations exécutées en un laps de temps donné est plus faible, le gain d'efficacité par simulation est suffisant pour qu'on gagne un facteur environ sur l'efficacité globale.

Remarque 3.14. Il est particulièrement flagrant sur cet exemple que les intervalles de confiance donnés avec ou sans échantillonnage préférentiel sont tout-à-fait compatibles. Cela n'a rien qui doive nous surprendre, dans la mesure où il s'agit de deux méthodes différentes pour calculer la *même* quantité. Pour triviale qu'elle soit, cette remarque est néanmoins utile pour détecter d'éventuels problèmes, soit algorithmiques (erreur dans le calcul de la densité, ...), soit pratique (sous-échantillonnage, ...).

3.2.2 Recherche d'une bonne loi d'échantillonnage préférentiel

La technique d'échantillonnage préférentiel nous laisse le choix de la probabilité d'échantillonnage Q — sous réserve évidemment que nous sachions simuler Q et calculer dQ/dP. Puisque la technique d'échantillonnage repose sur l'espoir qu'échantillonner selon Q sera plus efficace qu'échantillonner selon P, c'est qu'on s'attend à ce que certaines Q soient meilleures que d'autres... Mais lesquelles ? Y a-t-il des heuristiques pour trouver quelles sont les meilleures lois d'échantillonnage Q ? C'est à cette problématique que cette sous-section se propose de répondre.

De manière générale, l'efficacité d'une simulation dépend non seulement de la variance de la quantité dont on calcule l'espérance, mais aussi du cout de calcul par simulation. Cependant, ce dernier est difficile à modéliser mathématiquement, sans compter qu'il est susceptible de dépendre des détails de l'implémentation et même de la machine utilisée... Nous nous limiterons donc ici à essayer d'optimiser l'efficacité *par simulation*, dans l'espoir que cela aidera aussi à améliorer l'efficacité tout court.

Proposition 3.15. *L'efficacité par simulation de la technique d'échantillonnage préférentiel est égale à l'inverse de* $\big(\mathbb{E}_P(dP/dQ \times f^2) - \mathbb{E}_P(f)^2\big)$.

Démonstration. L'efficacité par simulation est par définition l'inverse de la variance $\mathrm{Var}_Q(dP/dQ \times f)$, qui est égale à $\mathbb{E}_Q\big((dP/dQ)^2 \times f^2\big) - \mathbb{E}_Q(dP/dQ \times f)^2 = \mathbb{E}_P(dP/dQ \times f^2) - \mathbb{E}_P(f)^2$ d'après la propriété caractéristique de la densité dP/dQ. □

Théorème 3.16. *L'efficacité par simulation par la méthode d'échantillonnage préférentiel est maximale quand la densité* dQ/dP de Q par rapport à P *est proportionnelle à* $|f|$.

Démonstration. Notons $\rho := dQ/dP = (dP/dQ)^{-1}$ (on fera comme si les mesures P et Q étaient équivalentes, ce qui justifie cette expression). D'après la proposition 3.15, notre objectif alors est de minimiser $\mathbb{E}_P(\rho^{-1} f^2)$, et ce sous la contrainte que Q soit une mesure de probabilité, càd. que $\rho \geqslant 0$ et $\mathbb{E}_P(\rho) = 1$. Faisons ici comme si f était non nulle presque-partout ; alors il est clair le ρ optimal vérifiera $\rho^* > 0$ presque-partout, car sinon on aurait $\mathbb{E}_P((\rho^*)^{-1} f^2) = +\infty$; de sorte que nous n'avons qu'à considérer la contrainte $\mathbb{E}_P(\rho) = 1$. On applique la méthode du multiplicateur de Lagrange : pour un paramètre λ à fixer, on cherche un ρ^* vérifiant $\rho^* \geqslant 0$ et $\mathbb{E}_P(\rho^*) = 1$ tel que ρ^* soit un point critique de la fonctionnelle $\rho \mapsto \mathbb{E}_P(\rho^{-1} f^2) + \lambda\, \mathbb{E}_P(\rho) =: F(\rho)$. On calcule que $\big(DF(\rho)\big) \cdot h = \mathbb{E}_P\big((-\rho^{-2} f^2 + \lambda) h\big)$, de sorte qu'un point critique ρ^* doit vérifier $-(\rho^*)^{-2} f^2 + \lambda \equiv 0$, d'où $\rho^* \propto |f|$. □

En pratique, la probabilité Q optimale, même si elle est simulable, ne peut pas être utilisée car on ne sait pas calculer exactement dQ/dP. Ce qu'il faut retenir est

plutôt qu'on doit privilégier une loi d'échantillonnage Q dont la densité par rapport à P soit « aussi proportionnelle que possible » à $|f|$. En particulier, s'il existe un ensemble restreint d'éventualités ω pour lesquelles f prend une valeur particulièrement grande, il faudra cercher une loi de simulation Q qui donne plus de poids à ces éventualités-là !

Remarque 3.17. Il y a deux façons de mal "coller" à la probabilité d'échantillonnage optimale : la première est le *suréchantillonnage*, qui consiste à simuler beaucoup trop souvent des zones où la valeur de $|f|$ est plutôt petite ; la seconde, à l'inverse, est le *sous-échantillonnage*, qui consiste à simuler beaucoup trop rarement des zones où la valeur de $|f|$ est plutôt grande. Ces deux défauts n'ont pas du tout la même gravité : *le sous-échantillonnage est beaucoup plus grave que le suréchantillonnage !* On le voit par exemple dans le théorème 3.15, où ce qui est susceptible de faire exploser $\mathbb{E}_P(dP/dQ \times f^2)$ est que dP/dQ soit très grande (autrement dit que dQ/dP soit très petite) à un endroit où f^2 est très grande également. Dans certains cas, le sous-échantillonnage peut même nous faire perdre le caractère L^2 de la quantité à intégrer. Pire encore : un échantillonnage très marqué, correspondant à l'oubli pratiquement complet d'une zone contribuant à l'espérance de f, peut conduire à un estimateur apparemment cohérent... mais en fait faux car il ne tient pas compte de la zone sous-échantillonnée ! Il est donc essentiel, chaque fois qu'on veut appliquer la technique d'échantillonnage préférentiel, de se demander d'abord « ne suis-je pas en train de sous-échantillonner gravement certaines zones ? ».

Exemple 3.18. Soient P^* et P_* deux mesures de probabilité sur un espace Ω se décomposant comme une union disjointe $\Omega_0 \cup \Omega_1$. On suppose que P^* (qui sera la mesure *sur*-échantillonnée) donne la masse ε^* à Ω_1 et $(1-\varepsilon^*)$ à Ω_0, et que P_* (qui sera la mesure *sous*-échantillonnée) leur donne quant à elle respectivement les masses ε_* et $(1-\varepsilon)_*$; et on suppose que P_* et P^* ont des densités l'une par rapport à l'autre qui sont constantes sur chaque Ω_i. On prend ici $\varepsilon_* \ll \varepsilon^* \ll 1$, de sorte que les mesures P^* et P_* sont pratiquement identiques sur Ω_0 (la densité relative dP^*/dP_* valant $(1-\varepsilon^*)/(1-\varepsilon_*) \simeq 1$), mais très différentes sur Ω_1.

Que se passe-t-il alors si nous utilisons la mesure d'échantillonnage P^* là où c'est P_* qui aurait été la plus adaptée ; autrement dit si nous essayons d'utiliser la loi d'échantillonnage préférentiel P^* pour évaluer $\mathbb{E}_{P_*}(1)$? L'efficacité par simulation est alors l'inverse de

$$\mathbb{E}_{P_*}(dP_*/dP^*) - 1 = (1-\varepsilon_*) \times \frac{1-\varepsilon_*}{1-\varepsilon^*} + \varepsilon_* \times \frac{\varepsilon_*}{\varepsilon^*} - 1 \simeq \varepsilon^*.$$

À l'inverse, en échantillonnant avec P_* là où P^* aurait été plus adaptée, on tombe sur une efficacité par simulation inverse de

$$\frac{(1-\varepsilon^*)^2}{1-\varepsilon_*} + \frac{(\varepsilon^*)^2}{\varepsilon_*} - 1 \simeq (\varepsilon^*)^2 \,/\, \varepsilon_*,$$

ce qui est beaucoup plus grand (et donc beaucoup moins bon) que ε^* !

Remarque 3.19. Historiquement, c'est l'idée d'échantillonnage préférentiel qui a véritablement lancé la méthode de Monte-Carlo, celle-ci ayant conduit à des améliorations spectaculaires du calcul de certaines quantités physiques correspondant à des évènements de probabilités très faibles.

3.3 Conditionnement

Définition 3.20. Supposons qu'on cherche à évaluer $\mathbb{E}(f)$ par la méthode de Monte-Carlo, f étant une v.a. L^2 définie sur l'espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$, et supposons qu'il y ait une sous-tribu $\mathcal{B}$ de $\mathcal{A}$ telle qu'on sache calculer $\mathbb{E}(f|\mathcal{B}) =: f^{\mathcal{B}}$. Alors la *méthode de conditionnement* consiste à observer que $\mathbb{E}(f) = \mathbb{E}(f^{\mathcal{B}})$ et à estimer $\mathbb{E}(f)$ par l'estimateur

$$\hat{m}^{\text{cond}} := N^{-1} \sum_{i=1}^{N} f^{\mathcal{B}}(\omega_i).$$

Remarque 3.21. On peut résumer la philosophie du conditionnement ainsi : ne pas utiliser Monte-Carlo quand on a accès à un résultat exact !

Exemple 3.22. Reprenons l'exemple du Keno que nous avions vu dans l'exemple 2.14. (après amélioration de la méthode de simulation aléatoire, cf. exemple 3.2). De la façon dont nous avons procédé, l'aléa du de chaque simulation est produit à l'issue de seize étapes : on regarde si le numéro coché '1' est sorti, puis si le numéro coché '2' est sorti, etc. J'affirme que cela correspond à raffirner successivement la tribu par rapport à laquelle nous réalisons notre simulation aléatoire.

En effet, notons Ω l'ensemble des tirages possibles (vus seulement au travers de la question de savoir quels numéros cochés ont été tirée). On a $\Omega = \{0,1\}^{\{1,\dots,16\}}$, une éventualité $\omega =: (\omega_i)_{i\in\{1,\dots,16\}}$ s'interprétant en disant ω_i vaut 1 si la numéro coché pour i points a été tiré, et 0 sinon. Lors de notre simulation, nous déterminions ainsi successivement $\omega_1, \omega_2, \dots, \omega_{16}$ (qui ne sont pas indépendants, NdA). J'affirme qu'à chacune des étapes de cette détermination successive est associé une tribu. Définissions en effet, pour $i \in \{0, \dots, 16\}$,

$$\mathcal{F}_i := \{A_{\leqslant i} \times \{0,1\}^{\{i+1,\dots,16\}} \mid A \in \mathit{Borel}(\{0,1\}^{\{1,\dots,i\}})\} \subset \mathfrak{P}(\Omega).$$

Alors, au bout du tirage de i numéros, nous savons précisément quelles sont les valeurs de ω_j pour $i \leqslant j$, mais rien de plus ; autrement dit, nous savons dans quels ensembles de $\mathcal{F}_i$ le point ω se trouve, mais nous n'avons aucune information supplémentaire. C'est très précisément ce que signifie « tirer ω dans Ω relativement à la tribu $\mathcal{F}_i$ ».

Remarque 3.23. Conformément à nos attentes, la tribu $\mathcal{F}_{16}$ est exactement la tribu borélienne sur Ω ; ce qui signifie qu'à l'issue de notre tirage, nous avons parfaitement déterminé ω. De même, le fait que la tribu $\mathcal{F}_0$ soit la tribu grossière $\{\emptyset, \Omega\}$ signifie qu'avant de commencer notre tirage, on ne connaissait évidemment rien de ω.

Maintenant, notant $\mathit{gain}(\omega)$ le gain correspondant au tirage ω, à quoi correspond $\mathbb{E}(\mathit{gain}|\mathcal{F}_{15})$? Eh bien, cela est l'espérance du gain quand on a déjà déterminé lesquels des numéros sont sortis parmi les ceux cochés de 1 à 15 points. Or, une fois qu'on sait tout cela, nous savons exactement quelle est la probabilité que ω_{16} vaille 0 ou 1 (nous avions justement utilisé cette probabilité dans la suite de notre simulation) ; et bien sûr, selon ce que vaudra ω_{16}, nous connaitrons la valeur correspondante de gain ; de sorte que nous sommes en mesure de déterminer explicitement $\mathbb{E}(\mathit{gain}|\mathcal{F}_{15})$ dès lors que nous

avons déterminé ω par rapport à la tribu $\mathcal{F}_{15}$ [*]. (Voir la §A.3.4 pour le calcul explicite de cette espérance conditionnelle).

Théorème 3.24. *L'efficacité par simulation de l'estimateur conditionné est toujours meilleure que celle de l'estimateur non conditionné.*

Démonstration. L'efficacité par simulation de l'estimateur non conditionné est l'inverse de la variance de f, tandis que celle de l'estimateur conditionné est l'inverse de la variance de $f^{\mathcal{B}}$.

Par l'inégalité de Jensen, $\mathbb{E}((f^{\mathcal{B}})^2) \leqslant \mathbb{E}(f^2)$, et comme en outre on a $\mathbb{E}(f^{\mathcal{B}}) = \mathbb{E}(f)$, il s'ensuit que $\mathrm{Var}(f^{\mathcal{B}}) = \mathbb{E}((f^{\mathcal{B}})^2) - \mathbb{E}(f^{\mathcal{B}})^2 \leqslant \mathbb{E}(f^2) - \mathbb{E}(f)^2 = \mathrm{Var}(f)$, ce qui prouve que l'efficacité par simulation est effectivement meilleure pour $f^{\mathcal{B}}$ que pour f. □

Remarque 3.25. Non seulement la variance par simulation est réduite quand on utilise le conditionnement, mais en général le cout de calcul par simulation va *aussi* diminuer ! En effet, il y aura besoin de pousser la simulation moins loin pour savoir où on tombe dans la tribu $\mathcal{B}$ que pour savoir où on tombe dans la tribu $\mathcal{A}$, vu que la tribu $\mathcal{B}$ est plus grossière. La seule chose qui pourrait éventuellement prendre plus de temps est l'évaluation de f, mais elle-ci représente quasiment toujours une fraction négligeable du cout de calcul. En conclusion, il ne faut jamais hésiter à implémenter une technique de réduction de variance par conditionnement quand on en voit la possibilité.

[∗]. Pour bien s'y retrouver dans les questions d'espérance conditionnelle, il faut garder à l'esprit qu'une espérance conditionnelle, disons $\mathbb{E}(f|\mathcal{B}')$, est un objet "hybride" : d'un côté, c'est une *variable aléatoire* ("la meilleure approximation de f qui soit $\mathcal{B}'$-mesurable"), de sorte qu'on peut parler de sa loi, de son espérance, etc. Mais, dans la mesure où cette variable aléatoire est $\mathcal{B}'$-mesurable, on peut aussi dire que, quand on raisonne *conditionnellement à la tribu* $\mathcal{B}'$ (qui n'est, en général, pas la tribu complète), il s'agit d'une constante... [†]

[†]. La signification de l'expression « raisonner conditionnellement à la tribu $\mathcal{B}$ » est particulièrement facile à comprendre lorsqu'on est sur un univers discret. Supposons ainsi que notre univers est un ensemble dénombrable Ω muni de sa tribu naturelle $\mathcal{B}$ qui n'est autre alors que $\mathfrak{P}(\Omega)$. Si $\mathcal{B}' \subset \mathcal{B}$ est une sous-tribu, pour $\omega \in \Omega$, je peux noter $\bar{\omega}$ ce que j'appellerai « l'évènement élémentaire de la tribu $\mathcal{B}'$ correspondant à ω », défini par

$$\bar{\omega} := \min\{B \in \mathcal{B}' \mid \mathcal{B}' \ni \omega\}.$$

Quand je connais toute l'information relative à la tribu $\mathcal{B}'$, et rien de plus, tout ce que je sais sur mon éventualité ω est qu'elle est située dans $\bar{\omega}$. À part cela, si je sais que ω était distribuée initialement selon la loi $\mathbb{P}$, une fois que je sais que ω est en fait dans $\bar{\omega}$, je peux dire que la probabilité conditionnelle que ω vaille telle ou telle valeur correspond à la probabilité $\mathbb{P}_{\bar{\omega}}$ définie par

$$\mathbb{P}_{\bar{\omega}}(X \subset \bar{\omega}) = \frac{\mathbb{P}(X)}{\mathbb{P}(\bar{\omega})}.$$

J'appellerai cette probabilité la *loi conditionnelle sachant* $\mathcal{B}'$ [‡] [§].

[‡]. Attention : bien que je parle de « la » loi conditionnelle sachant $\mathcal{B}'$, il y a en réalité une loi conditionnelle pour chaque évènement élémentaire $\bar{\omega}$ de $\mathcal{B}'$! Dans les situations où j'emploierai cette expression néanmoins, on sera toujours dans un contexte où $\bar{\omega}$ est connu, de sorte que c'est de cette loi conditionnelle-là que je parlerai.

[§]. Dans les situations où l'univers Ω n'est pas discret, le formalisme mathématique est un peu plus compliqué, mais l'idée de base reste la même.

3.4 Couplage [¶]

Théorème 3.26. *Soient $(\Omega_1, \mathbb{P}_1), (\Omega_2, \mathbb{P}_2)$ des espaces probabilisés (munis de leurs tribus standard) ; soient $f_1 \colon \Omega_1 \to \mathbf{R} \in L^1(\mathbb{P}_1)$, $f_2 \colon \Omega_2 \to \mathbf{R} \in L^1(\mathbb{P}_2)$. Supposons qu'il existe un espace probabilisé $(\Omega, \mathbb{P})$ et des applications $\pi_1 \colon \Omega \to \Omega_1$ et $\pi_2 \colon \Omega \to \Omega_2$ telles que ${\pi_1}_* \mathbb{P} = \mathbb{P}_1$ et ${\pi_2}_* \mathbb{P} = \mathbb{P}_2$. Alors : (notant resp. $\mathbb{E}_1, \mathbb{E}_2, \mathbb{E}$ les espérances relatives aux probabilités $\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}$),*

$$\mathbb{E}_2(f_2) - \mathbb{E}_1(f_1) = \mathbb{E}(f_2 \circ \pi_2 - f_1 \circ \pi_1).$$

Démonstration. Trivial à partir de la linératiré de l'espérance et de la définition de l'espérance. □

Définition 3.27. Lorsque, pour estimer une différence d'espérances par la méthode de Monte-Carlo, on commence par modifier les calculs faits en appliquant le théorème 3.26, on dit qu'on utilise le *méthode de couplage*.

Théorème 3.28. *Supposons que f_1 soit dans $L^2(\mathbb{P}_1)$, resp. f_2 dans $L^2(\mathbb{P}_2)$, et que sous la loi $\mathbb{P}$, les variables aléatoires $f_1 \circ \pi_1$ et $f_2 \circ \pi_2$ soient positivement corrélées [|||]. Alors l'estimateur de Monte-Carlo pour $(f_1 \circ \pi_1 - f_2 \circ \pi_2)$ sous $\mathbb{P}$, avec N_{sim} simulations, a une variance (effective) moins grande que la différence entre l'estimateur de Monte-Carlo pour f_1 sous $\mathbb{P}_1$ et l'estimateur de Monte-Carlo pour f_2 sous $\mathbb{P}_2$, chacun avec N_{sim} simulations.*

Démonstration. Notons $\hat{m}_1$ l'estimateur de Monte-Carlo (naïf) de $\mathbb{E}_1(f_1)$, $\hat{m}_2$ l'estimateur de Monte-Carlo de $\mathbb{E}_2(f_2)$, de sorte que $\hat{m}_1 - \hat{m}_2$, où $\hat{m}_1$ et $\hat{m}_2$ sont indépendants, correspond à l'estimateur sans couplage de $(\mathbb{E}_1(f_1) - \mathbb{E}_2(f_2))$; et notons $\hat{m}$ l'estimateur de $\mathbb{E}(f_1 \circ \pi_1 - f_2 \circ \pi_2)$, càd. l'estimateur avec couplage. Il s'agit donc de comparer les variances de $(\hat{m}_1 - \hat{m}_2)$ d'une part, et de $\hat{m}$ d'autre part. L'estimateur du second est tout simplement :

$$\operatorname{Var}(\hat{m}) = {N_{\text{sim}}}^{-1} \operatorname{Var}(f_1 \circ \pi_1 - f_2 \circ \pi_2);$$

et comme $f_1 \circ \pi_1$ et $f_2 \circ \pi_2$ sont positivement corrélées, en développant la variance, on en déduit que

$$\operatorname{Var}(\hat{m}) \leqslant {N_{\text{sim}}}^{-1} (\operatorname{Var}(f_1 \circ \pi_1) + \operatorname{Var}(f_2 \circ \pi_2)).$$

Pour l'estimateur non couplé ; dans la mesure où celui-ci s'écrit comme la différence de deux estimateurs *indépendants*, sa variance est la somme des variances individuelles de ces estimateurs, autrement dit :

$$\operatorname{Var}(\hat{m}_1 - \hat{m}_2) = {N_{\text{sim}}}^{-1} \operatorname{Var}_1(f_1) + {N_{\text{sim}}}^{-1} \operatorname{Var}_2(f_2).$$

[¶]. En anglais : *Common random numbers*

[|||]. Rappelons que, pour $f_1, f_2 \in L^2(\mathbb{P})$, dire que des variables sont *positivement corrélées* signifie que $\operatorname{Cov}(f_1, f_2) \geqslant 0$. On peut encore donner du sens à ce concept même dans un cadre non L^2, par exemple : « on dit que f_1 et f_2 sont positivement corrélées si, lorsque (f_1, f_2) et (f_1', f_2') sont des réalisations indépendantes de la loi jointe du couple de variables aléatoires (f_1, f_2), la variable aléatoire $(f_1 - f_1')(f_2 - f_2')$ est d'espérance positive (la notion d'espérance positive pour une variable aléatoire pouvant être définie dès lors que celle-ci est de partie négative intégrable) ». Avec une telle définition, si f_2 est une fonction croissante de f_1, f_1 et f_2 seront toujours positivement corrélées, même en cas de défaut d'intégrabilité.

Or la loi de f_1 sous $\mathbb{P}_1$ est aussi la loi de $f_1 \circ \pi_1$ sous $\mathbb{P}$, et la loi de f_2 sous $\mathbb{P}_2$ est aussi la loi de $f_2 \circ \pi_2$ sous $\mathbb{P}$; de sorte que

$$\mathrm{Var}(\hat{m}_1 - \hat{m}_2) = {N_{\mathrm{sim}}}^{-1}(\mathrm{Var}(f_1 \circ \pi_1) + \mathrm{Var}(f_2 \circ \pi_2)).$$

Au final, nous en déduisons bien que la variance de l'estimateur couplé est plus petite. □

Remarque 3.29. Dans le théorème 3.28, le résultat s'exprime sous la forme d'une amélioration de l'efficacité *par simulation* ; alors que, comme nous l'avons dit et redit, ce qui est intéressant est de savoir si l'efficacité *tout court* est améliorée. Néanmoins, dans les situations pratiques auxquelles nous serons confrontés, le cout de calcul par simulation est *plus faible* dans la situation couplée que dans la situation non couplée ! En effet, pour la plupart des couplages naturels, simuler $\mathbb{P}$ puis π_1 n'est guère plus long que de simuler $\mathbb{P}_1$ directement (idem concernant la simulation de $\mathbb{P}_2$ via $\mathbb{P}$ puis π_2). Du coup, si nous négligeons le cout des étapes "techniques" (calcul de la somme de simulations, de la somme des carrés, de la moyenne, etc. ; et dans le cas présent, calcul de la différence entre les deux fonctions) [**], nous voyons que le cout d'une étape de calcul est, pour la situation couplée, correspond à

— Simulation de $\mathbb{P}$;
— Calcul de π_1 ;
— Calcul de π_2 ;
— Calcul de f_1 ;
— Calcul de f_2 ;

et pour la situation non couplée, à

— Simulation de $\mathbb{P}_1$, *égale* [††] Simulation de $\mathbb{P}$ *plus* Simulation de π_1 ;
— Calcul de f_1 ;
— Simulation de $\mathbb{P}_2$, *égale* Simulation de $\mathbb{P}$ *plus* Simulation de π_1.

Le calcul couplé est ainsi (modulo les approximations simplificatrices que nous avons faites) strictement plus court que le calcul couplé — on gagne à peu près le temps de simulation de $\mathbb{P}$ à chaque étape.

Il convient aussi d'observer que, dans les situations pratiques, la possibilité d'appliquer une forme de stratification (de type « à postériori ») (cf. §**??**) pour l'évaluation naïve de $(\mathbb{E}_1(f_1) - \mathbb{E}_2(f_2))$ n'offre aucune amélioration par rapport à la méthode proposée ici consistant à procéder au même nombre de simulations pour chaque espérance individuelle ; car le temps de simulation de f_1 sous $\mathbb{P}_1$ et f_2 sous $\mathbb{P}_2$ sera quasiment identique, et $\mathrm{Var}_1(f_1)$ et $\mathrm{Var}_2(f_2)$ seront très proches ; de sorte que l'allocation optimale des calculs correspond bien à faire faire à peu près autant de simulations pour chaque espérance.

Remarque 3.30. Contrairement à la plupart des autres techniques de réduction de la variance, la possibilité de procéder par couplage ou pas pour améliorer l'efficacité d'une estimation apparait en général de façon assez flagrante. *Les situations qui se prêtent bien*

[**]. En pratique, ces étapes techniques étant très simples par rapport aux étapes de simulation et d'évaluation des fonctions à intégrer, elles ne prennent effectivement qu'une part extrêmement mineure du cout de calcul total.

[††]. L'égalité s'entend ici au sujet du cout de calcul.

au couplage correspondent en effet à celles où on doit estimer la différence entre deux situations qui sont essentiellement les mêmes, ne différant que par un paramètre mineur, comme dans l'exemple **??** ci-dessous. Dans ce cas, la façon judicieuse de procéder au couplage est d'utiliser le couplage *naturel* qui existe entre les deux situations ; où par « couplage naturel », j'entends qu'on utilise la *même* source d'aléa d'une situation à l'autre, les paramètres différant n'intervenant qu'aussi discrètement que possible (là encore, cf. exemple 3.31). Dès lors, $f_1 \circ \pi_1$ et $f_2 \circ \pi_2$ se "ressembleront" beaucoup, ce qui fait que la corrélation positive (et même une *forte* corrélation positive) sera quasi-automatiquement assurée.

Exemple 3.31 (Le yahtzee de la mort). Un groupe d'explorateurs a été capturé par une tribu de sauvages. Ceux-ci décident que le sort des explorateurs sera tranché par les dieux de la forêt *via* le jeu du yahtzee. À savoir, les explorateurs disposent de 5 dés équilibrés, et lancent ceux-ci une première fois ; puis, à deux reprises, il ont le droit de relancer tout ou partie (ou aucun) des dés, en choisissant ceux qu'ils veulent relancer en fonction du résultat obtenu. Les primitifs, qui attribuent aux dieux les résultats des dés, libèreront les explorateurs si le total des dés final atteint ou dépasse 24, et les exécuteront sinon. Le jeu ne devant avoir lieu que le lendemain matin, les explorateurs (qui pour leur part croient plus aux probabilités qu'au jugement des dieux) se concertent sur la stratégie optimale à suivre. Or, deux propositions s'opposent au sujet du meilleur choix des dés à relancer : l'une due à Charlotte et l'autre due à Xénia (voir la §**??** pour les détails). Comment savoir quelle stratégie est la meilleure ? Les explorateurs ayant justement 5 dés sur eux, le plus simple est de profiter de la nuit pour appliquer la méthode de Monte-Carlo "à la main" et évaluer les probabilités de succès respectives avec chaque stratégie.

Voilà à quoi pourraient ressembler les codes qui simuleraient les stratégies respectives de Charlotte et Xénia :

```
% La fonction "probaC" évalue la probabilité de succès en suivant la
% stratégie de Charlotte.
function probaC ()
Nsim = 120;
Nsucc = 0;
for i = 1: Nsim
    Nsucc = Nsucc + strategieC ();
end
p = Nsucc / Nsim;
var_f = p - p * p;
ect_estr = sqrt (var_f / Nsim);
fprintf ('Probabilité de succès pour la stratégie de Charlotte : ');
fprintf ('%f ± %f.\n', moy, 1.96 * ect_estr);
return
end
```

Dans ce contexte-ci, il faut aussi montrer la fonction auxiliaire `strategieC` :

```
% La fonction "strategieC" fait une simulation du jeu en suivant la
% stratégie de Charlotte. Elle renvoie true en cas de succes, et false en
```

```
% cas d'échec.
function resultat = strategieC ()
% La variable "des" est un 5-vecteur qui stocke les valeurs des dés devant
% le joueur. On commence par le premier lancer.
des = ceil (6 * rand (1, 5));
% On va relancer deux fois les dés, d'où la boucle ci-dessous.
for i = 1: 2
    % La fonction auxiliaire "choixC" est au centre de la stratégie de
    % Charlotte: elle dit, sachant à quel stade du jeu on en est, pour un
    % ensemble de dés fixés, quels sont les dés que Charlotte choisit de
    % relancer ou pas.
    choix = choixC (i, des);
    % On relance les dés que Charlotte veut relancer.
    for j = 1: 5
        if (choix(j))
            des(j) = ceil (6 * rand ());
        end
    end
end
resultat = (sum (des) >= 24);
return
end
```

Bien entendu, on définirait similairement des fonctions `probaX` et `strategieX`. Ce qui est intéressant, c'est de voir comment on pourrait coupler ces deux stratégies. Voici un couplage naturel : on imagine qu'il y a une même source d'aléa quelle que soit la stratgéie, qui réside dans le résultat du premier lancer de dés, puis dans le résultat du premier dé éventuellement relancé, le résultat du second dé éventuellement relancé, etc. Dans le programme ci-dessous, cette source d'aléa est appelée "source". Notons que nous faisons le choix de ne pas remplir cette source à l'avance, mais seulement en fonction des résultats dont on a besoin. le programme est le suivant :

```
% La fonction "diffprobaCX" évalue la différence des probabilités de succès
% selon qu'on suit la stratégie de Charlotte ou celle de Xénia.
function diffprobaCX ()
Nsim = 120;
total_f = 0;
total_f2 = 0;
for i = 1: Nsim
    % Toute la technique de couplage sera contenu dans la fonction de
    % simulation couplée "diffstrategieCX"
    f = diffstrategieCX ();
    total_f = total_f + f;
    total_f2 = total_f2 + f * f;
end
moy = total_f / Nsim;
var_f = total_f2 / Nsim - moy * moy;
ect_estr = sqrt (var_f / Nsim);
fprintf ('Différence des probabilités de succès : ');
```

```
fprintf ('%f ± %f.\n', moy, 1.96 * ect_estr);
return
end

% La fonction "diffstrategieCX" renvoie, dans une situation couplée, +1
% quand la stratégie de Charlotte a réussi mais pas celle de Xénia, -1
% quand c'est l'inverse, et 0 sinon.
function diff = diffstrategieCX ()
% On alloue directement la mémoire pour "source", sachant qu'on aura besoin
% d'au plus 15 valeurs.
source = zeros (1, 15);
% On commence par regarder ce que donne la stratégie de Charlotte. Les
% tirages de dés que nous serons amenés à faire seront stockés dans
% "source".
des = ceil (6 * rand (1, 5));
% Ce lancer de dés initial constitue évidemment les 5 premières valeurs de
% la source.
source (1: 5) = des;
% La variable "donnees_puisees" nous indique combien on a déjà tiré de dés
% dans "source".
donnees_puisees = 5;
% On passe aux dés à relancer.
for i = 1: 2
    choix = choixC (i, des);
    for j = 1: 5
        if (choix(j))
            % "r" est le résultat du dé supplémentaire tiré.
            r = ceil (6 * rand ());
            % On met ce résultat supplémentaire dans la source de données.
            donnees_puisees = donnees_puisees + 1;
            source(donnees_puisees) = r;
            des(j) = r;
        end
    end
end
% On stocke le résultat de Charlotte dans "resultatC".
resultatC = (sum (des) >= 24);
% On stocke aussi le nombre total de donnees écrites, dans
% "donnees_ecrites".
donnees_ecrites = donnees_puisees;
% Maintenant on regarde ce que donne la stratégie de Xénia. Toute l'astuce
% est qu'on utilisera la source de données déjà utilisée par Charlotte,
% autant que possible!
des = source (1: 5);
donnees_puisees = 5;
for i = 1: 2
    choix = choixX (i, des);
    for j = 1: 5
        if choix(j)
```

```
            % Observez comment on va forcer le couplage en allant lire dans
            % "source" quand c'est possible.
            if donnees_puisees < donnees_ecrites
                r = source(donnees_puisees + 1);
                donnees_puisees = donnees_puisees + 1;
            else
                r = ceil (6 * rand ());
            end
            des(j) = r;
        end
    end
end
resultatX = (sum (des) >= 24);
diff = resultatC - resultatX;
return
end
```

3.5 Stratification

3.5.1 Stratification avec échantillonnage uniforme

Théorème 3.32. *Supposons qu'on cherche à évaluer $\mathbb{E}(f)$ par la méthode de Monte-Carlo, et que l'espace Ω sur lequel est définie la probabilité $\mathbb{P}$ est partitionné en différentes « strates » $A_1, \dots, A_k$ dont les probabilités respectives (supposées toutes non nulles) sont connues exactement, notées p_k. Réalisons N simulations indépendantes de $\mathbb{P}$ et notons n_k le nombre de ce simulations qui tombent respectivement dans A_k ; alors l'*estimateur stratifié

$$\hat{m}^{\text{strat}} := \sum_{k=1}^{K} p_k n_k^{-1} \sum_{\omega_i \in A_k} f(\omega_i)$$

a une meilleure efficacité (par simulation) que l'estimateur "simple" (2.10).

Démonstration. Notons $\mathbb{P}_k$ la loi conditionnelle de $\mathbb{P}$ sous A_k (càd. $\mathbb{P}_k(B) := p_k^{-1}\,\mathbb{P}(B \cap A_k)$), et $m_k := \mathbb{E}_k(f)$. On a alors $\mathbb{P} = \sum_k p_k\,\mathbb{P}_k$, d'où $\mathbb{E}(f) = \sum p_k m_k$. Or on peut réécrire (3.32) comme $\hat{m}_{\text{strat}} = \sum_k p_k \hat{m}_k$, où $\hat{m}_k$ est l'estimateur de Monte-Carlo "ordinaire" de m_k, réalisé avec un nombre simulations égal à n_k. Puisque N est très grand, par la loi des grands nombres on a $n_k \sim p_k N$; dans la suite, nous ferons comme si on avait exactement $n_k = p_k N$, ce qui ne changera pas le calcul de la valeur de l'efficacité. L'estimateur $\hat{m}_k$ a une variance égale à $n_k^{-1} \operatorname{Var}_{\mathbb{P}_k}(f)$, et les différents $\hat{m}_k$ sont indépendants, donc la variance globale de $\hat{m}^{\text{strat}}$ vaut

$$\sum_{k=1}^{K} p_k^2 n_k^{-1} \operatorname{Var}_{\mathbb{P}_k}(f) = N^{-1} \sum_{k=1}^{K} p_k \operatorname{Var}_{\mathbb{P}_k}(f),$$

d'où une efficacité par simulation égale à celle qu'aurait une variable de variance $\sum_k p_k \operatorname{Var}_{\mathbb{P}_k}(f)$: cette quantité est appelée la *variance intrastrates*.

Montrons que la variance globale $\mathrm{Var}(f)$ est plus grande que la variance intrastrates, ce qui prouvera le théorème. On a

$$\begin{aligned}\mathrm{Var}(f) = \mathbb{E}(f^2) - \mathbb{E}(f)^2 &= \sum_k p_k \, \mathbb{E}_k(f^2) - \Big(\sum_k p_k \, \mathbb{E}(f_k)\Big)^2 \\ &= \sum_k p_k \, \mathrm{Var}_{\mathbb{P}_k}(f) + \Big[\sum_k p_k \, \mathbb{E}_k(f)^2 - \Big(\sum_k p_k \, \mathbb{E}(f_k)\Big)^2\Big]. \quad (3.1)\end{aligned}$$

Le premier terme de cette somme correspond à la variance intrastrates, tandis que le terme entre crochets est positif par l'inégalité de Cauchy–Schwarz (vu que $\sum_k p_k = 1$). □

Exemple 3.33 (La Lorraine indépendante ?). Un référendum va être organisé en Lorraine, pour ou contre l'indépendance de la région. Un institut de sondage essaye de prédire les résultats. Nous ferons les hypothèses suivantes :

— L'institut de sondage dispose d'un mécanisme qui lui permet de tirer au hasard un électeur lorrain de façon exactement uniforme.
— Tous les électeurs interrogés répondent, et répondent honnêtement.
— Les électeurs ne changeront pas d'avis d'ici le scrutin définitif.

Dans ce cas, faire un sondage est exactement équivalent à évaluer la différence entre la proportion d'électeurs qui voteront « oui » et celle de « non » par la méthode de Monte-Carlo, ce qui est doit ainsi effectivement prédire le résultat du référendum.

Ayant interrogé 1 024 électeurs, notre institut de sondage a obtenu les résultats suivants :

Réponse	Nombre	Pourcentage
Pour	???	??,?? %
Blanc	???	??,?? %
Contre	???	??,?? %

D'après ces résultats, la méthode de Monte-Carlo lui permettra d'annoncer comme estimateur une victoire du « oui » par une avance de ?,?? % des inscrits, avec une marge d'erreur à 5 % sur cette avance de ±?,?? % — ce qui n'est pas assez pour conclure de façon catégorique...

Mais il se trouve que, en plus de demander leur avis aux sondés, les sondeurs leur ont demandé de quel département de la Lorraine ils étaient : Meurthe-et-Moselle (*54*), Meuse (*55*), Moselle (*57*) ou Vosges (*88*). Et les résultats détaillés montrent une nette disparité d'un département à l'autre :

Département	Sondés	Pour	(% pour)	Blanc	(% blanc)	Contre	(% contre)
54	???	???	??,??	???	??,??	???	??,??
55	???	???	??,??	???	??,??	???	??,??
57	???	???	??,??	???	??,??	???	??,??
88	???	???	??,??	???	??,??	???	??,??
Total	???	???	??,??	???	??,??	???	??,??

Évidemment, sans information supplémentaire, de telles données détaillées ne permettent pas d'affiner l'estimation en quoi que ce soit. Mais information supplémentaire il y a ! Car notre institut de sondage connait grâce au recensement le nombre exact d'électeurs de chaque département... Il peut donc considérer qu'il a en fait procédé à quatre sondages partiels indépendants, chacun de ces sondages partiels lui donnant une estimation de la contribution du département concerné au résultat global :

Département	Résultat local	Proportion	Contribution
54	$??,??\pm?,??$	$0,??????$	$??,??\pm?,??$
55	$??,??\pm?,??$	$0,??????$	$??,??\pm?,??$
57	$??,??\pm?,??$	$0,??????$	$??,??\pm?,??$
88	$??,??\pm?,??$	$0,??????$	$??,??\pm?,??$
Total	$??,??\pm?,??$	1	$??,??\pm?,??$

Pour obtenir la dernière case de la ligne « Total », on a ajouté les estimateurs. Concernant la marge d'erreur, étant donné que les différentes estimations sont indépendantes, il faut ajouter les *variances* et non pas simplement les intervalles : puisque la largeur de l'intervalle de confiance est proportionnelle à l'écart-type, on obtient une largeur pour l'intervalle global qui est la racine de la somme des carrés des longueurs, soit $?,??$, nettement plus petite que la somme des longueurs qui serait $?,??$.

On constate donc deux choses :

— L'estimateur prenant en compte la stratification par département n'est pas le même que l'estimateur "brut" ! En l'occurrence, ...

— La précision de l'estimateur stratifié est meilleure que celle de l'estimateur brut ! L'utilisation intelligente des données sur les départements nous a donc permis d'améliorer notre estimation.

3.6 Allocation interstrates optimale

Théorème 3.34. *Supposons qu'on cherche à évaluer par la méthode de Monte-Carlo une quantité de la forme* $\mathbb{E}(f_1) + \cdots + \mathbb{E}(f_K)$ *en estimant séparément les différents* $\mathbb{E}(f_k)$*. La façon optimale de diviser le cout entre les différents calculs consiste alors à consacrer au calcul de* $\mathbb{E}(f_k)$ *un cout proportionnel à* $\mathit{Eff}_k^{-1/2}$*, où* Eff_k *désigne l'efficacité du calcul de* $\mathbb{E}(f_k)$*.*

Démonstration. Notons Z le cout total de calcul, et q_k la proportion de Z qu'on consacre au calcul de $\mathbb{E}(f_k)$. Le cout de calcul total consacré à l'estimation de $\mathbb{E}(f)$ est alors $q_k Z$, et la variance de l'estimateur obtenu est donc de $(\mathit{Eff}_k q_k Z)^{-1}$. Comme les estimateurs des différents $\mathbb{E}(f_k)$ sont indépendants, leurs variances d'ajoutent, de sorte que la variance globale pour l'estimateur de $\mathbb{E}(f_1) + \cdots + \mathbb{E}(f_K)$ vaut $\sum_k (\mathit{Eff}_k q_k)^{-1} \times Z^{-1}$, ce qui correspond à une efficacité globale de

$$\Big(\sum_{k=1}^{K} (\mathit{Eff}_k q_k)^{-1}\Big)^{-1}. \tag{3.2}$$

Il faut donc maximiser (3.2), ou encore minimiser $\sum_k \mathit{Eff}_k^{-1} q_k^{-1}$, sous la contrainte que $\sum_k q_k = 1$. On va utiliser la méthode du multiplicateur de Lagrange : on introduit un paramètre λ et on cherche une valeur critique de $f(q_1, \dots, q_K) := \sum_k \mathit{Eff}_k^{-1} q_k^{-1} + \lambda \sum_k q_k$, qui vérifie la contrainte. Vu que $\partial f / \partial q_k = -\mathit{Eff}_k^{-1} q_k^{-2} + \lambda$, il y a un unique point critique pour $q_k = \lambda^{-1/2} \mathit{Eff}_k^{-1/2}$ $\forall k$, ce qui montre (sans même avoir à calculer λ) que les q_k optimaux sont proportionnels aux $\mathit{Eff}_k^{-1/2}$. □

Exemple 3.35 (La Lorraine indépendante, suite). Reprenons l'exemple de notre sondage sur l'indépendance de la Lorraine. Un institut concurrent de notre premier institut de sondage veut procéder à sa propre estimation. Pour faire mieux que le premier institut, il désire arriver à une précision meilleure pour un investissement financier égal. Nous supposerons ici que le cout du sondage est directement proportionnel au nombre d'individus sondés, et ce indépendamment de leur département, de sorte que l'efficacité du sondage relatif à un département donné sera directement proportionnelle à l'inverse de la variance par habitant pour ce département. *(À poursuivre...).*

3.7 Variables de contrôle

3.7.1 Version rudimentaire

Définition 3.36. Supposons qu'on cherche à évaluer $\mathbb{E}(f)$ par la méthode de Monte-Carlo, et qu'on disponse d'une variable aléatoire g de classe L^2 sur Ω dont on connaisse l'espérance exactement. Dire qu'on utilise g comme *variable de contrôle* pour évaluer $\mathbb{E}(f)$ par la méthode de Monte-Carlo signifie, dans sa version "rudimentaire", qu'on s'appuie sur l'écriture « $\mathbb{E}(f) = \mathbb{E}(f - g) + \mathbb{E}(g)$ », autrement dit qu'on considère l'estimateur

$$\hat{m}^g := N^{-1} \sum_{i=1}^{N} \big(f(\omega_i) - g(\omega_i) \big) + \mathbb{E}(g).$$

Proposition 3.37. *L'efficacité par simulation de la méthode de Monte-Carlo utilisant la variable de contrôle g (au sens rudimentaire) est l'inverse de la variance de $(f - g)$.*

Démonstration. Évident. □

Exemple 3.38. En finance, certains contrats permettent de vendre un actif à la valeur maximale que son cours prendra sur une certaine période. Dans ce cadre, il est important d'être capable d'évaluer l'espérance du maximum ou du minimum d'une trajectoire ; cela motive l'exemple simplifié que nous présentons maintenant.

On considère une marche aléatoire de 60 pas sur $\mathbf{R}$ issue de 0 dont les incréments sont de loi $\mathcal{N}(1)$; autrement dit, pour $(S_j)_{1 \leqslant j \leqslant 60}$ des v.a. i.i.d. $\mathcal{N}(1)$, on considère la suite finie $(X_i)_{0 \leqslant i \leqslant 60}$ définie par $X_j = \sum_{j=1}^{i} S_j$. On définit la variable aléatoire X_* comme la plus grande valeur atteinte par la suite aléatoire $(X_i)_i$, c'est-à-dire que pour tout ω, on pose $X_*(\omega) := \max_{0 \leqslant i \leqslant 60} X_i(\omega)$. Notre but est d'évaluer $\mathbb{E}(X_*)$.

En première approximation, si on imagine que le comportement que $(X_i)_i$ au cours du temps est à peu près linéaire, on peut considérer que $X_* \simeq \max\{X_{60}, 0\} =: Y$. Or

on sait calculer exactement $\mathbb{E}(Y)$: en effet, $Y \sim \mathcal{N}(60)$, donc, notant $\sigma := \sqrt{60}$,

$$\mathbb{E}(Y) = \int_{-\infty}^{\infty} \max\{x, 0\} \frac{e^{-x^2/2\sigma^2}}{\sqrt{2\pi}\sigma}\, dx = \frac{1}{\sqrt{2\pi}\sigma} \int_0^{\infty} x e^{-x^2/2\sigma^2} dx$$
$$= \frac{1}{\sqrt{2\pi}\sigma} \left[-\sigma^2 e^{-x^2/2\sigma^2}\right]_0^{\infty} = \frac{\sigma}{\sqrt{2\pi}} = 3{,}090\ 193... \quad (3.3)$$

Cela suggère d'utiliser $X_{60} \wedge 0 =: Y$ comme variable de contrôle pour $\mathbb{E}(X_*)$.

L'annexe A.7 montre comment implémenter l'utilisation de cette variable de contrôle. Sans variable de contrôle, on obtient le résultat suivant :

```
>> tic; mathfi_naif(100000); toc;
Estimation de E(X_*) : 5.62716 ± 0.02893
Elapsed time is 1.086699 seconds.
```

Et avec l'utilisation rudimentaire de la variable de contrôle Y :

```
>> tic; mathfi_ctrl0(100000); toc;
Estimation de E(X_*) : 5.61888 ± 0.01427
Elapsed time is 0.998897 seconds.
```

On voit donc que, pour un temps de calcul à peu près égal, l'incertitude a été réduite d'un peu plus d'un facteur 2, soit une amélioration de l'efficacité d'un facteur 4.

3.7.2 Version paramétrée

Si on dispose d'une variable de contrôle g, on peut aussi se servir de λg comme variable de contrôle pour n'importe quel $\lambda \in \mathbf{R}$, puisqu'on connait alors $\mathbb{E}(\lambda g) = \lambda\, \mathbb{E}(g)$. Cela suggère donc de choisir la valeur la plus judicieuse possible pour λ.

Théorème 3.39. *La valeur de λ qui minimise $\mathrm{Var}(f - \lambda g)$ est*

$$\lambda^* = \mathrm{Cov}(f, g)\, \mathrm{Var}(g)^{-1}.$$

Notant $r := \mathrm{Cov}(f, g) \,/\, \mathrm{Var}(f)^{1/2}\, \mathrm{Var}(g)^{1/2}$ le coefficient de corrélation entre f et g, l'efficacité est alors améliorée d'un facteur $(1-r^2)^{-1}$: cette technique est donc d'autant plus efficace que $|r|$ est très proche de 1.

Démonstration. On développe par bilinéarité $\mathrm{Var}(f - \lambda g) = \mathrm{Var}(f) - 2\lambda\, \mathrm{Cov}(f, g) + \lambda^2\, \mathrm{Var}(g)$. Il s'agit est un polynôme de degré 2 en λ dont la minimisation est élémentaire : le minimum de $\mathrm{Var}(f - \lambda g)$ est atteint en λ^* et vaut $\mathrm{Var}(f) - \mathrm{Var}(g)^{-1}\, \mathrm{Cov}(f, g)^2 = (1 - r^2)\, \mathrm{Var}(f)$, d'où les résultats annoncés. □

Exemple 3.40. Si nous reprenons l'exemple des mathématiques financières avec toujours la variable de contrôle Y, mais cette fois-ci dans sa version paramétrée, nous arrivons au programme de l'annexe **??**. L'exécution donne :

```
>> tic; mathfi_ctrl1(100000); toc;
Estimation de E(X_*) : 5.61753 ± 0.01404
Elapsed time is 1.029657 seconds.
```

En l'occurrence on ne gagne presque rien par rapport à la version simple de la variable de contrôle, parce qu'il se trouve que la λ optimal vaut à peu près 0,9, de sort que la version simple (correspondant à $\lambda = 1$) était déjà pratiquement optimale.

3.7.3 Variables de contrôle multiples

On peut étendre l'idée du paragraphe précédent au cas de plusieurs variables de contrôle $g_1, \dots, g_K$, en appliquant (3.36) avec $\lambda_1 g_1 + \cdots + \lambda_K g_K$, $(\lambda_1, \dots, \lambda_K)$ étant un jeu de K paramètres à optimiser.

Théorème 3.41. *La valeur de* $(\lambda_1, \dots, \lambda_K) =: \vec{\lambda}$ *qui minimise* $\mathrm{Var}(f - \lambda_1 g_1 - \cdots - \lambda_K g_K)$ *est donnée par*

$$\vec{\lambda}^* = R_{fg} \Sigma_g^{-1},$$

où R_{fg} *est le vecteur* $\big(\mathrm{Cov}(f, g_1), \dots, \mathrm{Cov}(f, g_K)\big)$ *et* Σ_Y *est la matrice de covariance (supposée non dégénérée) de* $(g_1, \dots, g_K)$.

Démonstration. Par bilinéarité,

$$\mathrm{Var}(f - \lambda_1 g_1 - \cdots - \lambda_K g_K) = \mathrm{Var}(f) - 2\vec{\lambda} R_{fg}^{\mathsf{T}} + \vec{\lambda} \Sigma_g \vec{\lambda}^{\mathsf{T}},$$

où nous rappelons que Σ_g, en tant que matrice de covariance, est symétrique et positive (définie). Ce polynôme de degré 2 en λ se réduit en

$$\mathrm{Var}(f) + \big\| \vec{\lambda} \Sigma_g^{1/2} - R_{fg} \Sigma^{-1/2} \big\|^2 - R_{fg} \Sigma_g^{-1} R_{fg}^{\mathrm{T}},$$

où $\|v\|^2$ désigne la norme vv^{T} d'un vecteur v de $L^2(\mathbf{R}^K)$. Le seul terme de (3.7.3) dépendant de $\vec{\lambda}$ étant alors le carré de la norme de $\vec{\lambda}\Sigma_g^{1/2} - R_{fg}\Sigma^{-1/2}$, l'expression est minimale quand ce vecteur s'annule, ce qui donne bien $\vec{\lambda}^* = R_{fg}\Sigma_g^{-1}$. □

Exemple 3.42. Reprenons encore une fois l'exemple des mathématiques financières. Je prends trois variables de contrôle : $X_{20} \wedge 0 := Y_1$, $X_{40} \wedge 0 := Y_2$ et $X_{60} \wedge 0 := Y_3$, qui ont pour espérances respectives $\sqrt{20\,/\,2\pi}$, $\sqrt{40\,/\,2\pi}$ et $\sqrt{60\,/\,2\pi}$. Cela conduit au programme de l'annexe **??**, dans laquelle nous avons utilisé le formalisme vectoriel de Matlab pour traiter Y_1, Y_2 et Y_3 avec une seule ligne de code à chaque fois. L'exécution donne :

```
>> tic;mathfi_ctrl3(100000);toc;
Estimation de E(X_*) : 5.62414 ± 0.01012
Elapsed time is 1.827719 seconds.
```

On a donc encore gagné un facteur 2 sur l'efficacité par simulation par rapport à l'utilisation d'une seule variable de contrôle.

3.8 Variables antithétiques

Théorème 3.43. *Supposons que nous cherchons à évaluer* $\mathbb{E}(f)$ *par la méthode de Monte-Carlo. Notant* Ω *l'espace sur lequel vivent nos simulations, supposons que nous disposions d'une application non triviale* $\gamma\colon \Omega \to \Omega$ *qui préserve la mesure* $\mathbb{P}$ *(càd. que la mesure-image de* $\mathbb{P}$ *par* γ *est encore* $\mathbb{P}$*), et supposons que simuler* $\big(f(\omega), f(\gamma(\omega))\big)$

coute deux fois plus cher que simuler $f(\omega)$. Alors, si les variables aléatoires f et $f \circ \gamma$ sont négativement corrélées, l'estimateur

$$\hat{m}^{\text{anti}} := (2N)^{-1} \sum_{i=1}^{N} \big(f(\omega_i) + f(\gamma(\omega_i)) \big)$$

est plus efficace que l'estimateur de Monte-Carlo "simple".

Démonstration. L'efficacité de cette méthode par simulation est l'inverse de la variance de $\frac{1}{2}\big(f(\omega_i)+f(\gamma(\omega_i))\big)$, laquelle se développe en $\frac{1}{4}\big(\mathrm{Var}(f)+\mathrm{Var}(f\circ\gamma)+\mathrm{Cov}(f, f\circ\gamma)\big) = \frac{1}{2}\mathrm{Var}(f) + \frac{1}{4}\mathrm{Cov}(f, f\circ\gamma) \leqslant \frac{1}{2}\mathrm{Var}(f)$ en utilisant successivement que $f \circ \gamma$ a la même loi que f et l'hypothèse de corrélation négative entre f et $f \circ \gamma$. Ainsi l'efficacité par simulation est au moins 2 fois meilleure pour la méthode antithétique ; comme en outre le coût par simulation est au plus 2 fois plus important, au final on obtient bien que l'efficacité tout court est meilleure avec la méthode antithétique. □

Exemple 3.44. Nous prenons encore une fois l'exemple des mathématiques financières. Nous allons adjoindre à la simulation de notre processus financier une simulation "antithétique" où la trajectoire du processus financier sera symétrique par rapport à 0 ! Il est clair que la trajectoire symétrique a bien la même loi que la trajectoire originale, de sorte que celle-ci correspond toujours à un échantillonnage sous la loi de probabilité qui nous intéresse. En outre, on peut raisonnablement supposer que la corrélation entre les valeurs de X_* pour des deux trajectoires sera négative, puisque quand X_* est grand pour la première trajectoire, c'est que celle-ci a eu tendance à prendre de grandes valeurs, et donc la trajectoire symétrique a eu tendance à prendre de petites valeurs, et donc X_* est petit pour la trajectoire symétrique. Voir l'implémentation dans l'annexe A.7.5, qui donne à l'exécution :

```
>> tic;mathfi_anti(100000);toc;
Estimation de E(X_*) : 5.62541 ± 0.01627
Elapsed time is 0.615886 seconds.
```

On voit qu'on a gagné par rapport à l'algorithme de base, non seulement en termes de variance (d'un facteur plus de 3), mais aussi sur la vitesse par simulation, ce qui s'explique par le fait que nous avons eu à effectuer deux fois moins de tirages aléatoires, chaque tirage étant effectué deux fois.

Deuxième partie

Initiation aux processus aléatoires

Chapitre 4

Généralités sur les processus aléatoires

Définition 4.1. Pour E un espace métrique, une fonction $f\colon \mathbf{R}_+ \to E$ est dite *càdlàg* (**c**ontinue **à d**roite avec **l**imite **à g**auche) si $f(t) = \lim_{\substack{u\to t\\ u>t}} f(u)$ pour tout $t \geqslant 0$, et que pour tout $t > 0$, $\lim_{\substack{u\to t\\ u>t}} f(u)$ existe dans E (cette limite étant alors notée $f(t-)$). Dans la suite, nous appellerons *trajectoire à valeurs dans E* une fonction càdlàg de $\mathbf{R}_+$ dans E.

Définition 4.2. Pour E un espace métrique séparable, un *processus aléatoire à valeurs dans E* est une variable aléatoire qui prend ses valeurs dans l'esapce des trajectoires de $\mathbf{R}_+$ dans E ; autrement dit c'est une trajectoire aléatoire $(Z_t)_{t\geqslant 0}$.

Remarque 4.3. Pour traiter la trajectoire $(Z_t)_{t\geqslant 0}$ comme une variable aléatoire unique, il faut définir une topologie sur l'espace des trajectoires, ce qui soulève quelques problèmes. On pourrait utiliser la topologie de la convergence simple, pour laquelle la notion de convergence en loi est simplement la convergence au sens des marginales finidimensionnelles dont nous parlerons dans la suite du cours. Cependant cette topologie est mauvaise pour faire des probabilités, parce qu'elle n'est pas « de Luzin », ce qui signifie en gros qu'on ne peut pas simuler les trajectoires pour la loi-limite d'une façon compatible avec la topologie. En outre, ce n'est pas non plus une topologie satisfaisante du point de vue analytique : par exemple, le fait d'être une trajectoire continue ne correspond pas à une partie borélienne de cette topologie !

Il se trouve heureusement qu'il existe d'autres topologies présentant de bonnes propriétés probabilistes et analytiques, en particulier la *topologie de Skorokhod* qui est celle qu'on utilise généralement pour parler des processus aléatoires. Mais cette topologie est assez compliquée à décrire et à manipuler... Nous ferons donc le choix dans ce cours de ne pas nous encombrer de ces considérations techniques, mais il faudra garder à l'esprit que les propriétés trajectorielles des processus aléatoires n'ont en fait de sens que pour la topologie de Skorokhod.

Définition 4.4. Un processus aléatoire $(Z_t)_{t\geqslant 0}$ est qualifié de *markovien (homogène)* quand la prédiction de son futur au-delà d'un instant t_1 ne demande que de connaitre la valeur de Z_{t_1} et pas l'ensemble du passé de la trajectoire jusqu'à cet instant, et qu'en outre la loi conditionnelle du futur est invariante par translation en temps. Formellement, cela signifie qu'on a :

$$Loi\big((Z_t)_{t>t_1}\big|(Z_t)_{t\leqslant t_1}\big) = Loi\big((Z_t)_{t>t_1}\big|Z_{t_1}\big) \tag{4.1}$$

et

$$Loi\big((Z_{t_1+u})_{t\geqslant 0}\big|Z_{t_1}=z\big)=Loi\big((Z_u)_{u\geqslant 0}\big|Z_0=z\big). \tag{4.2}$$

L'ensemble (4.1)-(4.2) est appelé « propriété de Markov (faible) ».

Définition 4.5. Un temps aléatoire T est qualifié de *temps d'arrêt* quand on peut déterminer le moment de ce temps à partir de la seule connaissance du passé de la trajectoire avant ce moment, sans avoir à savoir ce qui se passe ensuite. Informellement, il s'agit d'une règle pour crier « STOP ! » à un certain moment suivant une règle déterministe dépendant de la trajectoire, *trajectoire qu'on découvre au fur et à mesure.*

Exemple 4.6.

- Si Z est une processus aléatoire à valeurs dans $\mathbf{R}$, « le premier instant T tel que $|Z_T - Z_{T-1}| \geqslant 3$ » définit un temps d'arrêt.
- En revanche, « l'instant $T \in [0,1]$ auquel Z_t atteint sa plus grande valeur » ne définit *pas* un temps d'arrêt (sauf exception), car on ne peut pas savoir que Z_T sera la plus grande valeur tant qu'on ne connait pas Z_t sur $(T,1]$.

Théorème 4.7. *(Sous certaines hypothèses techniques qui seront toujours satisfaites ici), tout processus aléatoire vérifiant la propriété de Markov faible vérifie aussi la* propriété de Markov forte. *Cette propriété signifie que* (4.1) *s'applique aussi à un temps d'arrêt : pour tout temps d'arrêt T,*

$$Loi\big((Z_t)_{t>T}\big|T=\tau \text{ et } (Z_t)_{t\leqslant\tau}=(z_t)_{t\leqslant\tau}\big)=Loi\big((Z_{t-\tau})_{t>\tau}\big|Z_0=z_\tau\big).$$

Chapitre 5

Processus de type brownien

5.1 Mouvement brownien

Dans cette section nous allons décrire la star de tous les processus aléatoires : le *mouvement brownien.*

5.1.1 Heuristique et définition

Soit X une loi à valeurs dans $\mathbf{R}^d$ de classe L^2, centrée, avec pour matrice de covariance $\sigma^{\mathsf{T}}\sigma$. Considérons la marche aléatoire de pas X, càd. la suite des $S_n = \sum_{i=1}^n X_i$ où les X_i sont i.i.d. de loi X. On se demande ce que cette loi devient à grande échelle, càd. que pour une très grande valeur N on s'intéresse au comportement de la trajectoire aléatoire $(S_{\lfloor tN \rfloor})_{t\geqslant 0}$. Comme $S_{\lfloor tN \rfloor}$ sera de l'ordre de $N^{1/2}$ d'après le théorème-limite central, on s'intéressera plus exactement à $(N^{-1/2}S_{\lfloor tN \rfloor})_{t\geqslant 0}$, abrégé dans la suite en $(B_t^N)_{t\geqslant 0}$.

À N fixé, le processus B^N est un certain processus aléatoire à valeurs dans $\mathbf{R}^d$. Quand $N \to \infty$, nous allons voir que ce processus converge (au sens de la convergence en loi pour les lois sur l'espace des trajectoires) vers une certaine limite, à savoir le mouvement brownien. La définition du mouvement brownien repose sur trois propriétés fondamentales qui sont le pendant de propriétés similaires pour la marche aléatoire $(S_n)_{n\in\mathbf{N}}$ (voir la preuve du théorème 5.3) :

Définition 5.1. On appelle *mouvement brownien d-dimensionnel de covariance par unité de temps* $\sigma^{\mathsf{T}}\sigma$ le processus $(B_t)_{t\geqslant 0}$ défini de la façon suivante :

- $B_0 = 0$ avec probabilité 1 ;
- Pour $0 = t_0 < t_1 < \cdots < t_k$, les vecteurs $(B_{t_{i+1}} - B_{t_i})$ sont indépendants ;
- Le vecteur $B_{t_1} - B_{t_0}$ est un vecteur gaussien centré de matrice de covariance $(t_1 - t_0)\sigma\sigma^{\mathsf{T}}$.

On appelle mouvement brownien d-dimensionnel est dit *standard* celui pour lequel $\sigma^{\mathsf{T}}\sigma = \boldsymbol{I}_d$.

Théorème 5.2. *Le mouvement brownien existe, c'est-à-dire qu'on peut bien construire un objet satisfaisant la définition ci-dessus. En outre, cette définition décrit complètement la loi du mouvement brownien (« au sens des marginales fini-dimensionnelles »),*

c'est-à-dire qu'elle permet de connaitre la loi de $(B_{t_1}, \dots, B_{t_k})$ *pour tout* k*-uplet* $(t_1, \dots, t_k) \in (\mathbf{R}_+)^k$ *(l'ensemble de ces lois constituant ce qu'on appelle les* marginales finidimensionnelles *du mouvement brownien), comme l'explicitera la proposition 5.12 ci-dessous.*

Théorème 5.3. *Quand* $N \to \infty$*, la loi de la trajectoire* $(B_t^N)_{t \geqslant 0}$ *converge vers le mouvement brownien (de variance par unité de temps* $\sigma\sigma^\mathsf{T}$*) « au sens des marginales finidimensionnelles », c'est-à-dire que pour pour* k*-uplet* $(t_1, \dots, t_k)$*, la loi de* $(B_{t_1}^N, \dots, B_{t_k}^N)$ *(qui est un vecteur aléatoire de l'espace fini-dimensionnel* $(\mathbf{R}^d)^k$*) converge vers celle de* $(B_{t_1}, \dots, B_{t_k})$*.*

Remarque 5.4. Le mouvement brownien est un objet *universel*, au sens où on retombe sur le même objet à la limite *quelle que soit la nature exacte de la loi de* X, seule la variance $\sigma^\mathsf{T}\sigma$ de celle-ci intervenant.

Exemple 5.5.

1. La figure 5.1 montre le graphe d'(un morceau de) la trajectoire d'une réalisation d'un mouvement brownien unidimensionnel. On remarquera le caractère imprévisible de la trajectoire, ainsi que son aspect très découpé.
2. La figure 5.2 montre (un morceau de) la trajectoire d'une réalisation d'un mouvement brownien 2-dimensionnel. (Comme notre dessin est lui-même 2-dimensionnel, cette fois-ci on ne peut pas représenter l'indication de temps). On voit que la trajectoire est complètement erratique et embrouillée... Il s'agit ici d'un mouvement brownien *anisotrope*, càd. que la matrice de covariance par unité de temps n'est pas diagonale. Cela se remarque au fait que la trajectoire "remue" manifestement beaucoup plus dans la direction de la première diagonale que dans celle de la seconde diagonale.

5.1.2 Simulation

Généralités

Le mouvement brownien étant un processus à temps continu, simuler complètement sa trajectoire, même sur un intervalle de temps borné, recquerrait de calculer sa valeur en un nombre infini d'instants, ce qui est évidemment impossible. On est donc obligé en pratique de *discrétiser* le temps ; c'est-à-dire qu'on ne simule pas la valeur du mouvement brownien en tous les instants, mais seulement en un nombre fini grand d'entre eux, espacés par de tout petits intervalles. Si on a besoin de connaître des aspects trajectoriels du processus, il faudra proposer une *interpolation* entre les instants de discrétisation ; c'est-à-dire, si on a simulé le processus aux temps t_1 et $t_1 + \varepsilon$, dire à quoi ressemble "en gros" la trajectoire complète sur $t \in [t_1, t_1 + \varepsilon]$. Dans le cas du mouvement brownien et des processus apparentés, la meilleure solution est presque toujours d'interpoler de façon linéaire, càd. de prendre la trajectoire affine $\tilde{B}_t = B_{t_1} + (t - t_1) / \varepsilon \times (B_{t_1+\varepsilon} - B_{t_1})$ pour le processus interpolé.

La discrétisation la plus simple consiste à considérer les instants de discrétisation $t = 0, \varepsilon, 2\varepsilon, 3\varepsilon, \dots$ pour ε très petit. D'autres choix de discrétisation sont possibles, notamment quand il importe de connaitre la trajectoire avec plus de précision sur certaines parties spécifiques. Éventuellement, les instants de discrétisation pourront dépendre de

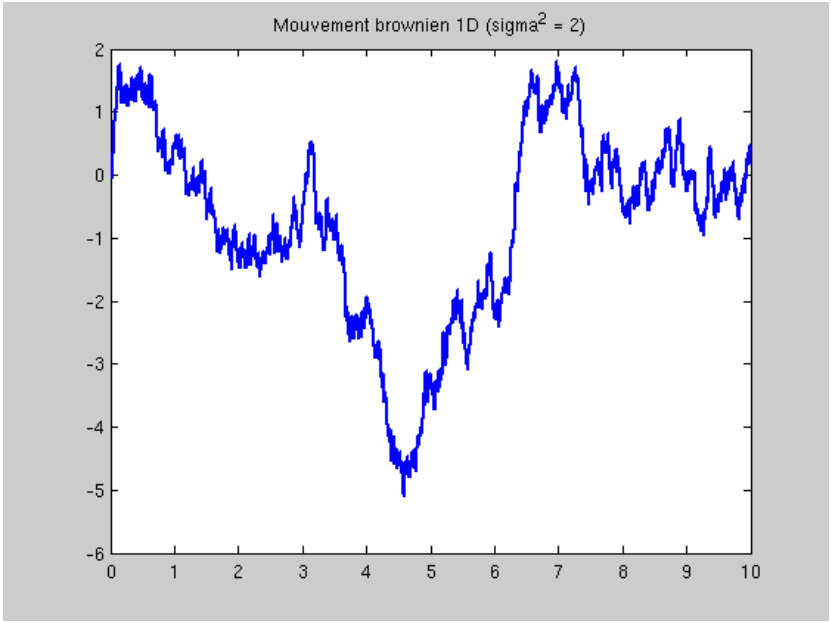

FIGURE 5.1 – Graphe d'un mouvement brownien unidimensionnel sur l'intervalle de temps $[0, 10]$.

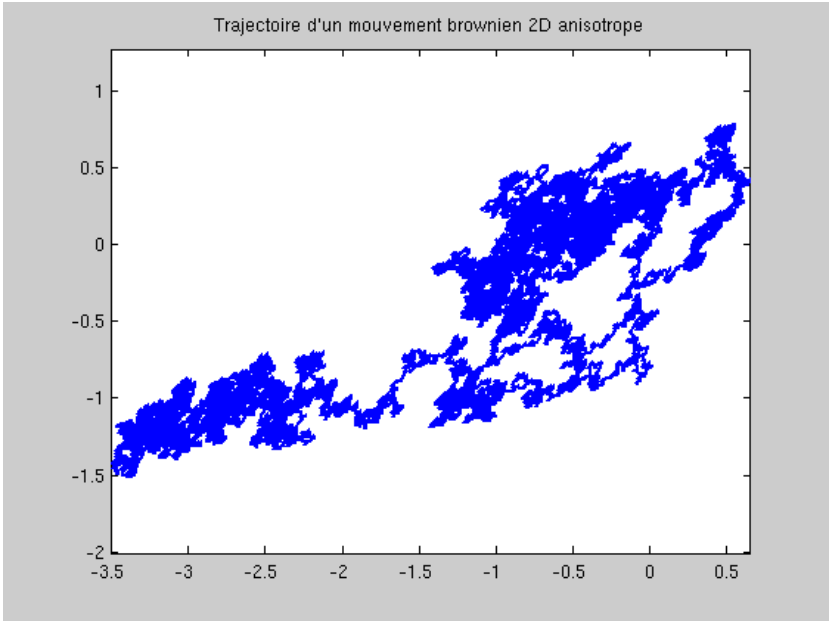

FIGURE 5.2 – Trajectoire d'un mouvement brownien 2-dimensionnel. La matrice de covariance par unité de temps a été choisie de sorte que dX_t^1 et $dX^2(t)$ aient un coefficient de corrélation de +0,5.

la simulation elle-même : ainsi, si par exemple on cherche à savoir à quel instant la trajectoire dépasse un certain seuil pour la première fois, cela conduira à "zoomer" en choisissant des intervalles de temps plus petits quand la valeur du processus approche ce seuil, et le moment où cela se produira dépendra évidemment de la simulation. Le choix de la discrétisation du temps doit assurer le meilleur compromis entre deux objectifs antithétiques : d'un côté, que la simulation discrétisée (avec interpolation) ressemble le plus possible à la vraie trajectoire, ce qui suggère de prendre de très petits intervalles de temps ε ; de l'autre, que le cout de simulation reste raisonnable, ce qui impose de ne pas prendre trop d'instants de discrétisation. L'équilibre précis entre ces eux contraintes dépend des exigences exactes du problème considéré ; dans ce cours, nous prendrons autour de 1 000 pas de discrétisation, ce qui est généralement une valeur appropriée.

Cette question de la discrétisation a son importance quand on voudra ensuite appliquer la méthode de Monte-Carlo à des processus aléatoires : car en plus de l'incertitude intrinsèque de la méthode de Monte-Carlo, il faudra aussi tenir compte de l'*erreur de discrétisation*, c'est-à-dire du fait que ce n'est pas tout-à-fait le vrai processus qu'on simule. Dans certains cas, l'erreur de discrétisation sera très petite par rapport à l'erreur de Monte-Carlo, au point qu'on pourra la négliger et faire comme si la simulation était exacte ; mais encore faut-il être capable de dire si on est effectivement dans un tel cas... Pour plus de rigueur, il faut disposer d'*estimations sur l'erreur* de discrétisation : nous verrons quelques telles estimations dans la suite.

Méthodes de simulation

Définition 5.6. La *méthode d'Euler stochastique* pour simuler $(B_{t_0=0}, B_{t_1}, \dots, B_{t_k})$ (où $t_0 < t_1 < \dots < t_k$ et les $t_{i+1} - t_i$ sont très petits) consiste à simuler successivement $B_{t_1}, B_{t_2}, \dots$ en utilisant que $B_{t_{i+1}} - B_{t_i}$ est indépendant du passé, de loi $\mathcal{N}\big((t_{i+1}-t_i)\sigma^{\mathsf{T}}\sigma\big)$.

Exemple 5.7. Les programmes `brownien1D.m` et `brownien2D.m` de l'annexe **??** implémentent la méthode d'Euler stochastique pour simuler respectivement des pouvement browniens 1- et 2-dimensionnels. Leurs rendus correspondent aux figures 5.1 et 5.2 présentées un peu plus haut.

Définition 5.8. La *méthode du point médian* permet de raffiner une simulation déjà effectuée en calculant la valeur du processus en des points de discrétisation intermédiaires sans modifier la simulation des points de discrétisation déjà calculés. Celle-ci consiste à observer que la propriété de Markov implique que la loi de $(B_t)_{t\in[t_1,t_2]}$ ne dépend que de B_{t_1} et B_{t_2} et pas de ce qui se passe pour $t < t_1$ ou $t > t_2$, puis à utiliser la loi de $B_{(t_1+t_2)/2}$ conditionnellement à B_{t_1} et B_{t_2}, donnée par la proposition qui suit.

Proposition 5.9. *Conditionnellement à B_{t_1} et B_{t_2}, $B_{(t_1+t_2)/2}$ suit la loi $(B_{t_1}+B_{t_2})/2+\mathcal{N}\big((t_2-t_1)\sigma^{\mathsf{T}}\sigma\,/\,4\big)$.*

Erreur de simulation

Théorème 5.10. *Soit $(B_t)_{t\geqslant 0}$ un mouvement brownien unidimensionnel de variance par unité de temps σ^2. Conditionnellement à B_{t_1} et B_{t_2}, la probabilité que le véritable*

mouvement s'écarte de plus de ε de son interpolation linéaire sur l'intervalle $[t_1, t_2]$ est majorée par $2\exp\bigl(-2\varepsilon^2 \,/\, (t_2 - t_1)\sigma^2\bigr)$.

Remarque 5.11. La borne du théorème ci-dessus étant expnentielle, il suffit donc de prendre $t_2 - t_1$ nettement plus petit que $\varepsilon^2 \,/\, \sigma$ pour être assuré que la probabilité d'une erreur supérieure à ε soit infime.

5.1.3 Propriétés

Loi du mouvement brownien

Proposition 5.12. *Pour tout $(t_1, \dots, t_k) \in (\mathbf{R}_+)^k$, le k-uplet $(B_{t_1}, \dots, B_{t_k})$ est un vecteur gaussien centré dont la matrice de covariance est donnée (par blocs) par*

$$\mathrm{Cov}(B_{t_i}, B_{t_j}) = (t_i \wedge t_j)\sigma^{\mathsf{T}}\sigma.$$

Proposition 5.13. *Si σ est une matrice de $\mathbf{R}^{d\times d}$ et $(B_t)_{t\geqslant 0}$ un mouvement brownien standard, alors $(B_t\sigma)_{t\geqslant 0}$ est un mouvement brownien de covariance par unité de temps $\sigma^{\mathsf{T}}\sigma$.*

Proposition 5.14. *Si $(B^1_t)_{t\geqslant 0}, \dots, (B^d_t)_{t\geqslant 0}$ sont des mouvements browniens unidimensionnels standard indépendants, alors $(B^1_t, \dots, B^d_t)_{t\geqslant 0}$ est un mouvement brownien d-dimensionnel standard.*

Propriétés du processus

Théorème 5.15. *Le mouvement brownien est un processus markovien. En outre, on connait exactement la loi conditionnelle de son futur :*

$$Loi\bigl((B_{t_1+u})_{u\geqslant 0}\big|B_{t_1} = x\bigr) \sim \bigl(x + \tilde{B}_u\bigr)_{u\geqslant 0},$$

où $(\tilde{B}_u)_{u\geqslant 0}$ suit la loi d'un (autre) mouvement brownien de même variance par unité de temps.

Théorème 5.16. *Pour tout $t_1 \geqslant 0$, $(B_{t_1+u} - B_{t_1})_{u\geqslant 0}$ suit la même loi que $(B_u)_{u\geqslant 0}$: on dit que le mouvement brownien est* à accroissements stationnaires.

Théorème 5.17. *Soit B un mouvement brownien unidimensionnel. Si T est un temps d'arrêt* borné, *alors $\mathbb{E}(B_T) = B_0$: on dit que le mouvement brownien vérifie la* propriété de martingale (globale).

Corolaire 5.18. *Soit B un mouvement brownien unidimensionnel. Pour $M < \infty$, notons τ le premier instant t pour lequel $|B_t| \geqslant M$. Ce processus définit un temps d'arrêt ; notons $\tilde{B}_t := B_{t\wedge\tau}$ le « mouvement brownien stoppé au temps τ ». Alors pour tout temps d'arrêt T fini presque-sûrement, $\mathbb{E}(\tilde{B}_T) = B_0$: on dit que le mouvement brownien vérifie la* propriété de martingale locale.

Comportement des trajectoires

Théorème 5.19. *Avec probabilité* 1, *les trajectoires du mouvement brownien sont continues.*

5.2 Équations différentielles stochastiques

5.2.1 Le bruit blanc gaussien

☛ *Ce paragraphe est présenté dans un but purement culturel ; la suite de la section doit être lue sans en tenir compte.*

Le mouvement brownien n'est pas seulement la star de tous les processus aléatoires, c'est aussi un objet qui est au cœur de toute la théorie des équations différentielles stochastiques, à laquelle nous allon maintenant donner une introduction. Plus exactement, ce n'est pas le mouvement brownien lui-même qui est l'objet fondamental, mais la *dérivée* du mouvement brownien, qu'on appelle *bruit blanc gaussien.*

« *Quelle* dérivée ?! » protesterez-vous. « Le mouvement brownien n'est pas dérivable ! ». Certes, pas au sens classique. Mais il est toutefois dérivable *au sens des distributions.* Le bruit blanc gaussien est donc une *distribution aléatoire* sur $\mathbf{R}$. Mais comme ce ne serait pas un objet pratique à manipuler, on préfère en général tout écrire en termes de son intégrale, à savoir le mouvement brownien, qui lui est une "brave" fonction continue.

Nous donnons quand même la définition et les propriétés fondamentales du bruit blanc gaussien, pour la culture mathématique. Notez que le bruit blanc gaussien se définit en général sur tout $\mathbf{R}$ et pas seulement sur $\mathbf{R}_+$.

Définition 5.20. On appelle *bruit blanc gaussien* (unidimensionnel) d'intensité σ^2 une distribution aléatoire T sur $\mathbf{R}$ (à valeurs réelles) telle que, pour toute fonction-test $\varphi \in \mathcal{D}(\mathbf{R})$, on ait $\langle T, \varphi\rangle \sim \mathcal{N}\big(\int_{\mathbf{R}} \varphi(x)^2\,dx\big)$. Nous admettrons que, sous réserve que donner une bonne définition formelle pour ce qu'est une distribution aléatoire, cela définit bien la loi de T de façon unique.

Théorème 5.21. *Si T est un bruit blanc gaussien,*

1. *Les valeurs de T sur des ouverts disjoints sont indépendantes, càd. que si $\varphi_1, \dots, \varphi_k$ sont des fonctions-test à supports disjoints, $\langle T, \varphi_1\rangle, \dots, \langle T, \varphi_k\rangle$ sont indépendants.*
2. *T est invariante par translation, càd. que pour tout $a \in \mathbf{R}$, $\boldsymbol{\delta}_a * T$ a la même loi que T.*

Démonstration. Il suffit de revenir à la définition. □

5.2.2 Principe général des équations différentielles stochastiques

Définition 5.22. Une équation différentielle stochastique (markovienne) à valeurs dans $\mathbf{R}^n$ est une équation de la forme

$$dX_t = dW_t\sigma(X_t) + b(X_t)dt,$$

où "dW_t" note l'incrément d'un mouvement brownien standard de $\mathbf{R}^d$ (on aura généralement $d = n$, car on peut toujours se ramener à ce cas), et $\sigma\colon \mathbf{R}^n \to \mathbf{R}^{m\times n}$ et $b\colon \mathbf{R}^n \to \mathbf{R}^n$ sont certaines fonction déterministes (continues).

Nous ne chercherons pas ici à savoir sous quelles conditions une telle équation a un sens. On simulera les solutions de cette équation par la méthode d'Euler stochastique, consistant à simuler successivement $X_0, X_\varepsilon, X_{2\varepsilon}, \ldots$ en assimilant « $dX_{n\varepsilon}$ » à « $X_{n\varepsilon+\varepsilon} - X_{n\varepsilon}$ », etc.

Remarque 5.23. Quand nous simulions le mouvement brownien, il y avait une erreur due à la discrétisation de temps (l'interpolation linéaire entre deux points de discrétisation ne correspondant pas exactement au vrai mouvement brownien), mais en revanche la simulation aux points de discrétisation était *exacte*. Quand nous simulons une équation différentielle stochastique par la méthode d'Euler stochastique, en revanche, *même les points de discrétisation ne sont pas simulés exactement*, car le schéma d'Euler n'est valable en toute rigueur qu'*asymptotiquement*, quand ε tend vers 0.

Il faut donc *aussi* tenir compte de cette *erreur de simulation* quand nous appliquons la méthode de Monte-Carlo à des processus aléatoires. Comme dans le cas du mouvement brownien, on dispose de bornes permettant de contrôler cette erreur de simulation, mais celles-ci ne sont pas simples. *En pratique*, comme pour le mouvement brownien, on supposera que l'erreur de simulation est bien plus faible que l'erreur de Monte-Carlo, et on fera donc comme si la simulation était exacte ; mais il faudra garder à l'esprit que cette hypothèse n'est pas toujours vérifiée et peut donc conduire à des erreurs dans certaiens situations.

Remarque 5.24. La notation « dW_t » dans l'équation différentielle stochastique laisse à penser qu'on a besoin de simuler le mouvement brownien $(W_t)_{t\geqslant 0}$ pour appliquer le schéma d'Euler. C'est certes une possibilité, mais ce serait inutilement lourd ; car l'objet dW_t (qui désigne les accroissements du mouvement brownien) est en fait *plus simple* que le mouvement brownien lui-même : par définition du mouvement brownien en effet, les dW_t sont indépendants et de loi $\mathcal{N}(\varepsilon)$ (où ε est la largeur du pas de temps concerné). Simuler $(W_t)_{t\geqslant 0}$ par la méthode d'Euler stochastique avant de prendre ses accroissements reviendrait à simuler les dW_t, à les sommer pour avoir le mouvement brownien, puis à prendre les différences pour... récupérer les incréments qu'on avait déjà simulés, ce qui est clairement idiot. (Au sujet de l'importance intrinsèque de $(dW_t)_{t\geqslant 0}$, voir aussi la § 5.2.1 sur le bruit blanc gaussien).

Exemple 5.25. Des exemples d'implémentation de simulation de processus stochastiques sont donnés par les progammes `browngeom.m` et `OU.m` de l'annexe **??** ; ils simulent respectivement un mouvement brownien géométrique et un processus d'Ornstein-Uhlenbeck, dont les définitions seront données dans la § 5.2.4. Les rendus de ces programmes correspondent respectivement aux figures 5.3 et 5.4.

5.2.3 Changement de variables

Nous admettrons la formule de changement de variables suivante, appelée *formule d'Itô* :

Théorème 5.26. *Soit un processus X_t suivant l'équation différentielle stochastique « $dX_t = dW_t\sigma(X_t) + b(X_t)dt$ ». Si $f\colon \mathbf{R}^n \to \mathbf{R}^m$ est de classe $\mathcal{C}^2$, alors $Y_t = f(X_t)$ suit l'équation différentielle stochastique :*

$$dY_t = Df(X_t)\cdot\big(dW_t\sigma(X_t)\big) + \big[Df(X_t)\cdot b(X_t) + \tfrac{1}{2}D^2f(X_t)\cdot\big(\sigma(X_t),\sigma(X_t)\big)\big]dt.$$

En particulier si $n, m = 1$, cela dit que si X_t suit l'équation différentielle stochastique « $dX_t = \sigma(X_t)dW_t + b(X_t)dt$ », alors $Y_t = f(X_t)$ vérifie :

$$dY_t = f'(X_t)\sigma(X_t)dW_t + [f'(X_t) + \tfrac{1}{2}f''(X_t)\sigma(X_t)^2]dt.$$

On voit ainsi que la formule d'Itô s'apparente à un changement de variables classique, à ceci près qu'il apparait en plus un terme de dérive faisant intervenir la dérivée seconde de f, à savoir $\frac{1}{2}D^2 f(X_t) \cdot \big(\sigma(X_t), \sigma(X_t)\big)dt$ (ce terme est appelé le *terme d'Itô*). D'où vient ce terme, et comment retenir facilement la formule ? En fait, une formule de changement de variables classique correspond à un développement de Taylor limité à l'ordre 1. C'est la même chose ici, à ceci près qu'il faut savoir que le terme dW_t soit être considéré comme en terme *d'ordre* 1/2... Du coup, pour obtenir le développement limité à l'ordre 1, il faut pousser jusqu'aux dérivées secondes et tenir compte du terme en $dW_t \otimes dW_t$ qui apparait : la formule d'Itô dit en substance que ce terme est égal à $\boldsymbol{I}_d dt$.

5.2.4 Exemples

1. Si $\sigma(X_t)$ et $b(X_t)$ sont constants, la solution de l'équation différentielle stochastique (qui est alors évidemment égale à $X_0 + W_t + bt$) s'appelle un *mouvement brownien avec dérive*, de variance par unité de temps $\sigma\sigma^\mathsf{T}$ et de vitesse de dérive b.
2. On appelle *mouvement brownien géométrique* de paramètres σ^2 et b la solution de l'équation différentielle stochastique :

   $$dX_t = \sigma X_t dW_t + (b + \sigma^2/2)X_t dt.$$

   D'après la formule d'Itô, si $(B_t)_{t\geqslant 0}$ est un mouvement brownien unidimensionnel avec dérive de paramètres σ^2 et μ, alors $(e^{B_t})_{t\geqslant 0}$ est un mouvement brownien géométrique de mêmes paramètres.
3. La solution de l'équation différentielle stochastique (en dimension 1)

   $$dX_t = \sigma dW_t - \lambda X_t dt \qquad (\lambda > 0)$$

   est appelée *processus d'Ornstein-Uhlenbeck* de paramètres σ et λ. Si on part d'une condition initiale X_0 distribuée selon la loi $\mathcal{N}(\sigma^2 \,/\, 2\lambda)$ (étant entendu que le bruit blanc gaussien est indépendant de cette condition initiale), alors on peut montrer ce processus est « stationnaire », càd. que pour tout $t_1 \geqslant 0$, $(X_{t_1+t})_{t\geqslant 0}$ a la même loi que $(X_t)_{t\geqslant 0}$. On dit que la loi $\mathcal{N}(\sigma^2 \,/\, 2\lambda)$ est une *mesure d'équilibre* du processus.

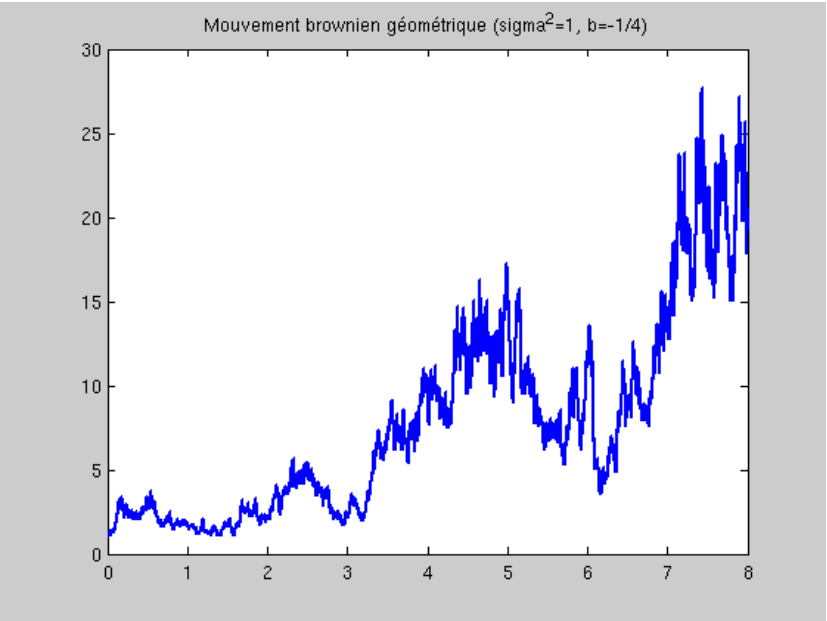

FIGURE 5.3 – Mouvement brownien géométrique. Remarquez qu'on voit bien comme le bruit est proportionnel à la valeur du processus.

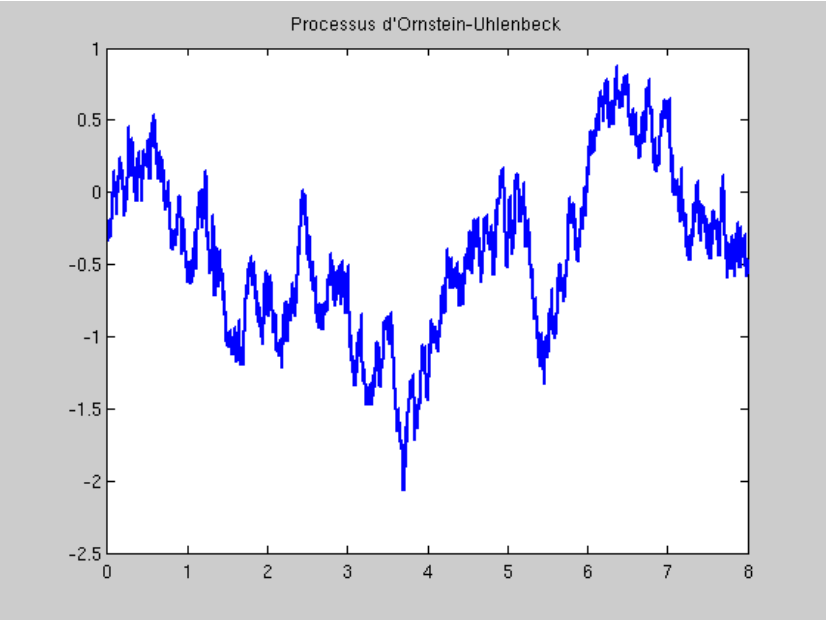

FIGURE 5.4 – Processus d'Ornstein-Uhlenbeck. On devine sur ce dessin qu'une "force" empêche le processus de trop s'éloigner de zéro...

Chapitre 6

Autres types de processus stochastiques

6.1 Processus à sauts

6.1.1 Processus de Poisson simple

Définition 6.1. Le processus de Poisson d'intensité λ est un processus de $\mathbf{R}_+$ dans $\mathbf{N}$ qui augmente par incréments d'une unité. L'occurrence (ou pas) d'incréments à différents instants est totalement indépendante, et la probabilité qu'un incrément se produise à l'instant t vaut λdt. En d'autres termes, les intervalles de temps entre deux incréments successifs sont indépendants et suivent chacun une loi *Exponentielle*(λ).

Remarque 6.2. Le processus de Poisson est la star des processus à sauts, de même que le mouvement brownien était la star des processus continus. Et de même que c'était plus le bruit blanc gaussien que le mouvement brownien lui-même qui était l'objet fondamental, ici c'est l'ensemble des points où un saut se produit qui est le plus intéressant : c'est ce qu'on appelle un *processus ponctuel de Poisson* (d'intensité λ).

6.1.2 Processus de Poisson composés

Si λ est une mesure de masse totale finie sur $\mathbf{R}$, le processus de Poisson composé d'intensité λ est un processus de $\mathbf{R}_+$ dans $\mathbf{R}$ qui évolue par sauts, de la façon suivante : l'occurrence (ou pas) de sauts et la valeur éventuelle de ces sauts à différents instants est totalement indépendante, et la probabilité qu'un saut d'amplitude y se produise à l'instant t vaut $d\lambda(y)dt$. En d'autres termes, notant Λ la masse totale de λ, les intervalles de temps entre deux sauts successifs sont indépendants et suivent chacun une loi Λ, et sachant les instants de sauts, les amplitudes des différents sauts sont i.i.d. suivant la loi $\Lambda^{-1}\lambda$.

Annexe A

Annexe : Codes MATLAB

A.1 Le championnat de basket

A.1.1 Détails techniques

Le championnat national de basketball met en lice 16 =: Z équipes. Chacune de ces équipes rencontre chaque autre une fois et une seule. À la fin du championnat, l'équipe qui a remporté le plus de victoires [*] gagne le championnat. Dans l'hypothèse où plusieurs équipes sont premières ex-æquo au nombre de victoires, la gagnante est tirée au sort uniformément parmi ces premières ex-æquo.

La modélisation de notre passionné consiste à attribuer à chaque équipe un numéro de 1 (la plus forte sur le papier) à 16 (la moins forte). L'équipe de Nancy est associée au nombre 3 =: *Nancy*. Lorsque l'équipe i rencontre l'équipe j, la probabilité que ce soit i qui gagne vaut alors $i^{-1/2} \,/\, (i^{-1/2} + j^{-1/2}) =: p_{vict}(i,j)$ [†]. En outre, comme nous l'avons déjà dit, les résultats des différents matchs sont indépendants.

A.1.2 Méthode de rejet : `basket_rejet.m`

```
% La fonction "basket_rejet" estime la probabilité de victoire de l'équipe
% de Nancy par la méthode de rejet.
function basket_rejet ()
% "Nsim" est le nombre de simulations à effectuer.
Nsim = 60000;
% "NANCY" est le numéro associé à l'équipe de Nancy.
NANCY = 3;
% bilan est le nombre de simulations où l'équipe de Nancy a été championne.
bilan = 0;
% On lance les simulations
for i = 0 : Nsim - 1
    % On lance la fonction "championne" qui simule la vainqueure du
    % championnat. S'il s'agit de Nancy, on ajoute 1 à "bilan" ; sinon on
    % lui ajoute 0.
    bilan = bilan + (championne () == NANCY);
end
```

[*]. Il n'y a pas de match nul au basketball.

[†]. Cette modélisation est bien cohérente, dans la mesure où $p(i,j) + p(j,i) = 1$.

```
% "proportion" est la proportion de victoires de Nancy sur cet ensemble de
% simulations.
proportion = bilan / Nsim;
% On affiche le résultat.
fprintf ('La probabilité estimée de victoire pour l''équipe de Nancy ');
fprintf ('est de %f %%.\n', 100 * proportion);
return
end
```

A.1.3 Variante : basket_14.m

```
% La fonction "basket_14" est identique à la fonction "basket_rejet", à
% ceci près que Nancy porte le no 14, et qu'on calcule aussi l'intervalle
% de confiance.
function basket_14 ()
Nsim = 60000;
NANCY = 14;
bilan = 0;
for i = 0 : Nsim - 1
    bilan = bilan + (championne () == NANCY);
end
proportion = bilan / Nsim;
ect_estr = sqrt ((proportion * (1 - proportion)) / Nsim);
fprintf ('La probabilité de victoire pour de Nancy est ');
fprintf ('%f ± %f %%.\n', 100 * proportion, 1.96 * 100 * ect_estr);
return
end
```

A.1.4 Fonction auxiliaire championne.m

```
% La fonction "championne" simule le championnat de basket et renvoie le
% numéro associé à l'équipe championne.
function ch = championne ()
% On désigne par "Neq" le nombre d'équipes en lice.
Neq = 16;
% "victoires(a)" est le nombre de matchs remportés par l'équipe a.
victoires = zeros (1, Neq);
% On fait jouer chaque équipe contre l'autre une fois.
for i = 1 : Neq
    for j = i + 1 : Neq
        % On simule le résultat du match, et on met à jour "victoires" en
        % fonction.
        if rand () < pvict (i, j)
            victoires(i) = victoires(i) + 1;
        else
            victoires(j) = victoires(j) + 1;
        end
    end
end
% Une méthode pour départager uniformément les ex-æquo est d'ajouter une
% partie fractionnaire aléatoire à chaque score.
victoires = victoires + rand (1, Neq);
% On regarde qui a le plus de victoires.
[~, ch] = max (victoires);
```

```
% On renvoie l'identité de la championne.
return
end
```

A.1.5 Fonction auxiliaire `pvict.m`

```
% La fonction "pvict" prend en argument les numéros associés à deux équipes
% "i" et "j" qui s'affrontent, et renvoie la probabilité que ce soit i qui
% gagne.
function p = pvict (i, j)
% On calcule la formule du modèle.
p = i ^ (-1 / 2) / (i ^ (-1 / 2) + j ^ (-1 / 2));
% On renvoie le résultat.
return
end
```

A.1.6 Calcul de la densité dP/dQ

L'univers que nous considérons pour ce calcul de densité est $\Omega := \{0,1\}^E$, où $E := \{(i,j) \in \{1,\dots,16\}^2 \,:\, i<j\}$. Pour $\omega \in \Omega$, les résultats du championnat sous cette éventualité sont décrits en disant que pour $(i,j) \in E$, $\omega_{(i,j)}$ vaut 1 si l'équipe n$^{\text{o}}$ i a battu l'équipe n$^{\text{o}}$ j lors de leur confrontation, et 0 si c'est n$^{\text{o}}$ j qui a battu n$^{\text{o}}$ i. Sous la loi P comme sous la loi Q, les $(\omega_e)_{e\in E}$ sont indépendants, de sorte que

$$\frac{dP}{dQ}\big((\omega_e)_{e\in E} = (r_e)_{e\in E}\big) = \prod_{e\in E} \frac{d(\omega_e * P)}{d(\omega_e * Q)}(b_e) = \prod_{e\in E} \frac{dP(\omega_e = b_e)}{dQ(\omega_e = b_e)}.$$

Par définition de Q, on a :

$$\frac{P(\omega_{(i,j)}=1)}{Q(\omega_{(i,j)}=1)} = \begin{cases} 1 & \text{si } i,j \neq 14; \\ (1+j^{-1/2}) \,/\, (1+\sqrt{14}j^{-1/2}) & \text{si } i = 14\,[\ddagger]; \\ (i^{-1/2}+1) \,/\, (i^{-1/2}+1/\sqrt{14}) & \text{si } j = 14; \end{cases}$$

et

$$\frac{P(\omega_{(i,j)}=0)}{Q(\omega_{(i,j)}=0)} = \begin{cases} 1 & \text{si } i,j \neq 14; \\ (1+j^{-1/2}) \,/\, (1/\sqrt{14}+j^{-1/2}) & \text{si } i = 14; \\ (i^{-1/2}+1) \,/\, (\sqrt{14}i^{-1/2}+1) & \text{si } j = 14; \end{cases}$$

Au final,

$$\frac{dP}{dQ}\big((\omega_e)_{e\in E} = (r_e)_{e\in E}\big) = \prod_{i<14} \frac{P(\omega_{(i,14)} = r_{(i,14)})}{Q(\omega_{(i,14)} = r_{(i,14)})} \prod_{j>14} \frac{P(\omega_{(14,j)} = r_{(14,j)})}{Q(\omega_{(14,j)} = r_{(14,j)})}.$$

Remarque A.1. Comme la densité dP/dQ apparait comme un produit de fonctions des ω_e, et que les ω_e sont simulés indépendamment, on va pouvoir calculer la densité en même temps qu'on simule les ω_e, en maintenant un produit à jour.

[‡]. Explication du calcul (il en va de même pour les calculs analogues) :

$$\frac{P(\omega_{(i,j)}=1)}{Q(\omega_{(i,j)}=1)} = \frac{i^{-1/2} \,/\, (i^{-1/2}+j^{-1/2})}{1^{-1/2} \,/\, (1^{-1/2}+j^{-1/2})} = \frac{1+j^{-1/2}}{1+i^{1/2}j^{-1/2}} = \frac{1+j^{-1/2}}{1+\sqrt{14}j^{-1/2}}.$$

A.2 Le Keno

A.2.1 Détails techniques

Le jeu de Keno se joue sur une grille comportant les numéros de 1 à 64. Le joueur doit choisir 16 de ces numéros, dont un auquel il attribue la valeur 1, un auquel il attribue la valeur 2, ..., et un auquel il attribue la valeur 16. Lors du tirage, la loterie nationale tire au hasard 16 boules parmi un jeu de boules numérotées de 1 à 64. Le *score* d'un joueur est alors défini comme la somme de valeurs qu'il a associées aux numéros qu'il a cochés et qui ont effectivement été tirés. La gain est ensuite défini de la façon suivante. En-dessous d'un score de 50, le joueur ne gagne rien. Pour un score de 50, le joueur gagne 5 €. Puis, à chaque point de score supplémentaire, le joueur augmente son gain de 10 %, en arrondissant à chaque fois au nombre entiers d'euros le plus proche (arrondi par excès en cas de demi-entier) : ainsi un score de 51 points rapporte 6 €, un score de 52 points rapporte 7 €, ..., un score de 60 points rapporte 15 €, un score de 61 points rapporte 17 €, ..., et un score parfait de 136 points rapporte 22 470 € !

A.2.2 Premier code : `keno_simple.m`

```
% La fonction "keno_simple" estime l'espérance de gain d'un joueur de Keno
% par la méthode de Monte-Carlo simple.
function keno_simple ()
% "Ncoch" est le nombre de numéros à cocher.
Ncoch = 16;
% "Nsim" est le nombre de simulations effectuées.
Nsim = 1000000;
% "total" est le total des résultats obtenus au cours des diverses
% simulations.
total = 0;
% On lance la boucle de simulations.
for i = 1 : Nsim
    % On appelle "carton" la donnée des numéros cochés par le joueur,
    % "carton(p)" désignant le numéro choisi à p points. Puisque la loi du
    % gain du joueur est la même quels que soient les numéros cochés, on
    % utilisera toujours le même carton.
    carton = 1 : Ncoch;
    % À l'aide de la fonction "tirage" pour simuler le tirage, de la
    % fonction "score" pour calculer le score à partir de "carton" et
    % "tirage", et de la fonction "gain" qui associe le gain correspondant
    % au score, on détermine la quantité d'argent gagnée par le joueur,
    % qu'on ajoute à "total".
    total = total + gain (score (carton, tirage ()));
end
% "moyenne" est le gain moyen, qui estime l'espérance de gain.
moyenne = total / Nsim;
% On affiche le résultat.
fprintf ('L''espérance de gain du Keno est estimée à ');
fprintf ('%f €.\n', moyenne);
% On quitte le programme.
return
end
```

A.3 Avec intervalle de confiance : keno_IC.m

```
% La fonction "keno_IC" estime l'espérance de gain d'un joueur de Keno
% par la méthode de Monte-Carlo, en donnant un intervalle de confiance.
% Nous ne commentons que les différences par rapport à "keno_simple".
function keno_IC ()
Ncoch = 16;
Nsim = 1000000;
total = 0;
% Pour calculer l'intervalle de confiance on a aussi besoin de calculer le
% total des carrés des quantités simulées.
total_carres = 0;
for i = 1 : Nsim
    carton = 1 : Ncoch;
    % "legain" est le gain obtenu par le joueur. Comme on va l'utiliser
    % deux fois, on doit le stocker dans une variable d'abord.
    legain = gain (score (carton, tirage ()))
    total = total + legain;
    % On met aussi à jour "total_carres".
    total_carres = total_carres + legain * legain;
end
moyenne = total / Nsim;
% On détermine aussi la variance (estimée) du gain.
variance_gain = total_carres / Nsim - moyenne * moyenne;
% On en déduit l'écart-type (estimé) de la moyenne empirique
ect_estimtr = sqrt (variance_gain / Nsim);
% On calcule les coefficients pour l'intervalle de confiance - noter qu'on
% peut aussi faire cette étape au préalable et juste entrer les résultats
% numériquement; en l'occurrence c0 vaut -1.65 et c1 vaut +3.10.
c0 = sqrt (2) * erfinv (-.90);
c1 = sqrt (2) * erfinv (+.998);
% On affiche le résultat.
fprintf ('Avec un risque de 1 %% pour la sous-estimation, )
fprintf ('resp. de 5 %% pour la surestimation,\n');
fprintf (L''espérance de gain du Keno se situe dans l'intervalle ');
fprintf ('[%f, %f] €', moyenne + c0 *ect_estimtr, ...
    moyenne + c1 * ect_estimtr);
return
end
```

A.3.1 Fonction auxiliaire : tirage.m

```
% La fonction "tirage" simule le tirage du jeu de Keno. Le résultat est
% renvoyé sous la forme d'un vecteur de "t" constitué de true et de false,
% t(n) étant vrai si et seulement si le numéro n a été tiré.
function t = tirage ()
% "Nnum" note le nombre total de numéros existant.
Nnum = 64;
% "Ntir" désigne le nombre de numéros à tirer.
Ntir = 16;
% "t" est le tirage qui va être effectué.
t = zeros (1, Nnum);
for i = 1 : Ntir
    % Boucle de tirage du numéro.
```

```
    while (true)
        % "numero" est le numéro tiré, à priori uniforme entre 1 et Nnum.
        numero = ceil (rand () * Nnum);
        % Cependant, on ne peut accepter ce numéro que s'il est différent
        % de ceux déjà tirés; sinon il faut recommencer.
        if (~t(numero))
            t(numero) = true;
            break;
        end
    end
end
return
end
```

A.3.2 Fonction auxiliaire : `score.m`

```
% La fonction "score" établit le score d'un carton "carton" en fonction du
% tirage "tirage". Rappelons que "carton(p)" désigne le numéro que le
% joueur a coché pour p points, et que tirage(n) est vrai si et seulement
% si le numéro n a été tiré.
function s = score (carton, tirage)
% "Ncoch" note le nombre de numéros à cocher.
Ncoch = 16;
% "s" sera le score obtenu.
s = 0;
% On regarde un par un si chacun des numéros cochés a été tiré, et on
% ajuste le score en conséquence.
for p = 1 : Ncoch
    if tirage(carton(p))
        s = s + p;
    end
end
% On renvoie le résultat.
return
end
```

A.3.3 Fonction auxiliaire : `gain.m`

```
% La fonction "gain" établit le gain en fonction du score "score", selon la
% règle édictée par la loterie nationale.
function g = gain (score)
% "seuil" est le seuil à partir duquel on gagne un gain.
seuil = 50;
if score < 50
    g = 0;
else
    g = 5;
    for i = seuil + 1 : score
        g = floor ((11 * g + 5) / 10);
    end
end
return
end
```

A.3.4 Calcul de l'espérance conditionnellement à $\mathcal{F}_{15}$

Dans cette sous-section nous explicitons la valeur de l'espérance conditionnelle du gain sous la tribu $\mathcal{F}_{15}$, c'est-à-dire lorsqu'on sait déjà quels numéros ont été tirés ou pas parmi les numéros cochés de 1 à 15 points. Nous reprenons les notations du texte principal. Notons $\bar{\omega}$ la donnée de $(\omega_1, \ldots, \omega_{15})$ dont nous disposons après la quinzième étape de la simulation du tirage du Keno ; notons $\bar{n}_{\text{sorties}}$ le nombre de boules sorties parmi les numéros cochés de 1 à 15 (càd. $\bar{n}_{\text{sorties}} := \sum_{i=1}^{15} \omega_i$), et $\bar{s}$ la part du score due à ces boules (càd. $\bar{s} := \sum_{i=1}^{15} \omega_i i$). Notons que $\bar{n}_{\text{sorties}}$ et $\bar{s}$ sont toutes les deux des fonctions de $\bar{\omega}$, autrement dit des v.a. $\mathcal{F}_{15}$-mesurables. Sous la loi conditionnelle relativement à $\mathcal{F}_{15}$, nous savons que $\bar{n}_{\text{sorties}}$ boules ont été tirées parmi les numéros cochés de 1 à 15, mais nous ne savons rien concernant les autres numéros, de sorte qu'il nous reste $16 - \bar{n}_{\text{sorties}}$ boules à tirer parmi les 49 numéros restants. Ainsi, la probabilité conditionnelle que la boule cochée pour 16 points soit tirée vaut $(16 - \bar{n}_{\text{sorties}}) \,/\, 49$ — et la probabilité qu'elle ne soit pas tirée vaut donc $(33 + \bar{n}_{\text{sorties}}) \,/\, 49$. Sous la première éventualité, le score total sera de $\bar{s} + 16$, et sous la seconde, de $\bar{s}$; ce qui correspond, en notant g la fonction qui associe le gain à un score donné, à des gains respectifs de $g(\bar{s} + 16)$ et $g(\bar{s})$. Finalement, l'espérance conditionnelle est donc :

$$\mathbb{E}(gain) = \frac{33 + \bar{n}_{\text{sorties}}}{49} g(\bar{s}) + \frac{16 - \bar{n}_{\text{sorties}}}{49} g(\bar{s} + 16).$$

A.4 Le lapin de Douady

A.4.1 Détails techniques

Définition A.2. Dans cette section, c désigne l'unique solution complexe dont la partie imaginaire soit strictement positive de l'équation :

$$c^3 + 2c^2 + c + 1 = 0.$$

On a $c \simeq -0{,}123 + 0{,}745i$.

Définition A.3. Un point $(x, y) \in \mathbf{R}^2$, représenté par le nombre complexe $z := x + yi \in \mathbf{C}$, appartient au lapin de Douady si et seulement si la suite $(u_n^{(z)})_{n \in \mathbf{N}}$ définie par :

$$\begin{cases} u_0^{(z)} := z ; \\ u_{n+1}^{(z)} := z^2 + c \end{cases}$$

ne diverge pas.

Un point $z \in \mathbf{C}$ appartient à l'ensemble de Douady si et seulement si la suite définie par $u_0 = z$ et $u_{n+1} = u_n^2 + c$ ne diverge pas : on peut montrer que cela est le cas dès qu'une itérée de la suite est inférieure à $1/4$ en module, qu'à l'inverse la suite diverge dès qu'une itérée dépasse 2 en module, et que presque-tout point de départ finit par se retrouver dans un de ces deux cas, ce qui permet de trancher. Le lapin de Douady a l'allure donnée par la figure 2.1. Comme celle-ci est très irrégulière, il n'est pas sûr qu'un maillage régulier soit le plus adapté ; nous allons donc procéder ici par méthode de Monte-Carlo.

A.4.2 Fonction principale : `lapin.m`

```
% La fonction "lapin" estime l'aire du lapin de Douady par la méthode de
% Monte-Carlo, en échantillonnant selon la loi uniforme sur le carré
% [-2,2]×[-2,2].
function lapin
% "Nsim" est le nombre de simulations.
Nsim = 15000000;
% "somme" est la somme des simulations effectuées.
somme = 0;
% Boucle de Monte-Carlo.
for i = 1 : Nsim
    % On tire z uniformément sur le carré [-2,2]×[-2,2].
    z = (-2 + rand () * 4) + (-2 + rand () * 4) * 1i;
    % Calcule la fonction dont on prendra l'espérance. La fonction
    % "enlapin" renvoie 1 si on est dans le lapin, et 0 sinon.
    somme = somme + enlapin (z) * 16;
end
% "moyenne" est la moyenne des simulations effectuées.
moyenne = somme / Nsim;
% Affichage du résultat.
fprintf ('L''aire du lapin de Douady est estimée à ');
fprintf ('%f.\n', moyenne);
end
```

A.4.3 Fonction auxiliaire `enlapin.m`

```
% La fonction "enlapin" dit si le point représenté par le nombre complexe
% "z" appartient ou non au lapin de Douady.
function reponse = enlapin (z)
% On calcule la valeur exacte de la constante apparaissant dans la formule
% d'itération, notée "c".
c = calculDeC ();
% On lance la boucle des itérations de z jusqu'à pouvoir déterminer si
% c'est un point du lapin ou non.
while true
    if (abs(z) < .25)
        % Cas où z initial appartenait au lapin.
        reponse = true;
        break;
    elseif (abs(z) > 2.)
        % Le z initial n'appartenait pas au lapin.
        reponse = false;
        break;
    else
        z = z * z + c;
    end
end
return
end
```

A.4.4 Fonction auxiliaire `calculDeC.m`

```
% La fonction "calculDeC" calcule, par la méthode de Newton, la constante
```

```
% "c", solution imaginaire positive de l'équation « z³+2z²+z+1=0 ».
function c = calculDeC ()
% On initialise "c" près de la bonne valeur.
c = -.1 + .7i;
% erreur_anc est l'ancienne valeur de l'erreur commise, initialisée à
% l'infini.
erreur_anc = +Inf;
while true
    % "P" est la valeur du polynôme au point c.
    P = c * c * c + 2 * c * c + c + 1;
    % "erreur" l'erreur commise, à savoir la valeur absolue de "P", qu'il
    % s'agit de rendre nulle.
    erreur = abs (P);
    % On n'est pas sûr que l'erreur va arriver à exactement zéro ; c'est
    % pourquoi il faut prévoir une sortie de la boucle dès que l'erreur ne
    % décroit plus.
    if (erreur >= erreur_anc)
        break;
    end
    % On prépare la future valeur de "erreur_anc".
    erreur_anc = erreur;
    % "Pp" est la dérivée du polynôme en c.
    Pp = 3 * c * c + 4 * c + 1;
    % Itération de la méthode de Newton.
    c = c - P / Pp;
end
return
end
```

A.5 SOS analyse complexe

A.5.1 Simulation de $\mathbb{P}$

Nous allons non seulement expliquer comment simuler la loi $\mathbb{P}$ choisie dans l'exemple 2.21, mais aussi éclaircir d'où vient le choix de cette loi.

L'idée est de partir d'une loi de Pareto : considérons ainsi la loi $\mathbb{P}_2$ sur $[1, +\infty)$ caractérisée par

$$\mathbb{P}_2(\bullet \geqslant x) = 1/x \qquad \forall x \geqslant 1.$$

L'avantage de cette fonction est qu'elle est très facile à simuler, puisque sa fonction de répartition complémentaire est immédiatement donnée par $\bar{F}(x) = 1/x$ sur $[1, +\infty)$; on peut alors appliquer la méthode de la fonction de répartition, observant que la bijection inverse de $\bar{F}$ est $\bar{F}^{-1} : p \mapsto 1/p$ sur $(0, 1)$, pour en déduire que si U est uniforme sur $[0, 1]$, alors $1/U$ suit la loi $\mathbb{P}_2$. En outre, en dérivant $\bar{F}$, on montre que $\mathbb{P}_2$ est une loi à densité telle que $d\mathbb{P}_2(x) = \mathbf{1}_{\{x \geqslant 1\}} 1/x^2 \, dx$.

Maintenant, on va modifier cette loi de Pareto pour en faire une loi sur $\mathbf{R}$ tout entier. Pour commencer, on considère la loi $\mathbb{P}_1$ qui est la mesure-image de $\mathbb{P}_2$ par $x \mapsto x-1$, càd. que si X est une v.a. de loi $\mathbb{P}_2$, $X-1$ sera une v.a. de loi $\mathbb{P}_1$. Cela permet immédiatement la simulation de $\mathbb{P}_2$à partir de celle de $\mathbb{P}_1$; quant à la densité de $\mathbb{P}_2$, on la calcule à l'aide

de la formule de changement de variables :

$$\mathbb{E}_1(\varphi) = \mathbb{E}_2\big(\varphi(X-1)\big) = \int_{\mathbf{R}} \varphi(x-1)\mathbf{1}_{\{x\geqslant 1\}} 1/x^2 \, dx \overset{y:=x-1}{=} \int \varphi(y)\mathbf{1}_{\{y\geqslant 0\}} 1/(y+1)^2 \, dy,$$

de sorte que $\mathbb{P}_2$ est une loi à densité, avec $d\mathbb{P}_1(y) = \mathbf{1}_{\{y\geqslant 0\}} 1/(y+1)^2 \, dy$.

On peut aussi transformer $\mathbb{P}_1$ en une loi sur $\mathbf{R}_-$, notée $\mathbb{P}_1^-$, qui serait l'image de $\mathbb{P}_1$ par $x \mapsto -x$ (à nouveau, la simulation à partir de $\mathbb{P}_1$ est immédiate) ; la formule de changement de variables nous donne que $d\mathbb{P}_1^-(x) = \mathbf{1}_{\{x\leqslant 0\}} 1/(|x|+1)^2 \, dx$ [§].

Enfin, pour fabriquer une loi sur $\mathbf{R}$ tout entier, on prendra la loi $\mathbb{P} := 1/2 \times \mathbb{P}_1 + 1/2 \times \mathbb{P}_1^-$, dont la densité se calcule immédiatement par $d\mathbb{P}(x) = 1/2 \times dPr_1(x) + 1/2 \times \mathbb{P}_1^-(x) = 1/2(|x|+1)^2 \, dx$. Quant à la simulation, il suffit de tirer d'abord une variable de Bernoulli de paramètre $1/2$, puis, si elle tombe sur 1, de faire une simulation de $\mathbb{P}_1^-$ (indépendante du premier tirage), et si elle tombe sur 0 de faire une simulation de $\mathbb{P}_1$. Pour cette deuxième étape, d'après la façon de simuler $\mathbb{P}_2$ et la définition de $\mathbb{P}_1$ et $\mathbb{P}_1^-$, nous voyons que si U est une loi uniforme sur $[0,1]$, $1/U - 1$ suivra la loi $\mathbb{P}_1$ et que $1 - 1/U$ suivra la loi $\mathbb{P}_1^-$. Au final, cela conduit au programme de la §A.5.2 ci-après.

A.5.2 Calcul de l'intégrale : `SOS.m`

```
function SOS ()
N = 4000000;
total = 0;
for i = 1:N
x = echantillonnage_SOS ();
y = integrande_SOS (x) / densite_SOS (x);
total = total + y;
end
moyenne = total / N;
fprintf ('L''intégrale I est estimée à ');
fprintf ('%f.\n', moyenne);
end
```

A.5.3 Fonction auxiliaire : `echantillonnage_SOS.m`

```
% La fonction "echantillonnage_SOS" simule une réalisation "x" de la loi P
% du polycopié.
function x = echantillonnage_SOS ()
% On distingue deux situations suivant la réalisation d'une loi de
% Bernoulli.
if (rand () < 1 / 2)
    x = 1 / rand () - 1;
else
    x = 1 - 1 / rand ();
end
return
end
```

[§]. On prendra garde à ne pas oublier que pour une intégrale "au sens de la mesure", qui n'a pas de sens de parcours, le changement de variables $y = f(x)$ donne $dy = |Df|(x)$, avec une valeur absolue !

A.5.4 Fonction auxiliaire : densite_SOS.m

```
% La fonction "densite_SOS" calcule la densité (par rapport à la mesure de
% Lebesgue) "d" de la loi P au point "x".
function d = densite_SOS (x)
d = 1 / (2 * (1 + abs (x))^2);
return
end
```

A.5.5 Fonction auxiliaire : integrande_SOS.m

```
% La fonction "integrande_SOS" calcule la valeur "f" de la fonction à
% intégrer au point "x".
function f = integrande_SOS (x)
if (x == 0)
    % La fonction est prolongée analytiquement par 1 en x=0.
    f = 1;
else
    f = (x * x + 1) / (x * x + x + 1) * (sin (x) / x) ^ 2;
end
return
end
```

A.6 La centrale nucléaire

A.6.1 Détails techniques

La valeur moyenne des paramètres X, Y, Z, et leur matrice de covariance, sont respectivement

$$\vec{m} := \begin{pmatrix} -3 \\ -4 \\ -5 \end{pmatrix}; \qquad \boldsymbol{C} := \begin{pmatrix} 1 & 0{,}1 & 0{,}2 \\ 0{,}1 & 1 & 0{,}3 \\ 0{,}2 & 0{,}3 & 1 \end{pmatrix}$$

A.6.2 Algorithme de Monte-Carlo “naïf”

```
% La fonction "centrale_naif" évalue la probabilité quotidienne que
% survienne un accident, par une méthode de Monte-Carlo naïve. "Nsim" est
% le nombre de simulations demandées. Je ne commente que les parties non
% transparentes du code.
function centrale_naif (Nsim)
% Je donne aux variables "m", "C" et "R" une portée globale, afin de pouvoir
% aussi les utiliser dans la fonction "simuler_XYZ".
global m C R;
m = [-3 -4 -5];
C = [1, .1, .2; .1, 1, .3; .2, .3, 1];
R = chol(C);
Nacc = 0;
for i = 1:Nsim
    XYZ = simuler_XYZ ();
    accident = all (XYZ >= 0);
    Nacc = Nacc + accident;
end
```

```
p = Nacc / Nsim;
% Notez qu'ici on n'a pas eu besoin de créer une variable pour la somme des
% carrés des réalisations, car celle-ci est directement égale au nombre
% d'accidents, ce qui donne directement la variance (estimée) de notre
% variable de Bernoulli à partir de p.
variance_f = p - p * p;
ect_estr = sqrt (variance_f / Nsim);
fprintf ('La probabilité d''accident est estimée à ');
fprintf ('%e ± %e.\n', p, 1.96 * ect_estr);
fprintf ('(Note : Intervalle de confiance standard à 1,96 sigmas).\n');
return
end
```

A.6.3 Avec échantillonnage préférentiel

```
% La fonction "centrale_pref" applique la méthode d'échantillonnage
% préférentiel pour calculer le risque d'accident.
function centrale_pref (Nsim)
global m C R Cinv;
m = [-3 -4 -5];
C = [1, .1, .2; .1, 1, .3; .2, .3, 1];
R = chol(C);
% "Cinv" est la matrice inverse de C, utilisée pour le calcul de la
% densité.
Cinv = inv (C);
% La variable dont on estime l'espérance n'étant plus une indicatrice, il
% faut maintenir séparément la somme et la somme des carrés.
somme_f = 0;
somme_f2 = 0;
for i = 1: Nsim
    % Cette fois-ci on échantillonne XYZ selon la loi Q.
    XYZ = tirer_Q ();
    % La fonction dont on évalue l'espérance doit tenir compte de la
    % densité.
    f = all (XYZ >= 0) * densitePQ (XYZ);
    somme_f = somme_f + f;
    somme_f2 = somme_f2 + f * f;
end
moy = somme_f / Nsim;
var_f = somme_f2 / Nsim - moy * moy;
ect_estr = sqrt (var_f / Nsim);
fprintf ('La probabilité d''accident est estimée à ');
fprintf ('%e ± %e.\n', moy, 1.96 * ect_estr);
fprintf ('(Note : Intervalle de confiance standard à 1,96 sigmas).\n');
return
end
```

A.6.4 Fonction auxiliaire : `simuler_XYZ.m`

```
% La fonction "simuler_XYZ" simule un vecteur-ligne XYZ dont l'espérance
% est m et la matrice de covariance C.
function XYZ = simuler_XYZ
% On indique à Matlab que "m" et "R" ont été définies dans la
% fonction-mère.
```

```
global m R;
% randn(1,3) simule un 3-vecteur ligne de v.a. i.i.d. N(0,1).
XYZ = m + randn (1, 3) * R;
return;
end
```

A.6.5 Fonction auxiliaire : `tirer_Q.m`

```
% "tirer_Q ()" simule un vecteur-ligne "XYZ" de loi Q.
function XYZ = tirer_Q
global R;
XYZ = randn (1, 3) * R;
return;
end
```

A.6.6 Fonction auxiliaire : `densitePQ.m`

```
% La fonction "densitePQ" évalue la densité relative de P par rapport à Q
% au point XYZ.
function ladensite = densitePQ (XYZ)
global m Cinv;
ladensite = exp (m * Cinv * (XYZ - m / 2)');
return
end
```

A.7 Mathématiques financières

A.7.1 Algorithme “naïf” : `mathfi_naif.m`

```
% La fonction "mathfi_naif" évalue par la méthode de Monte-Carlo
% l'espérance du maximum d'une marche de 60 pas sur R issue de 0
% d'incréments N(1), sans utiliser de technique de réduction de variance
% particulière. "N" est le nombre de simulations demandées.
function mathfi_naif(N)
% On introduit le raccourci T pour le nombre de pas de temps.
T = 60;
% "somme" et "somme2" contiendront respectivement la somme des résultats
% simulés et la somme de leurs carrés.
somme = 0;
somme2 = 0;
for i = 1:N
    % On simule la trajectoire, resprésentée par le 61-vecteur "X".
    % (Attention, ce que nous appelons X_t dans le polycopié correspond
    % dans le programme à X(t+1), puisque sous MATLAB les indices
    % commencent à 1). Il sera plus rapide de pré-allouer la mémoire pour
    % X, en initialisant le vecteur avec des zéros.
    X = zeros(1,T+1);
    % Boucle pour remplir le vecteur X. "Xt" est la valeur courante de X,
    % et "t" le jour auquel on calcule Xt.
    %
    Xt = 0;
    for t = 1:T
```

```
    Xt = Xt + randn(1);
    X(t+1) = Xt;
    end
    % "Xmax".
    Xmax = max(X);
    % On met à jour somme et somme2.
    somme = somme + Xmax;
    somme2 = somme2 + Xmax * Xmax;
end
% "m" est l'estimateur de Monte-Carlo.
m = somme / N;
% "ectype" est l'écart-type (estimé) de X_*.
ectype = sqrt (somme2/N - m*m);
% "erreur" est le rayon de l'intervalle de confiance à 5 %.
erreur = 1.96 * ectype / sqrt(N);
% On affiche le résultat.
fprintf('Estimation de E(X_*) : %.5f ± %.5f\n', m, erreur);
% Le programme s'arrête.
return
end
```

A.7.2 Utilisation "rudimentaire" d'une variable de contrôle : mathfi_ctrl0.m

```
% "mathfi_ctrl0" effectue le même travail que "mathfi_naif", mais en
% utilisant la variable de contrôle Y. Je ne commente que les différences
% avec "mathfi_naif".
function mathfi_ctrl0(N)
T = 60;
% Attention : cette fois-ci les variables "somme" et "somme2" ne se
% réfèreront pas à la simulation de "Xmax", mais à celle de Xmax-Y, où "Y"
% est la variable de contrôle.
somme = 0;
somme2 = 0;
for i = 1:N
    X = zeros(1,T+1);
    Xt = 0;
    for t = 1:T
        Xt = Xt + randn(1);
        X(t+1) = Xt;
    end
    Xmax = max(X);
    % On calcule la variable de contrôle.
    Y = max(X(T+1),0);
    % La quantité à laquelle on va appliquer la méthode de Monte-Carlo à
    % proprement parler est la différence Xmax-Y, notée "valeur".
    valeur = Xmax - Y;
    somme = somme + valeur;
    somme2 = somme2 + valeur*valeur;
end
% On évalue l'estimateur et la marge d'erreur pour E(valeur).
m = somme / N;
ectype = sqrt (somme2/N - m*m);
erreur = 1.96 * ectype / sqrt(N);
```

```
% On en déduit l'estimateur pour E(Xmax) = E(valeur) + E(Y), sachant qu'on
% sait calculer exactement E(Y), appelée ici "EY". Noter que l'addition de
% EY ne modifie pas la marge d'erreur.
EY = sqrt(T / (2*pi));
mXmax = m + EY;
erreurXmax = erreur;
fprintf('Estimation de E(X_*) : %.5f ± %.5f\n', mXmax, erreurXmax);
return
end
```

A.7.3 Utilisation de la variable de contrôle en version paramétrée : `mathfi_ctrl1.m`

```
% "mathfi_ctrl1" utilise la même variable de contrôle que "mathfi_ctrl0",
% mais en version paramétrée. Je ne commente que les différences avec
% "mathfi_ctrl0".
function mathfi_ctrl1(N)
T = 60;
somme = 0;
somme2 = 0;
% Cette fois-ci, il faut aussi maintenir la somme des simulations de Y et
% des Y^2 ("sommeY" et "sommeY2"), ainsi que la somme des produits Xmax * Y
% ("sommeprod").
sommeY = 0;
sommeY2 = 0;
sommeprod = 0;
for i = 1:N
    X = zeros(1,T+1);
    Xt = 0;
    for t = 1:T
        Xt = Xt + randn(1);
        X(t+1) = Xt;
    end
    Xmax = max(X);
    Y = max(X(T+1),0);
    % Pour l'instant on ne sait pas encore quelle va être la valeur dont on
    % calcule l'espérance par Monte-Carlo, donc on maintient à jour toutes
    % les sommes introduites précédemment.
    somme = somme + Xmax;
    somme2 = somme2 + Xmax*Xmax;
    sommeY = sommeY + Y;
    sommeY2 = sommeY2 + Y*Y;
    sommeprod = sommeprod + Xmax*Y;
end
% On évalue la moyenne covariance "covariance" entre Xmax et Y, ainsi que
% la variance "varY" de Y. On en profite aussi pour calculer la variance
% "varXmax" de Xmax, qui nous servira plus loin.
moyenneXmax = somme / N;
moyenneY = sommeY / N;
covariance = sommeprod/N - moyenneXmax * moyenneY;
varY = sommeY2/N - moyenneY*moyenneY;
varXmax = somme2/N - moyenneXmax*moyenneXmax;
% On cherche maintenant le paramètre "lambda" optimal (ou du moins une
% estimation de celui-ci) pour lequel considérer la quantité d'intérêt
```

```
% Xmax-lambda*Y. On utilise pour ce faire la formule du cours.
lambda = covariance / varY;
% Maintenant, il s'agit de savoir quelles ont été l'espérance empirique "m"
% et l'écart-type "ectype" de la quantité d'intérêt Xmax-lambda*Y - toute
% l'astuce étant qu'on peut les calculées à partir dans quantités comptées
% dans la boucle de Monte-Carlo. m est l'estimateur de l'espérance de la
% quantité d'intérêt, dont on calcule la marge d'erreur à partir de ectype.
m = moyenneXmax - lambda * moyenneY;
ectype = sqrt (varXmax - 2*lambda*covariance + lambda*lambda*varY);
erreur = 1.96 * ectype / sqrt(N);
% On en déduit l'estimateur pour E(Xmax) = E(valeur) + lambda*E(Y). Comme
% précédemment, l'addition de EY ne modifie pas la marge d'erreur.
EY = sqrt(T / (2*pi));
mXmax = m + lambda*EY;
erreurXmax = erreur;
fprintf('Estimation de E(X_*) : %.5f ± %.5f\n', mXmax, erreurXmax);
return
end
```

A.7.4 Utilisation de trois variables de contrôle : `mathfi_ctrl3.m`

```
% "mathfi_ctrl3" fonctionne comme "mathfi_ctrl_1", mais en utilisant trois
% variables de contrôle Y_1, Y_2 et Y_3. Nous utilisons le formalisme
% vectoriel de MATLAB pour traiter les trois variables de contrôle
% collectivement. Je ne commente que les différences avec "mathfi_ctrl1".
function mathfi_ctrl3(N)
T = 60;
somme = 0;
somme2 = 0;
% Attention, "sommeY" est maintenant une quantité vectorielle, dont les
% entrées correspondent respectivement aux sommes des Y_1, des Y_2 et des
% Y_3.
sommeY = zeros(1,3);
% Attention, "sommeY2" est maintenant une quantité matricielle :
% "sommeY2(i,j)" contient la somme des produits Y_i * Y_j.
sommeY2 = zeros(3,3);
% Attention, "sommeprod" est maintenant une quantité vectorielle :
% "sommeprod(i)" contient la somme des produits X_* * Y_i.
sommeprod = zeros(1,3);
for i = 1:N
    X = zeros(1,T+1);
    Xt = 0;
    for t = 1:T
        Xt = Xt + randn(1);
        X(t+1) = Xt;
    end
    Xmax = max(X);
    % On calcule cette fois-ci trois variables de contrôle, regroupées dans
    % le vecteur "Y".
    Y = [max(X(21),0), max(X(41),0), max(X(61),0)];
    % Pour l'instant on ne sait pas encore quelle va être la valeur dont on
    % calcule l'espérance par Monte-Carlo, donc on maintient à jour toutes
    % les sommes introduites précédemment.
    somme = somme + Xmax;
```

```
    somme2 = somme2 + Xmax*Xmax;
    sommeY = sommeY + Y;
    % Noter le formalisme matriciel pour la mise à jour de sommeY2 (la
    % prime correspond à la transposition).
    sommeY2 = sommeY2 + Y'*Y;
    sommeprod = sommeprod + Xmax*Y;
end
moyenneXmax = somme / N;
moyenneY = sommeY / N;
covariance = sommeprod/N - moyenneXmax * moyenneY;
% Noter le formalisme matriciel pour le calcul de varY.
varY = sommeY2/N - moyenneY'*moyenneY;
varXmax = somme2/N - moyenneXmax*moyenneXmax;
% La paramètre "lambda" est maintenant vectoriel. Sous MATLAB, la commande
% "A / B", quand elle est appliquée à des matrices, calcule le produit A *
% B^{-1} (à l'inverse, "A \ B" calcule B^{-1} * A).
lambda = covariance / varY;
% Notre quantité d'intérêt est maintenant X_* - \lambda_1*Y_1 -
% \lambda_2*Y_2 - lambda_3*Y_3, soit, en formalisme vectoriel, X_* - lambda
% * Y'. À part le formalisme vectoriel auquel il faut faire attention, la
% façon de calculer l'espérance empirique et l'écart-type de X sont les
% mêmes que pour "mathfi_ctrl1".
m = moyenneXmax - lambda * moyenneY';
ectype = sqrt (varXmax - 2*lambda*covariance' + lambda*varY*lambda');
erreur = 1.96 * ectype / sqrt(N);
% "EY" est maintenant vectoriel.
EY = [sqrt(20/(2*pi)), sqrt(40/(2*pi)), sqrt(60/(2*pi))];
% Noter le formalisme vectoriel dans la ligne suivante.
mXmax = m + lambda*EY';
erreurXmax = erreur;
fprintf('Estimation de E(X_*) : %.5f ± %.5f\n', mXmax, erreurXmax);
return
end
```

A.7.5 Utilisation des variables antithétiques : `mathfi_anti.m`

```
% La fonction "mathfi_anti" améliore "mathfi_naif" en y implémentant la
% méthode des variables antithétiques. N est le nombre de simulations
% demandées. Je ne commente que les différences par rapport à
% "mathfi_naif".
function mathfi_anti(N)
% Pour commencer, comme les simulations vont en fait être effectuées par
% deux, le ‘‘vrai’’ N doit être divisé par deux par rapport à celui indiqué
% en paramètre.
N = ceil(N/2);
T = 60;
somme = 0;
somme2 = 0;
for i = 1:N
    % Il y a maintenant deux trajectoires à simuler, que nous appelons
    % respectivement "X" et "Xanti". Le tirage des gaussiennes qui servent
    % à simuler "Xanti" est inversé par rapport à celui de "X". (Noter
    % qu'ici la correspondance entre la trajectoire de X et celle de X anti
    % est triviale, mais la méthode de retournement des gaussiennes par
```

```
    % laquelle je suis passé a le mérite d'être plus générale).
    X = zeros(1,T+1);
    Xanti = zeros(1,T+1);
    Xt = 0;
    Xantit = 0;
    for t = 1:T
        r = randn(1);
        Xt = Xt + r;
        Xantit = Xantit - r;
        X(t+1) = Xt;
        Xanti(t+1) = Xantit;
    end
    % On évalue "Xmax" et "Xantimax" qui est la valeur de Xmax pour Xanti.
    Xmax = max(X);
    Xantimax = max(Xanti);
    % La valeur dont nous allons calculer l'espérance par Monte-Carlo est
    % celle de Xmax + Xantimax ; c'est par rapport à elle que nous
    % maintenons "somme" et "somme2".
    valeur = Xmax + Xantimax;
    somme = somme + valeur;
    somme2 = somme2 + valeur*valeur;
end
% "mvaleur", "ectype" et "errvaleur" sont les quantités de l'estimation de
% Monte-Carlo pour "valeur".
mvaleur = somme / N;
ectype = sqrt (somme2/N - mvaleur*mvaleur);
errvaleur = 1.96 * ectype / sqrt(N);
% L'estimation qui nous intéresse est celle de l'espérance de X_*, qui est
% la moitié de l'espérance de "valeur" : pour obtenir son estimation et son
% intervalle de confiance, il faut tout diviser par 2.
m = mvaleur / 2;
erreur = errvaleur / 2;
fprintf('Estimation de E(X_*) : %.5f ± %.5f\n', m, erreur);
return
end
```

A.8 Simulation d'équations différentielles stochastiques par la métode d'Euler

A.8.1 Graphe d'un mouvement brownien unidimensionnel (`brown1D.m`)

```
% La fonction "brownien1D" simule et trace une trajectoire de mouvement
% brownien unidimensionnel (avec sigma^2 = 2) sur l'intervalle [0,10].
function brownien1D
% "sigma2" est la variance par unité de temps ; ici 2.
sigma2 = 2;
% "tmax" est la longueur de l'intervalle de simulation, ici 10.
tmax = 10;
% "N" est le nombre de pas de temps en lequel on subdivise l'intervalle
% [0,tmax] pour la discrétisation, ici choisi à 2000.
N = 2000;
% "epsilon" est la largeur d'un pas de discrétisation (on prend ici des pas
% de même taille).
```

```
epsilon = tmax / N;
% "T" est la liste des temps auxquels on discrétise.
T = epsilon * (0:N);
% "X" est destiné à contenir la liste des valeurs du mouvement brownien aux
% différents temps de "T".
X = zeros(1,N+1);
% Le mouvement brownien part de 0; on entre donc cette valeur initiale.
X(1) = 0;
% Cette boucle correspond à la méthode d'Euler stochastique. "i" est
% l'indice du dernier point simulé dans la liste "X".
for i = 1:N
    % On incrémente le mouvement brownien. L'incrément est indépendant du
    % passé, de loi N(sigma^2*epsilon).
    X(i+1) = X(i) + sqrt(sigma2 * epsilon) * randn;
end
% On trace la courbe du mouvement brownien.
plot(T,X,'LineWidth',2);
title('Mouvement brownien 1D (sigma^2 = 2)');
% Le programme s'arrête.
return
end
```

A.8.2 Trajectoire d'un mouvement brownien 2-dimensionnel (`brown2D.m`)

```
% La fonction "brownien2D" simule et trace une trajectoire d'un mouvement
% brownien 2-dimensionnel sur l'intervalle de temps [0,4].
function brownien2D
% "sigmasigmaT" est la matrice de covariance par unité de temps, que je
% choisis ici non standard.
sigmasigmaT = [1,.5;.5,1];
% "sigma" est (la transposée de) la matrice de Cholesky associée à
% "sigmasigmaT", qui nous servira dans la suite pour les simulations.
sigma = chol(sigmasigmaT)';
% "tmax" est la longueur de l'intervalle de simulation, ici 4.
tmax = 4;
% "N" est le nombre de pas de temps en lequel on subdivise l'intervalle
% [0,tmax] pour la discrétisation, ici choisi à 40000.
N = 40000;
% "epsilon" est la largeur d'un pas de discrétisation (on prend ici des pas
% de même taille).
epsilon = tmax / N;
% "B" est destiné à contenir la liste des valeurs du mouvement brownien aux
% différents temps de discrétisation. Chaque colonne de "B" correspondra
% aux 2 coordonnées du mouvement brownien à un certain temps de
% discrétisation.
B = zeros(2,N+1);
% Le mouvement brownien part de (0,0); on entre donc cette valeur initiale.
B(:,1) = [0;0];
% Cette boucle correspond à la méthode d'Euler stochastique. "i" est
% l'indice du dernier point simulé dans la liste "B".
for i = 1:N
    % On incrémente le mouvement brownien. L'incrément est indépendant du
    % passé, de loi N(epsilon*sigmasigmaT).
    B(:,i+1) = B(:,i) + sqrt(epsilon) * sigma * randn(2,1);
```

```
end
% On trace la trajectoire du mouvement brownien.
plot(B(1,:),B(2,:),'LineWidth',2);
axis equal;
title('Trajectoire d''un mouvement brownien 2D anisotrope');
% Le programme s'arrête.
return
end
```

A.8.3 Mouvement brownien géométrique (browngeom.m)

```
% La fonction "browngeom" simule et trace une trajectoire de mouvement
% brownien géométrique avec sigma^2=1 et b=-1/4, sur l'intervalle de temps
% [0,12]. Le programme s'appuie exactement sur la même structure que
% "brown1D"; seuls la nomenclature, les paramètres et la formule
% d'incrémentation changent.
function browngeom
sigma2 = 1;
% "b" est le paramètre de dérive du mouvement brownien arithmétique
% sous-jacent.
b = -1/4;
tmax = 8;
N = 3000;
epsilon = tmax / N;
T = epsilon * (0:N);
X = zeros(1,N+1);
% Évidemment le mouvement brownien géométrique ne peut pas partir de 0 !
% Ici je choisis de le faire partir de 1.
X(1) = 1;
for i = 1:N
    % Cette fois-ci, l'incrément est de loi N((sigma*X)^2*epsilon) +
    % (b+sigma^2/2)*X*epsilon.
    X(i+1) = X(i) + sqrt(sigma2) * X(i) * sqrt(epsilon) * randn +  ...
        (b + sigma2 / 2) * X(i) * epsilon;
end
plot(T,X,'LineWidth',2);
set(gca,'Ylim',[0,Inf]);
title('Mouvement brownien géométrique (sigma^2=1, b=-1/4)');
return
end
```

A.8.4 Processus d'Ornstein–Uhlenbeck (OU.m)

```
% La fonction "OU" simule et trace un processus d'Ornstein-Uhlenbeck avec
% sigma^2=1 et lambda=1, sur l'intervalle de temps [0,8]. Le programme
% s'appuie sur la même structure que "brown1D" et "browngeom"; je ne
% commente que la partie spécifique à ce programme.
function OU
sigma2 = 1;
lambda = 1;
tmax = 8;
N = 2000;
epsilon = tmax / N;
T = epsilon * (0:N);
```

```
X = zeros(1,N+1);
% Ici je choisis de faire partir le processus d'Ornstein-Uhlenbeck sous sa
% mesure d'équilbre, que je sais être N(sigma^2/2lambda).
X(1) = sqrt(sigma2/lambda/2) * randn;
for i = 1:N
    X(i+1) = X(i) + sqrt(sigma2) * sqrt(epsilon) * randn - ...
        lambda * X(i) * epsilon;
end
plot(T,X,'LineWidth',2);
title('Processus d''Ornstein-Uhlenbeck');
return
end
```

Annexe B

Tables de la loi normale standard

Lorsqu'on manipule des intervalles de confiance, souvent on a besoin de connaitre la probabilité que la loi normale standard $\mathcal{N}(0,1)$ tombe dans tel ou tel intervalle, ce qui exige de savoir déterminer la fonction de répartition de cette loi, fonction que nous noterons ici Φ. On a :

$$\Phi(x) := \mathbb{P}\big(\mathcal{N}(0,1)\big) = (2\pi)^{-1/2} \int_{-\infty}^{x} e^{-x^2/2}\, dx.$$

La fonction Φ a le mauvais gout de ne pas être « élémentaire », càd. de ne pas pouvoir s'écrire à partir des fonctions de base [*]. Heureusement, la plupart des logiciels de calcul numérique implémentent, parmi leurs « fonctions spéciales », une fonction appelée *fonction d'erreur*, notée erf, définie par :

$$\begin{aligned} \operatorname{erf} \colon \mathbf{R} &\to (\pm 1) \\ x &\mapsto \frac{2}{\sqrt{x}} \int_0^x e^{-x^2}\, dx, \end{aligned}$$

de sorte qu'on a

$$\Phi(x) = \big(1 + \operatorname{erf}(x/\sqrt{2})\big)/2.$$

Il est aussi souvent utile de pouvoir calculer la bijection réciproque Φ^{-1} de la fonction de répartition Φ. Notant $\operatorname{erf}^{-1} \colon (-1,1) \to \mathbf{R}$ la bijection réciproque de erf (qui est elle aussi généralement programmée dans les logiciels de calcul numérique), on a d'après (B) :

$$\Phi^{-1}(p) = \sqrt{2}\operatorname{erf}^{-1}(2p-1).$$

Lorsqu'on ne dispose pas d'une implémentation des fonctions erf et erf^{-1}, il faut soit programmer soi-même leur calcul numérique, soit utiliser une table de valeurs lorsque c'est possible. La figure B.1 indique les valeurs les plus souvent utilisées de Φ et Φ^{-1}.

[*]. Les fonctions élémentaires de base sont les fonctions constantes rationnelles, les quatre opérations (addition, soustraction, multiplication, division), l'exponentielle, les fonctions trigonométriques, le logarithme et les fonctions trigonométriques réciproques ; et quand nous disons qu'une fonction « s'écrit à partir des fonctions de base », cela signifie qu'on peut la déduire de celles-ci par composition : par exemple, la fonction constante π est élémentaire, car c'est $\arccos(-1)$; la fonction $\sqrt{\cdot}$ est élémentaire, car $\sqrt{x} = \exp(\log(x)/2)$; la fonction argtanh est élémentaire, car $\operatorname{argtanh}(x) = \ln \sqrt{(1+x)\,/\,(1-x)}$; etc.

x	$\Phi(x)$
$-\infty$	0
-3	0,001 3
-2	0,023
-1	0,16
0	$1/2$
$+1$	0,84
$+2$	0,977
$+3$	0,998 7
$+\infty$	1

p	$\Phi^{-1}(p)$
0	$-\infty$
0,000 5	$-3,30$
0,001	$-3,10$
0,002 5	$-2,81$
0,005	$-2,58$
0,01	$-2,33$
0,025	$-1,96$
0,05	$-1,65$
0,1	$-1,29$

p	$\Phi^{-1}(p)$
0,9	$+1,29$
0,95	$+1,65$
0,975	$+1,96$
0,99	$+2,33$
0,995	$+2,58$
0,997 5	$+2,81$
0,999	$+3,10$
0,999 5	$+3,30$
1	$+\infty$

TABLE B.1 – Tables sommaires de la fonction Φ, de la fonction Φ^{-1} au voisinage de 0 et de la fonction Φ^{-1} au voisinage de 1. Les arrondis sont donnés respectivement au plus proche, par défaut et par excès.

.