

# Méthode de Monte-Carlo et Application aux processus aléatoires

Rémi Peyre

printemps 2013



# Chapitre 1

## Principe de la méthode de Monte-Carlo

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

#### 1.1.1 Estimation d'une probabilité par la méthode de rejet

**Définition 1.1.** Supposons qu'on cherche à évaluer la probabilité  $\mathbb{P}(A)$  d'un évènement  $A$  sous une loi  $\mathbb{P}$  qu'on sait simuler. La *méthode de Monte-Carlo* consiste à réaliser un grand nombre  $N$  de simulations indépendantes de la loi  $\mathbb{P}$ , à compter le nombre  $n$  de ces simulations pour lesquelles  $A$  a eu lieu, et à estimer  $\mathbb{P}(A)$  par la proportion de telles simulations :

$$\hat{p} := N^{-1}n.$$

**Théorème 1.2.** *L'estimateur  $\hat{m}$  converge en probabilité vers  $\mathbb{E}(f)$  quand  $N$  tend vers l'infini : on dit que  $\hat{m}$  est un estimateur convergent de  $\mathbb{E}(f)$ . L'estimateur est même fortement convergent, c'est-à-dire que si on réalise un nombre infini de simulation et qu'on regarde la suite des  $\hat{p}_N$  obtenus en ne considérant que les  $N$  premières,  $\hat{p}_N$  converge presque-surement vers  $\mathbb{E}(f)$  quand  $N \rightarrow \infty$ .*

*Démonstration.* C'est simplement une reformulation de la version fréquentielle de la loi des grands nombres.  $\square$

*Exemple 1.3.* Dans cet exemple, nous allons essayer d'évaluer la probabilité qu'un joueur de *yahtzee* cherchant à obtenir un yahtzee réussisse son objectif. Je précise ici le principe du jeu de *yahtzee* pour ceux qui ne le connaissent pas. Dans ce jeu, le joueur dispose de cinq dés qu'il lance sur le plateau. À l'issue de ce lancer initial, il relance une première fois ceux des dés qu'il souhaite (il peut éventuellement n'en relancer aucun ou les relancer tous). Puis à l'issue de ce premier relancer, à nouveau il relance ceux des dés qu'il souhaite (il a le droit de relancer à cette étape des dés qu'il n'avait pas relancés à l'étape précédente). Le but du jeu est d'avoir obtenu à l'issue de ces trois lancers la meilleure combinaison possible. Ici nous supposons que le joueur ne s'intéresse qu'à

la meilleure combinaison de toutes, elle-même appelée « yahtzee », consistant à obtenir cinq dés identiques.


Il est clair (au moins intuitivement) que la meilleure stratégie pour le joueur consiste, à l'issue de chaque étape, à regarder quel est le chiffre le plus présent parmi les dés et à relancer tous les dés portant un autre chiffre. Mais quelle probabilité cela donne-t-il au final d'obtenir un yahtzee ? Les calculs n'étant guère évidents, une idée est de recourir à la méthode de Monte-Carlo en simulant un grand nombre de fois cette stratégie pour évaluer numériquement la probabilité de succès par l'estimateur (1.1) !

Une implémentation de cette méthode (avec  $N = 200\,000$ ) est donnée dans l'annexe A.1 (`yahtzee.m`). En l'occurrence, le plus difficile est en fait de programmer la simulation, la méthode de Monte-Carlo en elle-même étant simplissime... Voici ce qu'a donné l'exécution de cette fonction sur ma machine :

```
>> yahtzee
Estimateur de la probabilit  de succ s : 0.04612
```

Si on compare avec la valeur exacte  $100\,194\,316/6^{12} \simeq 0,046\,029$ , on voit que l'estimateur de Monte-Carlo est effectivement tout- -fait fiable !

### 1.1.2 Estimation d'une esp rance


 La m thode pr sent e dans la d finition 1.1 est en fait un cas particulier de la m thode g n rale de Monte-Carlo, qui permet d' valuer des *esp rances* :

**D finition 1.4.** Supposons qu'on cherche    valuer l'esp rance  $\mathbb{E}(f)$  d'une variable al atoire  $f$  de classe  $L^1$  sous une loi  $\mathbb{P}$  qu'on sait simuler. La *m thode de Monte-Carlo* consiste   r aliser un grand nombre  $N$  de simulations de la loi  $\mathbb{P}$ ,   regarder la valeur  $f(\omega_i)$  de  $f$  pour chacun de ces simulations, et   estimer  $\mathbb{E}(f)$  par la valeur moyenne des  $f(\omega_i)$  :

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

*Remarque 1.5.* Dans le cas particulier o   $f$  est l'indicatrice  $\mathbf{1}_{\{A\}}$  d'un  v nement  $A$ , la d finition 1.4 redonne la d finition 1.1.

**Th or me 1.6.** *L'estimateur  $\hat{m}$  converge fortement vers  $\mathbb{E}(f)$ , et converge aussi dans  $L^1$ .*

 *D monstration.* C'est encore la loi (forte) des grands nombres, cette fois-ci dans sa version g n rale. □

*Remarque 1.7.* La m thode de Monte-Carlo est donc une fa on d' valuer une quantit  *d terministe* (l'esp rance) par un proc d  *al atoire* (les simulations) ! Ce paradoxe montre que le hasard peut parfois  tre un atout plut t qu'un handicap...

*Remarque 1.8.* Attention, les th or mes 1.2 et 1.6 ne sont valables en toute rigueur que si les simulations  $\omega_i$  sont *exactement* i.i.d.  $\mathbb{P}$ . En pratique, les calculs informatiques utilisent g n ralement des nombres seulement *pseudo*-al atoires, ce qui peut faire  chouer la m thode de Monte-Carlo dans les cas les plus extr mes.

*Exemple 1.9.* Une chaîne télévisée veut lancer un jeu appelé *Qui veut gagner du pognon ?* Dans ce jeu, les candidats doivent résoudre des questions de difficulté croissante jusqu'à ce qu'ils ne sachent plus répondre. À la première bonne réponse, le candidat gagne 1 000 €, et son gain double à chaque question suivante. La chaîne a décidé de calibrer la difficulté du jeu de façon que la probabilité qu'un candidat réponde correctement à la  $k$ -ième question soit de  $3/(k + 2)$ . En outre, les candidats ont droit à deux jokers, qui leur permettent de changer de question (en restant au même niveau de l'échelle) s'ils ne connaissent pas la réponse ; il ne quittent le jeu en cas d'échec qu'après avoir consommé ces jokers.

Naturellement, la chaîne se demande quel budget créditer pour ce jeu : il lui faut donc calculer le gain moyen d'un joueur compte tenu de la calibration des difficultés qu'elle a choisie. Comme les responsables de la chaîne ne sont pas très forts en analyse théorique, ils décident de recourir à la méthode de Monte-Carlo en calculant par simulation l'estimateur (1.4).

Une implémentation de cette méthode (avec  $N = 5\,000$ ) est donnée dans l'annexe A.2 (`qvmdp.m`). Sur ma machine, une exécution de ce programme a fourni le résultat suivant :

```
>> qvmdp
Estimateur de l'espérance de gain : 24814.8 €
```

À titre de comparaison, on peut calculer que la véritable valeur de l'espérance est (en arrondissant) de 25 441 € : là encore, la précision de la méthode de Monte-Carlo est tout-à-fait satisfaisante.

### 1.1.3 Application au calcul d'intégrales

La méthode de Monte-Carlo peut aussi être utilisée pour calculer des intégrales, en réécrivant celles-ci sous forme d'espérances par la technique suivante :



**Proposition 1.10.** *Pour  $f$  une fonction intégrable sur  $\mathbb{R}^d$ , et  $\mathbb{P}$  une mesure de probabilité sur  $\mathbb{R}^d$  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 :*

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

Cela permet alors d'utiliser l'estimateur de Monte-Carlo (1.4) pour évaluer  $\int f(x) dx$ .



*Démonstration.* Notons  $\Omega$  l'ensemble des points où  $d\mathbb{P}/dx$  est non nulle ; la mesure de Lebesgue a alors sur  $\Omega$  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  $\Omega$ , on a  $\int_{\mathbb{R}^d} f(x) dx = \int_{\Omega} f(x) dx$ , d'où  $\int_{\mathbb{R}^d} f(x) dx = \int_{\Omega} f(x) (dx/d\mathbb{P})(x) d\mathbb{P}(x) = \int_{\Omega} (d\mathbb{P}/dx)^{-1}(x) f(x) d\mathbb{P}(x) = \mathbb{E}((d\mathbb{P}/dx)^{-1} f)$ , CQFD.  $\square$

*Remarque 1.11.* En fait, il n'y a essentiellement aucune différence entre une espérance et une intégrale. Dans un sens en effet, la théorie de la mesure nous dit qu'une espérance n'est rien d'autre qu'une intégrale par rapport à une mesure *générale* (quand celle-ci est de probabilité) ; et dans l'autre, une intégrale par rapport à la mesure de Lebesgue peut toujours s'écrire comme une espérance, en introduisant une mesure de probabilité  $\mathbb{P}$  à densité partout non nulle par rapport à  $dx$  et en appliquant (1.10). Dans la suite de ce cours, tout ce que nous dirons sur l'application de la méthode de Monte-Carlo pour une espérance s'appliquera donc aussi au cas des intégrales, et réciproquement.

*Exemple 1.12.* À titre d'exemple, voyons comment évaluer l'intégrale

$$\int_0^\infty (1+x^3)^{-1} dx.$$

Puisque la mesure de Lebesgue sur  $(0, \infty)$  n'est pas de probabilité, quelle loi peut-on utiliser pour voir cette intégrale comme une espérance ? Je propose d'utiliser la loi de type Pareto de densité

$$\forall x \in (0, \infty) \quad d\mathbb{P}(x) = (x+1)^{-2} dx,$$

qu'on peut simuler à partir d'une loi uniforme  $U$  sur  $[0, 1]$  en prenant  $x = U^{-1} - 1$ . Cela donne le programme de l'annexe A.3 (`exdintegrale.m`), dont l'exécution ( $N = 80\,000\,000$ ) m'a donné :

```
>> exdintegrale
Estimateur de l'intégrale : 1.209228
```

soit une précision de 5 chiffres significatifs par rapport à la véritable valeur  $2\sqrt{3}\pi/9 \simeq 1,209\,200$ .

*Remarque 1.13.* Il y a très nombreuses possibilités pour choisir la loi  $\mathbb{P}$  destinée à appliquer la proposition 1.10, 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 § 2.7.

Une situation particulière qu'il est utile de connaître concernant l'application de la méthode de Monte-Carlo au calcul d'intégrales est la suivante :

**Proposition 1.14.** *Soit  $f: \mathbb{R}^d \rightarrow \mathbb{R}$  une fonction de classe  $L^1$  qui est nulle en-dehors d'un certain pavé  $\prod_{k=1}^d [a_k, b_k]$ , et soient  $((\omega_{ik})_{k \in \{1, \dots, d\}})_{i \in \mathbb{R}^*}$  des variables aléatoires i.i.d. de loi uniforme sur  $[0, 1]$ . Alors*

$$\hat{I} := \prod_{k=1}^d (b_k - a_k) \times N^{-1} \sum_{i=1}^N f(a_1 + (b_1 - a_1)\omega_{i1}, \dots, a_d + (b_d - a_d)\omega_{id})$$

*est un estimateur convergent de  $\int_{\mathbb{R}^d} f(x) dx$ .*

*Démonstration.* Il s'agit simplement d'appliquer la proposition 1.10 en prenant pour  $\mathbb{P}$  la loi uniforme sur le pavé  $Q := \prod_{k=1}^d [a_k, b_k]$ . En effet, cette loi a par rapport à la mesure de Lebesgue la densité constante  $\text{Vol}(Q)^{-1}$  sur  $Q$  (où  $\text{Vol}(Q) = \prod_{k=1}^d (b_k - a_k)$  désigne le volume du pavé) ; et les  $(a_1 + (b_1 - a_1)\omega_{i1}, \dots, a_d + (b_d - a_d)\omega_{id})$  en sont des échantillons i.i.d.  $\square$

*Exemple 1.15.* Nous allons évaluer le volume de la boule unité de dimension 4, c'est-à-dire l'intégrale

$$\int_{\mathbb{R}^4} \mathbf{1}_{\{\|(x_1, x_2, x_3, x_4)\| \leq 1\}} dx_1 dx_2 dx_3 dx_4,$$

où  $\|(x_1, x_2, x_3, x_4)\|$  désigne la norme euclidienne  $(x_1^2 + x_2^2 + x_3^2 + x_4^2)^{1/2}$ .

La fonction  $\mathbf{1}_{\{\|(x_1, x_2, x_3, x_4)\| \leq 1\}}$  est clairement nulle en-dehors du pavé  $[-1, +1]^4$ , ce qui permet d'appliquer la proposition en prenant ce pavé pour  $\prod_{k=1}^d (b_k - a_k)$ . Une implémentation de la méthode (avec  $N = 100\,000$ ) est donnée dans l'annexe A.4 (`boule4D.m`). Une exécution m'a donné :

```
>> boule4D
Volume estimé de la boule : 4.9573
```

La valeur exacte est  $\pi^2/2 \simeq 4,935$ , une fois de plus en bon accord avec l'estimateur de Monte-Carlo.

## 1.2 Intervalles de confiance

### 1.2.1 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 pourrions 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.  $p = 99\%$ ).

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



**Théorème 1.16.** *Supposons qu'on cherche à évaluer  $\mathbb{E}(f)$  par la méthode de Monte-Carlo avec  $f$  de classe  $L^2$ , et notons  $\sigma$  la variance (supposée non nulle) de  $f$ . Alors, quand  $N \rightarrow \infty$ , la probabilité que  $\mathbb{E}(f)$  soit effectivement dans l'intervalle  $[\hat{m} + c_- \sigma N^{-1/2},$*

$\hat{m} + c_+ \sigma N^{-1/2}]$  converge vers  $\mathbb{P}(\mathcal{N}(1) \in [c_-, c_+]) = (2\pi)^{-1/2} \int_{c_-}^{c_+} e^{-x^2/2} dx$ . En particulier,

$$\lim_{N \rightarrow \infty} \mathbb{P}(\mathbb{E}(f) \in [\hat{m} + c_- \sigma N^{-1/2}, \hat{m} + c_+ \sigma N^{-1/2}]) \geq (2\pi)^{-1/2} \int_{c_-}^{c_+} e^{-x^2/2} dx :$$

on dit que  $[\hat{m} + c_- \sigma N^{-1/2}, \hat{m} + c_+ \sigma N^{-1/2}]$  est un intervalle de confiance asymptotique à la probabilité  $(2\pi)^{-1/2} \int_{c_-}^{c_+} e^{-x^2/2} dx$  pour  $\mathbb{E}(f)$ .



*Démonstration.* Que  $\mathbb{E}(f)$  soit dans l'intervalle  $[\hat{m} + c_- \sigma N^{-1/2}, \hat{m} + c_+ \sigma N^{-1/2}]$  est équivalent à ce que  $\sigma^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m})$  soit dans  $[c_-, c_+]$ . Or

$$\sigma^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m}) = N^{-1/2} \sum_{i=1}^N \sigma^{-1}(\mathbb{E}(f) - f(\omega_i)),$$

où les  $\sigma^{-1}(\mathbb{E}(f) - f(\omega_i))$  sont des v.a. i.i.d.  $L^2$  centrées et de variance unitaire. Le théorème-limite central dit alors que  $\sigma^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m})$  converge en loi vers  $\mathcal{N}(1)$ , d'où le résultat annoncé.  $\square$



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

**Corollaire 1.17.** *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  $\omega_i$  :*

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

Alors, quand  $N \rightarrow \infty$ , l'intervalle  $[\hat{m} + c_- \hat{\sigma} N^{-1/2}, \hat{m} + c_+ \hat{\sigma} N^{-1/2}]$  est un intervalle de confiance asymptotique à la probabilité  $\mathbb{P}(\mathcal{N}(1) \in [c_-, c_+])$  pour  $\mathbb{E}(f)$ .



*Démonstration.* De même que pour la preuve du théorème 1.16, nous allons démontrer que  $\hat{\sigma}^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m})$  converge en loi vers  $\mathcal{N}(1)$ . Nous observons que  $\hat{\sigma}^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m}) = \sigma^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m}) / \sigma^{-1} \hat{\sigma}$ , où  $\sigma^{-1} \hat{\sigma}$  converge en probabilité vers la constante 1 d'après la loi des grands nombres.

Nous allons maintenant utiliser un lemme élémentaire concernant la convergence en loi : « si  $X_N$  est une variable aléatoire convergeant en loi vers  $X$ , que  $\Lambda_N$  est une variable aléatoire convergeant en probabilité vers la constante  $\lambda_0$ , et que  $\Phi$  est une application continue en tous les points de la forme  $(x, \lambda)$ , alors  $\Phi(X_N, \Lambda_N)$  converge en loi vers  $\Phi(X, \lambda_0)$  ». En appliquant ce lemme avec  $X_N = \sigma^{-1} N^{1/2}(\mathbb{E}(f) - \hat{m})$ ,  $X = \mathcal{N}(1)$ ,  $\Lambda_N = \sigma^{-1} \hat{\sigma}$ ,  $\lambda_0 = 1$  et  $\Phi(x, \lambda) = x / \lambda$ , on trouve alors le résultat annoncé.  $\square$



*Remarque 1.18.* Attention, les théorèmes 1.16 et 1.17 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 considérerons 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 1.17 sous peine d'obtenir des résultats complètement aberrants !

*Remarque 1.19.* Bien qu'on parle souvent simplement d'« intervalle de confiance à la probabilité  $p$  », il convient de garder à l'esprit que les intervalles de confiance que donne le théorème 1.17 sont seulement *asymptotiques*, c.à.d. que la probabilité de confiance  $p$  n'est valable que quand  $N$  tend vers l'infini. En pratique, on pose comme « acte de foi » que  $N$  a été choisi suffisamment grand pour que l'intervalle de confiance trouvé soit presque de probabilité  $p$ , mais il peut y avoir des cas où c'est complètement faux...

*Remarque 1.20.* En première approximation, on pourra retenir la règle empirique suivante : si on souhaite obtenir un intervalle de confiance de niveau  $1 - \varepsilon$ , il faut procéder à au moins  $0,001\varepsilon^{-2}$  simulations pour pouvoir considérer que le niveau réel de l'intervalle de confiance a à peu près atteint sa valeur asymptotique.

*Exemple 1.21.* Un casino a imaginé une variante du blackjack que nous appellerons ici le *vingt-et-zéro*. Ce jeu se pratique avec des cartes numérotées de 1 à 10, les différents numéros étant distribués équitablement (le nombre de cartes dans le sabot étant très grand, les différents tirages seront considérés comme indépendants). Le joueur mise 1 € et joue contre la banque. Au départ, la banque tire une carte pour elle, visible par le joueur. Puis elle distribue des cartes au joueur jusqu'à ce qu'il dise « stop ». Le but pour le joueur est que le total de ses cartes approche le plus possible la valeur 20, sans jamais dépasser celle-ci. Si le joueur dépasse 20, il a perdu et la banque empoche sa mise. Sinon, c'est à la banque de tirer ses cartes. Celle-ci joue en suivant la règle suivante : elle continue de tirer des cartes jusqu'à atteindre ou dépasser 17, et s'arrête alors. À la fin du tirage des cartes de la banque :

- Soit la banque a dépassé 20 : dans ce cas là elle a perdu et donne 2 € au joueur.
- Soit la banque n'a pas dépassé 20 et dans ce cas-là on regarde qui a le plus fort total :
  - Si c'est la banque, elle garde la mise du joueur ;
  - Si c'est le joueur, la banque lui donne 2 € ;
  - En cas d'égalité :
    - Dans le cas où les deux adversaires ont atteint 20, la banque rend sa mise au joueur ;
    - Dans les autres cas d'égalité, on considère que c'est le joueur qui a gagné et la banque lui donne 2 €.

L'espérance de gain du joueur dépend évidemment de la stratégie qu'il suit. Ici nous supposons qu'il suit la *stratégie de base*, que nous admettrons être la meilleure possible : cette stratégie consiste à fixer le seuil auquel on arrête de demander une carte en fonction de la carte affichée par la banque, selon la règle suivante :

Carte de la banque	1	2	3	4	5	6	7	8	9	10
Le joueur dit <i>stop</i> à	15	15	15	14	14	14	15	16	16	16

Pour qu'un jeu de casino soit rentable pour l'organisateur, il faut bien sûr que l'espérance des gains du joueur ne dépasse pas sa mise. Le casino souhaite donc évaluer cette espérance. Mais il convient d'être précis, car les jeux de casino ont typiquement une espérance de gain supérieure à 95 % de la mise (parce que sinon le joueur perd trop vite et le jeu n'est alors plus amusant pour lui) : du coup, si l'espérance de gain pour une mise de 1 € est de 1,01 € et que le casino calcule à tort que celle-ci vaut 0,99 €, il validera un jeu... qui lui fait perdre de l'argent !

Vous l'avez compris, c'est la méthode de Monte-Carlo que le casino a choisie pour calculer l'espérance de gain, et l'utilisation des intervalles de confiance lui permettra de s'assurer que son calcul a la précision requise. Ici on suppose que le casino souhaite un niveau de confiance (asymptotique) de 99,9 %, de sorte que d'après la figure 1.1 il doit choisir un intervalle de confiance « à 3,30 sigmas », c'est-à-dire prendre  $c_+ = -c_- = 3,30$  dans le théorème 1.17. On prend un nombre de simulations relativement faible de 8 000, car notre casino préfère effectuer ses simulations à la main par prudence... Le code MATLAB correspondant est donné dans l'annexe A.5 (`vingtzero.m`). Une exécution m'a donné :

```
>> vingtzero
```

```
Avec une certitude d'environ 99,9 %, le gain moyen pour le joueur est dans
l'intervalle [0.95993,1.0333].
```

Ainsi, bien que l'estimateur de Monte-Carlo (qui est en l'occurrence le centre de l'intervalle de confiance, soit environ 0,996 62) affirme ici que le jeu est rentable pour le casino, la prise en compte de l'intervalle de confiance montre qu'on ne peut pas en avoir une certitude très élevée ! Pour clarifier la situation, le casino aurait dû effectuer un plus grand nombre de simulations : ainsi, avec 40 000 tirages au lieu de 8 000, mon programme aurait donné un intervalle à 99,9 % de  $[0,966\ 4; 0,999\ 2]$ , ce qui cette fois-ci garantit (avec 0,1 % de risque) au casino que son jeu ne lui fera pas faire faillite ! On aurait en fait pu calculer exactement le gain moyen, qui vaut environ 0,982, ce qui est bien dans les intervalles de confiance que nous avons trouvés.



Pour pouvoir donner une valeur numérique à la probabilité de confiance ou à la largeur de l'intervalle, il faut disposer de la table de répartition de la loi normale. Nous donnons dans la figure 1.1 deux mini-tables de correspondance entre intervalle de confiance et probabilité de confiance dans le cas où  $c_- = -c_+$ . De manière générale, on peut calculer ce genre de tables à l'aide de ce qu'on appelle la *fonction d'erreur*  $\text{erf}$ , qui est implémentée sur la plupart des systèmes de calcul numérique (par exemple, sous MATLAB on l'obtient par “`erf`”) : on a  $\mathbb{P}(\mathcal{N}(1) \leq c) = (1 + \text{erf}(c/\sqrt{2}))/2$ . L'inverse  $\text{erf}^{-1}$  de la fonction d'erreur (qu'on obtient par “`erfinv`” sous MATLAB) permet de faire le cheminement inverse :  $c = \sqrt{2} \text{erf}^{-1}(2p - 1)$  est le seuil tel que  $\mathbb{P}(\mathcal{N}(1) \leq c) = p$ .



## 1.2.2 Notion d'efficacité

Le théorème 1.16 nous dit que, sous réserve que  $f$  soit de classe  $L^2$ , la largeur de l'intervalle de confiance décroît comme  $N^{-1/2}$  quand le nombre de simulations  $N$  augmente. Cela est dû au fait que l'estimateur  $\hat{m}$  tombe à une distance d'ordre  $O(N^{-1/2})$  de la véritable valeur  $\mathbb{E}(f)$  : en effet, la variance de  $\hat{m}$  décroît en  $O(N^{-1})$ , puisque

$c$	$\mathbb{P}( \mathcal{N}(1)  \leq c)$
0	0
0,5	0,38
1	0,68
1,5	0,86
2	0,954
2,5	0,987
3	0,997 3
3,5	0,999 53
4	0,999 936
$\infty$	1

$\mathbb{P}( \mathcal{N}(1)  \leq c)$	$c$
0	0
0,5	0,68
0,8	1,29
0,9	1,65
0,95	1,96
0,98	2,33
0,99	2,58
0,995	2,81

$\mathbb{P}( \mathcal{N}(1)  \leq c)$	$c$
0,998	3,10
0,999	3,30
0,999 5	3,49
0,999 8	3,72
0,999 9	3,90
0,999 95	4,06
0,999 98	4,27
1	$\infty$

TABLE 1.1 – Tables sommaires du niveau de confiance (asymptotique) de l'intervalle en fonction de sa largeur (à gauche), resp. de la largeur requise pour atteindre un certain niveau de confiance (à droite). Les arrondis sont fait dans le sens conservatif.

d'après la formule d'additivité des variances pour les variables indépendantes,  $\text{Var}(\hat{m}) = (N^{-1})^2 \times N \text{Var}(f) = N^{-1} \text{Var}(f)$ . 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 :

**Définition 1.22.** On appelle *efficacité* d'un calcul de Monte-Carlo l'inverse de la variance 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 de  $f$  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 (1.4), cette efficacité par simulation est simplement l'inverse de  $\text{Var}(f)$ .

*Remarque 1.23.* 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 1.24.* On voit ainsi que l'erreur commise par la méthode de Monte-Carlo décroît 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 1.1.3, 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.

# Chapitre 2

## Techniques de réduction de variance

### Motivation

Nous avons vu dans la § 1.2.2 que le cout d'un calcul de Monte-Carlo, à précision fixée, était proportionnel à l'inverse de l'efficacité de celui-ci. En pratique, on rencontre de nombreuses situations où les circonstances du problème font que le cout de calcul devient une véritable limitation ; de sorte qu'il devient particulièrement important d'améliorer l'efficacité. Cette partie du cours va présenter diverses méthodes pour réaliser cet objectif. L'idée générale sera, par le biais d'une manipulation algébrique, de ramener notre calcul de Monte-Carlo à un *autre* calcul de Monte-Carlo dont l'efficacité est supérieure. Ces méthodes sont appelées *techniques de réduction de variance*, car elles se focalisent sur l'amélioration de l'efficacité *par simulation*, dont nous avons vu qu'elles n'était autre que l'inverse de la variance de la fonction dont on estime l'espérance.

*Remarque 2.1.* 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.

### 2.1 Stratification “à posteriori”

**Théorème 2.2.** *Supposons qu'on cherche à évaluer  $\mathbb{E}(f)$  par la méthode de Monte-Carlo, et que l'espace  $\Omega$  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 ces 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” (1.4).*

*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  $\mathbb{E}(f) = \sum_k p_k m_k$ . Or on peut

réécrire (2.2) 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  $n_k^{-1} \text{Var}_{\mathbb{P}_k}(f)$ , et les différents  $\hat{m}_k$  sont indépendants, donc la variance globale de  $\hat{m}^{\text{strat}}$  vaut

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

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

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

$$\begin{aligned} \text{Var}(f) &= \mathbb{E}(f^2) - \mathbb{E}(f)^2 = \sum_k p_k \mathbb{E}_k(f^2) - \left( \sum_k p_k \mathbb{E}(f_k) \right)^2 \\ &= \sum_k p_k \text{Var}_{\mathbb{P}_k}(f) + \left[ \sum_k p_k \mathbb{E}_k(f)^2 - \left( \sum_k p_k \mathbb{E}(f_k) \right)^2 \right]. \end{aligned} \quad (2.1)$$

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  $\sum_k p_k = 1$ ).  $\square$

*Exemple 2.3.* Supposons qu'à l'approche du second tour d'une élection présidentielle, opposant un candidat de gauche et un candidat de droite, un institut d'études d'opinions souhaite évaluer quelle proportion des voix recueilleront chacun des deux candidats (nous supposons pour simplifier que tout le monde vote et qu'il n'y a pas de bulletins nuls). Il procède pour cela à un *sondage*, c'est-à-dire qu'il interroge quelques citoyens tirés au sort (on supposera ici pour simplifier qu'il n'y a pas de biais dans ce tirage au sort) et estimera les scores réels par les scores obtenus sur son échantillon. (Le nombre de citoyens interrogés doit être suffisamment grand dans l'absolu [par exemple 2 000], mais il est néanmoins très inférieur à l'effectif total de l'électorat [environ 50 000 000 pour dans le cas présent] dont l'interrogation exhaustive serait inenvisageable pour l'institut). Notez qu'un sondage n'est essentiellement rien d'autre qu'une méthode de Monte-Carlo ! En effet, la loi de probabilité considérée est la loi uniforme sur l'ensemble des électeurs, le score à estimer du candidat  $X$  est l'espérance selon cette probabilité de l'indicatrice de « vote pour  $X$  », et les tirages i.i.d. correspondent à interroger des citoyens aléatoirement (la seule différence étant qu'ici il y a *sondage* des citoyens, c'est-à-dire auscultation, et non *simulation*).

Les résultats s'annonçant particulièrement serrés, notre institut souhaite obtenir une précision maximale. Or il a remarqué qu'il y avait généralement une corrélation entre l'âge des électeurs et leurs préférences partisans : une fluctuation statistique sur l'âge des personnes interrogées risque de se traduire par un biais sur le résultat obtenu ! Le bon point, c'est que l'institut d'opinion connaît *exactement* la répartition par tranches d'âges des électeurs, grâce aux statistiques de l'INSEE. Il décide donc de réaliser une

stratification à posteriori par tranche d'âge, avec cinq strates : 18–24 ans, 25–34, 35–49, 50–64 et 65+. D'après l'INSEE, les effectifs respectifs de ces strates sont 5 595 129, 8 012 818, 13 344 860, 12 632 448 et 11 174 879, soit en proportion respectivement 11,02 %, 15,79 %, 26,29 %, 24,89 % et 22,01 %. Notre institut interroge 1 000 personnes et obtient les résultats suivants :

Tranche	Sondés	Gauche	Droite
18–24	112	92	20
25–34	156	118	38
35–49	265	136	129
50–64	253	87	166
65+	214	47	167
Total	1 000	480	520

Si notre institut de sondage ne tient pas compte de la stratification, il obtient alors un estimateur pour le score du candidat de gauche égal à 48,00 %, avec une marge d'erreur à 2 sigmas de  $\pm 3,16$  %. Si maintenant l'institut utilise la stratification, cela signifie qu'il estime séparément la fraction du score du candidat due à chaque tranche d'âge et qu'il somme ces estimations après coup :

Tranche	Pourcentage exact	Score conditionnel	Score partiel
18–24	11,02	$82,14 \pm 7,24$	$9,05 \pm 0,80$
25–34	15,79	$75,64 \pm 6,87$	$11,94 \pm 1,09$
35–49	26,29	$51,32 \pm 6,14$	$13,49 \pm 1,61$
50–64	24,89	$34,39 \pm 5,97$	$8,56 \pm 1,49$
65+	22,01	$21,96 \pm 5,66$	$4,83 \pm 1,25$
Total	100	<i>sans objet</i>	$47,88 \pm 2,86$

(Rappelons que quand on somme des estimations indépendantes, l'intervalle de confiance ne s'obtient pas en sommant les *écarts-types* mais les *variances* ; autrement dit la marge d'erreur est la somme *quadratique* des marges d'erreur individuelles (càd. la racine carrée de la somme des carrés).

Ainsi l'institut a légèrement amélioré la précision de son estimation, ce qui, compte tenu à quel point les résultats obtenus sont serrés, lui permet de renforcer sensiblement son niveau de confiance en l'élection du candidat de droite (de 0,897 à 0,931) — et effectivement, dans notre exemple celui-ci a gagné avec 51,46 % des voix.

## 2.2 Stratification “à priori”

**Théorème 2.4.** *Supposons qu'on cherche à évaluer par la méthode de Monte-Carlo une quantité de la forme  $\mathbb{E}(f_1) + \dots + \mathbb{E}(f_K)$  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  $\text{Eff}_k^{-1/2}$ , où  $\text{Eff}_k$  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  $(\text{Eff}_k q_k Z)^{-1}$ . 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  $\mathbb{E}(f_1) + \dots + \mathbb{E}(f_K)$  vaut  $\sum_k (\text{Eff}_k q_k)^{-1} \times Z^{-1}$ , ce qui correspond à une efficacité globale de

$$\left( \sum_{k=1}^K (\text{Eff}_k q_k)^{-1} \right)^{-1}.$$

Il faut donc maximiser (2.2), ou encore minimiser  $\sum_k \text{Eff}_k^{-1} q_k^{-1}$ , sous la contrainte que  $\sum_k q_k = 1$ . On va utiliser la méthode du multiplicateur de Lagrange : on introduit un paramètre  $\lambda$  et on cherche une valeur critique de  $f(q_1, \dots, q_K) := \sum_k \text{Eff}_k^{-1} q_k^{-1} + \lambda \sum_k q_k$ , qui vérifie la contrainte. Vu que  $\partial f / \partial q_k = -\text{Eff}_k^{-1} q_k^{-2} + \lambda$ , il y a un unique point critique pour  $q_k = \lambda^{-1/2} \text{Eff}_k^{-1/2} \forall k$ , ce qui montre (sans même avoir à calculer  $\lambda$ ) que les  $q_k$  optimaux sont proportionnels aux  $\text{Eff}_k^{-1/2}$ .  $\square$

*Exemple 2.5.* Imaginons que le Pape désire connaître avec précision nombre de catholiques sur l'île d'Irlande. Cette île est divisée en deux parties assez homogènes : l'Irlande du Sud, qui comporte une population totale d'exactement  $M_1 = 4\,500\,000$  hab (ce qu'on sait d'après le recensement), et l'Irlande du Nord, qui comporte exactement  $M_2 = 1\,800\,000$  hab. Par ailleurs, avant même d'avoir lancé son enquête, le Pape a déjà une idée approximative de la proportion respective de Catholiques au Sud et au Nord : *grosso-modo*  $p_1 \approx 90\%$  et  $p_2 \approx 50\%$ . Par ailleurs, il est relativement plus couteux de sonder les gens dans le Sud que dans le Nord : on peut ainsi estimer le cout du sondage à  $\alpha_1 = 3$  € par personne interrogée au Sud, contre  $\alpha_2 = 2$  € au Nord. Le Pape dispose d'une enveloppe de 10 000 € pour mener son sondage à bien. Comment l'utiliser au mieux pour obtenir une précision maximale sur le nombre total de ses ouailles ?

Commençons par ramener la situation au paradigme de Monte-Carlo. Notons  $\mathbb{P}_1$ , resp.  $\mathbb{P}_2$ , la loi uniforme sur les habitants du Sud, resp. du Nord. Le nombre  $C_1$  de Catholiques au Sud est alors l'espérance, sous  $\mathbb{P}_1$ , de «  $M_1$  fois l'indicatrice d'être catholique » (fonction que nous noterons  $f_1$ ), et on a la même chose pour le Nord avec des indices '2' ; et on cherche à évaluer  $\mathbb{E}_1(f_1) + \mathbb{E}_2(f_2)$ , ce qui nous place bien dans le cadre du théorème 2.4. Quelle est l'efficacité du calcul de  $\mathbb{E}_1(f_1)$  ? D'après les valeurs approximatives de  $p_1$  et  $p_2$ , nous savons que la variance  $\text{Var}_{\mathbb{P}_1}(f_1)$  sera environ  $M_1^2 p_1(1-p_1) \simeq 1,82 \cdot 10^{12}$ , d'où une efficacité de  $\text{Eff}_1 = \text{Var}_{\mathbb{P}_1}(f_1)^{-1} / \alpha_1 \simeq 1,83 \cdot 10^{13} \text{ €}^{-1}$ , et de même une efficacité au Nord  $\text{Eff}_2 \simeq 6,17 \cdot 10^{13} \text{ €}^{-1}$ . D'après notre théorème 2.4, l'allocation optimale du budget est donc d'allouer une proportion  $\text{Eff}_1^{-1/2} / (\text{Eff}_1^{-1/2} + \text{Eff}_2^{-1/2}) \simeq 64,8\%$  du budget à l'échantillonnage du Sud, soit 6 480 € lui permettant de sonder 2 160 Sudistes, avec de l'autre côté 3 520 pour le Nord lui permettant de sonder 1 760 habitants.

Avec ces paramètres, on calcule que l'écart-type sur le nombre total de Catholiques sera d'environ  $\pm 36\,100$  âmes. À titre de comparaison, une allocation des ressources à moitié-moitié aurait donné un écart-type  $\pm 37\,600$ , et un sondage au prorata de la population aurait conduit à  $\pm 38\,200$ .



## 2.3 Variables de contrôle

### 2.3.1 Version rudimentaire

**Définition 2.6.** Supposons qu'on cherche à évaluer  $\mathbb{E}(f)$  par la méthode de Monte-Carlo, et qu'on dispose d'une variable aléatoire  $g$  de classe  $L^2$  sur  $\Omega$  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

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

**Proposition 2.7.** 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 2.8.* 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  $\mathbb{R}$  issue de 0 dont les incréments sont de loi  $\mathcal{N}(1)$ ; autrement dit, pour  $(S_j)_{1 \leq j \leq 60}$  des v.a. i.i.d.  $\mathcal{N}(1)$ , on considère la suite finie  $(X_i)_{0 \leq i \leq 60}$  définie par  $X_j = \sum_{i=1}^j S_i$ . 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  $\omega$ , on pose  $X_*(\omega) := \max_{0 \leq i \leq 60} X_i(\omega)$ . 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  $\sigma := \sqrt{60}$ ,

$$\begin{aligned} \mathbb{E}(Y) &= \int_{-\infty}^{\infty} \max\{x, 0\} \frac{e^{-x^2/2\sigma^2}}{\sqrt{2\pi}\sigma} dx = \frac{1}{\sqrt{2\pi}\sigma} \int_0^{\infty} x e^{-x^2/2\sigma^2} dx \\ &= \frac{1}{\sqrt{2\pi}\sigma} [-\sigma^2 e^{-x^2/2\sigma^2}]_0^{\infty} = \frac{\sigma}{\sqrt{2\pi}} = 3,090\ 193... \end{aligned} \quad (2.2)$$

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

L'annexe A.6 montre comment implémenter l'utilisation de cette variable de contrôle. Sans variable de contrôle, on obtient le résultat suivant :

```
>> tic; maxmarche_spl(100000); toc;
E(X_*) : 5.6272 ± 0.029523
Elapsed time is 2.200732 seconds.
```

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

```
>> tic; maxmarche_ctrl0(100000); toc;
E(X_*) : 5.6155 ± 0.014469
Elapsed time is 2.264886 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.

### 2.3.2 Version paramétrée



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

**Théorème 2.9.** *La valeur de  $\lambda$  qui minimise  $\text{Var}(f - \lambda g)$  est*

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

Notant  $r := \text{Cov}(f, g) / \text{Var}(f)^{1/2} \text{Var}(g)^{1/2}$  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  $\text{Var}(f - \lambda g) = \text{Var}(f) - 2\lambda \text{Cov}(f, g) + \lambda^2 \text{Var}(g)$ . Il s'agit d'un polynôme de degré 2 en  $\lambda$  dont la minimisation est élémentaire : le minimum de  $\text{Var}(f - \lambda g)$  est atteint en  $\lambda^*$  et vaut  $\text{Var}(f) - \text{Var}(g)^{-1} \text{Cov}(f, g)^2 = (1 - r^2) \text{Var}(f)$ , d'où les résultats annoncés.  $\square$

*Exemple 2.10.* Un système de commande électronique utilise un circuit dédoublé pour plus de fiabilité : c'est-à-dire qu'il y a plusieurs circuits accomplissant la même tâche, et que le système ne tombe en panne que lorsque *tous* ces circuits sont hors d'usage. Dans la situation que nous considérons, on suppose que la durée de vie de chaque circuit est exponentielle. Les ingénieurs qui ont conçu le système l'ont équipé de trois circuits : un circuit principal dont le temps de vie caractéristique est de 5 ans, et deux circuits de secours moins onéreux dont le temps de vie caractéristique est de 2 ans. Ils se demandent quel sera l'espérance de vie globale du système.

Formalisé, le problème s'écrit ainsi : on a trois variables aléatoires  $T_1, T_2, T_3$  indépendantes de loi respectives *Exponentielle*(1/5), *Exponentielle*(1/2) et *Exponentielle*(1/2), et on s'intéresse à la variable  $T := \max\{T_1, T_2, T_3\}$ , dont on souhaite calculer l'espérance.

Quelle variable de contrôle pourrait-on utiliser ? Le choix de  $T_1$  semble judicieux, puisque.

### 2.3.3 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 (2.6) avec  $\lambda_1 g_1 + \dots + \lambda_K g_K$ ,  $(\lambda_1, \dots, \lambda_K)$  étant un jeu de  $K$  paramètres à optimiser.

**Théorème 2.11.** *La valeur de  $(\lambda_1, \dots, \lambda_K) =: \vec{\lambda}$  qui minimise  $\text{Var}(f - \lambda_1 g_1 - \dots - \lambda_K g_K)$  est donnée par*

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

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



*Démonstration.* Par bilinéarité,

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

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

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

où  $\|v\|^2$  désigne la norme  $vv^T$  d'un vecteur  $v$  de  $L^2(\mathbb{R}^K)$ . Le seul terme de (2.3.3) dépendant de  $\vec{\lambda}$  étant alors le carré de la norme de  $\vec{\lambda} \Sigma_g^{1/2} - R_{fg} \Sigma_g^{-1/2}$ , l'expression est minimale quand ce vecteur s'annule, ce qui donne bien  $\vec{\lambda}^* = R_{fg} \Sigma_g^{-1}$ .  $\square$

## 2.4 Variables antithétiques

**Théorème 2.12.** *Supposons que nous cherchons à évaluer  $\mathbb{E}(f)$  par la méthode de Monte-Carlo. Notant  $\Omega$  l'espace sur lequel vivent nos simulations, supposons que nous disposions d'une application non triviale  $\gamma: \Omega \rightarrow \Omega$  qui préserve la mesure  $\mathbb{P}$  (càd. que la mesure-image de  $\mathbb{P}$  par  $\gamma$  est encore  $\mathbb{P}$ ), et supposons que simuler  $(f(\omega), f(\gamma(\omega)))$  coûte deux fois plus cher que simuler  $f(\omega)$ . Alors, si les variables aléatoires  $f$  et  $f \circ \gamma$  sont négativement corrélées, l'estimateur*

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

*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  $\frac{1}{2}(f(\omega_i) + f(\gamma(\omega_i)))$ , laquelle se développe en  $\frac{1}{4}(\text{Var}(f) + \text{Var}(f \circ \gamma) + \text{Cov}(f, f \circ \gamma)) = \frac{1}{2} \text{Var}(f) + \frac{1}{4} \text{Cov}(f, f \circ \gamma) \leq \frac{1}{2} \text{Var}(f)$  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.  $\square$

## 2.5 Simulations communes

**Définition 2.13.** Supposons qu'on cherche à évaluer par la méthode de Monte-Carlo une quantité de la forme  $\mathbb{E}_1(f_1) - \mathbb{E}_0(f_0)$ , où  $f_1$  et  $f_0$  sont à priori définies sur des espaces de probabilités  $(\Omega_0, \mathbb{P}_0)$  et  $(\Omega_1, \mathbb{P}_1)$ . La *méthode des simulations couplées* consiste alors à trouver un couplage entre les lois  $\mathbb{P}_0$  et  $\mathbb{P}_1$  de façon à pouvoir définir  $f_0$  et  $f_1$  sur un même espace de probabilité  $(\Omega, \mathbb{P})$ , et à utiliser directement l'estimateur de Monte-Carlo  $\mathbb{E}(f_1 - f_0)$ .

**Proposition 2.14.** *Supposons que le cout de simulation de  $(f_1(\omega) - f_0(\omega))$  (avec couplage) ne soit pas plus grand que le cout de simulation de  $(f_1(\omega^{(1)}) - f_0(\omega^{(0)}))$  (sans couplage). Alors, si les variables aléatoires  $f$  et  $g$  (couplées) sont positivement corrélées, l'efficacité du calcul utilisant les simulations couplées est plus grande que de faire deux estimations séparées pour  $\mathbb{E}_0(f_0)$  et  $\mathbb{E}_1(f_1)$  (en supposant qu'on utiliserait le même nombre de simulations pour les évaluations de  $\mathbb{E}_0(f_0)$  et de  $\mathbb{E}_1(f_1)$ ).*



*Démonstration.* L'estimateur sans couplage, pour  $N$  simulations par espérance à évaluer, s'écrit

$$\hat{m}^{\text{naif}} = N^{-1} \sum_{i=1}^N f_1(\omega_i^{(1)}) - N^{-1} \sum_{i=1}^N f_0(\omega_i^{(0)}) = N^{-1} \sum_{i=1}^N (f_1(\omega_i^{(1)}) - f_0(\omega_i^{(0)})),$$

où  $f_1(\omega_i^{(1)})$  et  $f_0(\omega_i^{(0)})$  sont indépendants, d'où une variance pour  $(f_1(\omega_i^{(1)}) - f_0(\omega_i^{(0)}))$  égale à  $\text{Var}(f_0) + \text{Var}(f_1)$ , et une efficacité par (paire de) simulation(s) égale à l'inverse de cette quantité.

Pour l'estimateur avec couplage, par contre, la variance de  $(f_1(\omega_i) - f_0(\omega_i))$  vaut par bilinéarité  $\text{Var}(f_0) + \text{Var}(f_1) - 2 \text{Cov}(f_0, f_1) \leq \text{Var}(f_0) + \text{Var}(f_1)$  d'après l'hypothèse de corrélation positive. L'efficacité par simulation est donc meilleure avec les simulations couplées; et comme le coût par simulation n'est pas plus grand, à fortiori l'efficacité tout court est meilleure elle aussi.  $\square$

*Exemple 2.15.* Un match de rugby va avoir lieu entre les équipes d'Agen ( $A$ ) et Brive ( $B$ ). L'entraîneur de Brive se demande quelle est la meilleure stratégie à adopter pour son équipe, en particulier concernant la prise de risques.

Nous considérons ici un modèle simplifié du rugby. Dans ce modèle, chaque équipe dispose au cours du match de 12 séquences de jeu. À l'issue d'une séquence, ou bien l'équipe qui avait l'initiative est repoussée et ne marque alors aucun point, ou bien elle prend l'avantage. Quand une équipe a pris l'avantage, elle a le choix entre deux options : soit prendre le but et marquer systématiquement 3 points, soit chercher à marquer un essai. Dans ce dernier cas, l'équipe marque 7 points si elle parvient à inscrire l'essai, et aucun sinon.

Quand une équipe choisit de chercher l'essai, elle y parvient dans 35 % des cas, ce qui lui donne une espérance de 2,45 points par tentative : *en moyenne*, il est donc plus intéressant de prendre le but. Toutefois, l'essai permet de scorer plus de points d'un seul coup et représente ainsi une prise de risques qui peut être payante si on est face à une équipe plus forte que soi et qu'on doit compter sur la chance pour l'emporter...

En l'occurrence, l'équipe  $A$  prend l'avantage dans 60 % de ses séquences de jeu, contre 40 % pour l'équipe  $B$ . L'équipe  $A$  étant favorite, elle n'a pas intérêt à prendre de risques et choisit donc systématiquement le but quand elle peut. Par contre, l'équipe  $B$  a vraisemblablement intérêt à chercher l'essai de temps en temps... Oui, mais à quel point ? Exceptionnellement, rarement, souvent, toujours... ?

Formalisons le problème. Pour  $q \in [0, 1]$ , nous appelons  $S_q$  la stratégie consistant, lorsque l'équipe  $B$  a pris l'avantage, à chercher l'essai avec probabilité  $q$  et à prendre le but sinon; et nous notons  $\mathbb{P}_q$  la loi de probabilité décrivant le score d'un match où l'équipe  $B$  a suivi la stratégie  $S_q$ . On note  $B$  l'événement « l'équipe  $B$  gagne » (les

matches nuls étant comptabilisé comme des défaites). Ainsi formalisé, le problème pour l'entraîneur est donc de maximiser  $\mathbb{P}_q(B)$  en fonction de  $q$ . Plus exactement, on va comparer six choix possibles, correspondant à  $q \in \{0, 0,1, 0,2, \dots, 1\}$ .

Imaginons que l'entraîneur soit limité à 15 000 simulations par valeur de  $q$ . Le code de Monte-Carlo « naïf » correspondant est donné dans l'annexe A.7.1. L'entraîneur obtient les résultats suivants :

```
>> tic; for q = 0:.2:1; rugby_spl(q); end; toc;
Probabilité de victoire de B pour q = 0 : entre 0.10999 et 0.12041
Probabilité de victoire de B pour q = 0.2 : entre 0.13873 et 0.15021
Probabilité de victoire de B pour q = 0.4 : entre 0.14496 et 0.15664
Probabilité de victoire de B pour q = 0.6 : entre 0.1464 et 0.15813
Probabilité de victoire de B pour q = 0.8 : entre 0.14273 et 0.15434
Probabilité de victoire de B pour q = 1 : entre 0.13394 et 0.14526
Elapsed time is 2.687767 seconds.
```

Aïe : les intervalles de confiance se recouvrent largement... Tout ce que l'entraîneur peut dire, c'est que le meilleur choix pour  $q$  est entre 0,2 et 0,8.

Mais en réalité, ce ne sont pas les *valeurs* des  $\mathbb{P}_q(B)$  qui intéressent l'entraîneur, mais leurs *différences*... Du coup, s'il trouve un couplage entre les différentes lois  $\mathbb{P}_q$  tel que les événements  $B$  sous ces différentes lois se retrouvent fortement corrélés, il pourra utiliser la méthode de simulations communes et obtenir ainsi un résultat bien plus précis avec le même nombre de simulations !

Comment réaliser un tel couplage ? L'idée est toute simple : on va réaliser les différentes lois  $\mathbb{P}_q$  en simulant en parallèle six matchs où tous les phénomènes aléatoires (à savoir : quel est le score de l'autre équipe ? arrivons-nous à prendre l'avantage ? arrivons-nous à marquer l'essai ?) se produisent de la *même* façon pour les six simulations, la seule chose qui change étant la décision ou pas de chercher l'essai, de sorte que les six simulations correspondront bien respectivement à chacune des lois  $\mathbb{P}_q$  qu'on souhaite simuler. En outre, même les décisions de chercher ou pas l'essai seront corrélées entre les différents matchs simulés en parallèle : en effet, on prendra toutes ces décisions à partir du même tirage uniforme  $[0, 1]$ , de sorte que si p.ex.  $q_2 > q_1$ , à chaque fois que la simulation de  $\mathbb{P}_{q_1}$  choisira de chercher  $q_2$ , la simulation de  $\mathbb{P}_{q_2}$  le choisira également.

Le programme correspondant, donné dans l'annexe A.7.2, donne le résultat suivant :

```
>> tic; rugby_simcom(0:.2:1); toc;
Tableau des différences de probabilités:
      0   -0.0318   -0.0379   -0.0346   -0.0247   -0.0157
0.0318      0   -0.0061   -0.0028    0.0071    0.0161
0.0379    0.0061      0    0.0033    0.0132    0.0221
0.0346    0.0028   -0.0033      0    0.0099    0.0189
0.0247   -0.0071   -0.0132   -0.0099      0    0.0089
0.0157   -0.0161   -0.0221   -0.0189   -0.0089      0
```

```
±
      0    0.0015    0.0018    0.0019    0.0020    0.0022
```

0.0015	0	0.0011	0.0016	0.0018	0.0021
0.0018	0.0011	0	0.0010	0.0015	0.0018
0.0019	0.0016	0.0010	0	0.0010	0.0014
0.0020	0.0018	0.0015	0.0010	0	0.0009
0.0022	0.0021	0.0018	0.0014	0.0009	0

Elapsed time is 1.048773 seconds.

La troisième ligne du premier tableau ne contient que des valeurs positives ; c'est donc la valeur  $q = 0,4$  associée à cette ligne qui semble le meilleur choix. Mais surtout, le second tableau nous indique que les valeurs en question sont effectivement positives *même quand on tient compte de la marge d'erreur* : ainsi, contrairement au cas précédent où on était incapable de trancher, les simulations communes nous permettent d'affirmer que cette réponse tient avec un niveau de certitude d'au moins 95 % ! Et en prime, les calculs sont même plus rapides...

## 2.6 Conditionnement



**Définition 2.16.** 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

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

**Théorème 2.17.** *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,  $\mathbb{E}((f^{\mathcal{B}})^2) \leq \mathbb{E}(f^2)$ , et comme en outre on a  $\mathbb{E}(f^{\mathcal{B}}) = \mathbb{E}(f)$ , il s'ensuit que  $\text{Var}(f^{\mathcal{B}}) = \mathbb{E}((f^{\mathcal{B}})^2) - \mathbb{E}(f^{\mathcal{B}})^2 \leq \mathbb{E}(f^2) - \mathbb{E}(f)^2 = \text{Var}(f)$ , ce qui prouve que l'efficacité par simulation est effectivement meilleure pour  $f^{\mathcal{B}}$  que pour  $f$ .  $\square$

*Remarque 2.18.* 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.

*Exemple 2.19.* Reprenons l'exemple du *yahtzee* (exemple 1.3). Notons  $\mathbb{P}$  la loi décrivant les résultats successifs obtenus par le joueur au cours de ses trois lancers et  $f$  l'indicatrice du succès final du joueur, de sorte que la probabilité  $p$  recherchée s'écrit comme  $\mathbb{E}(f)$ . En termes de tribus, nous avons simulé  $f$  comme une fonction  $\mathcal{A}$ -mesurable, où  $\mathcal{A}$  est la tribu correspondant à l'ensemble des résultats des trois lancers effectués. Considérons maintenant la tribu  $\mathcal{B}$  la engendrée par les résultats des *deux* premiers lancers.  $\mathcal{B}$  est évidemment une sous-tribu de  $\mathcal{A}$ . Pour appliquer la technique de conditionnement, peut-on calculer l'espérance conditionnelle  $\mathbb{E}(f|\mathcal{B})$  ?

Ici, il vaut mieux raisonner en phrase plutôt qu'en formules. L'espérance conditionnelle de  $f$  sachant  $\mathcal{B}$  signifie ceci : le joueur vient de faire son deuxième lancer et s'apprête à effectuer le troisième. Il se demande : « Sachant tout ce qui s'est passé pur l'instant dans ce coup, que vaudra  $f$  en moyenne, autrement dit, quelle est la probabilité que j'obtienne un yahtzee à l'issue de mon dernier lancer ? ». Mais cela est facile à calculer ! Prenons un exemple pour mieux comprendre : à l'issue de son deuxième lancer, imaginons que le joueur a obtenu deux '4', deux '1' et un '6'. Il décide ne ne garder que les deux '4' (mais ça ne changerait bien sûr rien s'il ne gardait plutôt que les deux '1') et de relancer les trois autres dés en espérant obtenir un yahtzee de '4' (ce qui est évidemment le seul type de yahtzee qu'il puisse obtenir). Quelle est a probabilité que cela réussisse ? Il faudra que chacun des trois dés qu'il relance tombe sur '4', ce qui a une probabilité de survenir de  $(1/6)^3 = 1/216$ . Plus généralement, si le chiffre le plus présent à l'issue du second lancer figure en  $k$  exemplaires, le joueur va relancer les  $(5 - k)$  dés n'affichant pas ce chiffre en espérant que ceux-ci retomberont tous sur le chiffre en question ; et cela qui se produit avec probabilité  $1/6^{5-k}$ . En conclusion,

$$\mathbb{E}(f|\mathcal{B}) = \frac{1}{6^{5-k}},$$

où  $k$  est la variable aléatoire qui compte le plus grand nombre d'occurrences d'un dé à l'issue du second lancer, variable aléatoire qui est bien  $\mathcal{B}$ -mesurable.

Le code correspondant est donné à l'annexe A.8.2. La comparaison avec l'algorithme "naïf" (repris dans l'annexe A.8.2 pour qu'il calcule aussi l'intervalle de confiance) donne :

```
>> tic;yahtzee_spl(100000);toc;
Probabilite de succès : 0.04574 ± 0.0013213
Elapsed time is 13.012523 seconds.
>> tic;yahtzee_cond(100000);toc;
Probabilite de succès : 0.045711 ± 0.00074245
Elapsed time is 8.529767 seconds.
```

Non seulement la largeur de l'intervalle de confiance est réduite d'un facteur presque 2, mais en plus le programme va  $1\frac{1}{2}$  fois plus vite, soit au final un gain d'efficacité d'un facteur 5 environ.

## 2.7 Échantillonnage préférentiel

**Définition 2.20.** Supposons qu'on cherche à évaluer  $\mathbb{E}_P(f)$  par la méthode de Monte-Carlo,  $\mathbb{E}_P$  désignant l'espérance associée à une certaine loi  $P$  sur l'espace mesurable  $\Omega$ .



Supposons qu'on dispose d'une autre loi  $Q$  sur le même espace  $\Omega$  qu'on sait simuler, et que  $P$  a une densité  $(dP/dQ)(\omega)$  par rapport à  $Q$  qu'on connaît exactement. Alors la *technique d'échantillonnage préférentiel* consiste à observer que  $\mathbb{E}_P(f) = \mathbb{E}_Q(dP/dQ \times f)$  et à estimer  $\mathbb{E}_P(f)$  par

$$\hat{m}^{\text{pref}} := N^{-1} \sum_{i=1}^N \frac{dP}{dQ}(\tilde{\omega}_i) f(\tilde{\omega}_i),$$

où les  $\tilde{\omega}_i$  sont simulés i.i.d. selon la loi  $Q$ .

**Proposition 2.21.** *L'efficacité par simulation de la technique d'échantillonnage préférentiel est égale à l'inverse de  $(\mathbb{E}_P(dP/dQ \times f^2) - \mathbb{E}_P(f)^2)$ .*



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



**Théorème 2.22.** *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 2.21, 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, c.à.d. que  $\rho \geq 0$  et  $\mathbb{E}_P(\rho) = 1$ . Faisons ici comme si  $f$  était non nulle presque-partout ; alors il est clair le  $\rho$  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  $\lambda$  à fixer, on cherche un  $\rho^*$  vérifiant  $\rho^* \geq 0$  et  $\mathbb{E}_P(\rho^*) = 1$  tel que  $\rho^*$  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  $(DF(\rho)) \cdot h = \mathbb{E}_P((-\rho^{-2} f^2 + \lambda)h)$ , de sorte qu'un point critique  $\rho^*$  doit vérifier  $-(\rho^*)^{-2} f^2 + \lambda \equiv 0$ , d'où  $\rho^* \propto |f|$ .  $\square$



*Remarque 2.23.* Dans le cas où on utilise la méthode de Monte-Carlo pour évaluer une intégrale par rapport à la mesure de Lebesgue, le théorème 2.22 devient que la probabilité d'échantillonnage optimale est celle dont la densité *par rapport à la mesure de Lebesgue* est proportionnelle à  $|f|$  : la preuve est la même, *mutatis utandis*.

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  $\omega$  pour lesquelles  $f$  prend une valeur particulièrement grande, il faudra chercher une loi de simulation  $Q$  qui donne plus de poids à ces éventualités-là !



*Remarque 2.24.* 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 2.21, 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 ? ».



*Remarque 2.25.* 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 événements de probabilités très faibles.

*Exemple 2.26.* 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 la conclusion que trois facteurs sont impliqués dans un tel risque, facteurs qu'elle appelle respectivement  $X$ ,  $Y$  et  $Z$ , et que ces trois vecteurs sont distribués selon la loi normale

$$(X, Y, Z) \sim (-3, -4, -5) + \mathcal{N} \begin{pmatrix} 1 & 0,2 & 0,3 \\ 0,2 & 1 & 0,4 \\ 0,3 & 0,4 & 1 \end{pmatrix}.$$

L'accident se produit quand les trois facteurs  $X$ ,  $Y$  et  $Z$  deviennent *tous les trois* positifs ; il faut donc évaluer  $\mathbb{P}((X, Y, Z) \in \mathbb{R}_+^3) =: p$ . La norme de certification impose que  $p$  soit inférieur à un milliardième.

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.9.1. Avec un-million simulations, elle obtient le résultat suivant :

```
>> risque_spl(1000000)
Probabilité d'accident : 0 ± 0
```

À première vue, il n'y a donc strictement aucun risque. Mais en fait, réfléchit l'ingénieure, de résultat est dégénéré : il signifie simplement que toutes les simulations ont été négatives ! Mais si le risque véritable est d'un cent-millionième, ce n'est pas en faisant seulement un-million simulations que je vais pouvoir voir quelque chose... Le problème, c'est que les capacités de calcul de l'ordinateur ne permettent guère de dépasser un million de simulations... D'où l'idée d'utiliser une technique d'échantillonnage préférentiel.

Appelons  $f = \mathbf{1}_{\{\text{accident}\}}$  l'indicatrice dont l'ingénieure veut évaluer l'espérance et  $P$  la loi du phénomène.  $f$  étant  $(X, Y, Z)$ -mesurable, on peut voir  $P$  comme une loi sur les valeurs de  $(X, Y, Z)$ . Nous abrégeons (2.26) sous la forme «  $(X, Y, Z) \sim M + \mathcal{N}(\Sigma)$  ». Quelle loi d'échantillonnage  $Q$  allons-nous considérer ? On veut que  $Q$  donne un poids assez important aux triplets  $(X, Y, Z)$  de  $\mathbb{R}_+^3$ , sans trop en sous-échantillonner aucun. Une idée est de prendre  $Q$  telle que, sous  $Q$ ,  $(X, Y, Z) \sim \mathcal{N}(\Sigma)$  : sous cette nouvelle loi les valeurs de  $(X, Y, Z)$  seront proches de  $(0, 0, 0)$ , donc elles couvriront bien mieux la zone qui nous concerne que sous  $P$ . Peut-on calculer  $dP/dQ$  ? Oui, assez facilement. En effet, nous connaissons la formule de la densité pour les variables gaussiennes, qui nous dit que sous  $P$ ,

$$dP((X, Y, Z) = (x, y, z) =: \vec{v}) = \frac{(2\pi)^{-3/2}}{\det C} \exp\left(-\frac{1}{2}(\vec{v} - M)C(\vec{v} - M)^\top\right) dx dy dz,$$

tandis que sous  $Q$ , il faut remplacer  $M$  par 0 :

$$dQ((X, Y, Z) = (x, y, z) =: \vec{v}) = \frac{(2\pi)^{-3/2}}{\det C} \exp\left(-\frac{1}{2}\vec{v}C\vec{v}^\top\right) dx dy dz ;$$

de sorte qu'en prenant le quotient, on trouve

$$\begin{aligned} \frac{dP}{dQ}((X, Y, Z) = (x, y, z) =: \vec{v}) &= \exp\left(-\frac{1}{2}(\vec{v} - M)C^{-1}(\vec{v} - M)^\top + \frac{1}{2}\vec{v}C^{-1}\vec{v}^\top\right) \\ &= \exp\left(\frac{1}{2}\vec{v}C^{-1}M^\top + \frac{1}{2}\vec{v}^\top C^{-1}M - \frac{1}{2}MC^{-1}M^\top\right) = e^{-MC^{-1}M^\top/2} \exp(\vec{v}C^{-1}M^\top). \end{aligned}$$

Cela conduit au code de l'annexe A.9.2, dont l'exécution donne :

```
>> risque_pref(200000)
Probabilité d'accident : 5.1108e-10 ± 1.4781e-11
```

Cette fois-ci le résultat est tout-à-fait exploitable, comme le confirme la finesse de l'intervalle de confiance. On trouve que les accidents arriveront avec une probabilité d'environ  $\frac{1}{2}$  milliardième, ce qui permet donc de certifier la centrale. (Ici le temps de calcul par simulation a été environ 2 fois plus lent dans la version préférentielle à cause de la complexité supplémentaire due au calcul et à la prise en compte de la densité, mais le gain de variance est tellement important qu'on a pu pallier cet inconvénient en prenant 5 fois moins de simulations, tout en obtenant encore un résultat incomparablement meilleur).

## 2.8 Optimisation du code

### 2.8.1 Considérations générales



Il ne faut pas oublier que le cout d'exécution d'un algorithme ne dépend pas seulement des calculs mathématiques qui le sous-tendent, mais aussi de la façon dont ces calculs sont implémentés. Le niveau zéro du gain d'efficacité consiste donc tout simplement à optimiser son code !



## 2.8.2 Parallélisation

Dans le cas précis des méthodes de Monte-Carlo, une méthode d'optimisation du code utile est d'utiliser si possible les propriétés de parallélisabilité de la méthode. La parallélisation peut intervenir à deux niveaux :

- En observant que les différentes simulations sont soumises aux mêmes calculs, on peut utiliser un parallélisme de type SIMD (*Single Instruction, Multiple Data*), également appelé « vectorisation ». Cela implique d'utiliser un langage de programmation conçu pour exploiter les capacités vectorielles du processeur.
- En observant d'autre part que le traitement des différentes simulations est indépendant dans la mesure où ce n'est qu'à la fin qu'on est obligé de sommer les résultats, on peut aussi utiliser un parallélisme de type MIMD (*Multiple Instruction, Multiple Data*), également appelé « calcul distribué ». Cela implique de disposer d'un ferme de processeurs sur lesquels on peut effectuer des calculs indépendants.

*Remarque 2.27.* Le calcul peut être distribué non seulement dans l'espace mais aussi *dans le temps*. Supposons ainsi qu'après avoir lancé une méthode de Monte-Carlo avec 1 million de simulations, l'ingénieur s'aperçoive que la précision obtenue est insuffisante et qu'il aurait en fait besoin de 2 millions de simulations. A-t-il besoin pour autant de relancer 2 millions de simulations ? Non ! Il peut aussi considérer le million de simulations effectués comme étant la première moitié de ses simulations, et lancer (indépendamment) 1 million de nouvelles simulations pour obtenir un échantillon de 2 millions. Notamment, il n'a pas besoin pour cela d'avoir gardé en mémoire tous les résultats obtenus pour le premier million de simulations, mais seulement la somme ces résultats (ce qui est équivalent à mémoriser l'estimateur avec une précision suffisante), somme à laquelle il ajoutera ensuite les résultats du second million de simulations.

## 2.9 Combiner les différentes techniques

Les différentes techniques de réduction de la variance que vous avons présentées ci-dessus ne sont pas mutuellement exclusives : on peut très bien combiner plusieurs d'entre elles pour une efficacité encore meilleure.

# Chapitre 3

## Initiation aux processus aléatoires

### 3.1 Généralités sur les processus aléatoires

**Définition 3.1.** Pour  $E$  un espace métrique, une fonction  $f: \mathbb{R}_+ \rightarrow E$  est dite *càdlàg* (continue à droite avec limite à gauche) si  $f(t) = \lim_{u \rightarrow t} f(u)$  pour tout  $t \geq 0$ , et que pour tout  $t > 0$ ,  $\lim_{u \rightarrow t} f(u)$  existe dans  $E$  (cette limite étant alors notée  $f(t-)$ ). Dans la suite, nous appellerons *trajectoire à valeurs dans  $E$*  une fonction càdlàg de  $\mathbb{R}_+$  dans  $E$ .



**Définition 3.2.** Pour  $E$  un espace métrique séparable, un *processus aléatoire à valeurs dans  $E$*  est une variable aléatoire qui prend ses valeurs dans l'espace des trajectoires de  $\mathbb{R}_+$  dans  $E$ ; autrement dit c'est une trajectoire aléatoire  $(Z_t)_{t \geq 0}$ .



*Remarque 3.3.* Pour traiter la trajectoire  $(Z_t)_{t \geq 0}$  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 3.4.** Un processus aléatoire  $(Z_t)_{t \geq 0}$  est qualifié de *markovien (homogène)* quand la prédiction de son futur au-delà d'un instant  $t_1$  ne demande que de connaître

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 :

$$\text{Loi}((Z_t)_{t>t_1} | (Z_t)_{t\leq t_1}) = \text{Loi}((Z_t)_{t>t_1} | Z_{t_1})$$

et

$$\text{Loi}((Z_{t_1+u})_{u\geq 0} | Z_{t_1} = z) = \text{Loi}((Z_u)_{u\geq 0} | Z_0 = z).$$

L'ensemble (3.4)-(3.4) est appelé « propriété de Markov (faible) ».

**Définition 3.5.** 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 3.6.*

- Si  $Z$  est un processus aléatoire à valeurs dans  $\mathbb{R}$ , « le premier instant  $T$  tel que  $|Z_T - Z_{T-1}| \geq 3$  » 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 connaît pas  $Z_t$  sur  $(T, 1]$ .

**Théorème 3.7.** (*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 (3.4) s'applique aussi à un temps d'arrêt : pour tout temps d'arrêt  $T$ ,

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



## 3.2 Mouvement brownien

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

### 3.2.1 Heuristique et définition

Soit  $X$  une loi à valeurs dans  $\mathbb{R}^d$  de classe  $L^2$ , centrée, avec pour matrice de covariance  $\sigma^\top \sigma$ . Considérons la marche aléatoire de pas  $X$ , c.à.d. 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, c.à.d. que pour une très grande valeur  $N$  on s'intéresse au comportement de la trajectoire aléatoire  $(S_{[tN]})_{t\geq 0}$ . Comme  $S_{[tN]}$  sera de l'ordre de  $N^{1/2}$  d'après le théorème-limite central, on s'intéressera plus exactement à  $(N^{-1/2} S_{[tN]})_{t\geq 0}$ , abrégé dans la suite en  $(B_t^N)_{t\geq 0}$ .

À  $N$  fixé, le processus  $B^N$  est un certain processus aléatoire à valeurs dans  $\mathbb{R}^d$ . Quand  $N \rightarrow \infty$ , 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  $(S_n)_{n \in \mathbb{R}}$  (voir la preuve du théorème 3.10) :

**Définition 3.8.** On appelle *mouvement brownien  $d$ -dimensionnel de covariance par unité de temps*  $\sigma^\top \sigma$  le processus  $(B_t)_{t \geq 0}$  défini de la façon suivante :

- $B_0 = 0$  avec probabilité 1 ;
- Pour  $0 = t_0 < t_1 < \dots < t_k$ , 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  $(t_1 - t_0)\sigma\sigma^\top$ .

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



**Théorème 3.9.** *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 connaître la loi de  $(B_{t_1}, \dots, B_{t_k})$  pour tout  $k$ -uplet  $(t_1, \dots, t_k) \in (\mathbb{R}_+)^k$  (l'ensemble de ces lois constituant ce qu'on appelle les marginales finidimensionnelles du mouvement brownien), comme l'explicitera la proposition 3.19 ci-dessous.*

**Théorème 3.10.** *Quand  $N \rightarrow \infty$ , la loi de la trajectoire  $(B_t^N)_{t \geq 0}$  converge vers le mouvement brownien (de variance par unité de temps  $\sigma\sigma^\top$ ) « au sens des marginales fini-dimensionnelles », c'est-à-dire que pour tout  $k$ -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  $(\mathbb{R}^d)^k$ ) converge vers celle de  $(B_{t_1}, \dots, B_{t_k})$ .*

*Remarque 3.11.* 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  $\sigma^\top \sigma$  de celle-ci intervenant.

*Exemple 3.12.*

1. La figure 3.2 montre le graphe d'un (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 ?? 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*, c.à.d. 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.

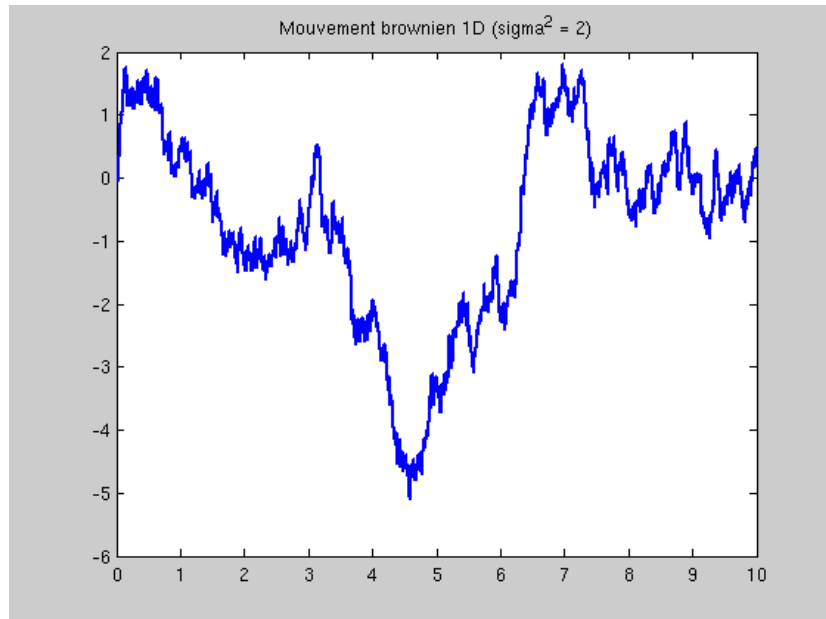


FIGURE 3.1 – Graphe d'un mouvement brownien unidimensionnel sur l'intervalle de temps  $[0, 10]$ .

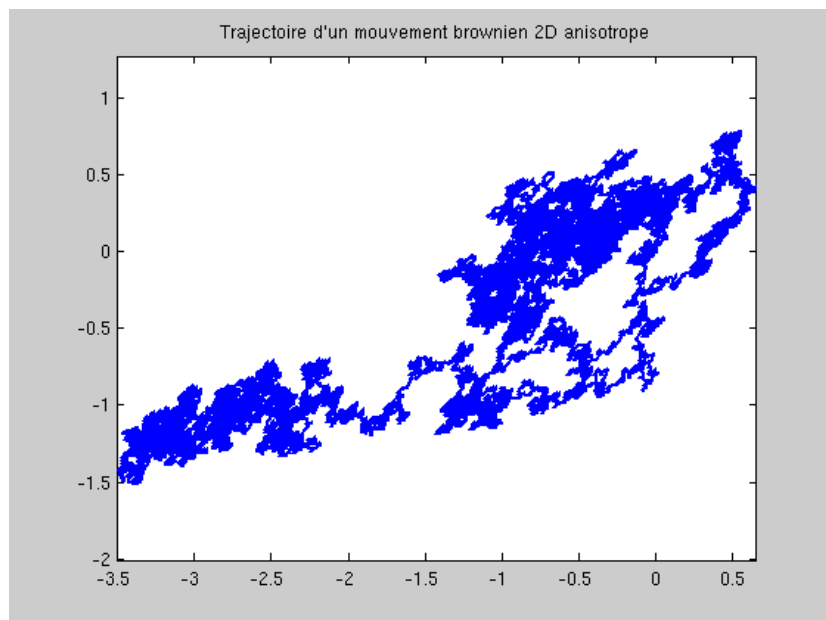


FIGURE 3.2 – Trajectoire d'un mouvement brownien 2-dimensionnel. La matrice de covariance par unité de temps a été choisie de sorte que  $dX_t^1$  et  $dX_t^2$  aient un coefficient de corrélation de  $+0,5$ .

### 3.2.2 Simulation

#### Généralités



Le mouvement brownien étant un processus à temps continu, simuler complètement sa trajectoire, même sur un intervalle de temps borné, requerrait 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, c.à.d. 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  $\varepsilon$  très petit. D'autres choix de discrétisation sont possibles, notamment quand il importe de connaître la trajectoire avec plus de précision sur certaines parties spécifiques. Éventuellement, les instants de discrétisation pourront dépendre de 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  $\varepsilon$  ; de l'autre, que le coût de simulation reste raisonnable, ce qui impose de ne pas prendre trop d'instants de discrétisation. L'équilibre précis entre ces deux 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 3.13.** 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  $\mathcal{N}((t_{i+1} - t_i)\sigma^{\top}\sigma)$ .





*Exemple 3.14.* Les programmes `brownien1D.m` et `brownien2D.m` de l'annexe ?? implémentent la méthode d'Euler stochastique pour simuler respectivement des mouvements browniens 1- et 2-dimensionnels. Leurs rendus correspondent aux figures 3.2 et ?? présentées un peu plus haut.

**Définition 3.15.** 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 3.16.** Conditionnellement à  $B_{t_1}$  et  $B_{t_2}$ ,  $B_{(t_1+t_2)/2}$  suit la loi  $(B_{t_1} + B_{t_2})/2 + \mathcal{N}((t_2 - t_1)\sigma^\top \sigma / 4)$ .



### Erreur de simulation



**Théorème 3.17.** Soit  $(B_t)_{t \geq 0}$  un mouvement brownien unidimensionnel de variance par unité de temps  $\sigma^2$ . Conditionnellement à  $B_{t_1}$  et  $B_{t_2}$ , la probabilité que le véritable mouvement s'écarte de plus de  $\varepsilon$  de son interpolation linéaire sur l'intervalle  $[t_1, t_2]$  est majorée par  $2 \exp(-2\varepsilon^2 / (t_2 - t_1)\sigma^2)$ .



*Remarque 3.18.* La borne du théorème ci-dessus étant exponentielle, 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  $\varepsilon$  soit infime.

## 3.2.3 Propriétés

### Loi du mouvement brownien

**Proposition 3.19.** Pour tout  $(t_1, \dots, t_k) \in (\mathbb{R}_+)^k$ , 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

$$\text{Cov}(B_{t_i}, B_{t_j}) = (t_i \wedge t_j) \sigma^\top \sigma.$$

**Proposition 3.20.** Si  $\sigma$  est une matrice de  $\mathbb{R}^{d \times d}$  et  $(B_t)_{t \geq 0}$  un mouvement brownien standard, alors  $(B_t \sigma)_{t \geq 0}$  est un mouvement brownien de covariance par unité de temps  $\sigma^\top \sigma$ .

**Proposition 3.21.** Si  $(B_t^1)_{t \geq 0}, \dots, (B_t^d)_{t \geq 0}$  sont des mouvements browniens unidimensionnels standard indépendants, alors  $(B_t^1, \dots, B_t^d)_{t \geq 0}$  est un mouvement brownien  $d$ -dimensionnel standard.

## Propriétés du processus

**Théorème 3.22.** *Le mouvement brownien est un processus markovien. En outre, on connaît exactement la loi conditionnelle de son futur :*

$$\text{Loi}((B_{t_1+u})_{u \geq 0} | B_{t_1} = x) \sim (x + \tilde{B}_u)_{u \geq 0},$$

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

**Théorème 3.23.** *Pour tout  $t_1 \geq 0$ ,  $(B_{t_1+u} - B_{t_1})_{u \geq 0}$  suit la même loi que  $(B_u)_{u \geq 0}$  : on dit que le mouvement brownien est à accroissements stationnaires.*

**Théorème 3.24.** *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).*

**Corollaire 3.25.** *Soit  $B$  un mouvement brownien unidimensionnel. Pour  $M < \infty$ , notons  $\tau$  le premier instant  $t$  pour lequel  $|B_t| \geq M$ . Ce processus définit un temps d'arrêt ; notons  $\tilde{B}_t := B_{t \wedge \tau}$  le « mouvement brownien stoppé au temps  $\tau$  ». 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 3.26.** *Avec probabilité 1, les trajectoires du mouvement brownien sont continues.*

## 3.3 Équations différentielles stochastiques

### 3.3.1 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 allons 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  $\mathbb{R}$ . 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  $\mathbb{R}$  et pas seulement sur  $\mathbb{R}_+$ .

**Définition 3.27.** On appelle *bruit blanc gaussien* (unidimensionnel) d'intensité  $\sigma^2$  une distribution aléatoire  $T$  sur  $\mathbb{R}$  (à valeurs réelles) telle que, pour toute fonction-test  $\varphi \in \mathcal{D}(\mathbb{R})$ , on ait  $\langle T, \varphi \rangle \sim \mathcal{N}(\int_{\mathbb{R}} \varphi(x)^2 dx)$ . 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 3.28.** 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  $a \in \mathbb{R}$ ,  $\delta_a * T$  a la même loi que  $T$ .

*Démonstration.* Il suffit de revenir à la définition. □

### 3.3.2 Principe général des équations différentielles stochastiques

**Définition 3.29.** Une équation différentielle stochastique (markovienne) à valeurs dans  $\mathbb{R}^n$  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  $\mathbb{R}^d$  (on aura généralement  $d = n$ , car on peut toujours se ramener à ce cas), et  $\sigma: \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$  et  $b: \mathbb{R}^n \rightarrow \mathbb{R}^n$  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  $X_0, X_\varepsilon, X_{2\varepsilon}, \dots$  en assimilant «  $dX_{n\varepsilon}$  » à «  $X_{n\varepsilon+\varepsilon} - X_{n\varepsilon}$  », etc. ⚠

*Remarque 3.30.* Quand nous simulons 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  $\varepsilon$  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 certains situations.

*Remarque 3.31.* La notation «  $dW_t$  » dans l’équation différentielle stochastique laisse à penser qu’on a besoin de simuler le mouvement brownien  $(W_t)_{t \geq 0}$  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  $\varepsilon$  est la largeur du pas de temps concerné). Simuler  $(W_t)_{t \geq 0}$  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  $(dW_t)_{t \geq 0}$ , voir aussi la § 3.3.1 sur le bruit blanc gaussien).

*Exemple 3.32.* Des exemples d'implémentation de simulation de processus stochastiques sont donnés par les programmes `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 § 3.3.4. Les rendus de ces programmes correspondent respectivement aux figures 3.3 et 3.4.

### 3.3.3 Changement de variables

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

**Théorème 3.33.** *Soit un processus  $X_t$  suivant l'équation différentielle stochastique «  $dX_t = dW_t \sigma(X_t) + b(X_t)dt$  ». Si  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  est de classe  $\mathcal{C}^2$ , alors  $Y_t = f(X_t)$  suit l'équation différentielle stochastique :*

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

*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 :*

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

On voit ainsi que la formule d'Itô s'apparente à un changement de variables classique, à ceci près qu'il apparaît en plus un terme de dérive faisant intervenir la dérivée seconde de  $f$ , à savoir  $\frac{1}{2} D^2 f(X_t) \cdot (\sigma(X_t), \sigma(X_t)) dt$  (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 apparaît : la formule d'Itô dit en substance que ce terme est égal à  $\mathbf{I}_d dt$ .

### 3.3.4 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  $\sigma \sigma^T$  et de vitesse de dérive  $b$ .

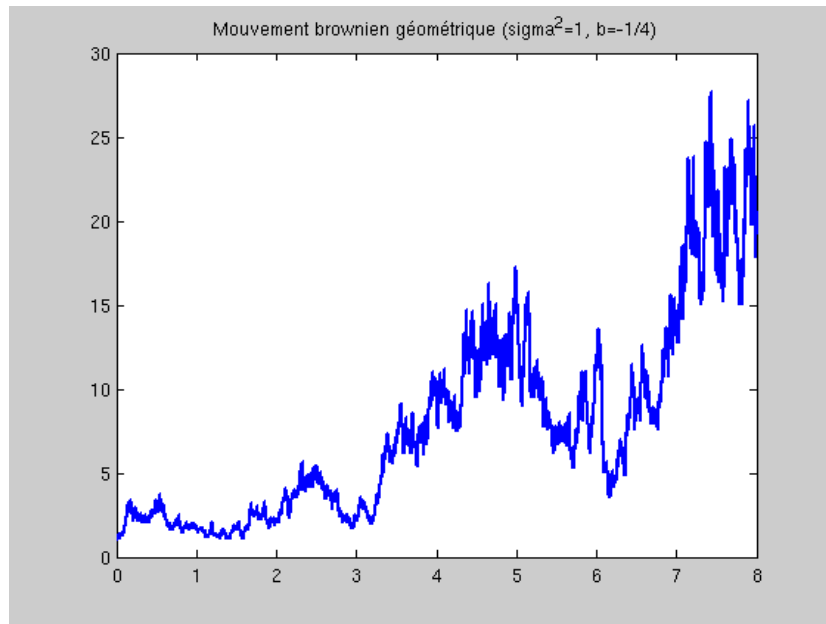


FIGURE 3.3 – Mouvement brownien géométrique. Remarquez qu’on voit bien comme le bruit est proportionnel à la valeur du processus.

2. On appelle *mouvement brownien géométrique* de paramètres  $\sigma^2$  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  $(B_t)_{t \geq 0}$  est un mouvement brownien unidimensionnel avec dérive de paramètres  $\sigma^2$  et  $\mu$ , alors  $(e^{B_t})_{t \geq 0}$  est un mouvement brownien géométrique de mêmes paramètres.

3. La solution de l’équation différentielle stochastique (en dimension 1)

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

est appelée *processus d’Ornstein-Uhlenbeck* de paramètres  $\sigma$  et  $\lambda$ . 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 », c.à.d. que pour tout  $t_1 \geq 0$ ,  $(X_{t_1+t})_{t \geq 0}$  a la même loi que  $(X_t)_{t \geq 0}$ . On dit que la loi  $\mathcal{N}(\sigma^2 / 2\lambda)$  est une *mesure d’équilibre* du processus.



## 3.4 Processus à sauts

### 3.4.1 Processus de Poisson simple

**Définition 3.34.** Le processus de Poisson d’intensité  $\lambda$  est un processus de  $\mathbb{R}_+$  dans  $\mathbb{R}$  qui augmente par incréments d’une unité. L’occurrence (ou pas) d’incrément à différents instants est totalement indépendante, et la probabilité qu’un incrément se produise



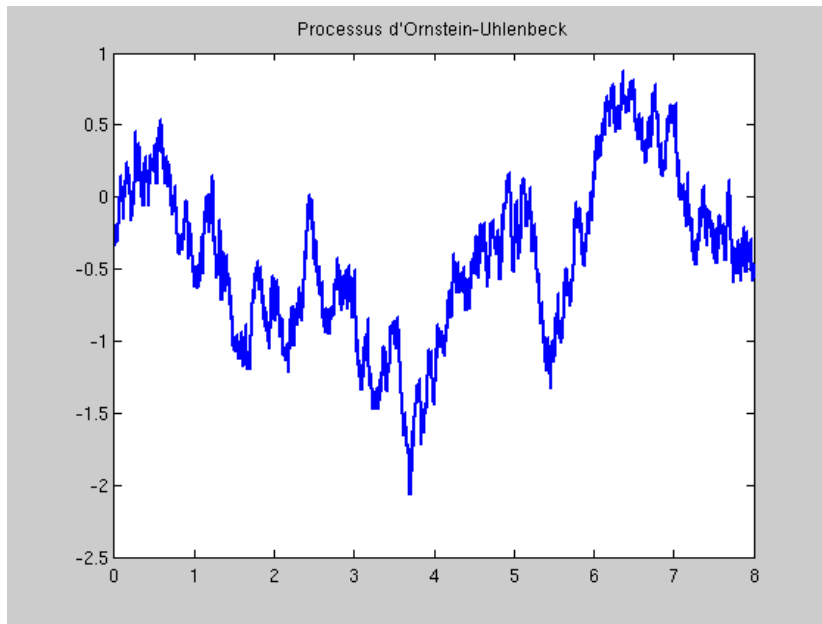


FIGURE 3.4 – Processus d’Ornstein-Uhlenbeck. On devine sur ce dessin qu’une “force” empêche le processus de trop s’éloigner de zéro...

à l’instant  $t$  vaut  $\lambda dt$ . En d’autres termes, les intervalles de temps entre deux incréments successifs sont indépendants et suivent chacun une loi *Exponentielle*( $\lambda$ ).

⚠ *Remarque 3.35.* 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  $\lambda$ ).

### 3.4.2 Processus de Poisson composés

ⓘ Si  $\lambda$  est une mesure de masse totale finie sur  $\mathbb{R}$ , le processus de Poisson composé d’intensité  $\lambda$  est un processus de  $\mathbb{R}_+$  dans  $\mathbb{R}$  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  $\Lambda$  la masse totale de  $\lambda$ , les intervalles de temps entre deux sauts successifs sont indépendants et suivent chacun une loi  $\Lambda$ , 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 *yahtzee* : yahtzee.m

```
%% Fonction principale
% La fonction "yahtzee" évalue par la méthode de Monte-Carlo la probabilité
% d'obtenir un yahtzee.
function yahtzee
% "N" note le nombre d'expériences choisi, ici 200000.
N = 200000;
% La variable "succes" comptera le nombre de fois qu'on a réussi à obtenir
% un yahtzee.
succes = 0;
% On lance une boucle de N essais, comptés par l'indice "i".
for i = 1:N
    % On simule un essai du joueur. Cela est effectué par la sous-routine
    % "troislancers" (voir plus loin), qui renvoie 1 si le joueur a réussi
    % et 0 s'il a échoué. Le résultat est stocké dans "r" .
    r = troislancers;
    % On comptabilise le succès éventuel du joueur.
    succes = succes + r;
end
% On calcule l'estimateur "p" de la probabilité de succès.
p = succes / N;
% On affiche le résultat.
disp(['Estimateur de la probabilite de succès : ',num2str(p)]);
% Le programme s'arrête.
return
end

%% Sous-routine "troislancers"
% La fonction "troislancers" simule le comportement du joueur effectuant
% ses trois lancers en cherchant à obtenir un yahtzee. Elle renvoie 1 si
% l'essai se solde par un succès et 0 si c'est un échec.
function resultat = troislancers
% On simule le premier lancer. Le 5-uplet "des" représente les chiffres
```

```

% affichés par les cinq dés.
des = randi(6,[1,5]);
% On relance les dés une première fois. Cette étape est simulée par la
% sous-routine "relancer" (voir plus loin), qui renvoie le nouvel affichage
% des dés après ce lancer.
des = relancer(des);
% On relance les dés une seconde fois.
des = relancer(des);
% On teste si on a obtenu un yahtzee ou pas ; la réponse est stockée dans
% "resultat".
resultat = all(des == des(1));
% La fonction retourne "resultat".
return
end

%% Sous-routine "relancer"
% La fonction "relancer" simule le comportement du joueur relançant les dés
% et renvoie le nouvel affichage des dés. Elle prend en paramètre la liste
% des valeurs affichées.
function desrelances = relancer(des)
% On va regarder combien de fois chaque chiffre apparait. Les résultats
% seront stockés dans le 6-uplet "occurrences".
occurrences = zeros(1,6);
% On lance une boucle pour remplir la série "occurrences". La variable "i"
% indexe le dé qu'on est en train de regarder.
for i = 1:5
    % On regarde la chiffre affiché par le i-ième dé et on incrémente
    % l'entrée correspondante de "occurrences".
    occurrences(des(i)) = occurrences(des(i)) + 1;
end
% On regarde le chiffre qui apparaît le plus souvent sur le plateau, qu'on
% appelle "recordman".
[~,recordman] = max(occurrences);
% On va maintenant donner la série des dés relancés, représentés par le
% 5-uplet "desrelances". On part de la situation des dés avant le relancer.
desrelances = des;
% On passe les dés en revue pour savoir s'ils sont à relancer ou pas. La
% variable "i" indexe le dé qu'on est en train de regarder.
for i = 1:5
    % On relance le dé seulement si le chiffre qu'il affichait n'était pas
    % le recordman.
    if des(i) ~= recordman
        % On simule le relancer du dé.
        desrelances(i) = randi(6);
    end
end
end
% La fonction retourne "desrelances".
return

```



end

## A.2 *Qui veut gagner du pognon ?* : qvgdp.m

```
%% Fonction principale
% La fonction "qvgdp" évalue par la méthode de Monte-Carlo le gain moyen
% d'un participant du jeu « Qui veut gagner du pognon ? ».
function qvgdp
% "N" note le nombre de simulations effectuées, ici 5000.
N = 5000;
% "S" stockera la somme des résultats obtenus en vue d'en calculer la
% moyenne.
S = 0;
% On lance la boucle de la méthode de Monte-Carlo, indexée par "i".
for i=1:N
    % On simule les gains du joueur. Cela est assuré par la sous-routine
    % "pognon" (voir plus loin), qui renvoie le montant gagné.
    gains = pognon;
    % On ajoute les gains de la simulation à la somme "S".
    S = S + gains;
end
% On calcule l'estimateur "m" de l'espérance de gain.
m = S / N;
% On affiche le résultat.
disp(['Estimateur de l'espérance de gain : ',num2str(m),' €']);
% Le programme s'arrête.
return
end

%% Sous-routine "pognon"
% Cette fonction simule les gains d'un joueur à « Qui veut gagner du pognon
% ? ». Elle renvoie le montant de ces gains (en euros).
function gains = pognon
% "J" est le nombre de jokers restant au joueur.
J = 2;
% "gains" est le montant des gains obtenus par le joueur. Comme les
% paramètres du jeu assurent que le candidat répond toujours correctement à
% la première question (sans dépenser de joker), on initialise directement
% "gains" à 1000.
gains = 1000;
% On passe donc directement à la deuxième question.
k = 2;
% Cette boucle représente le joueur tentant de progresser dans l'échelle
% des questions, jusqu'à son élimination, qui correspond dans ce code à ce
% que "J" devienne négatif.
while J >= 0,
    % "jesais" vaut 1 si la candidat connaît la réponse, et 0 s'il
    % l'ignore.
```

```

jesais = (rand < 3 / (k + 2));
% On regarde si le candidat connaît ou non la réponse.
if jesais
    % Si la candidat connaît la réponse, on double ses gains et on
    % passe à la question suivante.
    gains = 2 * gains;
    k = k + 1;
else
    % Si le candidat ignore la réponse, il est obligé de consommer un
    % joker et de prendre une autre question. S'il n'avait plus de
    % jokers à ce moment-là, "J" devient négatif et on sort de la
    % boucle.
    J = J - 1;
end
end
% La fonction retourne "gains".
return
end

```

### A.3 Exemple de calcul d'intégrale par la méthode de Monte-Carlo : exdintegrale.m

```

% "exdintegrale" évalue  $\int_0^{\infty} (1+x^3)^{-1} dx$  par la méthode de
% Monte-Carlo en échantillonnant selon x la loi P de  $(U^{-1}-1)$  pour U
% uniforme sur (0,1), laquelle loi est telle que  $(dx/dP)(x) = (1+x)^2$ .
function exdintegrale
% "N" est le nombre de simulations, ici 80 000 000.
N = 80000000;
% "S" sera la somme des résultats obtenus.
S = 0;
for i = 1:N
    % On simule "x" selon la loi P.
    x = 1 / rand - 1;
    % On calcule  $(1 + x^3)^{-1} \times dx/dP$ , dont l'espérance sous P sera
    % l'intégrale recherchée, et on l'ajoute à "S".
    S = S + (1 + x)^2 / (1 + x^3);
end
% On calcule l'estimateur de Monte-Carlo "m" de l'intégrale.
m = S / N;
% On affiche le résultat.
disp(['Estimateur de l''intégrale : ', num2str(m, '%7f')]);
% Le programme s'arrête.
return
end

```

## A.4 Volume de la 4-boule unité : boule4D.m

```
% La fonction "boule4D" évalue la volume de la boule unité de dimension 4
% par la méthode de Monte-Carlo en échantillonnant sur le cube  $[-1,+1]^4$ .
function boule4D
% "N" note le nombre de simulation effectuées, ici 100000.
N = 100000;
% "S" stocke la somme des échantillonnages de l'indicatrice de la boule, en
% vue d'en calculer la moyenne.
S = 0;
for i = 1:N
    % On tire le quadruplet de nombres aléatoires ( $\omega_{i1}$ , ...,
    %  $\omega_{i4}$ ) de l'énoncé, appelé ici "omega".
    omega = rand(1,4);
    % On simule un point "x" uniforme sur le cube à partir de "omega".
    x = [-1,-1,-1,-1] + [2,2,2,2] .* omega;
    % On calcule l'indicatrice que "x" soit dans la boule, appelée "indic".
    indic = (sqrt(sum(x .* x)) <= 1);
    % On ajoute "indic" à la somme.
    S = S + indic;
end
% On calcule l'estimateur "I" (le produit des  $(b_k - a_k)$  est ici égal à 16).
I = 16 * S / N;
% On affiche le résultat.
disp(['Volume estimé de la boule : ', num2str(I)]);
% Le programme s'arrête.
return
end
```

## A.5 Moyenne des gains au jeu du *vingt-et-zéro* : vingtetzzero.

```
%% Fonction principale
% La fonction "vingtetzzero" évalue par la méthode de Monte-Carlo le gain
% moyen (pour une mise de 1) d'un joueur suivant la stratégie de base au
% jeu du vingt-et-zéro, avec un intervalle de confiance à 99 %.
function vingtetzzero
% N est le nombre de simulations effectuées, ici 8 000.
N = 8000;
% S sera la somme des gains obtenus par le joueur, en vue de calculer le
% gain moyen.
S = 0;
% S2 sera la somme des carrés des gains obtenus par le joueur, en vue de
% calculer l'écart-type empirique.
S2 = 0;
% On lance la boucle de simulation principale.
for i = 1:N
    % On simule une manche du joueur contre la banque. Cela est assuré par
    % la sous-routine "manche" (voir plus loin), qui renvoie le gain du
```

```

    % joueur.
    gain = manche;
    % On met à jour "S".
    S = S + gain;
    % On met à jour "S2".
    S2 = S2 + gain * gain;
end
% On calcule la moyenne empirique.
m = S / N;
% On calcule l'écart-type empirique.
sigma = sqrt(S2 / N - m * m);
% On calcule les bornes "mbas" et "mhaut" de l'intervalle de confiance.
mbas = m - 3.30 * sigma / sqrt(N);
mhaut = m + 3.30 * sigma / sqrt(N);
% On affiche le résultat.
disp(['Avec une certitude d'environ 99,9 %, le gain moyen pour le ',...
      'joueur est dans l'intervalle [',num2str(mbas),',',num2str(mhaut),...
      '].']);
% Le programme s'arrête.
return
end

%% Sous-routine "manche"
% La fonction "manche" simule une manche du joueur contre la banque et
% renvoie le gain "gain" du joueur.
function gain = manche
% On commence par tirer la première carte de la banque.
cartebanque = randi(10);
% En fonction de la carte affichée par la banque, le joueur choisit le
% seuil "stop" auquel il arrête de demander des cartes, selon la stratégie
% de base.
if cartebanque <= 3
    stop = 15;
elseif cartebanque <= 6
    stop = 14;
elseif cartebanque == 7
    stop = 15;
else
    stop = 16;
end
% Le joueur tire maintenant des cartes jusqu'à atteindre le seuil "stop".
% "totaljoueur" est le total des cartes du joueur.
totaljoueur = 0;
while(totaljoueur < stop)
    totaljoueur = totaljoueur + randi(10);
end
% Si le joueur a dépassé 20, il a perdu et le programme retourne un gain
% nul.

```

```

if totaljoueur > 20
    gain = 0;
    return
% C'est maintenant à la banque de tirer ses cartes.
else
    % "totalbanque" est le total des cartes de la banque ; rappelons que la
    % banque tire des cartes jusqu'à ce que ce total atteigne 17.
    totalbanque = cartebanque;
    while(totalbanque < 17)
        totalbanque = totalbanque + randi(10);
    end
% On regarde si la banque a dépassé le seuil de 20, auquel cas elle a perdu
% et le programme retourne un gain de 2.
    if totalbanque > 20
        gain = 2;
        return
    % On regarde maintenant si la banque a fait un meilleur score que le
    % joueur ou l'inverse ; le programme retournera 0 ou 2 selon le cas.
    elseif totalbanque > totaljoueur
        gain = 0;
        return
    elseif totaljoueur > totalbanque
        gain = 2;
        return
    % On traite pour finir les cas d'égalité : le programme retournera 1 ou
    % 2 selon que le total commun est respectivement 20 ou une autre
    % valeur.
    elseif totaljoueur == 20
        gain = 1;
        return
    else
        gain = 2;
        return
    end
end
end
end

```

## A.6 Utilisation d'une variable de contrôle

### A.6.1 Sans variable de contrôle : maxmarche\_spl.m

```

% La fonction "maxmarche_spl" 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 maxmarche_spl(N)
% "S" et "S2" contiendront respectivement la somme des résultats simulés et
% la somme de leurs carrés.

```

```

S = 0;
S2 = 0;
for i = 1:N
    % On simule la trajectoire, représentée par le 60-vecteur "X".
    % (Attention, "X" n'inclut pas le point X_0 = 0).
    X = cumsum(randn(1,60));
    % On regarde le maximum de la trajectoire (en incluant X_0), appelé
    % "Xmax".
    Xmax = max([0,X]);
    % On met à jour S et S2.
    S = S + Xmax;
    S2 = S2 + Xmax * Xmax;
end
% "m" est l'estimateur de Monte-Carlo.
m = S / N;
% "sigma" est la variance (estimée) de X_*.
sigma = sqrt(S2 / N - m * m);
% "erreur" est le rayon de l'intervalle de confiance à 2 sigmas.
erreur = 2 * sigma / sqrt(N);
% On affiche le résultat.
disp(['E(X_*) : ',num2str(m),' ± ',num2str(erreur)]);
% Le programme s'arrête.
return
end

```

### A.6.2 Utilisation « rudimentaire » d'une variable de contrôle : maxmarche\_ctrl0.m

```

% "maxmarche_ctrl0" effectue le même travail que "maxmarche_spl", mais en
% utilisant la variable de contrôle Y. Je ne commente que les différences
% avec "maxmarche_spl".
function maxmarche_ctrl0(N)
% Attention : cette fois-ci les variables "S" et "S2" ne se référeront pas
% à la simulation de "Xmax", mais à celle de Xmax-Y, où "Y" est la variable
% de contrôle.
S = 0;
S2 = 0;
for i = 1:N
    X = cumsum(randn(1,60));
    Xmax = max([0,X]);
    % On calcule la variable de contrôle.
    Y = max(0,X(60));
    % La quantité à laquelle on va appliquer la méthode de Monte-Carlo à
    % proprement parler est la différence Xmax-Y, notée "Z".
    Z = Xmax - Y;
    % On met à jour "S" et "S2".
    S = S + Z;
    S2 = S2 + Z * Z;
end

```

```

end
% On évalue l'estimeur et la marge d'erreur pour E(Z).
mZ = S / N;
sigma = sqrt(S2 / N - mZ * mZ);
erreur = 2 * sigma / sqrt(N);
% On en déduit l'estimateur pour E(Xmax) = E(Z) + 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(60 / (2 * pi));
m = mZ + EY;
disp(['E(X_*) : ', num2str(m), ' ± ', num2str(erreur)]);
return
end

```

## A.7 Effet du couplage entre simulations : le match de rugby

### A.7.1 Sans couplage (rugby\_spl.m)

```

% "rugby_spl" calcule la probabilité de victoire de l'équipe B lorsque
% celle-ci suit la stratégie S_q. La fonction prend en entrée la valeur "q"
% du paramètre de la stratégie.
function rugby_spl(q)
% "N" est le nombre de simulations, fixé à 15 000.
N = 15000;
% "n" contiendra le nombre de simulations où l'équipe B a gagné.
n = 0;
% Ici commence la boucle de simulation.
for i = 1:N
    % "Balavatange", "Bcherchelessai" et "Binscritlessai" sont trois
    % 12-vecteurs disant, pour chaque séquence de jeu, si l'équipe B prend
    % l'avantage lors de cette séquence, puis, *sous réserve qu'elle ait
    % pris l'avantage*, si elle cherche l'essai, puis *sous réserve qu'elle
    % ait pris l'avantage et chercher l'essai*, si elle marque l'essai.
    Balavatange = (rand(1,12) < .4);
    Bcherchelessai = (rand(1,12) < q);
    Binscritlessai = (rand(1,12) < .35);
    % On calcule le score "scoreB" de l'équipe B en fonction des trois
    % 12-vecteurs simulés ci-dessus.
    scoreB = sum(Balavatange .* (7 * Bcherchelessai .* Binscritlessai + ...
        3 * (1 - Bcherchelessai)));
    % "scoreA" est le score de l'équipe A. (Il suffit pour simuler ce score
    % de compter les séquences où A prend l'avantage, vu qu'elle choisit
    % systématiquement de prendre le but).
    scoreA = 3 * sum(rand(1,12) < .6);
    % On compare "scoreB" et "scoreA", et on ajoute 1 à "n" si l'équipe B a
    % gagné.

```

```

    n = n + (scoreB > scoreA);
end
% On calcule l'intervalle de confiance à 2 sigmas pour la probabilité de
% victoire : "p" est l'estimateur de la probabilité de victoire, "sigma"
% celui de l'écart-type de l'indicateur dont on évalue l'espérance, "pinf"
% la borne inférieure de l'intervalle de confiance et "psup" sa borne
% supérieure.
p = n / N;
sigma = sqrt(p - p * p);
pinf = p - 2 * sigma / sqrt(N);
psup = p + 2 * sigma / sqrt(N);
% On affiche le résultat.
disp(['Probabilité de victoire de B pour q = ', num2str(q), ' : entre ', ...
    num2str(pinf), ' et ', num2str(psup)]);
% Le programme s'arrête.
return
end

```

## A.7.2 En utilisant les simulations communes (rugby\_simcom.m)

```

% La fonction "rugby_simcom" évalue les /différences/ entre les
% probabilités de victoire de l'équipe B pour différentes valeurs du
% paramètre de stratégie q, en utilisant un couplage entre les simulations
% pour les différentes valeurs de q. La fonction prend en entrée la liste
% "qs" de paramètres q pour lesquels on souhaite évaluer cette différence.
% On ne commente que les parties qui diffèrent de "rugby_spl".
function rugby_simcom(qs)
% "k" est le nombre de valeurs de q qu'on souhaite comparer entre elles.
k = length(qs);
N = 15000;
% Ss sera le tableau des sommes des quantités simulées, et S2s le tableau
% des carrés de ces sommes.
Ss = zeros(k);
S2s = zeros(k);
% On lance la boucle de simulation. Comme on couple les différentes valeurs
% de q, il n'y a qu'une seule boucle pour toutes les simulations.
for i = 1:N
    % Cette fois-ci, il va falloir calculer les k scores qu'obtiendrait B
    % en suivant les k stratégies possibles. Le couplage consiste à prendre
    % des valeurs communes pour les 12-vecteurs "Balavantage" et
    % "Binscritlessai" (qui deviennent ainsi des k×12-matrices
    % "Balavantages" et "Binscritlessais"), et à coupler les valeurs des
    % 12-vecteurs "Bcherchelessai" (qui devient ainsi une k×12-matrice
    % "Bcherchelessais") en les obtenant à partir d'un même 12-vecteur de
    % variables uniformes sur [0,1].
    Balavantages = ones(k,1) * (rand(1,12) < .4);
    % Observez bien comment est obtenu "Bcherchelessais" : à gauche du
    % signe '<', on a dupliqué k fois le même vecteur de variables

```



```

% uniformes sur [0,1] ; et à droite, la valeur à laquelle on le compare
% dépend de la ligne.
Bcherchelessais = (ones(k,1) * rand(1,12) < qs' * ones(1,12)) ;
Binscritlessais = ones(k,1) * (rand(1,12) < .35);
% "scoreB" est devenu un k-vecteur (colonne) "scoreBs" contenant le
% score pour chacune des valeurs de q.
scoreBs = sum(Balavantages .* (7 * Bcherchelessais .*...
    Binscritlessais + 3 * (1 - Bcherchelessais)),2);
% Par contre, il n'y a toujours qu'un seul score pour A.
scoreA = 3 * sum(rand(1,12) < .6);
% "success" est la version k-vectorielle (en colonne) de "succes".
success = (scoreBs > scoreA);
% Ce qui nous intéresse ici n'est pas d'estimer les probabilités de
% succès, mais leurs /différences/. "differences" est donc la tableau
% k×k des différences entre les différentes valeurs "succes" que
% contient "success".
differences = success * ones(1,k) - ones(k,1) * success';
% On met à jour "Ss" et "S2s", parallèlement pour chaque entrée des
% tableaux.
Ss = Ss + differences;
S2s = S2s + differences .* differences;
end
% On calcule les estimateurs des différences de probabilités, les
% écarts-type associés, et les marges d'erreurs à 2 sigmas, tout cela
% parallèlement pour chaque couple de valeurs de q.
ms = Ss / N;
sigmas = S2s / N - ms .* ms;
erreurs = 2 * sigmas / sqrt(N);
% On affiche le résultat.
disp('Tableau des différences de probabilités: ');
disp(ms);
disp('±');
disp(erreurs);
return
end

```

## A.8 Effet du conditionnement : le *yahtzee*

### A.8.1 Sans conditionnement (yahtzee\_spl.m)

```

%% Fonction principale
% La fonction "yahtzee_spl" est une modification de la fonction "yahtzee"
% de sorte que celle-ci renvoie aussi l'intervalle de confiance à 2 sigmas.
% En outre, la version modifiée demande en argument le nombre "N" de
% simulations souhaitées. Je ne commente que les points qui diffèrent par
% rapport à "yahtzee".
function yahtzee_spl(N)
succes = 0;

```

```

for i = 1:N,
    r = troislancers;
    succes = succes + r;
end
p = succes / N;
% On évalue l'écart-type "sigma". Notez que comme ici la fonction qu'on
% simule est une indicatrice, l'écart-type s'exprime directement en
% fonction de la probabilité.
sigma = sqrt(p * (1 - p));
% "erreur" est la marge d'erreur à 2 sigmas.
erreur = 2 * sigma / sqrt(N);
% On affiche le résultat avec la marge d'erreur.
disp(['Probabilite de succès : ',num2str(p),' ± ',num2str(erreur)]);
return
end

```

```

%% Sous-routine "troislancers"
function resultat = troislancers
des = randi(6,[1,5]);
des = relancer(des);
des = relancer(des);
resultat = all(des == des(1));
return
end

```

```

%% Sous-routine "relancer"
function desrelances = relancer(des)
occurrences = zeros(1,6);
for i = 1:5,
    occurrences(des(i)) = occurrences(des(i)) + 1;
end
[~,recordman] = max(occurrences);
desrelances = des;
for i = 1:5,
    if des(i) ~= recordman
        desrelances(i) = randi([1,6]);
    end
end
return
end

```

### A.8.2 Avec conditionnement (yahtzee\_cond.m)

```

%% Fonction principale
% La fonction "yahtzee_cond" améliore la fonction "yahtzee_spl" en
% utilisant la technique de conditionnement par rapport à la tribu B
% engendrée par les résultats des deux premiers lancers. L'interface avec
% l'utilisateur est la même que pour "yahtzee_spl". Je ne commente que les

```

```

% points qui diffèrent par rapport à la version naïve.
function yahtzee_cond(N)
% "S" et "S2" contiendront respectivement la somme et la somme des carrés
% des variables simulées. Notez que cette fois-ci on est bien obligé de
% comptabiliser séparément la somme et la somme des carrés, vu qu'il ne
% s'agit plus de simplement compter les occurrences d'un événement.
S = 0;
S2 = 0;
for i = 1:N,
    % La variable aléatoire dont on va évaluer l'espérance ici est la
    % probabilité conditionnelle d'obtenir un yahtzee à l'issue des deux
    % premiers lancers. Celle-ci, appelée "r", est simulée par la
    % sous-routine "deuxlancers", programmée plus bas.
    r = deuxlancers;
    % On met à jour "S" et "S2"
    S = S + r;
    S2 = S2 + r * r;
end
% On calcule l'espérance "p" et l'écart-type empirique "sigma" de la
% variable aléatoire r.
p = S / N;
sigma = sqrt(S2 / N - p * p);
erreur = 2 * sigma / sqrt(N);
disp(['Probabilite de succès : ',num2str(p),' ± ',num2str(erreur)]);
return
end

%% Sous-routine "deuxlancers"
% La fonction "deuxlancers" simule les deux premiers lancers d'un joueur
% cherchant à obtenir un yahtzee, et renvoie la probabilité conditionnelle
% de succès à l'issue de ces deux lancers.
function resultat = deuxlancers
des = randi(6,[1,5]);
des = relancer(des);
% On va regarder combien de fois chaque chiffre apparait. Les résultats
% seront stockés dans le 6-uplet "occurrences".
occurrences = zeros(1,6);
% On lance une boucle pour remplir la série "occurrences". La variable "i"
% indexe le dé qu'on est en train de regarder.
for i = 1:5,
    % On regarde la chiffre affiché par le i-ième dé et on incrémente
    % l'entrée correspondante de "occurrences".
    occurrences(des(i)) = occurrences(des(i)) + 1;
end
% Ce qui nous intéresse est de savoir quel est le nombre maximal
% d'occurrences d'un chiffre. Nous appelons "k" ce nombre.
k = max(occurrences);
% La probabilité conditionnelle à retourner est  $1/6^{5-k}$ .

```

```

resultat = 6^(k-5);
return
end

%% Sous-routine "relancer"
% Cette sous-routine est inchangée par rapport à "yatzee_spl".
function desrelances = relancer(des)
occurrences = zeros(1,6);
for i = 1:5,
    occurrences(des(i)) = occurrences(des(i)) + 1;
end
[~,recordman] = max(occurrences);
desrelances = des;
for i = 1:5,
    if des(i) ~= recordman
        desrelances(i) = randi([1,6]);
    end
end
return
end

```

## A.9 Effet de l'échantillonnage préférentiel : risque d'accident pour la centrale nucléaire

### A.9.1 Sans échantillonnage préférentiel (risque\_spl.m)

```

% La fonction "risque_spl" évalue la probabilité quotidienne que survienne
% un accident, par une méthode de Monte-Carlo 'naïve'. "N" est le nombre de
% simulations demandées.
function risque_spl(N)
% "accidents" comptera le nombre d'accidents sur les N simulations.
accidents = 0;
% "C" est la matrice de covariance des facteurs de risque (X,Y,Z).
C = [1,.2,.3;.2,1,.4;.3,.4,1];
% "R" est une matrice telle que R'R=C, dont nous aurons à nous servir pour
% appliquer la méthode de Cholesky à chaque simulation.
R = chol(C);
% On lance la boucle de simulation.
for i = 1:N
    % Le 3-vecteur XYZ est une simulation du triplet (X,Y,Z), effectuée par
    % la méthode de Cholesky.
    XYZ = [-1,-2,-3] + randn(1,3) * R;
    % "resultat" vaut 1 si les trois composantes de XYZ sont positives et
    % qu'il y a donc accident, et 0 sinon.
    resultat = all(XYZ > 0);
    % On met à jour le compteur "accidents".
    accidents = accidents + resultat;
end

```

```

end
% "p" est la probabilité empirique d'accidents.
p = accidents / N;
% "sigma" est la variance empirique de l'indicatrice d'un accident, qui
% s'exprime directement en fonction de p.
sigma = sqrt(p - p * p);
% "erreur" est la marge d'erreur à 2 sigmas.
erreur = 2 * sigma / sqrt(N);
% On affiche le résultat.
disp(['Probabilité d''accident : ',num2str(p),' ± ',num2str(erreur)]);
% Le programme s'arrête.

```

## A.9.2 Avec échantillonnage préférentiel (risque\_pref.m)

```

% La fonction "risque_pref" améliore "risque_spl" en échantillonnant selon
% la loi Q. L'interface avec l'utilisateur est la même que pour
% "risque_spl" ; je ne commente que les différences par rapport au
% programme naïf.
function risque_pref(N)
% Cette fois-ci on ne se contente plus de compter les occurrences d'une
% indicatrice ; on a donc besoin de comptabiliser séparément la somme "S"
% et la somme des carrés "S2" de la quantité simulée.
S = 0;
S2 = 0;
C = [1,.2,.3;.2,1,.4;.3,.4,1];
R = chol(C);
% "M" est le vecteur-ligne qui intervient dans l'expression de la loi P;
% "invC" est l'inverse de la matrice "C". Ces deux quantités seront utiles
% par la suite.
M = [-3,-4,-5];
invC = eye(3) / C;
% Le scalaire "facteur" et le vecteur-colonne "coefficients" interviendront
% dans le calcul de la densité de P par rapport à Q, ce qui fait qu'il vaut
% mieux les calculer une fois pour toutes.
facteur = exp(-M * invC * M' / 2);
coefficients = invC * M';
for i = 1:N
    % Le 3-vecteur XYZ est une simulation du triplet (X,Y,Z) /sous la loi
    % Q/.
    XYZ = randn(1,3) * R;
    % "densite" est la densité de P par rapport à Q au point XYZ, calculée
    % d'après notre formule théorique.
    densite = facteur * exp(XYZ * coefficients);
    % "resultat_brut" est l'indicatrice de l'occurrence d'un accident.
    resultat_brut = all(XYZ > 0);
    % Il faut ponderer "resultat_brut" pour tenir compte qu'on
    % échantillonne sous Q; on appelle "resultat" de résultat pondéré par
    % dP/dQ.

```

```

    resultat = resultat_brut * densite;
    % On met à jour "S" et "S2".
    S = S + resultat;
    S2 = S2 + resultat * resultat;
end
% Il ne reste plus qu'à calculer, de façon tout-à-fait ordinaire,
% l'estimateur et la marge d'erreur de l'espérance de "resultat" sous Q.
p = S / N;
sigma = sqrt(S2 / N - p * p);
erreur = 2 * sigma / sqrt(N);
disp(['Probabilité d''accident : ', num2str(p), ' ± ', num2str(erreur)]);
return
end

```

## A.10 Simulation d'équations différentielles stochastiques par la méthode d'Euler

### A.10.1 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.10.2 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  $N(\epsilon \cdot \text{sigmasigmaT})$ .
    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.10.3 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'incrémentat. 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.10.4 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'équilibre, que je sais être  $N(\sigma^2/2\lambda)$ .
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

```

