Quelques éléments d'histoire

Les origines de la méthode de Monte Carlo coincident avec les origines du calcul scientifique et sont très associées aux besoins de calcul scientifique de la physique statistique. Le personnage le plus cité comme le père de la méthode de Monte Carlo est Métropolis[1]. Et pourtant le contenu de son article fondateur ne porte pas sur la méthode de Monte Carlo elle-même mais sur une nouvelle technique d'échantillonnage. Cela fait sens de lui faire porter la paternité de Monte Carlo, même si la méthode était déjà en place avant sa proposition, parce que cette proposition a tout changé en terme de praticabilité des calculs de physique statistique de l'équilibre et c'est la dynamique résultante qui a donné à la méthode de Monte Carlo toute la résonnance qu'elle a eu par la suite.

Au cours des vingt premières années de la méthode, tout va très vite et on peut aller jusqu'à dire qu'à la fin des années 60 tout est dit ! Les promesses sur la parallélisation, la dimension infinie, la gestion de physiques complexes, de données complexes sont toutes formulées[2, 3, 4, 5, 6]. Nous les reprenons aujourd'hui, presque à l'identique.

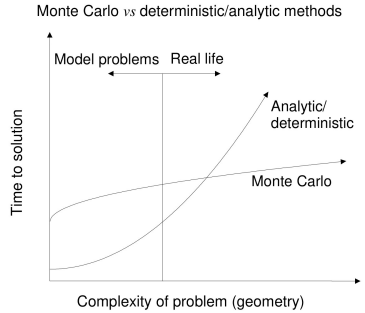

Figure 1: Time to solution using Monte Carlo *vs.* deterministic/analytic approaches.

FIGURE 1 – Premiere figure de [5]

Un champ d'application très spécifique mérite cependant une référence toute particulière : celui de la physique cinétique linéaire. La raison principale vient du fait que cette physique se dit initialement en des termes statistiques (distributions des libres parcours, probabilité du type de collision, etc). Il est alors possible de faire appel à la méthode de Monte Carlo sans jamais écrire le moindre développement mathématique supplémentaire : le code "reproduit" la physique. On parle d'*algorithmes analogues*. Mais bien sûr il est aussi possible de faire l'effort de traduire formellement les algorithmes en question. On parle alors de *formulation intégrale*. Ce double regard sur la même physique numérique a beaucoup contribué à stabiliser les pratiques[3, 7, 8, 9, 10, 11].

Au contraire, avec le travail de Feynmann et Kac, puis avec l'école russe qui s'est développé autour de Ermakov, c'est tout un savoir faire qui s'est développé concernant le souhait de partir d'une physique qui n'a initialement rien de statistique et la traduire en termes de processus stochastiques pour ensuite faire appel à la méthode de Monte Carlo dans l'esprit des physiciens de la cinétique linéaire. En tant que travail d'une dualité, ces recherches ont une portée qui dépasse très largement la méthode de Monte Carlo elle-même et on les retrouve dans des domaines applicatifs très divers, souvent très éloignés de la physique[12, 13, 14, 15].

Une étape majeure est ensuite l'arrivée de l'informatique graphique. Aujourd'hui il n'y a plus une seule image de synthèse produite par l'industrie du cinéma qui ne passe pas par la méthode de Monte Carlo, mais pour en arriver à ce stade il a fallu un investissement communautaire énorme pendant près d'une trentaine d'année. Il s'agissait avant tout de la promesse initiale de gérer les géométries complexes, les rapports d'échelles infinis ! De la promesse à la pratique effective il a fallu ajouter de nombreux concepts, à la fois géométriques et statistiques, dont nous héritons aujourd'hui pour toute l'ingénierie, totalement indépendamment de l'objectif de produire des images[16, 17, 18, 19].

Plus récemment encore, c'est un autre axe de recherche essentiel pour l'ingénierie qui a été ouvert simultanément dans plusieurs contextes applicatifs (physique

statistique, finance, physique énergétique, biologie, etc) : la gestion des non-linéarités. Elles sont principalement de trois types : i) lorsque un phénomène est régi par une équation différentielle ou intégrale non-linéaire ; ii) lorsque plusieurs phénomènes interviennent de façon couplée et leur couplage est non-linéaire ; iii) lorsqu'on cherche à comprendre, même pour une phénoménologie purement linéaire, comment une observable dépend des paramètres du problèmes (calcul de sensibilité) est que cette dépendance est non-linéaire[20, 21, 22, 23].

Ces deux dernières ouvertures (complexité de la donnée géométrique et non-linéarité) font que la méthode est à la fois très mature et continuellement secouée par un renouvellement des perspectives applicatives, dans le domaine académique aussi fortement que dans les différents contextes industrielles où elle est installée.

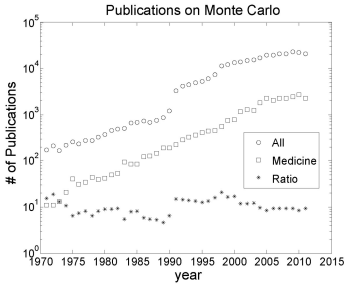

Figure 2: The number of papers published per year garnered from the Web of Knowledge ('All') and MedLine ('Medicine').

FIGURE 2 – Deuxiéme figure de [5]

RQ : Penser à rajouter essaim, kinetic MC, DSMC, etc.

L'estimation d'une espérance

$$\mathcal{A} = \mathbb{E}(W) \tag{1}$$

où

$$\mathbb{E}(W) = \int p_W(w) dw \; w \tag{2}$$

W est le *poids*. On utilise n réalisations du poids w_1, w_2 ... w_n pour estimer $\mathcal{A}$ comme

$$\mathcal{A} \approx m \tag{3}$$

avec

$$m = \frac{1}{n} \sum_{i=1}^{n} w_i \tag{4}$$

Commentaire 1 - L'estimateur

D'un point de vue probabiliste, on introduit n variables aléatoires W_1, W_2 ... W_n (les W sont majuscules) qui sont indépendantes et toutes distribuées comme le poids W (IID). On définit alors l'*estimateur* M comme

$$M = \frac{1}{n} \sum_{i=1}^{n} W_i \tag{1}$$

Dans l'équation 3, l'estimation m est alors vue comme *une unique réalisation* de l'estimateur M. Il est facile de prouver que $\mathbb{E}(M) = \mathbb{E}(W)$ et donc que

$$\mathcal{A} = \mathbb{E}(M) \tag{2}$$

m est une *estimation* de $\mathcal{A}$ et on associe à cette estimation une *incertitude de l'estimation* s :

$$s = \frac{1}{\sqrt{n-1}} \sqrt{\left(\frac{1}{n}\sum_{i=1}^{n} w_i^2\right) - m^2} \tag{5}$$

Un algorithme de type Monte Carlo a donc toujours la structure indiquée dans le pseudo-algorithme 1. L'enjeu se résume donc à la définition de la variable aléatoire W et la construction d'un algorithme permettant sa réalisation.

Algorithme 1 : Partie commune à tous les algorithmes de Monte Carlo

```
1 somme = 0;
2 sommeDesCarres = 0;
3 pour chaque i ∈< 1 : n > faire
4 |   Réalisation w de W;
5 |   somme = somme + w;
6 |   sommeDesCarres = sommeDesCarres + w^2;
7 fin
8 m = somme/n;
9 s = 1/sqrt(n-1) * sqrt(sommeDesCarres/n - m^2);
```

Du fait que l'espérance de l'estimateur est ainsi exactement égale à la grandeur recherchée $\mathcal{A}$, on dit que *l'estimateur est sans biais.*

De même, on définit la variable aléatoire S de la façon suivante (comme dans l'équation 5 mais avec des lettres majuscules) :

$$S = \frac{1}{\sqrt{n-1}} \sqrt{\left(\frac{1}{n}\sum_{i=1}^{n} W_i^2\right) - M^2} \tag{3}$$

et on montre que

$$\mathbb{E}(S^2) = \sigma^2(M) \tag{4}$$

et donc S^2 est *un estimateur sans biais de la variance de l'estimateur.* Dans l'equation 5 le carré de l'incertitude de l'estimation s est donc vu comme *une unique réalisation* de l'estimateur de la variance de l'estimateur. C'est donc s^2 qui s'interprète avec la confiance que l'on peut accorder à un estimateur sans biais et non pas s (qui n'est qu'un estimateur biaisé de l'écart-type de l'estimateur).

Formulation intégrale : un langage commun à la physique, les probabilités et l'algorithmique

Notre point de départ a été que lorsque l'on cherche à estimer une grandeur $\mathcal{A}$ par la méthode de Monte Carlo, il faut disposer d'une variable aléatoire W, définie sur un univers $\mathcal{U}_W$ et de densité de probabilité p_W, telle que $\mathcal{A}$ est égal à l'espérance de W, soit

$$\mathcal{A} = \int_{\mathcal{U}_W} p_W(w) dw \; w \tag{6}$$

Toute les finesses de la méthode seront dans la façon de définir W et de l'échantillonner pratiquement à l'aide d'un calculateur. On peut déjà retenir que W sera presque toujours défini à partir de variables aléatoires intermédiaires, elles-mêmes bien connues et faciles à échantillonner, par exemple deux variables aléatoires X et Y dans ce qui suit. Il y aura donc un algorithme d'échantillonnage de W qui passera par l'échantillonnage de X et l'échantillonnage de Y. Dans une pratique de type transfert thermique, X pourra typiquement être liée un chemin conductif au sein d'un solide convexe qui ne peut échanger avec l'extérieur que par rayonnement. X sera par exemple la position à laquelle le chemin conductif atteint la surface du solide et Y sera le chemin radiatif qui part de la surface du solide en X, l'espace des chemins radiatifs étant alors dépendant du point d'arrivée du chemin conductif, donc Y étant dépendant de X. Avec un tel exemple on voit tout de suite que les algorithmes dont nous allons parler peuvent être très élaborés, à la fois en termes phénoménologiques et en termes de rapport à la donnée (dans cet exemple essentiellement de rapport à la donnée géométrique).

Il sera donc essentiel de pouvoir garantir à toute étape la rigueur des choix probabilistes effectués, y compris lorsque le travail nous conduira au coeur de problématiques algorithmiques avancées, par exemple en termes d'accélération de l'accès à la donnée et de parallèlisme. Cette garantie de rigueur est heureusement obtenue assez simplement grace à la formulation intégrale.

Reprenons notre exemple avec deux variables aléatoires intermédiaires. On écrira typiquement

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x)dx \int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\ \tilde{w}(x,y) \tag{7}$$

On pose donc une égalité entre la grandeur étudiée $\mathcal{A}$ et une intégrale. Quelque soit l'origine de cette proposition, comme X et Y sont bien définies, alors $\mathcal{U}_X$, $\mathcal{U}_Y$, p_X et p_Y sont également bien définis et on peut réaliser le geste mathématique de vérification formelle de l'égalité proposée. Il s'agit alors d'un travail de formulation intégrale qui part de ce que l'on sait de $\mathcal{A}$ et qui est totalement indépendant du sens probabiliste qu'on accorde à l'équation 7. C'est la première utilité des écritures intégrales : *permettre la justification rigoureuse des choix de probabilisation.* Même lorsque ceux-ci ont été amenés par des raisonnements analogues, donc très intuitifs comme nous l'avons décrit en introduction à propos de la cinétique physique, si on fait l'effort de poser les écritures intégrales, on peut toujours vérifier a posteriori que la proposition faite est physiquement rigoureuse.

La seconde utilité d'une écriture intégrale est qu'elle remplace l'algorithme lui-même : *elle se lit comme un algorithme.* Avec un peu d'habitude on reconnait dans l'équation 7 le fait que $\mathcal{A}$ est l'espérance d'une variable aléatoire W. Et pourtant la convention selon laquelle une variable aléatoire est signifiée par une lettre majuscule n'est pas utilisée ici de façon directe pour cette variable aléatoire principale : la lettre W n'apparaît pas. Par contre les lettres X et Y apparaissent et signifient bien deux variables aléatoires, dont on repère les univers $\mathcal{U}_X$ et $\mathcal{U}_Y$ au niveau des signes d'intégration et qui définissent W comme $W = \tilde{w}(X,Y)$, c'est à dire comme une fonction des deux variables aléatoires X et Y.

Le signe $\tilde{w}(X,Y)$ n'apparaît pas car ce sont les lettres minuscules x et y qui sont utilisées dans la fonction $\tilde{w}$. Mais ces lettres minuscules sont porteuses de sens (bien qu'elles soient muettes du point de vue de la stricte grammaire mathématique). Ce sens vient de la présence des signes $p_X(x)dx$ et $p_Y(y;x)dy$ qui renvoient aux mesures probabilistes de X et Y.

Un cran plus loin, on lit que X et Y sont corrélés (p_Y fait appel à x) et que l'univers de Y dépend de X ($\mathcal{U}_Y$ est fonction de x). Puis l'intégrale sur $\mathcal{U}_Y$ signifie que la seconde partie de l'expression est l'espérance de $\tilde{w}(x,Y)$, soit

$$\int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\tilde{w}(x,y) = \mathbb{E}\left(\tilde{w}(x,Y)\right) \tag{8}$$

Ici seul Y est aléatoire car la lettre minuscule x n'est qu'une variable d'intégration. Mais dans l'expression complète de $\mathcal{A}$ l'intégration sur $\mathcal{U}_X$ signifie une seconde espérance et ont lit

$$\mathcal{A} = \mathbb{E}\left(\mathbb{E}\left(\tilde{w}(X,Y)\right)\right) \tag{9}$$

où X est bien une lettre majuscule et signifie cette fois la variable aléatoire. Enfin comme l'espérance d'une espérance est une espérance, on retient

$$\mathcal{A} = \mathbb{E}\left(\tilde{w}(X,Y)\right) \tag{10}$$

et donc

$$\mathcal{A} = \mathbb{E}\left(W\right) \tag{11}$$

Donc l'équation 7 signifie directement l'équation 11, c'est à dire renvoie à l'algorithme 1, avec

$$W = \tilde{w}(X,Y) \tag{12}$$

c'est à dire que l'échantillonnage de W passe par l'échantillonnage de X et l'échantillonnage de Y, puis l'application de la fonction $\tilde{w}$. On sait aussi que l'échantillonnage de X précède celui de Y car à la fois l'univers et la densité de probabilité de Y dépendent de X. On lit donc directement (après entraînement) l'algorithme 2

Algorithme 2 : Algorithme de Monte Carlo correspondant à l'équation 7

$somme = 0$;
$sommeDesCarres = 0$;
pour chaque $i \in < 1 : n >$ **faire**
    Réalisation x de X;
    Réalisation y de Y sachant $X = x$;
    Calcul de $w = \tilde{w}(x, y)$;
    $somme = somme + w$;
    $sommeDesCarres = sommeDesCarres + w^2$;
fin
$m = \frac{somme}{n}$;
$s = \frac{1}{\sqrt{n}} \sqrt{\frac{sommeDesCarres}{n} - m^2}$;

Echantillonnage par importance et variance nulle

Dans ce qui précède,

— nous avons affirmé que $\mathcal{A}$ existait, avec une définition découlant par exemple d'un modèle physique ;

— nous avons admis qu'il était aussi possible d'écrire $\mathcal{A}$ comme l'espérance d'une variable aléatoire W ;

— nous avons présenté la formulation intégrale comme un moyen de décrire l'algorithme de vérifier que l'écriture statistique est bien rigoureusement compatible avec la définition initiale de $\mathcal{A}$.

Considérons l'exemple de mise en oeuvre le plus simple : $\mathcal{A}$ est initialement défini comme l'intégrale d'une fonction f sur un domaine $\mathcal{D} \subseteq \mathbb{R}$.

$$\mathcal{A} = \int_{\mathcal{D}} f(x)dx \tag{13}$$

On peut alors choisir n'importe quelle variable aléatoire X sur $\mathcal{U}_X = \mathcal{D}$ de densité de probabilité p_X strictement positive et faire la transformation suivante :

$$\mathcal{A} = \int_{\mathcal{D}} f(x)dx = \int_{\mathcal{U}_X} p_X(x)dx \ \frac{f(x)}{p_X(x)} \tag{14}$$

On définit alors la fonction $\tilde{w}$ telle que

$$\tilde{w}(x) = \frac{f(x)}{p_X(x)} \tag{15}$$

pour retenir

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x)dx \ \tilde{w}(x) = \mathbb{E}\left(\tilde{w}(X)\right) \tag{16}$$

et en définissant la variable aléatoire W comme $W = \tilde{w}(X)$ on aboutit bien à

$$\mathcal{A} = \mathbb{E}\left(W\right) \tag{17}$$

L'idée de l'*échantillonnage par importance* part du constat que l'on peut jouer sur le choix de X, donc sur le choix de la densité de probabilité p_X utilisée lors de l'échantillonnage, afin que la variance de W soit la plus faible possible. Nous avons vu en effet que l'incertitude du calcul Monte Carlo est lié à la variance des poids : si toutes les réalisations du poids de Monte Carlo conduisaient exactement la même valeur, la variance de l'estimateur serait nulle ($s = 0$) et le calcul serait donc infiniment précis.

Or $W = \tilde{w}(X)$. On va donc chercher à réduire les variations de la fonction $\tilde{w}(x) = \frac{f(x)}{p_X(x)}$. Pour cela, dans la mesure du possible on va choisir p_X de sorte qu'elle reflète les variations de la fonction f : si p_X suit f exactement, alors $\tilde{w} = \frac{f}{p_X}$ est une constante et W est un dirac. Quelque soit la réalisation x de X, la réalisation correspondante $w = \tilde{w}(x)$ de W prend toujours la même valeur et W a bien une variance nulle.

On parle d'un *algorithme à variance nulle* lorsque cette limite idéale a pu être atteinte. Ce n'est bien sûr jamais le cas dans la pratique effective, mais dans les efforts d'optimisation de convergence nous sommes toujours guidés par cet objectif et nous investissons beaucoup d'effort dans la description détaillée de ces algorithmes limites, même s'ils ne sont pas praticables.

Prenons donc le temps de décrire un algorithme à variance nulle pour notre exemple, en indiçant d'une lettre I tous les choix d'échantillonnage idéaux correspondants. Pour tout x nous devons avoir $\frac{f(x)}{p_{X,I}(x)} = K$ où K est une constante, soit $p_{X,I}(x) = \frac{1}{K} f(x)$. De plus $p_{X,I}$ est une densité de probabilité donc elle est normalisée : $\int_{\mathcal{U}_X} p_{X,I}(x)dx = 1$. Cela conduit à

$$\int_{\mathcal{U}_X} \frac{1}{K} f(x)dx = \frac{1}{K} \int_{\mathcal{U}_X} f(x)dx = \frac{1}{K}\mathcal{A} = 1 \tag{18}$$

donc

$$K = \mathcal{A} \tag{19}$$

$$p_{X,I}(x) = \frac{f(x)}{\mathcal{A}} \tag{20}$$

et

$$W_I = \tilde{w}_I(X) = \frac{f(X)}{p_{X,I}(X)} = \mathcal{A} \tag{21}$$

La densité de probabilité d'échantillonnage idéale est donc la fonction f à intégrer divisée par la solution $\mathcal{A}$ de l'intégrale et le poids de Monte Carlo est toujours strictement égal à la solution recherchée. L'algorithme renvoie donc la solution exacte du problème dès la première réalisation.

On voit bien que l'on ne peut mettre en oeuvre concrètement cet algorithme que si l'on connaît déjà la solution. Dans une situation de calcul scientifique usuel, la solution n'étant pas connue, ce n'est pas praticable. Mais l'échantillonnage par importance pourra être guidé par cette limite au sens où on cherchera à utiliser une loi d'échantillonnage optimisée, $p_{X,opt}$, s'approche au mieux de $p_{X,I}$, par exemple en partant d'un algorithme non optimisé, en écrivant

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x)dx\ \tilde{w}(x) = \int_{\mathcal{U}_X} p_{X,opt}(x)dx\ \frac{p_X(x)}{p_{X,opt}(x)}\tilde{w}(x) = \int_{\mathcal{U}_X} p_{X,opt}(x)dx\ \tilde{w}_{opt}(x) \tag{22}$$

Commentaire 2 - Existe-t'il toujours un algorithme à variance nulle ?

Indépendamment de sa praticabilité, peut-on toujours se référer à un algorithme à variance nulle, comme un idéal de convergence, ou bien y-a-t'il des cas où aucun algorithme à variance nulle ne peut être envisagé ? En posant $p_{X,I}(x) = \frac{f(x)}{\mathcal{A}}$ nous avons supposé que les conditions étaient réunies pour que $p_{X,I}$ soit bien une densité de probabilité. L'intégrabilité et la normalisation sont assurées, mais $f(x)$ n'est pas nécessairement positif pour tout x.

Si f est partout négative, ce n'est pas un problème car on peut prendre $p_{X,I}(x) = -\frac{f(x)}{\mathcal{A}}$ et aboutir aux même conclusions. Mais si f change de signe cette astuce ne suffit pas. On doit alors faire un travail de reformulation intégrale avant de pouvoir envisager un algorithme à variance nulle. La réponse à notre question est donc négative : sans changer la forme de l'intégrale il est souvent impossible de définir un algorithme à variance nulle.

Mais la reformulation intégrale nécessaire peut être très simple. Admettons par exemple que f est parfois négative mais qu'elle est minorée par une valeur f_{min} connue et que le domaine d'intégration $\mathcal{D}$ est de mesure $\mu_{\mathcal{D}}$ finie. On peut alors écrire

$$\mathcal{A} = \int_{\mathcal{D}} f(x)dx = \int_{\mathcal{D}} (f(x) - f_{min})dx + \mu_{\mathcal{D}} f_{min} \tag{18}$$

Le problème se ramène alors à l'estimation d'une nouvelle intégrale $\mathcal{A}' = \int_{\mathcal{D}} f'(x)dx$ de la fonction $f' = f - f_{min}$ qui cette fois est toujours positive et on peut définir un algorithme à variance nulle pour $\mathcal{A}'$.

Commentaire 3 - Optimiser un algorithme à plusieurs variables

Reprenons l'exemple de l'équation 7 pour décliner la question de l'échantillonnage par importance dans le cas d'un échantillonnage multiple, ici échantillonage de X puis

avec

$$\tilde{w}_{opt}(x) = \frac{p_X(x)}{p_{X,opt}(x)}\tilde{w}(x) \tag{23}$$

Partant d'un algorithme de Monte Carlo donné, la pratique est donc que l'on cherche à améliorer sa convergence (réduire l'incertitude de l'estimateur) en ajustant la loi d'échantillonnage et en modifiant le poids de Monte Carlo de sorte que la solution reste exacte pour un nombre infini de réalisation (que l'estimateur reste non biaisé). Cette modification n'est rien d'autre que la multiplication de l'ancien poids par le rapport des densités de proabilité d'échantillonnage.

RQ : Penser à rajouter double randomisation.

échantillonnage de Y connaissant la réalasiation de X :

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x)dx \int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\tilde{w}(x,y) \tag{19}$$

La seule idée à retenir est que l'échantillonnage par importance se travaille pour chaque intégrale comme si elle était unique, donc comme nous l'avons décrit pour l'équation 13. Ici, pour la variable aléatoire X la fonction de x que nous devons tenter de suivre au mieux est $f(x) = p_X(x)\int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\tilde{w}(x,y)$. La densité de probabilité idéale est donc

$$p_{X,I}(x) = \frac{f(x)}{\int_{\mathcal{U}_X} f(x)dx} = \frac{1}{\mathcal{A}}p_X(x)\int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\ \tilde{w}(x,y) \tag{20}$$

Pour optimiser l'échantillonage de x il faut donc que pour toute valeur de x nous disposions d'une bonne approximation de l'intégrale de la seconde variable, soit de $\int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\tilde{w}(x,y)$.

Ensuite, pour optimiser l'échantillonage de y on est rammené au cas à une seule variable, mais il faut bien voir que nous devons être en mesure de nous approcher de

$$p_{Y,I}(y;x) = \frac{p_Y(y;x)\tilde{w}(x,y)}{\int_{\mathcal{U}_Y(x)} p_Y(y;x)dy\ \tilde{w}(x,y)} \tag{21}$$

pour toute valeur de x, ce qui peut parfois être très difficile à réaliser.

Reformulation intégrale

A partir d'exemples simples comme ceux de cette introduction, il est difficile de faire sentir ce qu'est l'essentiel de la pratique en termes d'amélioration de convergence. C'est la *reformulation intégrale*. L'échantillonnage par importance et les algorithmes à variance nulle ne viennent en effet que dans un second temps, seulement après avoir acquis la conviction que la forme intégrale que l'on a choisi de retenir est bien celle qui est la plus adaptée à un calcul de type Monte Carlo.

Il est difficile de décrire les critères qui fondent ce jugement ("la formulation choisie est bien la bonne") car ils sont le produit d'une culture pratique, différente dans chaque domaine applicatif et toujours très appuyée sur les images physiques disponibles. Nous donons ici un exemple issu de notre pratique, en le présentant d'abord sous un angle purement mathématique, puis en expliquant brièvement comment la stratégie retenue s'est imposée "naturellement" depuis la physique du rayonnement.

On cherche à estimer

$$\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2 \tag{24}$$

avec

$$\mathcal{A}_1 = \int_{-a}^{0} kf(x_1)e^{+kx_1}(1-e^{-kb})dx_1 \tag{25}$$

et

$$\mathcal{A}_2 = \int_{0}^{b} kf(x_2)e^{-kx_2}(1-e^{-ka})dx_2 \tag{26}$$

Initialement, $\mathcal{A}_1$ est $\mathcal{A}_2$ étaient vues comme des intégrales séparées que l'on estimait chacune séparément pour ensuite retenir la différence. Mais très vite on se rend compte que l'on rencontre des problèmes de convergence dès que la fonction f varie faiblement autour d'une valeur f_{ref} sur l'interval $[-a, b]$. On observe en effet que les deux intégrales prennent des valeurs proches de $f_{ref}(1-e^{-ka})(1-e^{-kb})$ et qu'il faut donc les estimer chacune très précisément avant de pouvoir prendre leur différence, sinon on ne voit que les incertitudes des deux Monte Carlo.

On peut cependant observer que

$$1-e^{-kb} = \int_{0}^{b} ke^{-kx_2}dx_2 \tag{27}$$

et

$$1-e^{-ka} = \int_{-a}^{0} ke^{+kx_1}dx_1 \tag{28}$$

ce qui invite à écrire $\mathcal{A}_1$ et $\mathcal{A}_2$ toutes les deux comme des intégrales sur le même double domaine :

$$\mathcal{A}_1 = \int_{-a}^{0} dx_1 \int_{0}^{b} dx_2 ke^{+kx_1}ke^{-kx_2}f(x_1) \tag{29}$$

et

$$\mathcal{A}_2 = \int_{-a}^{0} dx_1 \int_{0}^{b} dx_2 ke^{+kx_1}ke^{-kx_2}f(x_2) \tag{30}$$

On écrit alors $\mathcal{A}$ comme une seule intégrale :

$$\mathcal{A} = \int_{-a}^{0} dx_1 \int_{0}^{b} dx_2 ke^{+kx_1}ke^{-kx_2}(f(x_1)-f(x_2)) \tag{31}$$

Sur cette nouvelle base, on peut chercher à estimer $\mathcal{A}$ directement, à l'aide d'un seul algorithme de Monte Carlo, typiquement en raisonnant sur la base d'un échantillonnage par importance qui laisse de côté la différence $f(x_1)-f(x_2)$: on cherche à refléter seulement les variations de e^{+kx_1} et de e^{-kx_2} en fonction de x_1 et x_2, respectivement. Cela conduit à

$$p_{X_1}(x_1) = \frac{e^{+kx_1}}{\int_{-a}^{0} e^{+kx_1}dx_1} = \frac{ke^{+kx_1}}{1-e^{-ka}} \tag{32}$$

et

$$p_{X_2}(x_2) = \frac{e^{-kx_2}}{\int_{0}^{b} e^{-kx_2}dx_2} = \frac{ke^{-kx_2}}{1-e^{-kb}} \tag{33}$$

En reportant ces deux densités de probabilité on obtient la formulation intégrale suivante dans laquelle on voit bien qu'on disparu les difficultés de convergence liées au fait que f puisse n'avoir que de faibles variations autour d'une valeur de référence f_{ref}. En effet l'algorithme échantillonne directement la différence $f(x_1) - f(x_2)$ et si celle-ci tend vers zéro, la solution tend vers zéro mais elle est bien estimée car elle n'est plus construite comme la différence de deux estimateurs indépendants.

$$\mathcal{A} = \int_{-a}^{0} p_{X_1}(x_1)dx_1 \int_{0}^{b} p_{X_2}(x_2)dx_2 \ \frac{ke^{+kx_1}}{p_{X_1}(x_1)} \frac{ke^{-kx_2}}{p_{X_2}(x_2)} (f(x_1) - f(x_2)) = \int_{-a}^{0} p_{X_1}(x_1)dx_1 \int_{0}^{b} p_{X_2}(x_2)dx_2 \ (1 - e^{-ka})(1 - e^{-kb})(f(x_1) - f(x_2)) \tag{34}$$

soit

$$\mathcal{A} = \int_{-a}^{0} p_{X_1}(x_1)dx_1 \int_{0}^{b} p_{X_2}(x_2)dx_2 \ \tilde{w}(x_1, x_2) \tag{35}$$

avec

$$\tilde{w}(x_1, x_2) = (1 - e^{-ka})(1 - e^{-kb})(f(x_1) - f(x_2)) \tag{36}$$

et donc

$$W = (1 - e^{-ka})(1 - e^{-kb})(f(X_1) - f(X_2)) \tag{37}$$

Cet exemple est une version simplifiée d'un problème où l'on cherche à évaluée la puissance nette échangée par deux couches adjacentes d'épaisseurs a et b composées d'un même milieu semi-transparent de coeffcient d'absorption k. La fonction f traduit la luminance de Planck qui est fonction de la température en chaque point. $\mathcal{A}_1$ est la puissance du rayonnement émis par la couche d'épaisseur a et absorbé par la couche d'épaisseur b. Réciproquement, $\mathcal{A}_2$ est la puissance du rayonnement émis par la couche d'épaisseur b et absorbé par la couche d'épaisseur a. Dans la première stratégie on estime ces deux puissances séparément, or si les deux couches sont à la même température (i.e. $f(x)$ est le même pour tout x) alors le second principe de la Thermodynamique nous dit que ces deux puissances sont rigoureusement égales, donc que leur différence, la puissance nette échangée $\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2$, est nulle. C'est en partant de cette image physique que dans la pratique la reformulation proposée ci-dessus a été conçue.

Calcul de sensibilité

Supposons que nous disposons d'un algorithme de Monte Carlo estimant $\mathcal{A}(\alpha)$ de la façon suivante (voir la section sur la formulation intégrale) :

$$\mathcal{A}(\alpha) = \int_{\mathcal{U}_X} p_X(x;\alpha)dx \ \tilde{w}(x;\alpha) \tag{38}$$

Dans cet algorithme, le poids de Monte Carlo w et la densité de probabilité p_X de la variable aléatoire échantillonnée X dépendent tous les deux d'un paramètre α. La solution $\mathcal{A}$ dépend donc également de α. On peut alors se donner comme objectif d'estimer la sensibilité de $\mathcal{A}$ à α au sens linéaire, c'est à dire d'estimer la dérivée $\frac{\partial \mathcal{A}}{\partial \alpha}$.

Observant que le domaine d'intégration ne dépend pas de α (sinon la question est beaucoup plus délicate), on peut écrire

$$\frac{\partial \mathcal{A}}{\partial \alpha} = \int_{\mathcal{U}_X} dx \ \left(\frac{\partial p_X}{\partial \alpha} \tilde{w} + p_X \frac{\partial \tilde{w}}{\partial \alpha} \right) \tag{39}$$

Sous cette forme, on ne reconnais pas un algorithme de Monte Carlo. Mais si on décide de faire apparaître la même loi d'échantillonnage que celle de l'algorithme initial, on obtient

$$\frac{\partial \mathcal{A}}{\partial \alpha} = \int_{\mathcal{U}_X} p_X(x;\alpha)dx \ \left(\frac{1}{p_X} \frac{\partial p_X}{\partial \alpha} \tilde{w} + \frac{\partial \tilde{w}}{\partial \alpha} \right) \tag{40}$$

En comparant l'équation 39 et l'équation 40, on voit que les algorithmes sont identiques au sens où ils passent tous les deux par l'échantillonnage de la même variable aléatoire X, la seule différence étant le poids de Monte Carlo :

— pour l'estimation de $\mathcal{A}$ on utulise le poids w ;

— pour l'estimation de $\frac{\partial \mathcal{A}}{\partial \alpha}$ on utilise le poids $w_{sensib} = \frac{1}{p_X}\frac{\partial p_X}{\partial \alpha}\tilde{w} + \frac{\partial \tilde{w}}{\partial \alpha}$

Cette propriété est utilisée pour développer des algorithmes qui calculent simultanément la grandeur $\mathcal{A}$ recherchée et ses dérivées par rapport à l'ensemble des paramètres du problème. Comme souvent l'essentiel du temps de calcul est lié à l'échantillonnage des variables aléatoires, et pas du tout au calcul des poids, ces algorithmes enrichis peuvent avoir un coût de calcul quasi-identique à celui de l'algorithme d'origine et ils permettent de disposer non pas seulement d'une estimation de la valeur de $\mathcal{A}$ pour un jeu de paramètres donnés, mais aussi d'une information sur la façon avec laquelle $\mathcal{A}$ évolue lorsqu'on modifie les valeurs des paramètres autour du jeu choisi.

TP

Préparation

Se connecter

— user : etudiant

— passwd : "demander"

Lancer le serveur graphique

```
etudiant@machine:~ $  startx
```

Puis ouvrir un terminal avec la combinaison de la touche Alt et de la touche Entrée :

```
[Alt] [Entrée]
```

Créer un répertoire de travail à votre nom

```
etudiant@machine:~ $  mkdir votre_nom
etudiant@machine:~ $  cd votre_nom
```

Visiter les pages de manuel

```
etudiant@machine:~/votre_nom $  man edix_gestes_de_base
etudiant@machine:~/votre_nom $  man edix_premiers_pas
etudiant@machine:~/votre_nom $  man edix_interface_graphique
```

On quitte une page de manuel avec la touche q. La page de manuel *edix_fenetres* décrit l'utilisation du système de fenétrage (naviguer parmi les 9 bureaux possibles, ouvrir une console, passer d'une console à l'autre, etc). La page de manuel *edix_gestes_de_base* décrit les éléments de pratique dont vous aurez besoin pour les TPs (les commandes du shell, l'éditeur de texte, les outils de visualisation, etc).

TP1

Recopier le premier exemple de code

```
etudiant@machine:~/votre_nom $  cp -r /home/lafrier/livre_informatique_graphique/Codes/quatre_plus_un_parallelise \
      /home/etudiant/votre_nom/quatre_plus_un_parallelise
etudiant@machine:~/votre_nom $  cd quatre_plus_un_parallelise
```

Vous êtes maintenant dans le répertoire `/home/etudiant/votre_nom/quatre_plus_un_parallelise` et vous pouvez par exemple utilser la commande `ls` pour lister le contenu de ce répertoire :

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ls
```

Compiler le code et l'exécuter Le code que nous allons regarder propose d'estimer le résultat $\mathcal{A}$ de la somme $4+1$ en effectuant un test de Bernoulli de probabilité 0.5 (un "pile ou face") et en retenant soit $4/0.5 = 8$, soit $1/0.5 = 2$ et en faisant la moyenne d'un grand nombre n de telles réalisations. La moyenne m d'un grand nombre de 8 ou de 2 dont la sortie est équiprobable est effectivement proche de $\mathcal{A} = 4 + 1 = 5$, donc $m \approx \mathcal{A}$ si n est grand. Le code fourni également une estimation s de l'incertitude statistique. Cette incertitude est l'écart type de l'estimateur et nous savons que l'estimateur est gaussien. Avec une probabilité 0.9973, on peut donc s'attendre à ce que le résultat du calcul soit compatible avec la valeur exacte 5 à plus ou moins trois fois cette incertitude.

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  . ~lafrier/star-build/local/etc/stardis.profile
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  sh compilation.sh
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ./a.out
```

La première de ces trois commandes doit être relancée à chaque fois que vous ouvrez une nouvelle session du shell (e.g. une nouvelle console). Elle crée les variables d'environnement dont on a besoin pour utiliser les bibliothèques développées par la plateforme edstar. La seconde commande est le script shell `compilation.sh` qui compile le code (nommé `quatre_plus_un_parallelise.c`). Sans chercher à comprendre les détail de la ligne de compilation, vous pouvez afficher le contenu de ce script avec la commande `cat` :

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  cat compilation.sh
```

La troisième commande exécute le code compilé. On peut par exemple obtenir l'affichage suivant :

```
m=4.995200 +/- s=0.030001
```

Le symbole `+/-` est usuel mais est abusif. Comme annoncé ci-dessus, on ne peut parler qu'en termes de probabilités. Le raisonnement est le suivant : avec une probabilité 0.9973, on s'attend à ce que la solution exacte soit comprise entre $m - 3s$ et $m + 3s$, soit entre $4.979000 - 3 * 0.030001$ et $4.979000 + 3 * 0.030001$.

Ouvrir le code et le lire

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  vim quatre_plus_un_parallelise.c
vim:
  Se déplacer avec les flèches
  Sortir avec :q!
```

Voir la page de manuel *edix_ gestes_ de_ base* pour les premiers éléments d'utilisation de l'éditeur de texte *vim*. A ce stade, l'idée est simplement de prendre la mesure du nombre de lignes nécessaires pour écrire un tel code de Monte Carlo *dans les règles de l'art*, c'est à dire avec une gestion des exceptions et une assurance d'indépendance des générations aléatoires lorsqu'on fait appel à la parallélisation (exploitation de plusieurs processeurs pour le même calcul). Nous nous servirons de ce premier code comme point de départ pour les exemples qui suivent. On peut donc déjà reprérer quelques éléments de structure :

— la déclaration des variables globales ;

```
/* ================================================== */
/* Variables spécifiques communes à tous les processus */
/* Ces variables ne doivent pas être modifiées par les */
/* processus lancés en parallèle                       */
/* ================================================== */

double proba = 0.5; // probabilité du test de Bernoulli
```

— la définition de toutes les grandeurs que l'on peut passer en entrée lors de l'exécution du code ;

```
/************************************************************************/
/*  Gestion des entrées standard                                        */
/*  Entrée 1 : new_rng_serie (valeur 0 si non spécifiée)                */
/*  Entrée 2 : nrealisations (valeur 10000 si non spécifiée)            */
/*  Entrée 3 : nthreads (nombre de processeurs logiques de la machine si    */
/*             non spécifiée)                                           */
/************************************************************************/
et tous les tests qui suivent ...
```

— la définition des variables attachées à chaque processus (dans cet exemple il n'y en a aucune) ;

```
/* ============================================== */
/* Variables spécifiques propres à chaque processus */
/* Ces variables prendrons des valeurs différentes  */
/* pour chacun des processus lancés en parallèle    */
/* ============================================== */
```

— l'algorithme d'échantillonnage lui-même.

```
/* ==================================== */
/* Algorithme d'échantillonnage du poids */
/* ==================================== */

if(ssp_rng_canonical(rngs[ithread]) < proba) {
  w = 4./proba;
} else {
  w = 1./(1-proba);
}

/* ============================================ */
/* Fin de l'algorithme d'échantillonnage du poids */
/* ============================================ */
```

Pour tous les TPs envisagés, le reste du code est générique et nous ne le modifierons pas. Les parties du code où nous irons écrire pour coder de nouveaux algorithmes de Monte Carlo à partir de cet exemple ne sont donc pas nombreux : deux zones de déclaration de variables (générales ou associée au processus), une zone de définition des entrées et l'algorithme d'échantillonnage.

Exécuter le code avec des paramètres différents Si vous regardez la définition des entrées, vous verrez qu'il est possible de passer trois données d'entrée. La première est soit 0, soit 1. Si on met 0, à chaque nouvelle exécution du code c'est la même série de nombres aléatoires qui est utilisée. On obtient donc exactement le même résultat à chaque fois. Si au contraire on met 1, la série de nombres aléatoires est différente à chaque nouvelle exécution. Par exemple si vous lancez trois fois de suite la commande suivante

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ./a out 0
```

vous obtenez trois fois le même résultat. Si au contraire vous lancez trois fois de suite la commande suivante

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ./a out 1
```

vous obtenez trois résultats de simulation différents.

La seconde donnée d'entrée possible est le nombre de réalisations. Si vous lancez successivement les deux commandes suivantes,

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ./a out 0 10000
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ./a out 0 1000000
```

vous constaterez que l'incertitude statistique est dix fois plus faible dans le second cas que dans le premier. La raison est que nous avons utilisé cent fois plus de réalisations, or l'incertitude statistique évolue comme $\frac{1}{\sqrt{n}}$ où n est le nombre de réalisations. Avec cent fois plus de réaliasations on est donc dix fois plus précis.

La troisième donnée d'entrée est le nombre de processus lancés en parallèle. Par défaut on utilise autant de processus que de processeurs logiques. Si vous lancez

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  ./a out 0 10000 1
```

il n'y aura qu'un seul processus et vous ne ferez donc pas appel à la parallélisation.

Ouvrir le shell run.sh, le commenter et le lancer

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  vim run.sh
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  sh run.sh
```

Ce script lance cinq fois le code. On obtient donc cinq estimations différentes de $\mathcal{A}$, avec chacune le même nombre de réalisations.

Modifer le shell pour tracer un histogramme des estimations de $\mathcal{A}$ Au lieu de cinq estimations, on peut lancer un grand nombre d'estimations différentes et tracer la distribution statistique de ces estimations sous la forme d'un historgramme. Pour cela, on peut utiliser le logiciel de statistique R qui intègre des solutions de visualisation, ou bien faire les statistiques dans un programme en langage C et visualiser les résultats avec le logiciel de visualisation de données scientifiques *gnuplot*.

Le but d'une telle étude statistique est de vérifier (au moins à l'oeil) que la distribution des estimations m (quand on relance le code de Monte Carlo) est bien une gaussienne et que son écart type est bien celui prédit par s (l'estimation de l'incertitude). C'est cette propriété qui justifie le fait que dans la suite, plus jamais nous ne lancerons plusieurs fois de code de Monte Carlo avec les même paramètres : connaissant les valeurs de m et s obtenues lors d'un seul calcul, on a déjà toute l'information utile sur la distribution des estimations (celle que l'on obtiendrait en répétant la simulation). Répéter un calcul de Monte Carlo n'a pas plus de sens que de faire plusieurs sondages identiques en parallèle : si on peut accéder à plusieurs échantillons, alors on les regroupe pour travailler sur un échantillon de plus grande dimension, accédant alors à des conclusions statistiques plus précises. On ne relance pas un calcul de Monte Carlo pour voir comment le résultat fluctue car on sait comment il fluctuera. On travaille toujours avec *une seule estimation*. Par contre, quand c'est nécessaire on n'hésite à relancer un calcul de Monte Carlo avec plus de réalisations pour accéder à une estimation plus précise.

Reste que tracer une fois au moins la distribution des estimations sous la forme d'un histogramme, voir qu'elle est gaussienne et voir que s prédit sa largeur, est essentiel dans une phase de formation à la méthode. Ultérieurement dans la pratique on ne le fait plus jamais, mais l'image de cette distribution gaussienne est au coeur de tous les raisonnements. Vous pouvez essayer de tracer cet histogramme à votre façon, ou bien en vous inspirant de l'exemple proposé en solution.

Solution du TP1 Dans le répertoire

```
~/votre_nom/quatre_plus_un_parallelise/histogramme
```

Vous trouverez trois scripts :

- `run.sh` qui est l'équivalent du script `run.sh` que nous venons de regarder. La différence est que le nombre de tirages est plus élevé et que les résultats de calcul sont archivés dans le répertoire `/tmp/hist.dat`.
- `hist.R` qui est un script R qui trace un histogramme à partir du contenu du fichier `/tmp/hist.dat`. La figure contenant l'histogramme est `/tmp/hist.png`.
- `hist.sh` qui est un script shell qui lance le script R et visualise le fichier `/tmp/hist.png`.

Vous pouvez tester ces scripts avec les commandes suivantes :

```
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise $  cd histogramme
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise/histogramme $  sh run.sh
etudiant@machine:~/votre_nom/quatre_plus_un_parallelise/histogramme $  sh hist.sh
```

Vous n'obtiendrez pas encore l'histogramme que vous attendez car le nombre de tirage est $n = 2$. Les estimations possible sont donc $m = 2$ (si les deux tirages ont chacun conduit à retenir la valeur 2), $m = 8$ (si les deux tirages ont chacun conduit à retenir la valeur 8), où $m = 5$ (si un tirage a conduit à retenir la valeur 2, l'autre la valeur 8, ce qui est réalisé avec une probabilité double). Il n'y a donc que trois barres de l'histogrammes qui sont peuplées. Mais en modifiant le contenu du script `run.sh`, vous pourrez augmenter la valeur de n (e.g. en remplaçant `./a.out 1 2` par `./a.out 1 100`) et vous approcher de la gaussienne attendue. A vous ensuite de jouer sur les différents paramètre et comparer la largeur de la gaussienne à la valeur de s obtenue en lançant un seul calcul.

TP2

En repartant du code précédent, coder un nouvel algorithme de Monte Carlo qui estime l'intégrale suivante :

$$\mathcal{A}(p,q) = \int_0^1 dx \int_0^x dy\ x^p y^q \tag{41}$$

Pour x et pour y, on utilisera une loi d'échantillonnage uniforme.

$\mathcal{A}$ est connu analytiquement et vous pourrez donc vérifier que votre algorithme obtient bien, à l'incertitude statistique près,

$$\mathcal{A}(p,q) = \frac{1}{q+1}\frac{1}{p+q+2} \tag{42}$$

Remarque Dans un tel exemple, il est aussi possible de prévoir analytiquement la valeur de la variance de l'estimateur. On peut en effet calculer formellement l'espérance du carré de la variable aléatoire W et comme l'espérance de W est connue, puisqu'elle est égale à $\mathcal{A}$, on peut calculer l'espérance de W^2 moins le carré de $\mathcal{A}$ pour trouver

$$\sigma^2(W) = \frac{1}{2q+1}\frac{1}{2p+2q+3} - \frac{1}{(q+1)^2}\frac{1}{(p+q+2)^2} \tag{43}$$

Comme $\sigma^2(M) = \frac{1}{n}\sigma^2(W)$, vous pourrez ainsi prévoir les valeurs de l'incertitude statistique s produite à chaque exécution du code.

Solution du TP2 Un code correspondant au TP2 est proposé dans le répertoire `/home/lafrier/livre_informatique_graphique/Codes/somme_xp_yq`

TP3

Reprendre le TP2 avec un algorithme à variance nulle. Il suffit de donner l'algorithme. Il n'est pas utile de le coder car ce sera fait au TP5.

TP4

Estimer l'intégrale suivante :

$$\mathcal{A}(p,q,L,\alpha) = \int_0^1 dx \int_0^x dy\ x^p y^q \left(1 + \alpha \cos\left(2\pi \frac{y}{L}\right)\right) \tag{44}$$

Pour x et pour y, on utilisera une loi d'échantillonnage uniforme. Ce TP est très proche du TP2. Il est une étape pour les TP suivants. Pour $\alpha = 0$ vous devez notamment retrouver les résultats du TP2.

Solution du TP4 Un code correspondant au TP4 est proposé dans le répertoire `/home/lafrier/livre_informatique_graphique/Codes/somme_xp_yq_cosy`

TP5

Reprendre le TP4 en utilisant les lois d'échantillonnage du TP3. Pour $\alpha = 0$, vous serez exactement dans les conditions du TP3. L'algorithme doit donc être à variance nulle. Pour $\alpha \neq 0$ ce ne sera plus le cas. Vous pourrez tenter d'analyser la façon avec laquelle les estimations s de l'incertitude statistique dépendent de p, q et α. Quand la difficulté d'intégration est essentiellement liée à $x^p y^q$, les lois d'échantillonnage du TP3 sont bien adaptées. Quand au contraire c'est $\cos\left(2\pi \frac{y}{L}\right)$ qui est la source de la difficulté d'intégration, alors ces lois d'échantillonnage peuvent ne pas être adaptées et l'incertitude statistique sera plus élevée. Une des difficultés sera de décider si vous souhaitez raisonner en termes d'incertitude absolue, soit s, ou d'incertitude relative, soit $\frac{s}{m}$.

TP6

En repartant du TP4, concevoir et coder un algorithme qui estime la dérivée $\frac{\partial \mathcal{A}}{\partial \alpha}$.

Solution du TP6 Un code correspondant au TP6 est proposé dans le répertoire `/home/lafrier/livre_informatique_graphique/Codes/sensib_somme_xp_yq_cosy`

Références

[1] Nicholas Metropolis et al. « Equation of state calculations by fast computing machines ». In : *The journal of chemical physics* 21.6 (1953), p. 1087-1092.

[2] John Michael Hammersley, David Christopher Handscomb et Françoise Rostand. *Les méthodes de Monte-Carlo*. Dunod, 1967.

[3] Malvin H Kalos et Paula A Whitlock. *Monte carlo methods*. John Wiley & Sons, 2009.

[4] Michael Evans et Timothy Swartz. *Approximating integrals via Monte Carlo and deterministic methods*. T. 20. OUP Oxford, 2000.

[5] Alex F Bielajew. « History of monte carlo ». In : *Monte Carlo techniques in radiation therapy*. CRC Press, 2021, p. 3-15.

[6] Jun S Liu et Jun S Liu. *Monte Carlo strategies in scientific computing*. T. 10. Springer, 2001.

[7] Alireza Haghighat. *Monte Carlo methods for particle transport*. Crc Press, 2020.

[8] Oleg N Vassiliev. *Monte Carlo Methods for Radiation Transport Fundamentals and Advanced Topics*. Springer.

[9] William L Dunn et J Kenneth Shultis. *Exploring monte carlo methods*. Elsevier, 2022.

[10] Dirk P Kroese, Thomas Taimre et Zdravko I Botev. *Handbook of monte carlo methods*. John Wiley & Sons, 2013.

[11] Nick T Thomopoulos. *Essentials of Monte Carlo simulation : Statistical methods for building simulation models*. Springer Science & Business Media, 2012.

[12] Bernard Lapeyre, Étienne Pardoux et Rémi Sentis. *Méthodes de Monte-Carlo pour les équations de transport et de diffusion*. Springer, 1998.

[13] Ivan T Dimov. *Monte Carlo methods for applied scientists*. World Scientific, 2008.

[14] Sergei Mikhailovich Ermakov, Vladimir Viktorovich Nekrutkin et Aleksandr Stepanovich Sipin. *Random processes for classical equations of mathematical physics*. T. 34. Springer Science & Business Media, 2013.

[15] Mervin E Muller. « Some continuous Monte Carlo methods for the Dirichlet problem ». In : *The Annals of Mathematical Statistics* (1956), p. 569-589.

[16] Eric Veach. *Robust Monte Carlo methods for light transport simulation*. Stanford University, 1998.

[17] Henrik Wann Jensen et al. « Monte Carlo ray tracing ». In : *ACM SIGGRAPH*. T. 5. 2003, p. 340769537.

[18] Matt Pharr, Wenzel Jakob et Greg Humphreys. *Physically based rendering : From theory to implementation*. MIT Press, 2023.

[19] Wenzel Alban Jakob. *Light transport on path-space manifolds*. Cornell University, 2013.

[20] Ivan Dimov et Todor Gurov. « Monte Carlo algorithm for solving integral equations with polynomial non-linearity. Parallel implementation ». In : *Pliska Studia Mathematica Bulgarica* 13.1 (2000).

[21] Emmanuel Gobet. *Monte-Carlo methods and stochastic processes : from linear to non-linear*. Chapman et Hall/CRC, 2016.

[22] Jérémi Dauchet et al. « Addressing nonlinearities in monte carlo ». In : *Scientific reports* 8.1 (2018), p. 13302.

[23] Guillaume Terrée et al. « Addressing the gas kinetics Boltzmann equation with branching-path statistics ». In : *Physical Review E* 105.2 (2022), p. 025305.

Index