

Méthode de Monte Carlo

1 Principe	2
1.1 Calcul numérique d'une espérance	2
1.2 Calcul numérique d'une probabilité	3
2 Applications	3
2.1 Calcul d'intégrales	3
2.2 Calcul d'aires	4
2.3 Calcul de la fonction de répartition d'une variable aléatoire	6
2.4 Comparaison de différents estimateurs ponctuels	6
2.5 Calcul du niveau de confiance réel d'un inter- valle de confiance	7

Compétences attendues.

- ✓ Mettre en oeuvre la méthode de Monte-Carlo pour estimer une probabilité, la valeur d'une intégrale, ...

Liste des commandes Python exigibles aux concours.

- Dans la librairie `numpy` :
 - Fonctions et constantes usuelles : `np.cos`, `np.sqrt`, `np.abs`, `np.pi`
 - Création de vecteurs : `np.linspace`
 - Opération sur les matrices : `np.sum`, `np.min`, `np.max`, `np.mean`, `np.std`
- Dans la librairie `numpy.random` : `rd.random`, `rd.exponential`, `rd.normal`, `rd.binomial`, `rd.geometric`
- Dans la librairie `scipy.special` : `sp.ndtr`
- Dans la librairie `matplotlib.pyplot` : `plt.plot`, `plt.hist`, `plt.show`

Mathieu Mansuy

Professeur en ECG deuxième année spécialité mathématiques approfondies au Lycée Louis Pergaud (Besançon)

Page personnelle : mathieu-mansuy.fr/

E-mail : mansuy.mathieu@hotmail.fr

Introduction

On appelle communément *méthode de Monte-Carlo* toute méthode visant à estimer une quantité numérique, difficile à calculer explicitement, à l'aide de procédés probabilistes qui eux sont plus faciles à mettre en oeuvre (mais avec un risque, bien que faible, que la valeur retournée ne soit pas bonne). Les applications sont variées, comme par exemple :

- le calcul d'aires ou de volumes ;
- l'estimation du risque d'une décision financière ;
- en physique nucléaire...

Nous présentons ici le principe de cette méthode, et des exemples d'applications.

Le saviez-vous ?

Le problème du jeu de Go était jusque très récemment considéré comme l'un des grands défis de l'intelligence artificielle. Il fut résolu par le développeur Google DeepMind et son programme informatique AlphaGo. Ses premières versions utilisaient justement la méthode de Monte-Carlo afin de déterminer la stratégie qui a la probabilité la plus grande de gagner la partie. En octobre 2015, il devient le premier programme à battre un joueur professionnel (le français Fan Hui) sur un goban de taille normale (19 × 19) sans handicap. Le 27 mai 2017, AlphaGo bat le champion du monde Ke Jie. Pour en savoir plus sur le sujet, vous pouvez consulter cette [vidéo](#).

1 Principe

1.1 Calcul numérique d'une espérance

On dispose d'une quantité C difficile à calculer directement, mais que l'on sait écrire comme l'espérance d'une variable aléatoire X . La méthode de Monte-Carlo consiste à simuler un échantillon (X_1, \dots, X_n) i.i.d. de la loi de X . Les deux résultats suivants permettent alors d'obtenir une estimation de $E(X) = C$:

- La *loi faible des grands nombres* assure que $\bar{X}_n = \frac{1}{n} \sum_{k=1}^n X_k$ est un estimateur sans biais et convergent de C .
- Le *théorème limite central* assure que $[\bar{X}_n - t_\alpha \frac{\sigma}{\sqrt{n}}, \bar{X}_n + t_\alpha \frac{\sigma}{\sqrt{n}}]$ est un intervalle de confiance asymptotique de C au niveau de confiance $1 - \alpha$, en notant t_α le réel vérifiant $\Phi(t_\alpha) = 1 - \frac{\alpha}{2}$ et σ l'écart-type de X .

Si on ne connaît pas σ , on peut le remplacer par une majoration si elle existe de cet écart-type.

Méthode. Méthode de Monte Carlo pour l'estimation d'une espérance.

Pour obtenir une estimation numérique de C , on procèdera ainsi :

- on exprime C comme l'espérance d'une variable aléatoire X : $C = E(X)$;
- on réalise un échantillon i.i.d. de la loi de X de taille n avec n grand ($n = 1000$ par exemple) ;
- on renvoie la moyenne de l'échantillon obtenu (à l'aide de la commande `np.mean`).

1.2 Calcul numérique d'une probabilité

Une probabilité $p = P(A)$ peut être considérée comme l'espérance d'une variable de Bernoulli $X \mapsto \mathcal{B}(p)$, avec $X = \begin{cases} 1 & \text{si } A \text{ est réalisée} \\ 0 & \text{sinon} \end{cases}$. La méthode de Monte-Carlo s'applique donc également dans ce cas pour obtenir une estimation de la probabilité d'un événement.



Méthode. Méthode de Monte Carlo pour l'estimation d'une probabilité.

Pour obtenir une estimation numérique de p , on procèdera ainsi :

- on identifie l'épreuve de Bernoulli associée à la probabilité $p = P(A)$;
- on réalise n fois l'épreuve de Bernoulli de manière indépendante avec n grand ;
- on compte le nombre c de succès dans cette succession d'épreuves de Bernoulli indépendantes ;
- on renvoie la fréquence de réussite $\frac{c}{n}$ de l'épreuve de Bernoulli.

2 Applications

Dans toute la suite, on suppose avoir exécuté les commandes suivantes afin d'importer les librairies qui nous seront nécessaires :

```

1 | import numpy as np
2 | import numpy.random as rd
3 | import matplotlib.pyplot as plt
4 | import scipy.special as sp # pour la fonction de rép de la loi N(0,1)

```

2.1 Calcul d'intégrales

Exercice 1 (★★)

Soit (U_n) une suite de variables aléatoires indépendantes suivant toutes la loi uniforme sur $[0, 1]$, et soit $g : [0, 1] \rightarrow \mathbb{R}$ une fonction continue. Pour tout $n \geq 1$, on pose $S_n = \frac{1}{n} \sum_{i=1}^n g(U_i)$.

1. En utilisant la loi faible des grands nombres, montrer que :

$$S_n \xrightarrow{P} \int_0^1 g(x) dx.$$

À quelle propriété de cours ce résultat vous fait-il penser ?

2. On souhaite déterminer une valeur approchée de l'intégrale $I = \int_0^1 \cos(x^2) dx$.

Compléter le programme suivant afin qu'il retourne une valeur approchée de I .

```

1 | def g(x) :
2 |     return ...
3 |
4 | n = 1000 ; S = 0
5 | for k in range(n):
6 |     u = ...
7 |     S = S + ...
8 | print(...)

```

Essayer ce programme avec différentes valeurs de n , en faisant plusieurs essais pour chaque valeur de n . Comparer les valeurs obtenues avec celle donnée par le code suivant :

```

1 | from scipy.integrate import quad
2 | res, err = quad(g,0,1)
3 | print("Résultat de l'intégrale :", res)
    
```

Remarque. Dans l'exemple précédent, l'intérêt d'une méthode probabiliste n'est pas flagrant, puisque nous aurions également pu utiliser la méthode de rectangles (c'est-à-dire les sommes de Riemann) pour obtenir une valeur approchée de l'intégrale, sans risque d'erreur (il n'y a pas de hasard dans la méthode des rectangles). En revanche, cette méthode peut aisément être adaptée pour des intégrales impropres, pour lesquelles la méthode des rectangles fonctionne mal.

Exercice 2 (★)

Estimer la valeur des intégrales suivantes à l'aide de la méthode de Monte-Carlo.

$$\bullet I = \int_0^{+\infty} \frac{e^{-x}}{1+x^4} dx \qquad \bullet J = \int_{-2}^4 e^{-x^2} dx.$$

Sur le même principe, on peut également estimer la valeur de sommes (finies ou infinies) à l'aide d'une méthode de Monte-Carlo.

Exercice 3 (★★)

Estimer la valeur des sommes suivantes à l'aide de la méthode de Monte-Carlo :

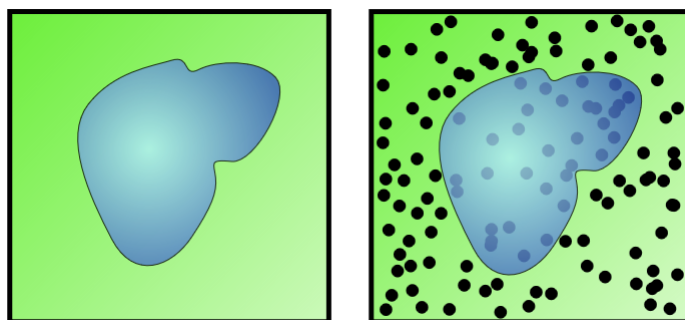
$$\bullet \sum_{k=0}^{10} \frac{\binom{10}{k}}{\sqrt{1+k^4}} \qquad \bullet \sum_{k=1}^{+\infty} \frac{1}{k^3 2^k}.$$

Indication : penser au théorème de transfert pour une variable suivant respectivement la loi $\mathcal{B}(10, 1/2)$ et la loi $\mathcal{G}(1/2)$.

2.2 Calcul d'aires

Il est également possible d'utiliser des raisonnements probabilistes pour estimer des aires.

Considérons par exemple un terrain carré dont la longueur des côtés, et donc l'aire $\mathcal{A}_{\text{terrain}}$, sont connues. Au sein de cette zone se trouve un lac dont la superficie est inconnue. Pour trouver l'aire du lac \mathcal{A}_{lac} , on demande à une armée de tirer N coups de canon de manière aléatoire sur cette zone.



La probabilité p qu'un boulet soit tombé dans le lac est égale à :

$$p = \frac{\mathcal{A}_{\text{lac}}}{\mathcal{A}_{\text{terrain}}}.$$

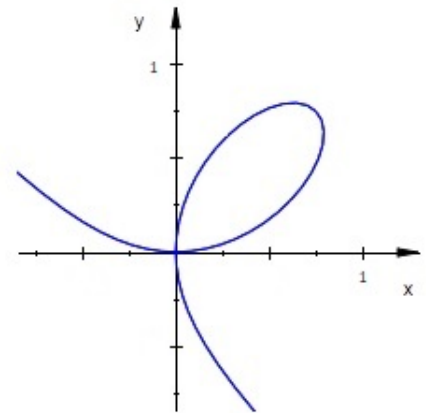
Pour estimer la superficie du lac, il suffit de compter le nombre n de boulets qui sont restés sur le terrain. En effet pour N grand, on a par le théorème d'or de Bernoulli :

$$p \approx \frac{N - n}{N} \quad \text{et donc} \quad \mathcal{A}_{\text{lac}} \approx \frac{N - n}{N} \times \mathcal{A}_{\text{terrain}}.$$

Exercice 4 (★)

Considérons la courbe ci-contre (appelée folium de Descartes), dont on cherche à calculer une valeur approchée de l'aire formée par la boucle. On admet que l'intérieur de cette boucle est l'ensemble :

$$\mathcal{B} = \{(x, y) \in \mathbb{R}^2 : x \geq 0, y \geq 0, x^3 + y^3 \leq \frac{3xy}{2}\}.$$



Folium de Descartes.

Compléter le programme suivant afin qu'il retourne une valeur approchée de l'aire de \mathcal{B} .

```

1 | n = 10000
2 | X = rd.random(n)
3 | Y = ...
4 | S = 0
5 | for i in range(n):
6 |     if ..... :
7 |         S = S+1
8 | print(.....)
    
```

Exercice 5 (★★ - Calcul d'une valeur approchée de π)

Soit $\mathcal{D} = \{(x, y) \in \mathbb{R}^2, x, y \geq 0, x^2 + y^2 \leq 1\}$

1. Représenter \mathcal{D} . Quelle est son aire ?
2. Soient $(X_i)_{1 \leq i \leq n}$ et $(Y_i)_{1 \leq i \leq n}$ des variables i.i.d. suivant la loi $\mathcal{U}([0, 1])$. On note T_i la variable qui vaut 1 si le point (X_i, Y_i) appartient à \mathcal{D} , et 0 sinon.
 - (a) Déterminer la loi de T_i .
 - (b) On pose $\overline{T}_n = \frac{1}{n} \sum_{i=1}^n T_i$. Montrer que $4\overline{T}_n$ est un estimateur convergent de π .
3. Écrire un programme qui calcule une valeur approchée de π .
4. (a) Déterminer un intervalle de confiance asymptotique de π , au niveau de risque $\alpha = 0.05$, dont les bornes dépendent de \overline{T}_n .
 - (b) Quelle valeur de n faut-il choisir pour obtenir une approximation de π à $\varepsilon = 0.01$ près, au niveau de risque $\alpha = 0.05$?

2.3 Calcul de la fonction de répartition d'une variable aléatoire

Exercice 6 (★★ - Fonction de répartition empirique de la loi $\mathcal{N}(0,1)$)

1. Écrire une fonction `phi_empirique` qui, à un réel x entré par l'utilisateur, renvoie une estimation de $\Phi(x)$ par la méthode de Monte-Carlo.
2. On rappelle que la fonction de répartition Φ de la loi $\mathcal{N}(0,1)$ est accessible dans la librairie `scipy.special` à l'aide de la commande `sp.ndtr`. Comparer les résultats obtenus à l'aide de la fonction `phi_empirique` avec ceux renvoyés par la commande `sp.ndtr`.

Remarque. La méthode de Monte-Carlo avait déjà été utilisée au TP5 (sans la nommer ainsi) pour tracer la fonction de répartition empirique G_X d'une variable aléatoire X à partir d'un échantillon \mathbf{E} . Rappelons-en le principe :

- on discrétise l'intervalle $[\min(\mathbf{E}), \max(\mathbf{E})]$ à l'aide de la commande

$$\mathbf{u} = \text{np.linspace}(\text{np.min}(\mathbf{E}), \text{np.max}(\mathbf{E}), 100) ;$$

- pour chaque élément `u[k]` de `u`, on détermine la fréquence des modalités inférieures ou égales à `u[k]` dans l'échantillon \mathbf{E} , qu'on stocke dans la k -ème composante d'un vecteur `v` ;
- on trace la ligne brisée d'abscisses données par `u` et d'ordonnées par `v`

La valeur stockée dans `v[k]` donne bien une estimation de $F_X(x) = P(X \leq x)$, comme nous avons pu le faire dans l'exercice précédent.

Cette méthode permet de représenter la courbe représentative de la fonction de répartition F_X de n'importe quelle variable aléatoire X , même dans les cas où il aurait été bien délicat d'obtenir une expression explicite de F_X .

Exercice 7 (★)

Tracer la fonction de répartition empirique de $Z = XY$ où $X \hookrightarrow \mathcal{P}(2)$ et $Y \hookrightarrow \mathcal{E}(1)$ sont indépendantes.

2.4 Comparaison de différents estimateurs ponctuels

Exercice 8 (★★)

Soit (X_1, \dots, X_n) un échantillon de loi mère $\mathcal{E}(\lambda)$. Nous disposons des deux estimateurs de $\theta = \frac{1}{\lambda}$:

- la moyenne empirique $\overline{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$;
- l'écart type empirique $S_n = \sqrt{\frac{1}{n} \sum_{k=1}^n (X_k - \overline{X}_n)^2}$.

On souhaite comparer ces deux estimateurs. Prenons pour commencer $\lambda = 1$ et $n = 100$.

1. On considère les instructions suivantes :

```

1 | lbd = 1 ; theta = 1/lbd ; n = 100 ; m = 1000
2 | E = rd.exponential(1, [n,m])
3 | S = np.std(E,0) ;
4 | bS = np.mean(S) - theta
5 | rS = np.std(S)**2 + (np.mean(S)-theta)**2
6 | print(bS,rS)

```

Que contiennent les variables `S`, `bS` et `rS` ?

2. Procéder de même avec \overline{X}_n comme estimateur de θ .

3. Recommencer avec d'autres valeurs de λ et de n . Quel estimateur privilégieriez vous ?
4. On souhaite maintenant comparer ces deux estimateurs à l'aide de leur histogramme des fréquences. Écrire pour cela un programme qui :

- simule $m = 10000$ n -échantillons de la loi $\mathcal{E}(\lambda)$;
- calcule, pour chaque échantillon observé, l'estimation correspondante obtenue à l'aide de chacun des estimateurs \overline{X}_n et S_n ;
- trace l'historgramme des 10000 estimations obtenues avec chacun des estimateurs.

On rappelle pour cela que la commande `plt.hist(x,50,density='True',edgecolor='k')` trace l'historgramme des fréquences de la série statistique x , en répartissant ses éléments en 50 classes équiréparties entre la plus petite et la plus grande valeur de x .

On pourra également utiliser les commandes `plt.subplot(1,2,1)` et `plt.subplot(1,2,2)` pour tracer les historgrammes sur la même fenêtre graphique l'un à côté de l'autre.

Comparer les deux historgrammes. Recommencer pour d'autres valeurs de λ . Quel semble être le meilleur estimateur ?

2.5 Calcul du niveau de confiance réel d'un intervalle de confiance

Afin de déterminer des intervalles de confiance, nous avons besoin d'inverser la fonction Φ , notamment pour déterminer le réel $t_\alpha = \Phi^{-1}(1 - \frac{\alpha}{2})$. Nous utiliserons à cet effet la commande `sp.ndtri`.

Par exemple, si l'on souhaite déterminer le nombre x tel que $\Phi(x) = 0.75$, on entre `sp.ndtri(0.75)`.

Exercice 9 (★ - Des valeurs classiques)

Avec les notations habituelles, déterminer les valeurs de t_α pour $\alpha = 0.05$ et $\alpha = 0.01$.

Exercice 10 (★★ - Intervalle de confiance de l'espérance d'une loi normale)

Nous avons montré en TD que si X_1, \dots, X_n sont des variables i.i.d. suivant la loi normale $\mathcal{N}(\theta, 1)$, alors $\left[\overline{X}_n - \frac{t_\alpha}{\sqrt{n}}, \overline{X}_n + \frac{t_\alpha}{\sqrt{n}} \right]$ est un intervalle de confiance de θ au niveau de confiance $1 - \alpha$.

1. Étendue de l'intervalle de confiance.

- (a) Écrire une fonction `def etendue(n,alpha)` : qui prend en paramètres un entier n et un réel $\alpha \in]0, 1[$, et renvoie l'étendue de l'intervalle de confiance de θ correspondant.
- (b) Essayer ce programme pour $\alpha = 0.05$ en faisant varier n . Puis à n fixé, étudier l'étendue de l'intervalle pour $\alpha = 0.05$, $\alpha = 0.01$, $\alpha = 0.001$ et $\alpha = 0.0001$. Que constate-t-on ?

2. Niveau de confiance réel.

Prenons dans la suite $\alpha = 0.05$ et $n = 1000$. On souhaite estimer le niveau de confiance réel de l'intervalle de confiance par la méthode de Monte-Carlo.

- (a) Qu'effectue le code suivant :

```

1 | theta = rd.exponential(1/1) ; n = 1000 ; m = 10000 ; t=sp.ndtri(0.975)
2 | E = rd.normal(theta,1, [n,m])
3 | Xbar = np.mean(E,0)
4 | c = 0
5 | for k in range(m):
6 |     if np.abs(Xbar[k]-theta)<t/np.sqrt(n):
7 |         c = c+1
8 | print("theta :",theta) ;
9 | print("Estimation du niveau de conf de l'IdC :",c/m)

```

(b) Exécuter plusieurs fois ce code. Est-ce conforme à ce que vous vous attendiez ?

 **Le saviez-vous ?**

Les méthodes de Monte-Carlo ont été développées par les physiciens J. Von Neumann et S. Ulam du Projet Manhattan lors de l'élaboration dans le plus grand secret des premières bombes atomiques au laboratoire national de Los Alamos (Nouveau-Mexique) au début des années 1940. Ils disposaient alors du tout premier ordinateur électronique, l'ENIAC, qui pesait 30 tonnes, calculait à peu près 10 millions de fois moins vite que le dernier Iphone et était très souvent en panne. Il s'agissait évidemment d'un énorme progrès autorisant des calculs qu'il n'aurait pas été possible de faire à la main, mais les capacités de calcul étaient tout de même limitées.

Les utilisateurs de l'ENIAC développèrent alors les méthodes expliquées précédemment pour faire des calculs approchés avec le moins d'opérations possible afin de minimiser le temps d'utilisation de leur ordinateur. L'avantage des méthodes expliquées précédemment est qu'on sait à l'avance combien d'opérations seront nécessaires, puisque nous fixons nous même la taille de l'échantillon. La contrepartie est qu'on obtient alors des valeurs qui sont peut-être inexactes... Puisque le projet était secret, il lui fallait un nom de code, et le nom de Monte-Carlo fut choisi en référence aux jeux de hasard des casinos monégasques.

Les algorithmes probabilistes sont généralement regroupés en trois catégories :

- les algorithmes de Monte-Carlo, qui retournent des valeurs « probablement correctes » en un temps déterminé à l'avance ;
- les algorithmes de Las Vegas qui donnent des valeurs exactes en temps probablement court (mais qui peut éventuellement être très long) ;
- les algorithmes d'Atlantic City qui donnent une réponse probablement correcte en un temps probablement court (mais qui peuvent éventuellement donner une « mauvaise » réponse avec un temps de calcul long).