⋆ Méthode de Monte Carlo ⋆

Compétences attendues

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

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.

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 $E(X) = C$ avec une garantie d'approximation.

- La *loi faible des grands nombres* assure que $\overline{X_n} = \frac{1}{n}\sum_{k=1}^{n} X_k$ est un estimateur sans biais et convergent de C.
- Le *théorème limite central* assure que $\left[\overline{X_n} - t_\alpha \frac{\sigma}{\sqrt{n}}, \overline{X_n} + t_\alpha \frac{\sigma}{\sqrt{n}}\right]$ 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 le remplacera par un estimateur convergent de σ.

Méthode 1.1 [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 mean).

1.2 Calcul numérique d'une probabilité

Une probabilité $p = P(A)$ peut être considérée comme l'espérance d'une variable de Bernoulli $X \hookrightarrow \mathscr{B}(p)$,

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 évènement.

Méthode 1.2 [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

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 $g : [0,1] \to \mathbb{R}$ 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 :

$$S_n \xrightarrow{P} \int_0^1 g(x)\mathrm{d}x$$

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

2. On souhaite déterminer une valeur approchée de l'intégrale $I = \int_0^1 \cos\left(x^2\right) \mathrm{d}x$.
   Compléter le programme suivant afin qu'il retourne une valeur approchée de I.

```
def g(x):
    return ...

N = 1000
S = 0
for i in range(N):
    S = S + ...
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 `scipy.integrate`.

Remarque 2.1. 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 marche mal.

Exercice 2

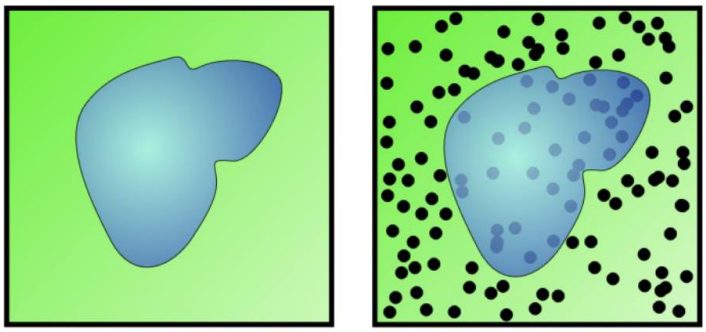
Estimer la valeur des intégrales suivantes à l'aide de la méthode de Monte-Carlo.

$$I = \int_0^{+\infty} \frac{\mathrm{e}^{-x}}{1+x^4}\,\mathrm{d}x \quad \text{et} \quad J = \int_{-2}^{4} \mathrm{e}^{-x^2}\,\mathrm{d}x.$$

2.2 Calcul d'aires

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

Considérons par exemple un terrain carré dont la longueur des côtés, et donc l'aire $\mathscr{A}_{\text{terrain, sont}}$ connues. Au sein de cette zone se trouve un lac dont la superficie est inconnue. Pour trouver l'aire du lac $\mathscr{A}_{\text{lac}}$, 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 :

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

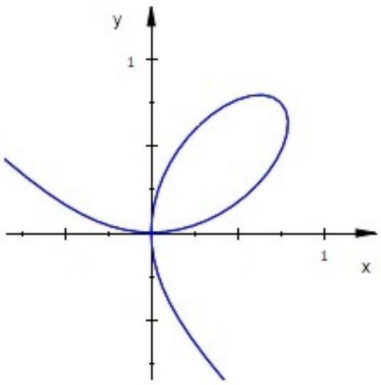
Pour estimer la superficie du lac, il suffit de compter le nombre n de boulets qui sont restés sur le terrain. En effet pour N grand, on a par le théorème d'or de Bernoulli :

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

Exercice 3

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 :

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



Folium de Descartes.

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

```
import numpy.random as rd
N = 10000
X = rd.random(N)
Y = ....
S = 0
for i in range(N):
    if ....... :
        S = S + 1
print(...)
```

Exercice 4 [Calcul d'une valeur approchée de π]

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

1. Représenter $\mathscr{D}$. 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 $\mathscr{U}([0,1])$. On note T_i la variable qui vaut 1 si le point (X_i, Y_i) appartient à $\mathscr{D}$, 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 $4\overline{T_n}$ converge en probabilité vers 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 5 [Fonction de répartition empirique de la loi $\mathscr{N}(0,1)$]

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 `scipy.special.ndtr(x)` renvoie $P(X \leq x)$ où $X \hookrightarrow \mathscr{N}(0,1)$. Comparer les résultats obtenus à l'aide de la fonction `phi_empirique` avec ceux renvoyés par la commande `scipy.special.ndtr`.

Remarque 2.2. La méthode de Monte-Carlo avait déjà été utilisée dans le TP9 (sans la nommer ainsi) pour tracer la fonction de répartition empirique G_X d'une variable aléatoire X. Rappelons-en le principe :

- on génère un échantillon observé E distribué suivant la loi de X ;
- on trie cet échantillon par modalités croissantes (à l'aide de la commande `y =np.sort(E)`);
- on trace la fonction en escalier G_X d'abscisse les modalités et d'ordonnée la fréquence cumulée croissante de ces modalités.

Pour tout $x \in \mathbb{R}, G_X(x)$ correspond à la fréquence des modalités inférieures ou égales à x dans l'échantillon E. Ce qui donne bien une estimation de $F_X(x) = P(X \leq x)$, comme nous avons pu le faire dans l'exercice précédent.

Exercice 6 Tracer la fonction de répartition empirique de $Z = XY$ où $X \hookrightarrow \mathscr{P}(2)$ et $Y \hookrightarrow \mathscr{E}(1)$ sont indépendantes.

2.4 Comparaison de différents estimateurs ponctuels

Exercice 7

Soit $(X_1, \ldots, X_n)$ un échantillon de loi mère $\mathscr{E}(\lambda)$. 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 $S_n = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}\left(X_k - \overline{X_n}\right)^2}$.

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

1. On considère les instructions suivantes :

```
import numpy as np
import numpy.random as rd
lamb = 1
theta = 1/lamb
n = 100
E = rd.exponential(1,(n,1000))
S = np.std(E, axis=0)
bS = np.mean(S) - theta
rS = np.mean((S-theta)**2)
```

   Que contiennent les variables `S`, `bS` et `rS` ?
2. Estimer de même le biais et le risque quadratique de $\overline{X_n}$ comme estimateur de θ.
3. Recommencer avec d'autres valeurs de λ et de n. Quel estimateur privilégieriez vous ?
4. On souhaite enfin comparer ces deux estimateurs à l'aide de leur histogramme des fréquences. Écrire pour cela un programme qui :
   - simule $m = 10000n$-échantillons de la loi $\mathscr{E}(\lambda)$;
   - calcule, pour chaque échantillon observé, l'estimation correspondante obtenue à l'aide de chacun des estimateurs $\overline{X_n}$ et S_n ;
   - trace l'histogramme des 10000 estimations obtenues avec chacun des estimateurs.
     On rappelle pour cela que la commande `plt.hist(x, bins=50)` trace l'histogramme 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 histogrammes sur la même fenêtre graphique l'un à côté de l'autre.

   Comparer les deux histogrammes. 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_α tel que $\Phi(t_\alpha) = 1 - \frac{\alpha}{2}$, soit encore $t_\alpha = \Phi^{-1}\left(1 - \frac{\alpha}{2}\right)$. Nous utiliserons à cet effet la commande `norm.ppf` sous `scipy.stats` de la manière suivante : pour déterminer l'unique réel x tel qu'une variable aléatoire $X \hookrightarrow \mathscr{N}(\mu, \sigma)$ vérifie $P(X \leq x) = p$, on saisit :

```
import scipy.stats as ss
x = ss.norm.ppf(p, mu, sigma)
```

Par exemple, si l'on souhaite déterminer le nombre x tel que $\Phi(x) = 0.75$, on entre `ss.norm.ppf(0.75, 0,1)` ou simplement `ss.norm.ppf(0.75)`.

Exercice 8 [Des valeurs classiques]

Avec les notations habituelles, déterminer les valeurs de t_α pour $\alpha = 0.05$ et $\alpha = 0.01$.
On pourra utiliser la commande `scipy.stats.norm.ppf` *(percent point function).*

Exercice 9 [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 $y = e$ 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 déterminer le niveau de confiance réel de l'intervalle de confiance par la méthode de Monte-Carlo.
   (a) Qu'effectue le code suivant :

```
import numpy as np
import numpy.random as rd
theta = rd.exponential(1)
n = 1000
m = 10000
t = 1.96
E = rd.normal(theta,1, (m,n))
Xbar = np.mean(E,axis=1)
c = 0
for k in range(m):
    if np.abs(Xbar[k] - theta)<t/np.sqrt(n) :
        c = c+1
print('Theta :', theta)
print('Proportion d'IdC contenant theta :', 100*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 valeur de n. 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).