Calcul d'intégrales : méthode de Monte-Carlo

1. Introduction

Ce document explique le principe du calcul d'une intégrale par la méthode de Monte-Carlo. On fera une comparaison avec la méthode des rectangles, pour mettre en évidence l'avantage de la méthode de Monte-Carlo sur les méthodes de quadrature pour les calculs d'intégrale dans un espace de dimension élevée.

2. Calcul numérique d'une intégrale : méthode des rectangles

On cherche à calculer une approximation numérique de l'intégrale d'une fonction f sur un intervalle $[a, b]$. Soient N valeurs x_i régulièrement réparties sur l'intervalle $[a, b]$, espacées de

$$h = \Delta x = \frac{b-a}{N} \tag{1}$$

Le calcul numérique d'une intégrale par une méthode de quadrature consiste à utiliser une approximation de l'intégrale sur chaque intervalle $[x_i, x_{i+1}]$ qui s'exprime en fonction des valeurs de f sur ses deux bornes, et éventuellement sur d'autres points de l'intervalle.

On se contente de citer la méthode la plus simple, la méthode des rectangles, qui consiste à assimiler la fonction sur chaque intervalle $[x_i, x_{i+1}]$ à une fonction constante :

$$\int_a^b f(x)\,dx \simeq \frac{b-a}{N}\sum_{i=0}^{N-1} f(x_i) \tag{2}$$

Considérons par exemple le calcul de l'intégrale suivante :

$$\int_0^\pi \sin(x)\,dx = 2 \tag{3}$$

La fonction suivante calcule cette intégrale avec la méthode des rectangles et renvoie l'erreur :

```
import numpy
from matplotlib.pyplot import *

def integrale(N):
    x = numpy.linspace(0,numpy.pi,N)
    f = numpy.sin(x)
    return abs(f.sum()*numpy.pi/N-2.0)
```

Voici un exemple :

```
e = integrale(100)
```

```
print(e)
--> 0.020166157744947677
```

Multiplions par 10 le nombre de points :

```
e = integrale(1000)
```

```
print(e)
--> 0.0020016465809187256
```

L'erreur est divisée par 10. La méthode des rectangles a une erreur d'ordre 1, proportionnelle à h^1. D'autres méthodes ont une erreur d'ordre plus élevé par rapport à h, qui permet de converger plus rapidement. Par exemple, la méthode des trapèzes a une erreur d'ordre 2 : une réduction de h d'un facteur 10 réduit l'erreur d'un facteur 100.

Considérons à présent le cas des intégrales doubles. Pour un domaine d'intégration carré [a,b]x[a,b], la méthode des rectangles s'écrit :

$$\int_a^b \int_a^b f(x,y)\,dxdy \simeq \left(\frac{b-a}{N}\right)^2 \sum_{i,j} f(x_i,y_j) \tag{4}$$

Faisons le calcul pour l'intégrale suivante :

$$\int_0^\pi \int_0^\pi \sin(x)\cos(y)\,dxdy = 0 \tag{5}$$

```
def integrale2(N):
    dx = numpy.pi/N
    def f(i,j):
        return numpy.sin(i*dx)*numpy.cos(j*dx)
    f = numpy.fromfunction(f,(N,N))
    return abs(f.sum()*(numpy.pi/N)**2)
```

```
e=integrale2(100)
```

```
print(e)
--> 0.062826685274008073
```

```
e=integrale2(1000)
```

```
print(e)
--> 0.0062831801394658184
```

La division du pas par 10 a le même effet que pour l'intégration simple : l'erreur est divisée par 10. Cependant, le nombre de calculs effectués (évaluation de f et somme) est multiplié par 100. De manière générale, un calcul d'intégrale en dimension d par la méthode des rectangles nécessite une augmentation du temps de calcul d'un facteur 10^d pour augmenter la précision d'un facteur 10. Pour les autres méthodes de quadrature plus précises, on retrouve cette dépendance en puissance par rapport à la dimension de l'espace.

3. Méthode de Monte-Carlo

3.a. Principe

On reprend l'exemple de l'intégrale d'une fonction à une variable sur l'intervalle $[a, b]$. Soient N réels x_i tirés aléatoirement sur l'intervalle $[a, b]$ avec une densité de probabilité uniforme, égale à

$$p(x) = \frac{1}{b-a} \tag{6}$$

L'évaluation de l'intégrale par la méthode de Monte-Carlo consiste (dans sa forme élémentaire) à calculer la somme suivante :

$$S_N = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(x_i) \tag{7}$$

Il s'agit de la même formule que celle de la méthode des rectangles, avec des points répartis aléatoirement sur l'intervalle.

En introduisant la variable aléatoire X, cette somme est une évaluation de l'espérance de $f(X)$, multipliée par $(b-a)$.

Pour obtenir une expression plus générale, considérons une variable aléatoire réelle X sur l'intervalle $[a, b]$ avec une densité de probabilité $p(x)$.

Pour une fonction g, l'espérance de $g(X)$ est :

$$E(g(X)) = \int_a^b p(x)g(x)\,dx \tag{8}$$

En posant $f(x) = p(x)g(x)$, on peut écrire :

$$E(f/p) = \int_a^b f(x)\,dx \tag{9}$$

Une approximation de l'intégrale est donc obtenue en évaluant l'espérance de f/p avec N échantillons x_i tirés dans l'intervalle $[a, b]$ avec une densité de probabilité p, soit :

$$S_N = moy(f/p, N) = \frac{1}{N}\sum_{i=0}^{N-1}\frac{f(x_i)}{p(x_i)} \tag{10}$$

Dans le cas d'une densité uniforme, on retrouve l'expression (7).

Notons σ^2 la variance de $f(X)/p(X)$, définie par :

$$\sigma^2 = E\left((f/p)^2\right) - E(f/p)^2 \tag{11}$$

La variance de la moyenne empirique est :

$$var(S_N) = \frac{\sigma^2}{N} \tag{12}$$

Il s'en suit que l'écart-type se réduit comme l'inverse de la racine carré de N lorsqu'on augmente le nombre d'échantillons. Ce résultat reste valable pour une intégrale double, triple, etc. L'évolution de la variance de la somme avec N est indépendante de la dimension d de l'espace. Pour une intégrale simple, cette convergence en $1/\sqrt{N}$ est plus lente que la convergence en $1/N$ de la méthode des rectangles, qui est la méthode de quadrature la plus lente. La méthode

de Monte-Carlo peut cependant devenir avantageuse pour les intégrales triples ou en dimension d supérieure à 3, lorsque la précision souhaitée est faible.

Pour évaluer la précision du résultat, il faut évaluer la variance σ^2. Cela peut se faire avec le calcul de la moyenne, en calculant la variance empirique :

$$v_N = var(f/p, N) = \frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{f(x_i)}{p(x_i)} \right)^2 - S_N^2 \tag{13}$$

En divisant par N, on obtient la variance de la somme. L'intervalle de confiance à 95 pour cent est :

$$\left[S_N - \frac{1,96\sqrt{v_N}}{\sqrt{N}}, S_N + \frac{1,96\sqrt{v_N}}{\sqrt{N}} \right] \tag{14}$$

L'intégrale considérée a une probabilité 0,95 de se trouver dans cet intervalle.

La dépendance de la variance de la somme en fonction de N est toujours $1/\sqrt{N}$, quelle que soit la dimension de l'espace et la fonction intégrée. Pour réduire la variance, il faut réduire la variance σ^2 de f/p. L'amélioration de la convergence consiste donc à choisir une densité de probabilité $p(x)$ qui minimise la variance et qui ne soit pas trop difficile à échantillonner. Supposons que $f(x) \geq 0$ et considérons la densité de probabilité particulière suivante :

$$p_o(x) = \frac{f(x)}{\int_a^b f(x)\,dx} \tag{15}$$

La variance est dans ce cas :

$$\sigma^2 = var\left(\int_a^b f(x)\,dx \right) = 0 \tag{16}$$

La variance est nulle, mais cette densité de probabilité $p_o(x)$ n'est pas réalisable car il faudrait pour cela connaître l'intégrale que l'on cherche. Pour réduire la variance, on cherche à définir une densité de probabilité qui s'approche au mieux de la forme de la fonction f, tout en étant simple et peu coûteuse à échantillonner. C'est ce qu'on appelle *l'échantillonnage préférentiel* (importance sampling). La densité de probabilité doit être grande là où la fonction f a une valeur absolue grande.

3.b. Exemple à une dimension

```
import numpy.random
import random
import math
import numpy
```

La fonction suivante effectue l'intégration de Monte-Carlo d'une fonction sur un intervalle [a,b], avec une densité de probabilité $p(x)$ uniforme. Elle renvoie l'estimation de l'intégrale et le demi-intervalle de confiance à 95 pour cent.

```
def integration1d(fonction,a,b,N):
    x = a+(b-a)*numpy.random.random_sample(N)
    p = 1.0/(b-a)
    f = fonction(x)
```

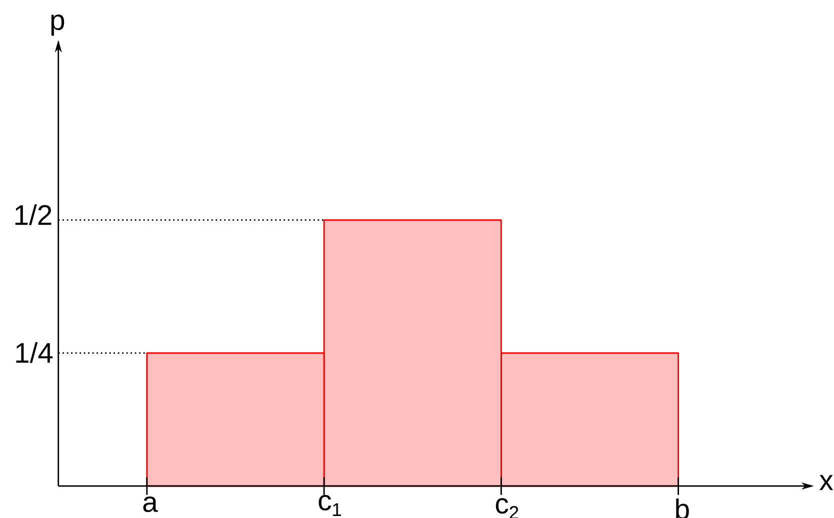
```
    moyenne = f.sum()/(N*p)
    g = f*f
    variance = g.sum()*1.0/(N*p*p)-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96)
```

Voici un exemple, avec l'intégrale déjà considérée plus haut :

```
def f(x):
    return numpy.sin(x)
(integrale,intervalle) = integration1d(f,0,math.pi,1000)
```

```
print((integrale,intervalle))
--> (1.9776925716412701, 0.06172322830900628)
```

Pour le même nombre de points calculés, la méthode des rectangles est beaucoup plus précise. Pour réduire la variance, on peut chercher à augmenter la densité de probabilité autour de $\pi/2$, où la fonction est maximale. Bien sûr, ce type d'amélioration suppose que l'on ait des informations sur la fonction à intégrer. Une densité constante par morceaux est un bon choix, car elle peut être échantillonnée simplement. Pour obtenir une distribution non uniforme simple, on peut diviser l'intervalle en trois parts égales, et attribuer une probabilité double à l'intervalle central :



Pour échantillonner cette distribution, il suffit de tirer un nombre aléatoire p parmi 1,2,3,4 avec une égale probabilité. Si p=1, on tire x dans le premier intervalle avec une densité uniforme. Si p=2 ou 3, on tire x dans le second intervalle. Si p=4, on tire x dans le troisième intervalle. Pour chaque tirage, on doit diviser la valeur de $f(x)$ par la densité de probabilité. Les densités de probabilité des trois intervalles sont respectivement égales à $1/4$, $1/2$ et $1/4$ divisés par la largeur de chaque intervalle.

```
def integration1d_importance(fonction,a,b,N):
    delta = (b-a)*1.0/3
    c1 = a+delta
    c2 = a+2*delta
    somme = 0.0
    somme2 = 0.0
    for i in range(N):
        p = random.randint(1,4)
        if p==1:
            x = random.uniform(a,c1)
            f = fonction(x)*4
        elif p==2 or p==3:
            x = random.uniform(c1,c2)
            f = fonction(x)*2
        else:
            x = random.uniform(c2,b)
            f = fonction(x)*4
        somme += f
        somme2 += f*f
    moyenne = somme*delta/N
    variance = somme2*delta*delta/N-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96)
```

```
(integrale,intervalle) = integration1d_importance(f,0,math.pi,1000)
```

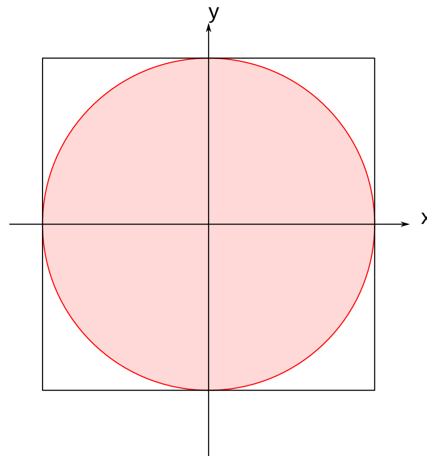
```
print((integrale,intervalle))
--> (1.9782931763755307, 0.04799482505984376)
```

On a bien une réduction de la variance pour le même nombre d'échantillons. Pour évaluer l'efficacité de la nouvelle méthode, il faudrait aussi tenir compte du surcoût dû au tirage d'un nombre aléatoire supplémentaire. Lorsque l'évaluation de f est très coûteuse (ce n'est pas le cas ici) et domine le temps de calcul, l'échantillonnage préférentiel est certainement un bon choix.

3.c. Exemple à deux dimensions

Considérons le domaine d'intégration Ω = [-1,1]x[-1,1] et le disque D de rayon 1. On cherche à calculer l'intégrale double suivante :

$$\int_D f(x,y)dxdy \tag{17}$$



Il est possible d'échantillonner avec une densité uniforme dans un disque (voir plus loin), mais une méthode plus générale consiste à échantillonner uniformément dans le domaine carré Ω. D'une manière plus générale, le domaine d'intégration D peut avoir une forme complexe, et on choisit le domaine rectangulaire Ω le plus petit (parallèle aux axes) qui contienne D. Il suffit alors de poser $f(x) = 0$ en dehors de D.

Voici une fonction qui fait l'intégration dans un domaine rectangulaire. La densité de probabilité est l'inverse de l'aire du rectangle.

```
def integration2d(fonction,a,b,c,d,N):
    x = a+(b-a)*numpy.random.random_sample(N)
    y = c+(d-c)*numpy.random.random_sample(N)
    p = 1.0/((b-a)*(d-c))
    somme = 0.0
    somme2 = 0.0
    for i in range(N):
        f = fonction(x[i],y[i])
        somme += f
        somme2 += f*f
    moyenne = somme/(p*N)
    variance = somme2/(p*p*N)-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96)
```

Voici par exemple le calcul de l'intégrale de la fonction qui vaut 1 sur le disque, qui est égale à π :

```
def f(x,y):
    if x*x+y*y <= 1:
        return 1
    else:
        return 0

(integrale,intervalle) = integration2d(f,-1,1,-1,1,10000)


print((integrale,intervalle))
--> (3.156, 0.0319886411440061)
```

On remarque que les points tirés en dehors du cercle ne sont pas pris en compte dans la somme (puisque f prend une valeur nulle). Cela revient à échantillonner des points dans le disque avec une méthode de rejet.

Pour revenir au cas plus général d'une fonction f à intégrer sur le disque, on remarque qu'il y a un échantillonnage préférentiel évident consistant à échantillonner seulement sur le disque. Pour ce faire, il ne faut pas utiliser une méthode de rejet, qui nous ramènerait au cas précédent. Dans le cas du disque, l'inversion de la fonction de répartition a une solution analytique. Pour deux variables aléatoires (u_1, u_2) de densité uniforme sur l'intervalle [0,1], les variables polaires suivantes :

$$\theta = 2\pi u_1 \tag{18}$$
$$r = R\sqrt{u_2} \tag{19}$$

donnent une distribution uniforme sur le disque de rayon R. La fonction suivante calcule une intégrale sur le disque de rayon R. Les variables de la fonction sont en coordonnées polaires.

```
def integrationDisque(fonction,R,N):
    theta = 2*numpy.pi*numpy.random.random_sample(N)
    r = R*numpy.sqrt(numpy.random.random_sample(N))
    p = 1.0/(numpy.pi*R*R)
    somme = 0.0
    somme2 = 0.0
    for i in range(N):
        f = fonction(r[i],theta[i])
        somme += f
        somme2 += f*f
    moyenne = somme/(p*N)
    variance = somme2/(p*p*N)-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96)
```

Dans ce cas, le calcul de π avec $f = 1$ n'a plus de sens puisque la densité est l'inverse de π. Considérons plutôt l'intégration d'une fonction :

```
def f(r,theta):
    return 1-r*r

(integrale,intervalle) = integrationDisque(f,1.0,10000)


print((integrale,intervalle))
--> (1.5654527243187746, 0.017746448191176947)
```

Lorsqu'on intègre une fonction décroissante, on a intérêt à mettre en place un échantillonnage préférentiel qui favorise la zone centrale.

3.d. Exemple en dimension d

Dans les systèmes à grand nombre de degrés de libertés, on est amené à calculer des intégrales dans un espace de dimension d qui peut être très grand (par exemple plusieurs millions).

On peut citer l'espace des phases utilisé en physique statistique pour représenter la configuration d'un système comportant un grand nombre d'atomes. Les méthodes de quadrature sont très inefficaces, voire inutilisables, lorsque la dimension est élevée. Par exemple pour la méthode des rectangles, le nombre de points nécessaires pour maintenir une précision constante évolue comme N^d. Le logarithme du nombre de points est donc proportionnel à la dimension d, une loi que l'on retrouve pour toutes les méthodes de quadrature. La méthode de Monte-Carlo ne présente pas cette dépendance par rapport à la dimension. La variance de la somme calculée est :

$$var(S_N) = \frac{\sigma^2}{N} \tag{20}$$

La variance σ^2 de la variable f/p est susceptible d'augmenter avec la dimension, mais pas selon une loi en puissance. La méthode de Monte-Carlo a donc un avantage décisif en dimension élevée.

Comme exemple, nous allons calculer, en dimension d, le volume compris entre les hypersphères de rayons R_1 et R_2. La fonction renvoie aussi l'estimation de la variance σ^2.

```
def volume(dim,R1,R2,N):
    somme = 0.0
    somme2 = 0.0
    R12 = R1*R1
    R22 = R2*R2
    p = 1.0/math.pow(2*R2,dim)
    for i in range(N):
        r2 = 0.0
        for d in range(dim):
            x = random.uniform(-R2,R2)
            r2 += x*x
        if r2>= R12 and r2<=R22:
            f = 1.0
        else:
            f = 0.0
        somme += f
        somme2 += f*f
    moyenne = somme/(p*N)
    variance = somme2/(p*p*N)-moyenne*moyenne
    return (moyenne,variance,math.sqrt(variance/N)*1.96)
```

On commence par la sphère de rayon 1 en dimension 3, dont le volume est $4\pi/3$:

```
(m,v,e) = volume(3,0,1,10000)
```

```
print((m,v,e))
--> (4.1168, 15.98635776, 0.07836656938441033)
```

Voyons la sphère en dimension 5 :

```
(m,v,e) = volume(5,0,1,10000)
```

```
print((m,v,e))
--> (5.4112, 143.87731456, 0.2350997855408839)
```

Comme on le voit sur cet exemple, la variance dépend de la dimension. Augmentons le nombre d'échantillons pour obtenir un écart comparable à celui de la dimension 3. Comme l'écart évolue comme $1/\sqrt{N}$, il faut augmenter N d'un facteur 90 environ :

```
(m,v,e) = volume(5,0,1,10000*90)
```

```
print((m,v,e))
--> (5.253582222222223, 140.51450494546174, 0.024490372761522387)
```

Avec la méthode des rectangles, il aurait fallu augmenter le nombre de points d'un facteur N^2 pour maintenir la précision constante. Même si la variance augmente avec la dimension, la méthode de Monte-Carlo est incomparablement plus efficace en dimension élevée.

Voyons le volume d'une coque d'épaisseur un dixième du rayon :

```
(m,v,e) = volume(5,0.9,1,10**5)
```

```
print((m,v,e))
--> (2.1328, 63.70076416, 0.04946846021426743)
```

Voici les mêmes calculs en dimension 7 (beaucoup plus longs) :

```
(m,v,e) = volume(7,0,1,10**7)
```

```
print((m,v,e))
--> (4.7254144, 582.5235019482726, 0.014959352543089837)
```

```
(m,v,e) = volume(7,0.9,1,10**7)
```

```
print((m,v,e))
--> (2.4641664, 309.34118315311105, 0.010901215937687829)
```

On voit apparaître une propriété de la couche sphérique lorsque la dimension augmente : son volume se rapproche du volume de la sphère complète.