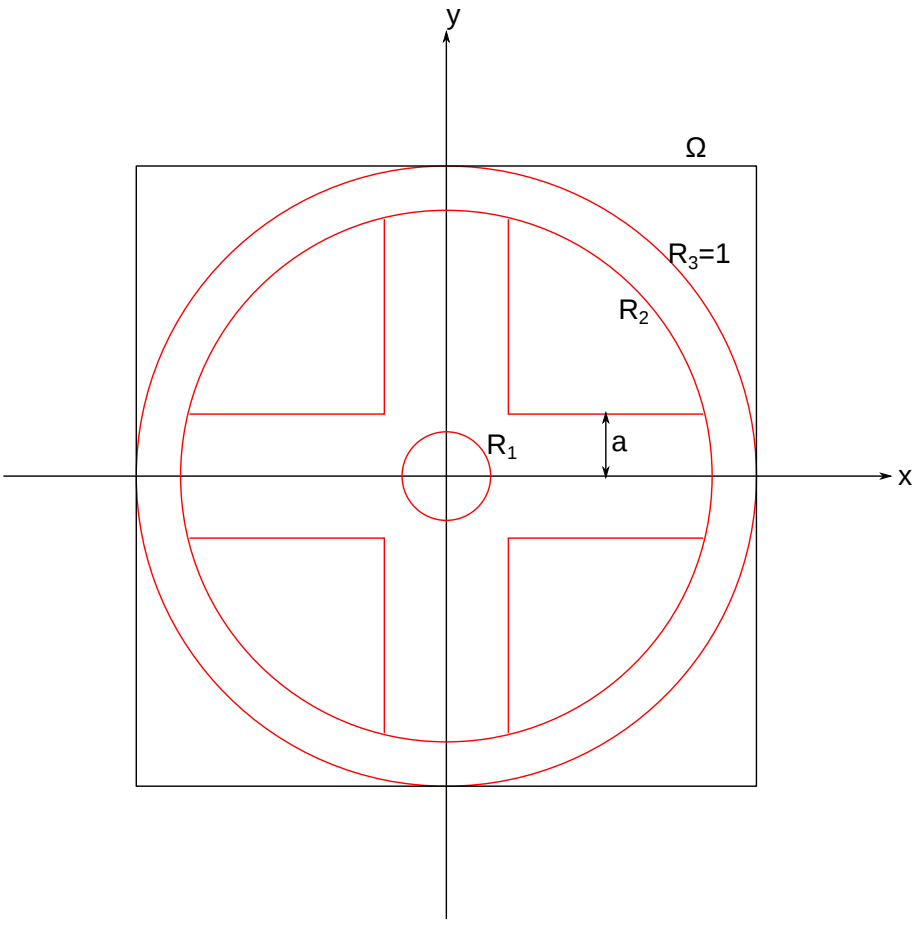
Calcul de moments d'inertie par intégration de Monte-Carlo

1. Introduction

L'objectif est de calculer les moments d'inertie d'un solide en utilisant la méthode d'intégration de Monte-Carlo. On verra dans un premier temps un échantillonnage uniforme sur un domaine carré contenant le solide, avant d'envisager un échantillonnage préférentiel, qui permettra de réduire la variance de la somme calculée.

2. Problème

On souhaite calculer les moments d'inertie d'un solide. On considère l'exemple d'une roue :



La roue est un cylindre d'axe Z et d'épaisseur e. Le rayon extérieur est $R_3 = 1$. le rayon intérieur est R_2. Le moyeu comporte un trou circulaire de diamètre R_1 et deux branches de largeur $2a$. On suppose que la densité de masse est uniforme.

On cherche à calculer le moment d'inertie de la roue par rapport à son axe Z : :

$$J_z = \int\int\int (x^2+y^2)\rho(x,y,z)\,dxdydz = e\int\int (x^2+y^2)\rho(x,y)\,dxdy \tag{1}$$

En raison du caractère bidimensionnel de ce solide, on est ramené à un calcul d'intégrale double, et on posera $e = 1$.

La densité de masse est $\rho = 1$ dans la matière (dans les deux branches du moyeu et dans la roue), et nulle en dehors.

Pour le calcul de l'intégrale, on utilisera le domaine carré Ω=[-1,1]x[-1,1]. Il s'agit donc de calculer :

$$J_z = \int_{-1}^{1}\int_{-1}^{1} (x^2+y^2)\rho(x,y)\,dxdy \tag{2}$$

On définit la fonction à intégrer :

$$f(x,y) = (x^2+y^2)\rho(x,y) \tag{3}$$

3. Méthode de Monte-Carlo avec échantillonnage uniforme

3.a. Principe

La méthode consiste à tirer aléatoirement N points dans le domaine carré Ω avec une densité de probabilité uniforme, égale à l'inverse de l'aire du carré :

$$p(x,y) = \frac{1}{4} \tag{4}$$

Pour chacun de ces N points, on calcule :

$$f_i = f(x_i,y_i) = (x_i^2+x_i^2)\rho(x_i,y_i) \tag{5}$$

Cette fonction est très facile à évaluer. Il suffit de faire des tests pour savoir si le point se trouve dans la matière, et de renvoyer x^2+y^2 si c'est le cas, 0 dans le cas contraire.

Une estimation de l'intégrale est obtenue en calculant la moyenne empirique de f/p, c'est-à-dire :

$$S_N = moy(f/p,N) = \frac{1}{N}\sum_{i=0}^{N-1}\frac{f_i}{p_i} \tag{6}$$

Dans le cas présent, la densité est uniforme ($p_i = p$), donc on peut diviser par p après le calcul de la somme. On calcule aussi la variance empirique de la manière suivante :

$$v_N = var(f/p,N) = \frac{1}{N}\sum_{i=0}^{N-1}\left(\frac{f_i}{p_i}\right)^2 - S_N^2 \tag{7}$$

L'intervalle de confiance à 95 pour cent pour l'intégrale est :

$$\left[S_N - \frac{1,96\sqrt{v_N}}{\sqrt{N}}, S_N + \frac{1,96\sqrt{v_N}}{\sqrt{N}}\right] \tag{8}$$

3.b. Réalisation

```
import numpy
import numpy.random
from matplotlib.pyplot import *
import math
```

On commence par écrire la fonction qui calcule $f(x, y)$:

```
def f_Jz(x,y,param):
    R1 = param[0]
    R2 = param[1]
    a = param[2]
    # ....
```

L'argument `param` est une liste qui contient les paramètres de la roue. Cette fonction doit être écrite avec le souci de minimiser le nombre de tests et de calculs.

```
def f_Jz(x,y,param):
    R1 = param[0]
    R2 = param[1]
    a = param[2]
    r2 = x*x+y*y
    if r2 > 1 or r2 < R1*R1:
        return 0.0
    if r2 > R2*R2:
        return r2
    if abs(x) < a:
        return r2
    if abs(y) < a:
        return r2
    return 0.0
```

La fonction de calcul de l'intégrale est :

```
integration2d(fonction,N,param,np)
```

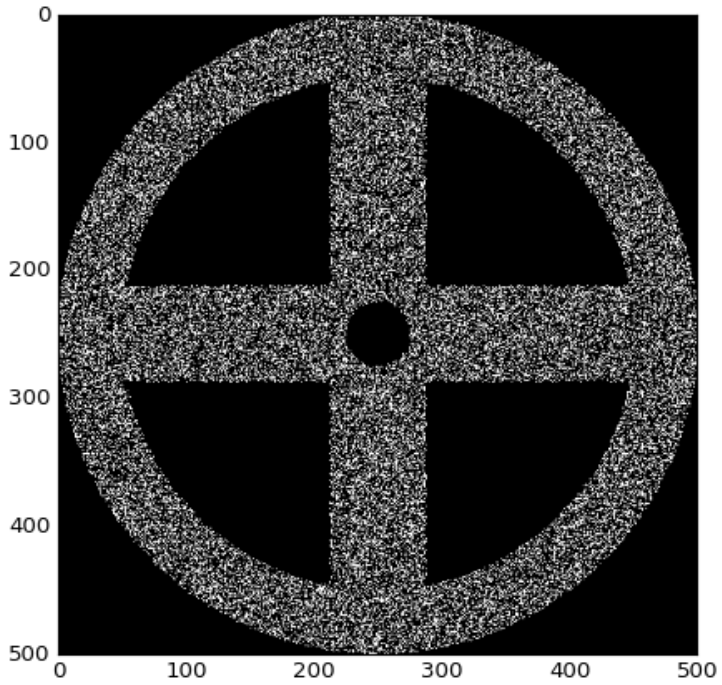
Cette fonction fait le calcul de la somme, de la variance et de l'intervalle de confiance à 95 pour cent. Elle génère aussi une image carrée dont le nombre de pixels sur chaque dimension est `np`. Cette image, noire au départ, est complétée par un pixel blanc pour chaque point tiré dont la valeur de f est non nulle.

```
def integration2d(fonction,N,param,np):
    image = numpy.zeros((np,np))
    x = -1+2*numpy.random.random_sample(N)
    y = -1+2*numpy.random.random_sample(N)
    p = 1.0/4
```

```
    somme = 0.0
    somme2 = 0.0
    for i in range(N):
        f = fonction(x[i],y[i],param)
        somme += f
        somme2 += f*f
        if f!=0:
            image[int((y[i]+1)/2*np),int((x[i]+1)/2*np)] = 1.0
    moyenne = somme/(p*N)
    variance = somme2/(p*p*N)-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96,image)
```

Voici un calcul :

```
param = [0.1,0.8,0.15]
N = 10**5
(integrale,intervalle,image) = integration2d(f_Jz,N,param,500)
figure()
imshow(image,cmap='gray')
```




```
print((integrale,intervalle))
--> (1.1361162182318325, 0.009031678004722264)
```

4. Méthode de Monte-Carlo avec échantillonnage préférentiel

4.a. Échantillonnage sur le disque

Une première méthode consiste à échantillonner uniformément sur le disque de rayon 1. On doit pour cela utiliser les coordonnées polaire (r, θ) et faire des tirages avec la densité de probabilité uniforme sur le disque :

$$p(r, \theta) = \frac{1}{\pi} \tag{9}$$

Pour réaliser cet échantillonnage à partir de variables à densité uniforme, on considère la fonction de répartition bidimensionnelle :

$$F(r, \theta) = \int_0^{\theta} \int_0^{r} \frac{r' d\theta' \, dr'}{\pi} = \frac{\theta}{2\pi} r^2 \tag{10}$$

L'inversion de la fonction de répartition permet d'échantillonner r et θ à partir de deux variables (u_1, u_2) uniformes sur l'intervalle $[0, 1]$:

$$\theta = 2\pi u_1 \tag{11}$$

$$r = \sqrt{u_2} \tag{12}$$

4.b. Réalisation

On redéfinit la fonction f en coordonnées polaires :

```
def f_Jz(r,theta,param):
    R1 = param[0]
    R2 = param[1]
    a = param[2]
    # .....


def f_Jz_polaire(r,theta,param):
    R1 = param[0]
    R2 = param[1]
    a = param[2]
    r2 = r*r
    if r2 > 1 or r2 < R1*R1:
        return 0.0
    if r2 > R2*R2:
        return r2
    if abs(r*math.cos(theta)) < a:
        return r2
    if abs(r*math.sin(theta)) < a:
        return r2
    return 0.0
```

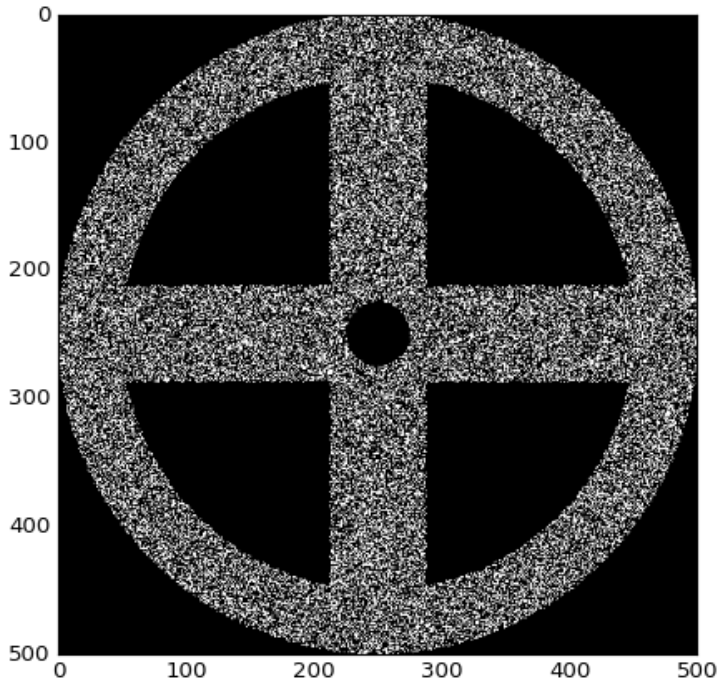
et la fonction d'intégration :

```
integration2d_disque(fonction,N,param,np)
```

```
def integration2d_disque(fonction,N,param,np):
    image = numpy.zeros((np,np))
    theta = 2*numpy.pi*numpy.random.random_sample(N)
    r = numpy.power(numpy.random.random_sample(N),0.5)
    somme = 0.0
    somme2 = 0.0
    p = 1.0/numpy.pi
    for i in range(N):
        f = fonction(r[i],theta[i],param)/p
        somme += f
        somme2 += f*f
        if f!=0:
            image[int((r[i]*math.cos(theta[i])+1)/2*np),int((r[i]*math.sin(theta[i])+
    moyenne = somme/N
    variance = somme2/N-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96,image)
```

Voici un calcul :

```
(integrale,intervalle,image) = integration2d_disque(f_Jz_polaire,N,param,500)
figure()
imshow(image,cmap='gray')
```



```
print((integrale,intervalle))
--> (1.1338844321238779, 0.007306451381173197)
```

L'écart est bien réduit par rapport à l'échantillonnage sur le domaine carré.

4.c. Échantillonnage non uniforme

Pour réduire encore la variance, on peut chercher à échantillonner sur le disque de rayon 1, avec une densité proportionnelle à $x^2 + y^2$, c'est-à-dire :

$$p(r, \theta) = \frac{2r^2}{\pi} \tag{13}$$

La fonction de répartition est :

$$F(r, \theta) = \frac{\theta}{2\pi} r^4 \tag{14}$$

On en déduit l'échantillonnage à partir de deux variables uniformes sur l'intervalle $[0, 1]$:

$$\theta = 2\pi u_1 \tag{15}$$

$$r = u_2^{\frac{1}{4}} \tag{16}$$

4.d. Réalisation

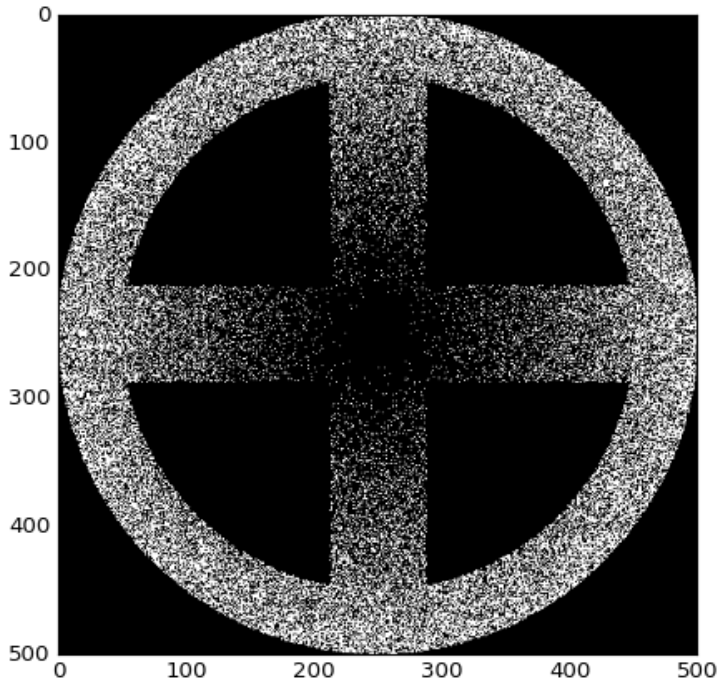
On écrit la fonction d'intégration :

```
integration2d_disque2(fonction,N,param,np)
```

```
def integration2d_disque2(fonction,N,param,np):
    image = numpy.zeros((np,np))
    theta = 2*numpy.pi*numpy.random.random_sample(N)
    r = numpy.power(numpy.random.random_sample(N),0.25)
    somme = 0.0
    somme2 = 0.0
    k = numpy.pi/2
    for i in range(N):
        f = fonction(r[i],theta[i],param)*k/(r[i]*r[i])
        somme += f
        somme2 += f*f
        if f!=0:
            image[int((r[i]*math.cos(theta[i])+1)/2*np),int((r[i]*math.sin(theta[i])+
    moyenne = somme/N
    variance = somme2/N-moyenne*moyenne
    return (moyenne,math.sqrt(variance/N)*1.96,image)
```

Voici un calcul :

```
(integrale,intervalle,image) = integration2d_disque2(f_Jz_polaire,N,param,500)
figure()
imshow(image,cmap='gray')
```

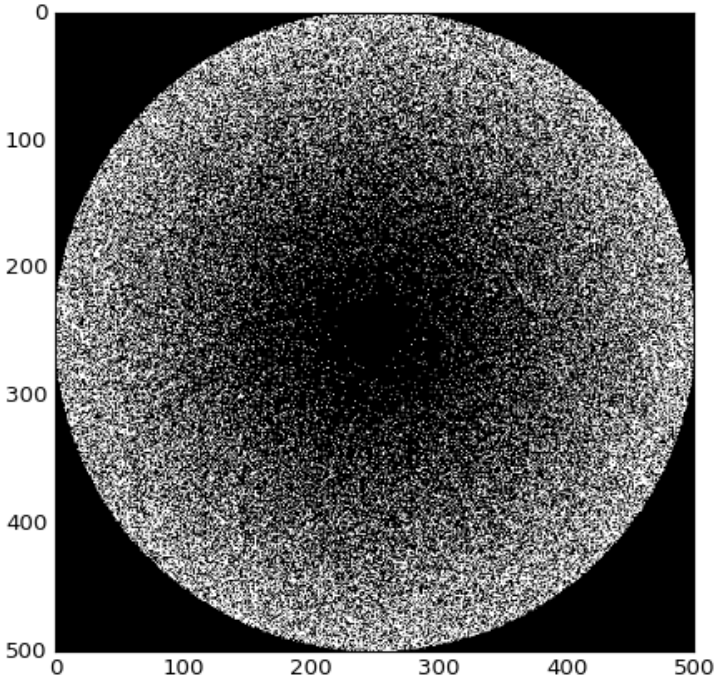



```
print((integrale,intervalle))
--> (1.1370994609655842, 0.004352596815941401)
```

Cet échantillonnage préférentiel permet de réduire la variance de manière importante. Il peut être utilisé pour différentes formes d'objets.

Il est intéressant de comparer avec le moment d'inertie d'une roue pleine (avec seulement le trou central) :

```
param = [0.1,0.1,0]
(integrale,intervalle,image) = integration2d_disque2(f_Jz_polaire,N,param,500)
figure()
imshow(image,cmap='gray')
```




```
print((integrale,intervalle))
--> (1.5706549551235363, 9.235866842716063e-05)
```

Dans ce cas la variance est très faible. En effet, un disque plein complet doit donner une variance théorique nulle avec cet échantillonnage. La contribution du trou central au moment d'inertie est très faible.

4.e. Distribution des intégrales évaluées

Obtenir un histogramme donnant la distribution des évaluations pour un ensemble de calculs faits avec un nombre N de tirages donné. Il faudra choisir judicieusement l'intervalle de valeurs et le nombre de sous-intervalles pour le calcul de cet histogramme. Pour des raisons de temps de calcul, la valeur de N sera moins grande que précédemment.