Année 2022 - 2023

RAPPORT DE PROJET

MÉTHODES NUMÉRIQUES

MOUVEMENT BROWNIEN

Projet réalisé par :
Dan NTAMBWE
Glen ROGER

Projet encadré par :
Hamid NESHATE

Sommaire

Introduction

Dans le cadre de notre premier semestre en première année du cycle ingénieurs à l'EIDD, il nous est proposé un projet de 4 mois nous permettant de mettre en pratique les méthodes numériques afin de résoudre des problèmes physiques et mathématiques.

Notre sujet se porte sur le mouvement brownien, qui est une description mathématique du mouvement aléatoire généralement d'une particule immergée dans un fluide et qui n'est soumis à aucune autre interaction que des chocs avec les molécules du fluide environnant.

La description physique la plus élémentaire du phénomène est la suivante :

- Entre deux chocs, la particule se déplace en ligne droite avec une vitesse constante.
- La particule est accélérée lorsqu'elle rencontre une molécule de fluide ou une paroi.

Cette dynamique est appelée mouvement brownien en l'honneur du botaniste Robert Brown qui a remarqué une agitation des grains de pollen au microscope.

En pratique, le mouvement Brownien est un cas particulier d'une discipline physique bien plus vaste appelée la théorie du transport. Cette discipline consiste à résoudre mathématiquement ce que l'on appelle des équations de diffusions afin d'étudier quantitativement l'évolution de différentes distributions initiales d'objets dont l'approximation au continu est possible.

Abstract

Afin d'étudier le phénomène du mouvement brownien, nous allons utiliser la simulation de Monte Carlo qui est une méthode d'estimation d'une quantité numérique qui utilise des nombres aléatoires. Le principe de la simulation est d'utiliser un modèle et d'étudier l'évolution de ce modèle sans faire fonctionner le système réel. Pour cela, nous sommes donc partis d'une loi de probabilité normal :

$$p(x) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

, avec $\mu : moyenne$ et $\sigma : variance$

Le physicien Albert Einstein, établit que le « déplacement moyen des particules » sur un intervalle de temps de longueur t est égal à $D * t$ où D est appelé constante de diffusion. La formule d'Einstein donne la valeur de la constante de diffusion D en termes de plusieurs paramètres :

$$D = \frac{k_B T}{3\pi\eta d}$$

$T : la\ température\ en\ Kelvin$
$k_B : la\ constant\ de\ Boltzmann$
$\eta : la\ viscosité\ du\ fluide$
$d : le\ diamètre\ de\ la\ particule$

Dans cette étude nous allons utiliser le langage C pour résoudre les différents problèmes et le langage Python pour tracer les graphiques.

I. Implémentation d'une loi normale

Comme marqué précédemment dans l'*abstract*, nous sommes donc partis d'une loi de probabilité normal :

$$p(x) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}} \quad \text{, avec} \quad \mu : moyenne \quad \text{et} \quad \sigma : variance$$

Pour résoudre cette équation numériquement, nous devons calculer la fonction inverse en partant de la fonction de répartition suivante :

$$F(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2\pi}}\right)\right] \quad \text{, avec} \quad \text{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2} dt$$

Malheureusement cette fonction de répartition n'est pas facile à résoudre et ainsi à inverser.

On a donc vu dans le cours qu'il existe des méthodes qui combinent plusieurs tirages simples, c'est le cas de la méthode de Box-Muller qui consiste à :

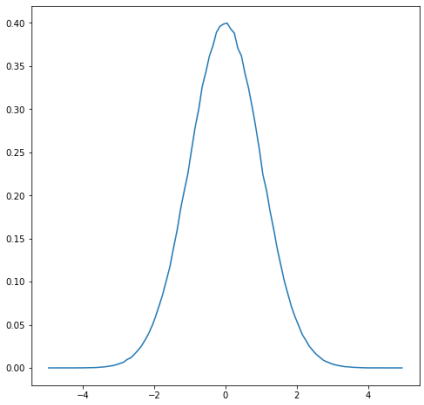
- Tirer des u et v uniformes sur [0,1]
- Calculer à chaque fois $x = \sqrt{-2\ln(u)}\cos(2\pi v)$

```c
double uniform(){
    double num = (double)rand()/(double)RAND_MAX;
    return num;
}

/* generate a random value weighted within the normal (gaussian) distribution */
double gauss(){
    double u = uniform();
    double v = uniform();
    double x = sqrtf(-2 * log(u)) * cos(2 * M_PI * v);
    return x;
}
```

Code en langage C pour une loi normale suivant distribution uniforme entre 0 et 1

Résultat :

Nous avons donc bien une courbe normale avec une forme dite « cloche » malheureusement, on a ici une distribution normal centrée réduite c'est-à-dire avec une moyenne égale à 0 et une variance égale à 1. Pour avoir un résultat généralisé de cette distribution, il faudra donc convertir notre résultat en distribution normal « non-standard » à l'aide des étapes suivantes :



Graphique d'une loi normale

- On pose Z, une variable aléatoire suivant une loi normale centrée réduite.
- On pose X, une variable aléatoire suivant une loi normale non centrée et non réduite.
- En utilisant la relation :

$$Z = \frac{X - \mu}{\sqrt{\sigma}}$$

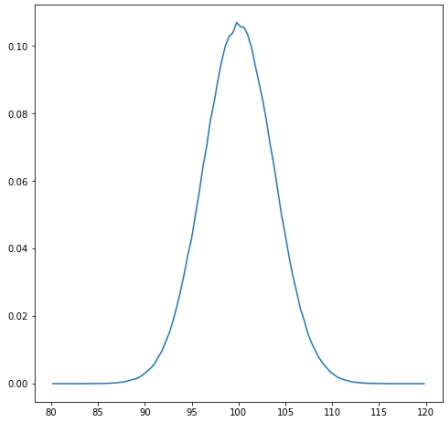
On obtient une loi centrée réduite. Alors que dans notre cas, on connait déjà Z à partir de la méthode de la fonction inverse, ainsi :

$$X = Z\sqrt{\sigma} + \mu$$

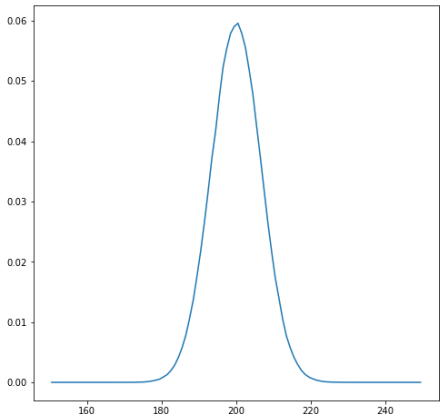
```
/* convert non-standard normal distribution to standard */
double convert(double m, double sig){
    return gauss()*sqrtf(sig)+m;
}
```

Code de conversion d'une loi normale centrée réduite en une loi normale non-centrée et non-réduite

Résultats :



Loi normale de moyenne 100 et de variance 14



Loi normale de moyenne 200 et de variance 45

On a donc bien une représentation d'une loi normale pour une moyenne et une variance quelconque qui montre que le programme fonctionne. On a maintenant une variable aléatoire suivant une loi normal, on passe donc au mettre à la simulation du mouvement brownien en 1D, 2D et 3D.

II. Simulation d'un mouvement brownien en plusieurs dimension

Dans toute cette partie, on a un nombre de point $N = 10^3$*, un temps* $t_{max} = 100\ s$ *et une moyenne pour la loi normale* $m = 0$.

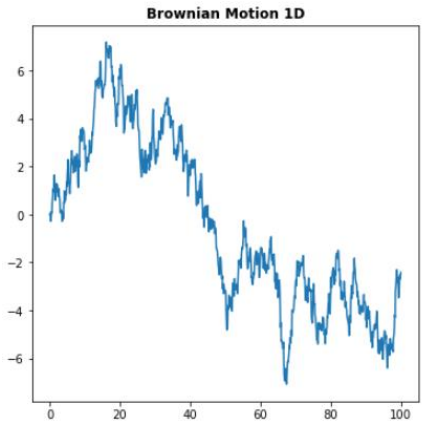
En une dimension la résolution du problème se fait facilement car on suit la trajectoire d'une particule de fluide suivant l'évolution du temps. Pour implémenter cette situation, il faut donc prendre en compte un pas de temps nommé *epsilon*. On se base donc il par une marche aléatoire qui se fait lorsqu'on calcule la position de la particule à chaque temps t grâce à la bibliothèque *random*.

```
double* brownian1D(int N, double m, double tmax){
    FILE *f;
    f=fopen("brownian1D.txt","w");                    //écriture dans un fichier
    double epsilon = tmax/(double)(N-1);              //pas de temps
    double tab[N];
    tab[0] = 0;                                       //initialise le temps à t=0
    double X[N];
    X[0]=0;                                           //condition initiale car la particule part de la position x=0
    fprintf(f,"%f %f\n",tab[0],X[0]);
    for(int i=1; i<N; i++){
        tab[i] = tab[0] + epsilon * i;                //incrémentation du temps
        X[i] = X[i-1] + convert(0,tab[i]-tab[i-1]);   //la position suivante dépend de la dernière position
        fprintf(f,"%f %f\n",tab[i],X[i]);
    }
    fclose(f);
}
```

Calcul des positions d'une particule de fluide en fonction du temps en 1D

```
import numpy as np
import matplotlib.pyplot as plt
import math

fig = plt.figure(figsize=(6,6))
data = np.loadtxt('./brownian1D.txt')
T = data[:,0]
X = data[:,1]
plt.plot(T,X)
plt.title('Brownian motion 1D',fontweight = 'bold')
plt.show()
```

Code pour la représentation graphique du mouvement brownien en 1D



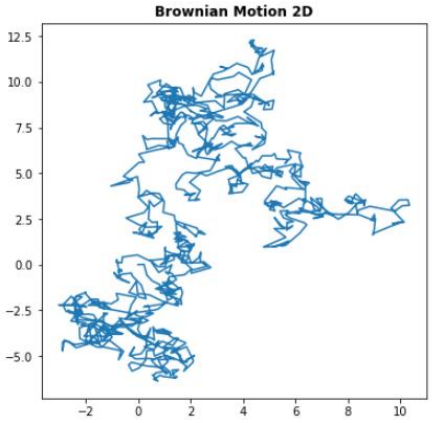
En deux dimensions pour implémenter le mouvement brownien, il faut deux axes, x et y qui seront implémenter en fonction du temps comme dans le code précédent. Ce qui est différent c'est lors de la représentation car on n'affiche pas en fonction du temps mais plutôt de x en fonction de y.

```
double* brownian2D(int N, double m, double tmax){
    FILE *f;
    f=fopen("brownian2D.txt","w");
    double epsilon = tmax/(double)(N-1);
    double tab[N];
    tab[0] = 0;
    double X[N], Y[N];
    X[0]=0;                                           //condition initiale sur x
    Y[0]=0;                                           //condition initiale sur y
    fprintf(f,"%f %f\n",X[0],Y[0]);
    for(int i=1; i<N; i++){
        tab[i] = tab[0] + epsilon * i;
        X[i] = X[i-1] + convert(0,tab[i]-tab[i-1]);
        Y[i] = Y[i-1] + convert(0,tab[i]-tab[i-1]);
        fprintf(f,"%f %f\n",X[i],Y[i]);
    }
    fclose(f);
}
```

Calcul des positions d'une particule de fluide en fonction du temps en 2D

```python
import numpy as np
import matplotlib.pyplot as plt
import math

fig = plt.figure(figsize=(6,6))
data = np.loadtxt('./brownian2D.txt')
X = data[:,0]
Y = data[:,1]
plt.plot(X,Y)
plt.title('Brownian Motion 2D',fontweight = 'bold')
plt.show()
```

Code pour la représentation graphique du mouvement brownien en 2D



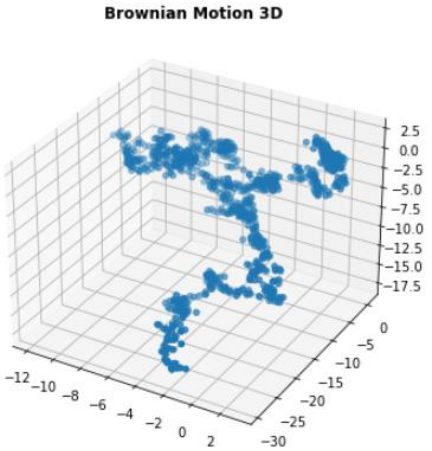
En trois dimensions on fait exactement la même chose qu'en deux dimensions mais en ajoutant un axe z.

```c
double* brownian3D(int N, double m, double tmax){
    FILE *f;
    f=fopen("brownian3D.txt","w");
    double epsilon = tmax/(double)(N-1);
    double tab[N];
    tab[0] = 0;
    double X[N], Y[N], Z[N];
    X[0]=0;                                          //condition initiale sur x
    Y[0]=0;                                          //condition initiale sur y
    Z[0]=0;                                          //condition initiale sur z
    fprintf(f,"%f %f %f\n",X[0],Y[0],Z[0]);
    for(int i=1; i<N; i++){
        tab[i] = tab[0] + epsilon * i;
        X[i] = X[i-1] + convert(0,tab[i]-tab[i-1]);
        Y[i] = Y[i-1] + convert(0,tab[i]-tab[i-1]);
        Z[i] = Z[i-1] + convert(0,tab[i]-tab[i-1]);
        fprintf(f,"%f %f %F\n",X[i],Y[i],Z[i]);
    }
    fclose(f);
}
```

Calcul des positions d'une particule de fluide en fonction du temps en 3D

```python
import numpy as np
import matplotlib.pyplot as plt
import math

fig = plt.figure()
fig = plt.figure(figsize=(6,6))
data = np.loadtxt('./brownian3D.txt')
X = data[:,0]
Y = data[:,1]
Z = data[:,2]
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X,Y,Z,s=20)
plt.title('Brownian Motion 3D',fontweight = 'bold')
plt.show()
```

Code pour la représentation graphique du mouvement brownien en 3D



III. Etude de la diffusion de l'encre

L'encre diffuse dans l'eau à cause du mouvement aléatoire de l'eau et des molécules d'encre. Sur une grande échelle, nous ne voyons pas les molécules individuelles en mouvement. Au lieu de cela, nous voyons à quel point l'encre est sombre à différents points de la solution, ce qui indique en fait sa concentration.

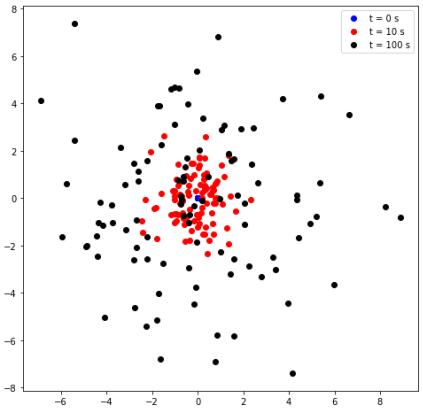
Pour la simulation de ce phénomène, on a considéré 100 particules de fluide qui se propagent aléatoirement selon une loi normale et représenter par un mouvement brownien. Pour montrer la direction que prend les particules, on regarde à différents temps selon le code ci-dessous :

```
double ink(int N, double m, double tmax){
    FILE* t0=fopen("t0.txt","w");                                   //création d'un fichier à t=0
    FILE* t10=fopen("t10.txt","w");                                 //création d'un fichier à t=10
    FILE* t100=fopen("t100.txt","w");                               //création d'un fichier à t=100
    double epsilon = tmax/(double)(N-1);
    double tab[N];
    double X[N], Y[N], distance[N];
    double dist = 0;
    double q = 0;
    for(int j=0; j<100; j++){
        X[0]=0;                                                     //condition initiale sur x
        Y[0]=0;                                                     //condition initiale sur y
        tab[0] = 0;
        distance[0]=0;                                              //condition initiale sur la distance
        fprintf(t0,"%f %f\n",X[0],Y[0]);
        fprintf(quadra,"%f %f %f\n",X[0],Y[0],distance[0]);
        for(int i=1; i<N+1; i++){
            tab[i] = tab[0] + epsilon * i;
            X[i] = X[i-1] + convert(0,tab[i]-tab[i-1]);
            Y[i] = Y[i-1] + convert(0,tab[i]-tab[i-1]);
            distance[i] = sqrt(pow(X[i]-X[i-1],2)+pow(Y[i]-Y[i-1],2));      //calcule la distance de la particule i par rapport sa position précédente
            q += pow(distance[i]-distance[i-1],2);                          //numérateur de la distance quadratique moyenne
        }
        fprintf(t10,"%f %f\n",X[10],Y[10]);
        fprintf(t100,"%f %f\n",X[100],Y[100]);
        dist += sqrt(pow(X[N]-X[0],2)+pow(Y[N]-Y[0],2));                    //somme de la distance de la dernière particule par rapport à sa position initiale
    }
    fclose(t0);
    fclose(t10);
    fclose(t100);
    printf("%f\n",dist/N);                                          //distance moyenne
    printf("%f\n",q/N);                                             //distance quadratique moyenne
    return q/N;
}
```

Code pour calculer la position de chaque particule à t = 0 s, t = 10 s et t = 100 s

```
import numpy as np
import matplotlib.pyplot as plt
import math

fig = plt.figure(figsize=(8,8))
data = np.loadtxt('./t0.txt')
data10 = np.loadtxt('./t10.txt')
data100 = np.loadtxt('./t100.txt')
plt.scatter(data[:,0],data[:,1],color='blue',label='t = 0 s')
plt.scatter(data10[:,0],data10[:,1],color='red',label='t = 10 s')
plt.scatter(data100[:,0],data100[:,1],color='black',label='t = 100 s')
plt.legend()
plt.show()
```

Code pour la représentation graphique de la diffusion de l'encre



Représentation graphique de la diffusion de l'encre

Pour cette représentation avec les valeurs qu'on a obtenu à partir de notre simulation, on trouve **une distance moyenne, au bout de 100 s, d'environ 15,8 mm**.

Pour avoir la distance quadratique moyenne, il faut utiliser la formule suivante :

$$MSD(\tau) = < (x(t + \tau) - x(t))^2 >$$

, avec τ : $le\ pas\ de\ temps$ et MSD : $mean\ squared\ displacement$

Les parties du code de la fonction *ink* qui traite la distance moyenne quadratique sont :

```
        distance[i] = sqrt(pow(X[i]-X[i-1],2)+pow(Y[i]-Y[i-1],2));      //calcule la distance de la particule i par rapport sa position précédente
        q += pow(distance[i]-distance[i-1],2);                          //numérateur de la distance quadratique moyenne
```

```
    printf("%f\n",q/N);                                                 //distance quadratique moyenne
    return q/N;
```

Pour donner donc **une distance moyenne quadratique d'environ 85,8 mm²**.

Selon la théorie, le déplacement carré moyen de la particule est proportionnel à l'intervalle de temps, selon la relation :

$$< (x(t + \tau) - x(t))^2 > = 2dD\tau$$

$x(t)$: $la\ position$ à t
d : $nombre\ de\ dimensions$
D : $coefficient\ de\ diffusion$
τ : $pas\ de\ temps$

On peut ainsi avoir le coefficient de diffusion avec :

$$D = \frac{MSD(\tau)}{2d\tau}$$

Comme on sait qu'on se place en deux dimensions et $\tau = \frac{100}{10^3} = 0.1\ s$.

On a ainsi :

$$D = \frac{85.8}{2 * 2 * 0.1} = 214.5\ mm^2/s = 2.1 * 10^{-4}\ m^2/s$$

IV. La probabilité d'échappement d'une sphère

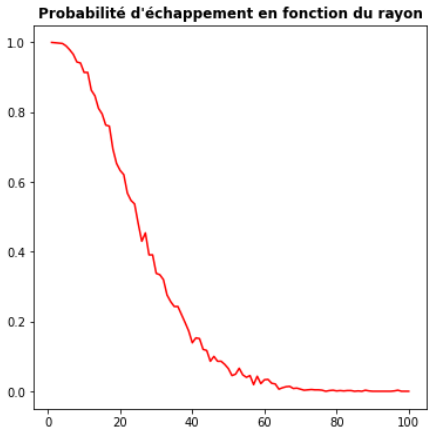
Pour cette étude, nous sommes partis du fait de prendre l'aire d'une sphère. Ce qu'on cherche à calculer c'est la probabilité à la quelle une particule, après un certain temps, sorte de la sphère de **rayon r variable**. Pour cela nous avons calculer la probabilité pour 1000 particules partant d'une coordonnée (0,0,0).
Ce calcule se fait en trois dimensions en calculant à chaque fois la distance entre la position après N-itérations et la position de départ. Il faut ensuite compter le nombre de particules qui sort de la sphère de rayon r pour connaitre la probabilité de particule à l'extérieur de la sphère comme ce code le suggère :

```c
double echappement(int N, double rayon, int tmax, int fois){
    double epsilon = tmax/(double)(N-1);
    double tab[N], X[N], Y[N], Z[N];                                    //on se place en 3D avec x,y,z
    tab[0] = 0;
    X[0]=0;
    Y[0]=0;
    Z[0]=0;
    double p_ext = 0;
    double p_tot = 0;
    double distance = 0;
    for(int j=1; j<fois+1; j++){                                        //on a ici 1000 particules
        for(int i=1; i<N+1; i++){
            tab[i] = tab[0] + epsilon * i;
            X[i] = X[i-1] + convert(0,tab[i]-tab[i-1]);
            Y[i] = Y[i-1] + convert(0,tab[i]-tab[i-1]);
            Z[i] = Z[i-1] + convert(0,tab[i]-tab[i-1]);
        }
        distance = sqrt(pow(X[N]-X[0],2)+pow(Y[N]-Y[0],2)+pow(Z[N]-Z[0],2));        //calcule de la distance après N-itérations
        if(distance>rayon){                                             //regarde si la particule est à l'extérieure du sphère
            p_ext++;                                                    //compte le nombre de particule à l'extérieur
        }
    }
    return p_ext/(double)fois;
}

void plot_echap(int N, double rayon){
    FILE* fich = fopen("echappement.txt","w");
    double p_ext = 0;
    for(int i=1;i<rayon+1;i++){
        p_ext = echappement(N,i,100,1000);
        fprintf(fich,"%i %f\n",i,p_ext);
    }
    fclose(fich);
}
```

Code pour calculer la probabilité d'échappement

```python
import numpy as np
import matplotlib.pyplot as plt
import math

fig = plt.figure(figsize=(6,6))
data = np.loadtxt('./echappement.txt')
plt.plot(data[:,0],data[:,1],color='r')
plt.title("Probabilité d'échappement en fonction du rayon",fontweight = 'bold')
plt.show()
```

Code pour la représentation graphique de la probabilité d'échappement



Cette courbe montre que lorsqu'on à un rayon très petit, la probabilité tend vers 1 car toutes les particules se situe à l'extérieur de la sphère après N-itérations et que lorsque le rayon est grand, la probabilité tend vers 0 car toutes les particules sont à l'intérieur de la sphère.

Conclusion

En conclusion, on a pu simuler un mouvement brownien en 1D, 2D et 3D à partir de la fonction de répartition d'une loi normale de probabilité. En étudiant la diffusion de l'encre dans un de l'eau, on a pu constater que le coefficient de diffusion de l'encre n'est pas en accord avec l'ordre de grandeur donné dans les documents. Cette différence peut être dû à la température que l'on ne prend pas en compte. En général, nos résultats présentent bien un mouvement brownien issue d'une marche aléatoire qui est bien montré par la simulation de la probabilité d'échappement d'une sphère.

Annexe

Bibliographie

Sites internet

Simulating brownian motion
https://www.epfl.ch/labs/lben/wp-content/uploads/2018/07/Simulating-Brownian-Motion.pdf

Méthode de Monte Carlo pour la simulation du mouvement brownien
https://www.mathenjeans.fr/sites/default/files/comptes-rendus/methode_de_monte_carlo_mouvement_brownien_perigueux_2020.pdf

Standardisation d'une loi normale
http://www.hec.unil.ch/amattei/EXOIDS/groupe-2/stdtion%20loi%20N.pdf

Institut de France – Académie des sciences
Mouvement brownien et marche au hasard, conférence de Jean-François Le Gall devant les lycéens
www.academie-sciences.fr/fr/Seances-publiques/mouvement-brownien-marche-au-hasard.html

Document

Real applications of normal distribution
Envoyé par M. Hamid NESHATE

Cours

```c
void histogram(int N, double x[N], double xmin, double xmax, int nbin){
    int i,bin;
    FILE *f;
    f=fopen("test.txt","w");
    double hist[nbin];
    double dx = (xmax-xmin)/nbin;
    // initialisation de l'histogramme a 0
    for(bin=0;bin<nbin;bin++){hist[bin]=0.0;}
    // on ajoute chacune des N valeur de x dans le bonn nbin de l'histogramme
    for(i=0;i<N;i++){
        if (x[i]<xmax && x[i]>xmin){
            bin = (int) ((x[i]-xmin)/dx) ;
            hist[bin] += 1.0/N/dx ;
        }
    }
    // on ecrit les valeurs de l'histogram dans un fichier
    for(bin=0;bin<nbin;bin++){
        fprintf(f,"%13.6e %13.6e \n",xmin+dx*(bin+0.5),hist[bin]);
    }
    fclose(f);
}
```