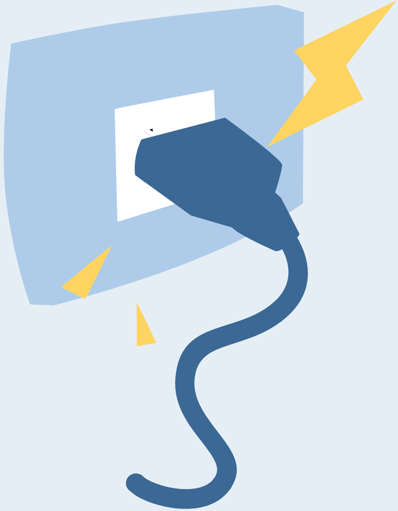
Mouvement Brownien

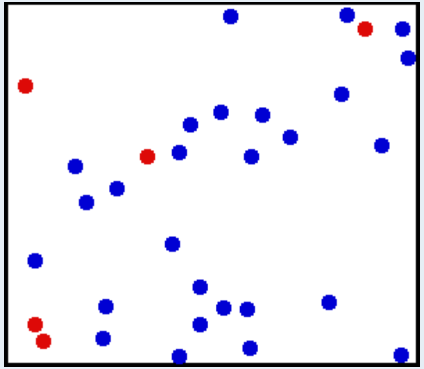
Dan NTAMBWE MAKEPA

Glen ROGER

ÉCOLE D'INGÉNIEUR DENIS DIDEROT

Introduction





Déplacement des particules

Robert Brown
(1849 – 1853)

Plan

01
Implémentation d’une loi normale

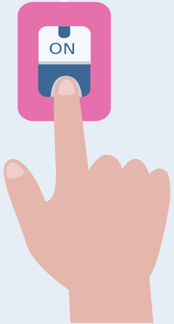
02
Simulation du mouvement brownien

03
Etude de la diffusion de l’encre

04
La probabilité d’échappement d’une sphère

01

Implémentation d’une loi normale



Loi de probabilité

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

Fonction de répartition

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

Fonction inverse

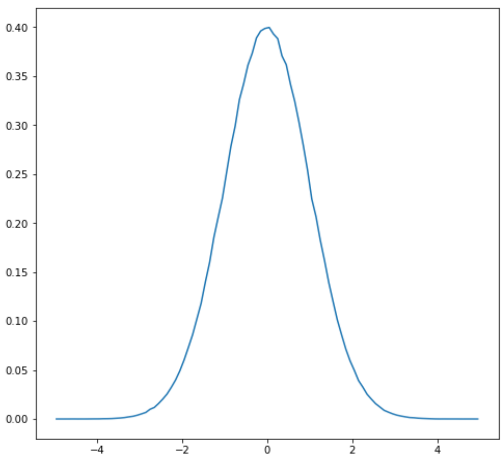
Méthode de Box-Muller

- 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



Graphique d'une loi normale

Interprétation

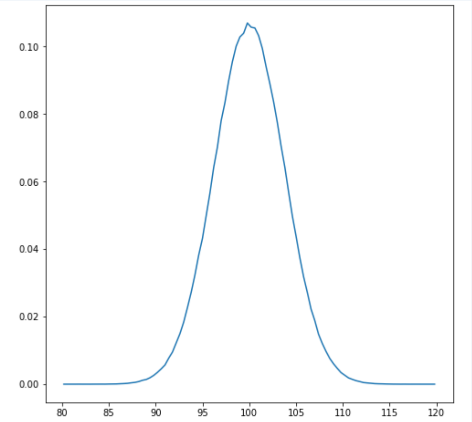
- Courbe normale avec une forme dite « cloche ».
- Distribution normale centrée réduite:
  - Moyenne = 0
  - Variance = 1
- Convertir en une loi normale non centrée et non réduite :

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

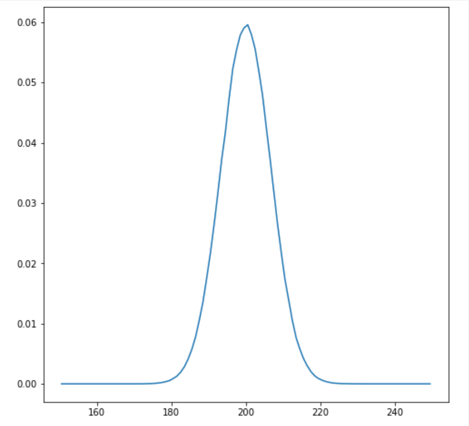
```c
/* convert non-standard normal distribution to standard */
double convert(double m, double sig){
    return gauss()*sqrtf(sig)+m;
}
```

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

Interprétation

- Loi normale pour une moyenne et une variance quelconque.

02

Simulation du mouvement brownien

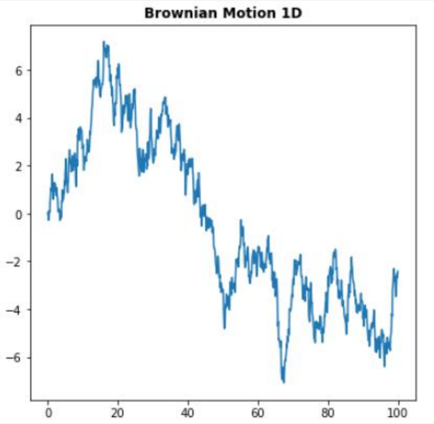
En 1D :

```c
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

```python
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



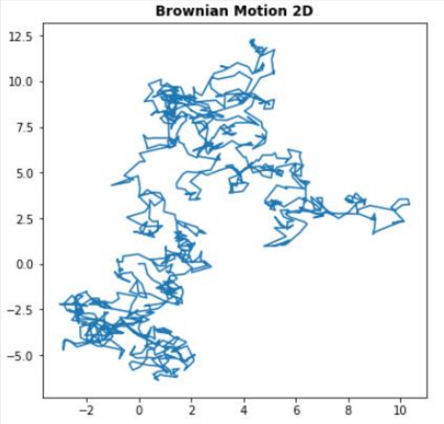
En 2D :

```c
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



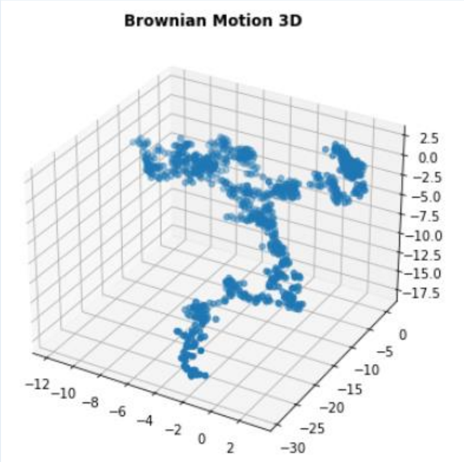
En 3D :

```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

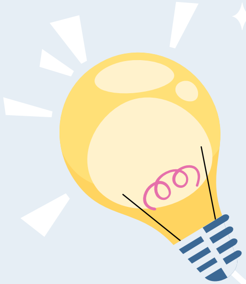
```python
import numpy as np
import matplotlib.pyplot as plt
import math

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()
```

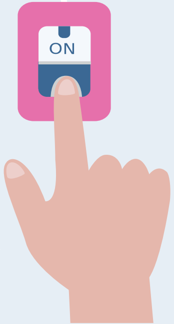
Code pour la représentation graphique du mouvement brownien en 3D



03

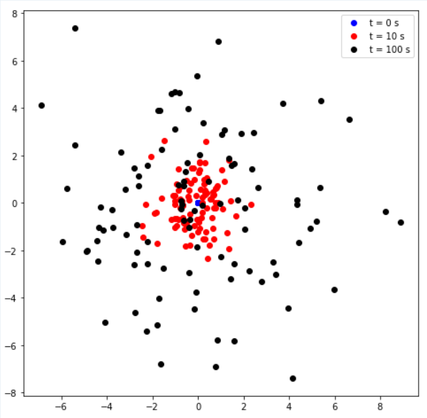
Etude de
la diffusion de l’encre





```c
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



Représentation graphique de la diffusion de l'encre

```python
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

Résultat du code :

- Après 100 secondes pour 100 particules :
  distance moyenne ≈ 15,8 mm

Distance moyenne quadratique

$MSD(\tau) = <(x(t+\tau) - x(t))^2>$, avec τ : *le pas de temps* et MSD : *mean squared displacement*

De plus :

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

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

On a donc :

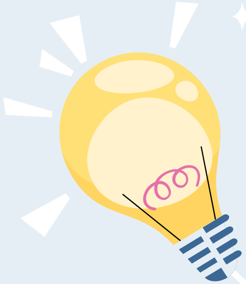
$$D = \frac{85.8}{2 * 2 * 0.1} = 214.5\ mm^2/s = 2.1 * 10^{-4}\ m^2/s$$

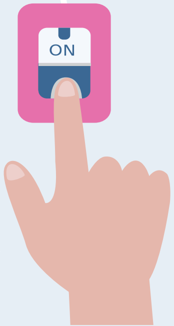
Car :

on sait qu'on se place en deux dimensions $d = 2$ et $\tau = \frac{100}{10^3} = 0.1\ s$.

04

Probabilité d’échappement d’une sphère



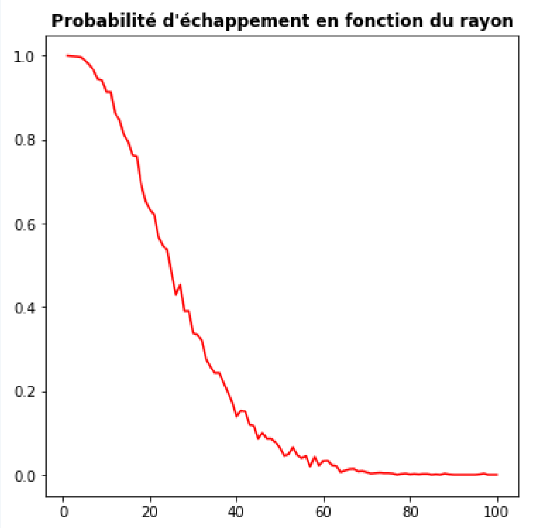


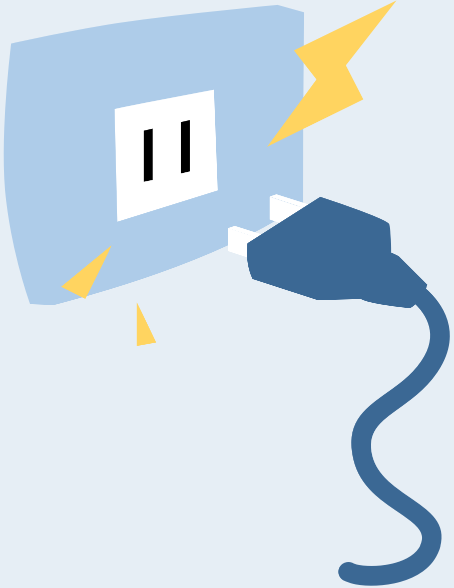
```c
double echappement(int N, double rayon, int tmax, int fois){
    double epsilon = tmax/(double)(N-1);
    double tab[N], X[N], Y[N], Z[N];                                          //on se place en 3D avec x,y,z
    tab[0] = 0;
    X[0]=0;
    Y[0]=0;
    Z[0]=0;
    double p_ext = 0;
    double 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





Conclusion

Merci de votre attention !