Méthode de Monte Carlo pour la simulation du mouvement brownien

Atelier MATH.en.JEANS, Lycée A. Claveille année 2019-2020

Enseignant chercheur : MR Vincent COUALLIER

Elèves :

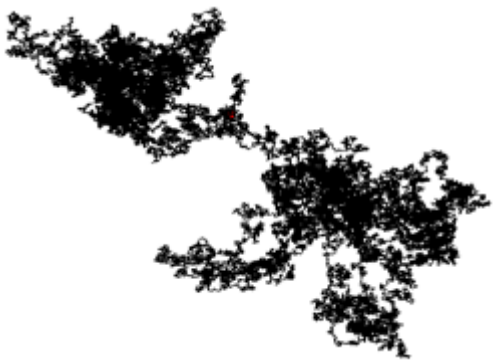
TS : Beney Dorian, Bernard Mathéo, Bernard Paul, Bodin Louise, Brambatti Gwenael, Dauphin Yohann, De Sousa Germain, Devaux Lisa, Triquet Maël

1G : Jayat Adrien, Laffez Julien, Marty Thomas

Enseignant : P. Lefebvre

1- Un peu d'histoire

Le mouvement brownien n'est ni un groupe de mangeurs de Brownies, ni le fan club de James Brown . C'est un phénomène naturel, et un objet mathématique.
Le phénomène naturel a été observé par Richard Brown. Ce grand botaniste écossais du début du 19ème siècle, qui s'intéressait à l'action du pollen dans la reproduction des plantes, a été amené, comme d'autres, à observer le mouvement erratique de fines particules de pollen en suspension dans l'eau ou dans un gaz.
En 1901, Louis Bachelier propose un premier modèle mathématique du mouvement brownien et l'applique à la finance.
Puis les physiciens pressentent que le mouvement des particules peut tenir à l'agitation moléculaire. En 1905, Einstein veut tester la théorie cinétique moléculaire de la chaleur dans les liquides. Il aboutit à une formule qui permet, à partir de l'observation du mouvement brownien, de calculer le nombre d'Avogadro. Puis Jean Perrin s'en inspire pour présenter sa théorie de l'Atome.
Norbert Wiener définit enfin une construction mathématique rigoureuse du mouvement brownien : il bâtit, à partir du modèle physique de trajectoires continues avec des vitesses infinies en chaque point, une courbe continue sans tangence.



2- La méthode de Monte Carlo

Une méthode de Monte Carlo est une méthode algorithmique qui explore aléatoirement les états possibles d'une grandeur pour la mesurer. C'est une technique probabiliste. Cette méthode, qui fait allusion aux jeux de hasard pratiqués au casino de Monte-Carlo, a été inventée en 1947 par Nicholas Metropolis.

Illustration par le calcul de l'intégrale $I = \int_0^1 f(t)dt$, où f est une fonction intégrable de [0 ; 1] dans $\mathbb{R}$.

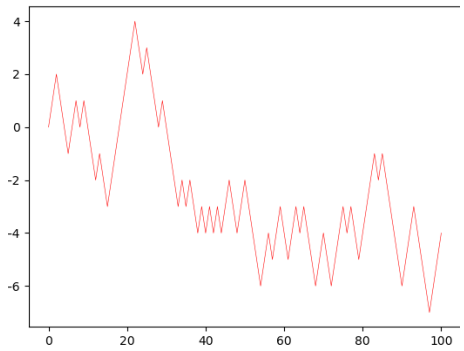
```
from random import random

def f(x):
    return x**3

S=0
N=int(input("Nombre de points ?"))
for i in range(N):
    S=S+f(random())
print (S/N)
```

On s'intéresse dans cet exemple à la fonction cube.
On simule N variables aléatoires indépendantes de loi uniforme sur [0,1].
L'estimateur de Monte Carlo $I_N = \frac{1}{N}\sum_{i=1}^{N} f(U_i)$ donne une estimation de la valeur moyenne de f .
La loi des grands nombres assure qu'avec une probabilité de 1,

$$lim_{N\to+\infty} \quad \sum_{i=1}^{N} f(U_i) \quad = \quad \int_0^1 f(t)dt$$

3- Approche du mouvement aléatoire

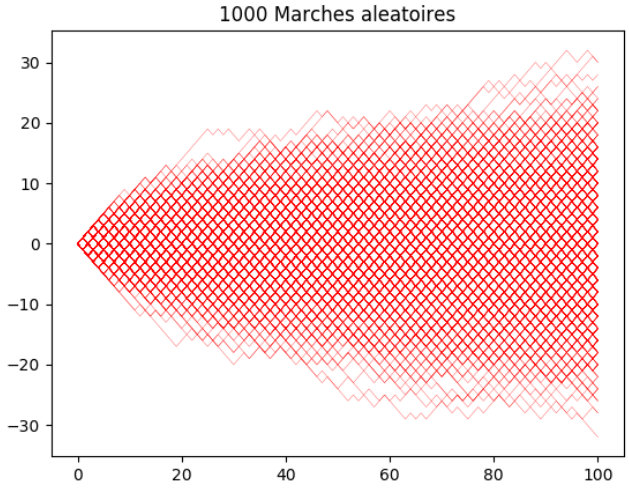
a- *Marche aléatoire*



On s'intéresse à une particule qui se déplace sur un axe gradué aléatoirement à gauche ou à droite avec la même probabilité, d'un pas constant égal à 1.
Elle part de l'origine et effectue 100 pas.
On s'intéresse à sa position en fonction du nombre de pas effectué.
La position finale se situe entre -100 et +100. On obtient par exemple la simulation ci-contre :

Pour avoir une idée de la valeur moyenne de la position finale, nous pouvons ensuite simuler 1000 marches aléatoires de 100 pas chacune (méthode de Monte Carlo).

```
from tkinter import *
import numpy as np
from math import *
import matplotlib.pyplot as plt
from random import *

nbMarches, nbPas = 1000, 100
x = np.linspace(0, 100, num=nbPas+1)  #fonction qui génére 100 points à intervalle régulier
positions, frequences = [], []
plt.title('1000 Marches aleatoires')

for i in range(nbMarches):
    y = [0] #intilialise la liste des y (non vide pour permettre l'accès à la dernière valeur)
    for i in range(nbPas):
        y.append(y[-1]+choice([-1,1]))  #ajoute à la liste des y la pos précédante + le mvt au hasard
    plt.plot(x, y, 'red', lw=0.2)
    positions.append(y[-1]) #j'incrémente mon nb de l'issue y[-1] (paire donc (y[-1]+nbPas)//2 pour commencer à 0

plt.show()
```

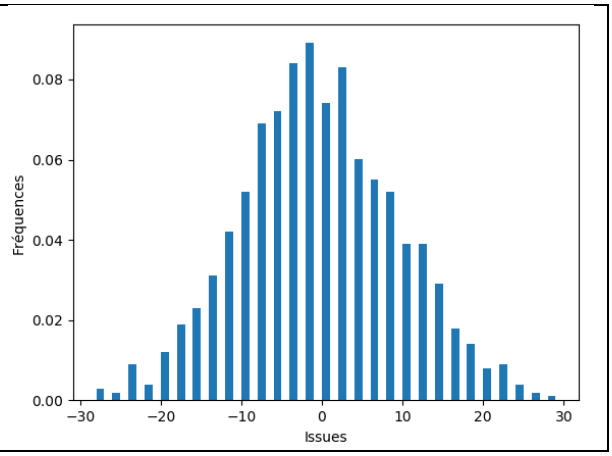


Les positions finales semblent se concentrer autour de 0, mais de quelle manière ?

b- Distribution des positions finales

Nous avons ensuite étudié la distribution des fréquences des positions finales associées aux 1000 marches aléatoires.

```
from random import random
from math import log
from matplotlib import pyplot as plt
import numpy as np

I=[]
N=100
for i in range(1000):
    S=0
    for i in range(N):
        if random()>0.5:
            S=S+1
        else:
            S=S-1
    I.append(S)

#plt.hist(I, bins='sturges') #sturges correspond à log2(n)+1
#plt.ylabel('Occurrences')

###option 2 avec vrais fréquences###
plt.hist(I, bins=range(min(I),max(I)), normed=1)
plt.ylabel('Fréquences')
plt.xlabel('Issues')
plt.show()
```



On observe une distribution gaussienne de ces positions finales, centrée sur 0. Intéressons-nous à son écart-type.

c- Observation de l'écart-type

Nous allons utiliser la formule de König-Huygens $\sqrt{E(X^2) - E(X)^2}$

```
from random import random
from matplotlib import pyplot as plt

def marches_aleatoires(n):
    I=[]
    for i in range(n):
        S=0
        for i in range(900):
            if random()>0.5:
                S=S+1
            else:
                S=S-1
        I.append(S)
    return(I)

from math import sqrt
def moyenneEcartType(listeVal):
    nombre = 0
    sommeValeurs = 0.0
    sommeCarres = 0.0
    for x in listeVal:
        nombre += 1
        sommeValeurs += x
        sommeCarres += x * x
    moyenne = sommeValeurs / nombre
    ecartt = sqrt(sommeCarres / nombre - moyenne * moyenne)
    return (moyenne, ecartt)
```

Nous allons nous intéresser à l'influence du nombre de pas de chaque marche aléatoire, sur la valeur de l'écart-type

Nombre Pas	100	900	10000
Ecart-type	10.84	29.34	99.95

On peut conjecturer que l' écart-type est égal à $\sqrt{nombre\ de\ pas}$.

d- Conclusion

La distribution des positions finales d'une marche aléatoire de N pas, suit donc une distribution normale de moyenne 0 et d'écart-type $\sqrt{N}$. Nous allons par la suite remplacer ces N pas par une durée T : on aura donc une distribution normale de moyenne 0 et d'écart-type $\sqrt{T}$

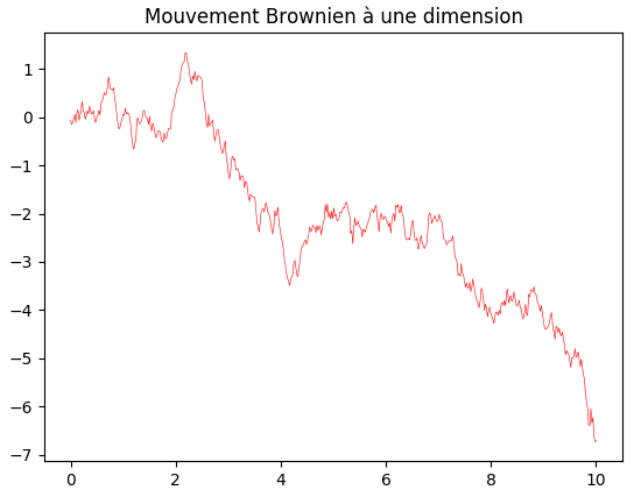
4- Simulation du mouvement brownien

a- Mouvement brownien unidimensionnel

Nous allons utiliser la méthode qui fait intervenir la marche aléatoire (théorème de Donsker) .
Le mouvement brownien en dimension 1 est un processus stochastique $(W_t)_{t\geq 0}$, à temps et trajectoires continus tel que :

- $W_0 = 0$
- Tout accroissement entre 2 temps t_1 et t_2, suit une loi gaussienne centrée de variance $t_2 - t_1$
- Les accroissements sont indépendants

```
import numpy as np       #génération des échantillons
import matplotlib.pyplot as plt      #creation de graphiques
from math import (log, sqrt)
T=int(input('Durée '))
N = int(input('Nombre de pas: '))
ecart_type = sqrt(T/N)       #propriété imposée
Y=[0]
Y = ecart_type*np.random.randn(N)
Y = np.cumsum(Y)
X=[]
for i in np.linspace(0,T,N):
    X.append(i)


plt.title('Mouvement Brownien à une dimension')
plt.plot(X, Y, color='red', lw=0.4)
plt.axis('on')
plt.show() #on affiche tous les graphiques
```

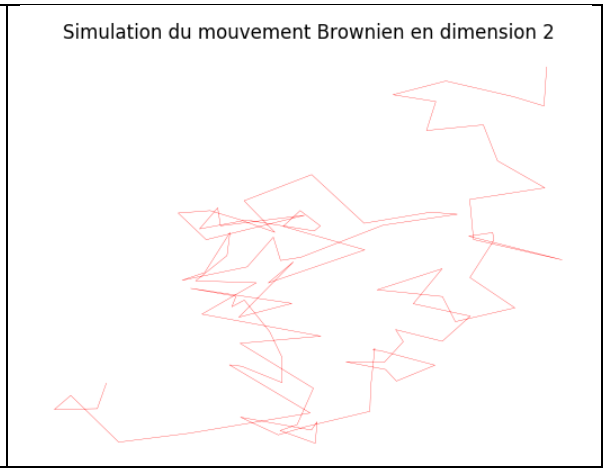


Cette représentation est trompeuse car le programme fait une interpolation linéaire entre 2 temps. On a donc une représentation approchée du mouvement Brownien.

La qualité de cette représentation est meilleure lorsque N, le nombre d'étapes, est plus grand

b- Mouvement brownien à 2 dimensions

```
import numpy as np       #génération des échantillons
import matplotlib.pyplot as plt      #creation de graphiques
from math import (log, sqrt)

nbPas = int(input('Nombre d''étapes: '))
ecart_type = sqrt(nbPas)       #propriété imposée

X = ecart_type*np.random.randn(nbPas)
X = np.cumsum(X)
Y = ecart_type*np.random.randn(nbPas)
Y = np.cumsum(Y)

plt.title('Simulation du mouvement Brownien en dimension 2')
plt.plot(X, Y, color='red', lw=0.2)
plt.axis('off')
plt.show() #on affiche tous les graphiques
```



c- Et en dimension 3

```
import numpy as np       #génération des échantillons
from mpl_toolkits . mplot3d import Axes3D
import matplotlib.pyplot as plt #surnommée en plt
from math import (log, sqrt)

nbPas = int(input('Nombre d'' étapes: '))
ecart_type = sqrt(nbPas)       #propriété imposée

X, Y, Z = (ecart_type*np.random.randn(nbPas) for _ in range(3))
X, Y, Z = (np.cumsum(liste) for liste in (X, Y, Z))

fig = plt.figure(figsize=(9,7), facecolor='black')
ax = fig.add_subplot(111, projection = "3d", facecolor='white')
ax.plot(X, Y, Z, linewidth = 0.5, color="cyan") #taille à définir en fonction de nbPas pour plus de visibilité
ax.axis('off')
plt.title("Mouvement Brownien ({:e}) pas".format(nbPas), color='red')  #titre
plt.show()
```

5- Application du mouvement brownien aux mathématiques financières

a- Modèle de Black-Scholes

On cherche à modéliser le cours d'un actif coté en temps continu par un processus aléatoire $(S_t)_{t\geq 0}$.

Le modèle de Black-Scholes de prix initial $S_0 > 0$, rendement instantané μ et volatilité $\sigma > 0$ est donné par :

$$S_t = S_0 exp(\mu t - \frac{\sigma^2}{2} t + \sigma B_t), t \geq 0,$$

où $(B_t)_{t\geq 0}$ est un mouvement Brownien standard.

b- Application

Nous avons récupéré les cours de clôture du CAC 40 et nous avons cherché à modéliser les cours observés.

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

def extract_data(name: str):
    '''Returns an array of dates, stock market prices and variations in %'''
    import csv, re, datetime

    def date_fmt(date: str):
        d, m, y, *h = map(int, re.split(r'\D', date))
        return datetime.date(day=d, month=m, year=y)

    def number_fmt(value: str):
        return float(value.replace(',','.'))

    date_fmt = np.vectorize(date_fmt)
    number_fmt = np.vectorize(number_fmt)

    with open('spreadsheets/Comparaison ecart-type - '+name+'.csv', 'r', newline='') as f:
        array = csv.reader(f, delimiter=',')
        data = np.transpose(list(array)[1:])

    return date_fmt(data[0]), number_fmt(data[1]), number_fmt(data[2])/100
```

```
def black_scholes(price, variations):
    '''Returns a numpy array of values â€‹â€‹obtained by the Black Scholes differential equation'''
    n = len(price)

    t = np.linspace(0,1,n)
    S0 = price[0]
    mu = np.log(price/S0)
    sigma = np.std(variations)
    var_alea = np.random.normal(loc=0, scale=(1/n)**2, size=(n))
    Bt = np.cumsum(var_alea)

    St = S0*np.exp(mu*1-sigma**2/2*1+sigma*Bt)
    return St
```

```python
def draw(name: str):
    '''Draw a graph from the spreadsheet values â€‹â€‹and the other from the Black-Scholes equation'''
    dates, price, variations = extract_data(name)
    bs_curve = black_scholes(price, variations)

    fig, ax = plt.subplots()
    ax.plot(dates, bs_curve, color='red', lw=0.5)
    ax.plot(dates, price, color='green', lw=0.3)
    ax.grid(True)
    fig.autofmt_xdate()
    ax.legend(['Black-Scholes', 'Stock price'], loc=4)
    plt.title("Cours de bourse %s et équation de Black-Scholes"%name)
    plt.xlabel('Temps')
    plt.ylabel('Valeurs en clôture')

draw("CAC40")  #add graphics this way
plt.show()
```

On observe une bonne corrélation entre les variations réelles et notre simulation.