Simulation of Brownian motion

Daniel Stefanik

1. Explanation of aim

The aim of this exercise was to create model of physical phenomenon - Brownian motion. Implementation of computer simulation using Monte Carlo method helps to understand the mechanisms describing the object of observation and theoretical concept. Another objective is to validate some of the properties related to this issue.

2. Used tools

Simulation was created in python language using Jupyter Notebook application. Libraries and their use:

- *matplotlib* - draw the diagrams of moving particles and the relationship between mean square of displacement and time,
- *numpy* - create pseudo random values with normal probability distribution.

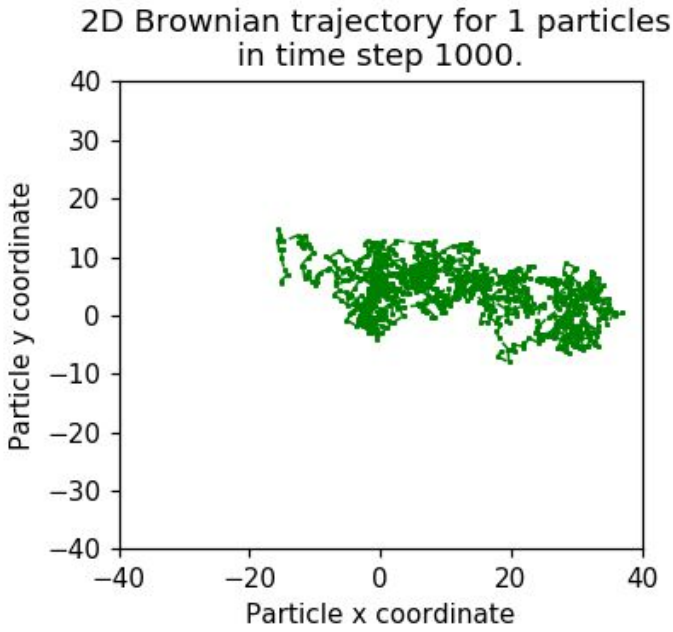
3. Results

a. Single particle trajectory

Created scripts visualize Brownian motion in two dimensions. First diagram shows the trajectory of 1 particle where number of time steps is equal to 1000. Starting point is (0,0) and next positions are calculated by adding to current position coordinates random values. Random parameters are:

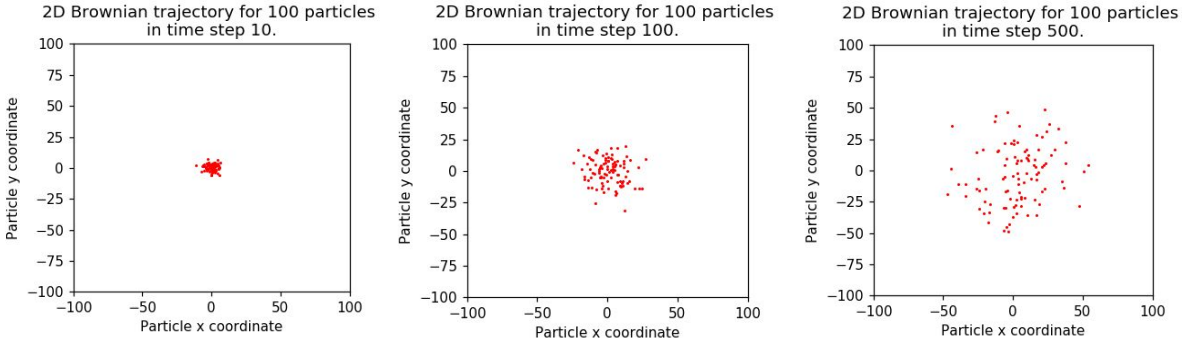
- mean = 0,
- standard deviation = 1.

We can observe that the random moves of this particle forms some patterns similar to each other.



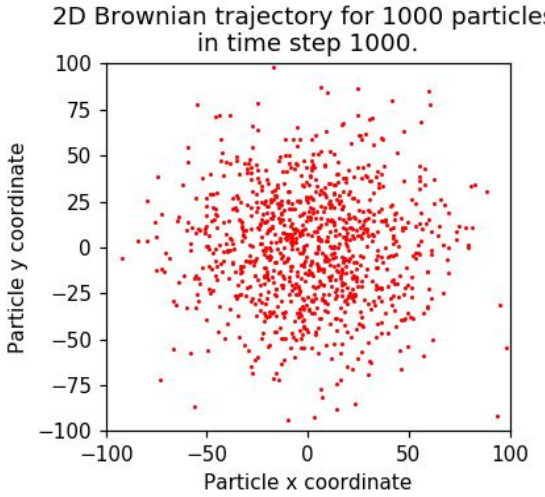
b. Multiple particles

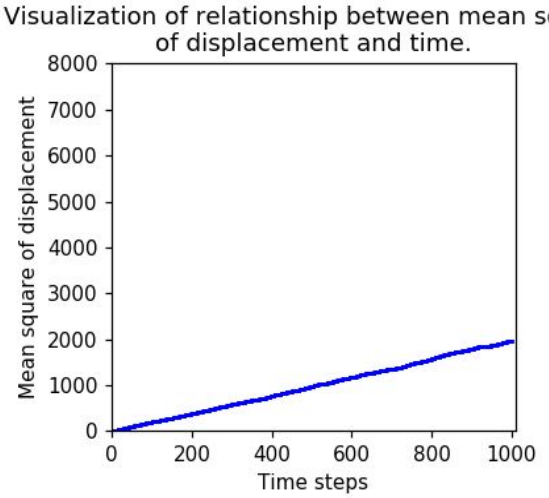
Second diagram presents 100 particles in time step 10, 100 and 500 all starting from point (0,0). Visualisation shows that along the increase of time steps density of principles decreases.



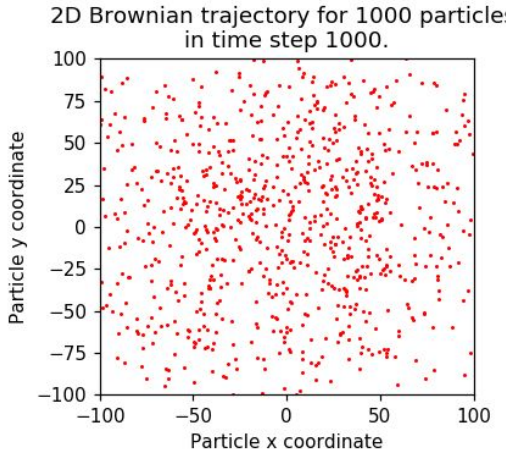
c. Relationship between mean square of displacement and time

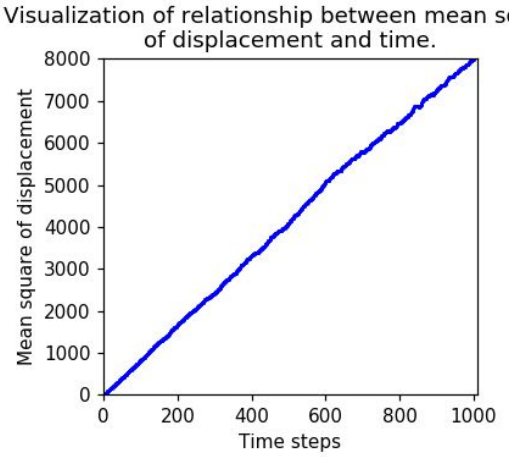
As we can see on the third diagram there is a relationship between mean square of displacement and time.





When comparing the third and the fourth diagram we can notice that the density of particles are different. This is due to a change of standard deviation parameter for random numbers generator to 2. This change causes increase of angle between the x axis and the line designated by mean square of displacement over time.





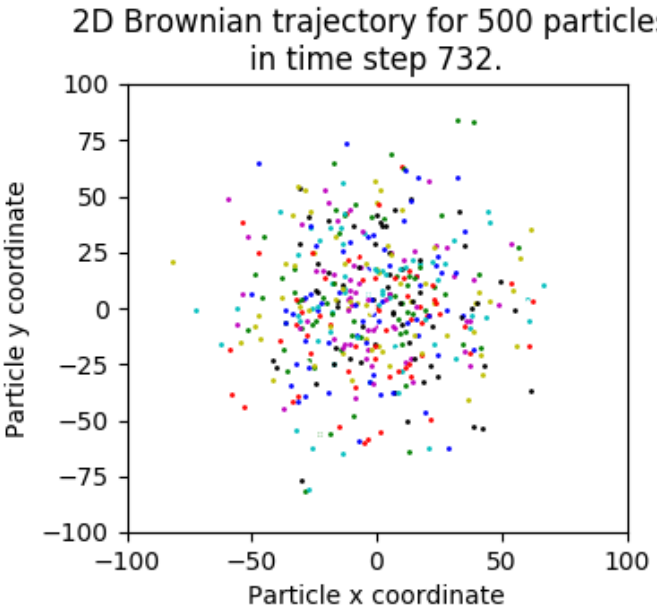
4. Discussion

a. About simulation

Simulation was created step by step:

- model of single particle as a class,
- trajectory of single particle,
- multiple particles and trajectories,
- visualize just last position of particles,
- compute and create diagram of relationship between mean square of displacement and time,
- add colors to particles on the diagram.

To start simulation with colored particles run method *runAllColors(particles,epochs,ax,axMsod)* see diagram below. Uncommenting lines *ax.cla()* and *drawLabels(ax,particles[0].epoch)* in plot methods removes trajectories of particles and shows only the last position of them.



b. Conclusions

Creating simulation of Brownian motion using Monte Carlo method helped us to understand theoretical concept and mechanisms of this physical phenomenon. We also validated relationship between mean square of displacement and time what was another aim of this exercise.

5. Code listening

```
%matplotlib notebook

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


mu, sigma = 0, 2 # mean and standard deviation
particlesAxisX = [-100, 100]
particlesAxisY = [-100, 100]
msodAxisX = [0,1010]
msodAxisY = [0,8000]
```

```
epochs = 1000
amount_particles = 1000
```

```
### Graph
def initGraphs():
    ax = fig.add_subplot(1, 2, 1)
    drawLabels(ax,0)

    axMsod = fig.add_subplot(1, 2, 2)
    drawMsodLabels(axMsod)

    plt.subplots_adjust(bottom=0.25, top=0.75, wspace=0.5, hspace=0.5)
    fig.show()
    return ax, axMsod

def drawMsodLabels(axMsod):
    axMsod.set_xlim(msodAxisX)
    axMsod.set_ylim(msodAxisY)
    axMsod.set_xlabel('Time steps')
    axMsod.set_ylabel('Mean square of displacement')
    axMsod.set_title('Visualization of relationship between mean square
\nof displacement and time.')

def drawLabels(ax, epoch):
    ax.set_xlim(particlesAxisX)
    ax.set_ylim(particlesAxisY)
    ax.set_xlabel('Particle x coordinate')
    ax.set_ylabel('Particle y coordinate')
    ax.set_title('2D Brownian trajectory for '+str(amount_particles)+'
particles \nin time step '+str(epoch)+".")

def plotParticles(particles, ax):
    ax.cla()
    drawLabels(ax,particles[0].epoch)
    ax.plot(map(lambda m: m.x, particles),map(lambda m: m.y,
particles),"ro",markersize=1)

def plotParticlesColors(particles,ax):
    #     ax.cla()
    #     drawLabels(ax,particles[0].epoch)
    for m in particles:
        #         ax.plot(m.x,m.y,m.color+"o",markersize=1)

ax.plot([m.bx,m.x],[m.by,m.y],m.color+'o--',markersize=1,linewidth=1)
```

```
def plotMsod(epoch,msod,ax):
    ax.plot(epoch,msod,"bo",markersize=1)


### Calculations
class Particles:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.bx = x
        self.by = y
        self.color = color
        self.epoch = 0
        self.squareFullDistance = 0

    def __str__(self):
        return "Particles ("+str(self.x)+","+str(self.y)+")
"+str(self.color)+", epoch: "+str(self.epoch)

    def move(self, coordinates):
        self.bx = self.x
        self.by = self.y
        self.x += coordinates[0]
        self.y += coordinates[1]
        self.epoch += 1
        self.squareFullDistance = self.x*self.x + self.y*self.y

def meanSquareOfDisplacement(movementPerTimeUnit, squareDistance, time):
    return movementPerTimeUnit * squareDistance * time

def calculateMsod(particles):
    avr = sum(map(lambda m: m.squareFullDistance, particles))
    return meanSquareOfDisplacement(1,avr/len(particles),1)

def randomCoordinates():
    return np.random.normal(mu, sigma, 2)

def setInitCoordinatesZeros(amount_particles,colors):
    particles = []
    for i in range(0,amount_particles):
        particles.append(Particles(0,0,colors[i%len(colors)]))
    return particles

def setInitCoordinates(amount_particles,axis,border,colors):
```

```
    particles = []
    if amount_particles == 1:
        particles.append(Particles(0,0,colors[0]))
        return particles

    for i in range(0,amount_particles):
        s_amount = (math.sqrt(amount_particles))

        offset = (axis*2 - border*2) / s_amount

        x = -axis+border+offset*(i/s_amount)
        y = -axis+border+offset*(i%s_amount)
        particles.append(Particles(x,y,colors[i%len(colors)]))
    return particles


### Run methods
def runAll(particles,epochs,ax,axMsod):
    for i in range(0,epochs):
        for m in particles:
            m.move(randomCoordinates())

        plotParticles(particles,ax)
        plotMsod(particles[0].epoch,calculateMsod(particles),axMsod)
        fig.canvas.draw()

def runAllColors(particles,epochs,ax,axMsod):
    for i in range(0,epochs):
        for m in particles:
            m.move(randomCoordinates())

        plotParticlesColors(particles,ax)
        plotMsod(particles[0].epoch,calculateMsod(particles),axMsod)
        fig.canvas.draw()
```

```
axis = 100
border=40
colors = ['b','g','r','c','m','y','k','w']
```

```
fig = plt.figure(figsize=(9, 5))
ax, axMsod = initGraphs()
```

```
# particles = setInitCoordinates(amount_particles,axis,border,colors)
particles = setInitCoordinatesZeros(amount_particles,colors)

runAll(particles,epochs,ax,axMsod)
# runAllColors(particles,epochs,ax,axMsod)
```