Project Report
Modeling and Visualization of Brownian Motion with python

Author: J???? Song
Physics Department, University of D?????

Brownian motion is widely used in the field of physics, mathematics, and economics, refers to the random motion of particles in a fluid, such as smoke particles in a gas. This study models and visualizes the Brownian motion in two dimensions by using random walk process and python visual package respectively. The final program is capable to performs a million steps of random walk process in a finite size lattice.

I. INTRODUCETION

A. Outline

This report contains five part. The first part introduces the content of this report and shows the problem we need to solve for this project. The second part explains the algorithm and implementation of the python program. The third part includes some modification based on the project problem. In the fourth part, we demonstrate the visualized Brownian motion and conclusions. The last part discovers some further study opportunities based on this project.

B. Core problem overview

"Brownian motion is the motion of a particle, such as a smoke or dust particle, in a gas, as it is buffeted by random collisions with gas molecules. Make a simple computer simulation of such a particle in two dimensions as follows. The particle is confined to a square grid or lattice $L * L$ squares on a side, so that its position can be represented by two integers i, $j = 0...L - 1$. It starts in the middle of the grid. On each step of the simulation, choose a random direction—up, down, left, or right—and move the particle one step in that direction. This process is called a random walk. The particle is not allowed to move outside the limits of the lattice—if it tries to do so, choose a new random direction to move in."

"Write a program to perform a million steps of this process on a lattice with $L = 101$ and make an animation on the screen of the position of the particle. (We choose an odd length for the side of the square so that there is one lattice site exactly in the center.)"

C. Motivation

Brownian motion denotes the random motion of particles suspended inside a fluid. This phenomenon was first discovered by the botanist Robert Brown in 1827, it revolutionized physics and chemistry at that time by demonstrated the nature of matter.

As the final project of Computational Physics, modeling and visualization of Brownian Motion is a good opportunity to practice the knowledge we learned in this course. It also helps me a lot to improve my scientific programming ability and strengthened my knowledge of mathematics. In this project, I learned how to combine physics with math and computer science to solve problems.

II. PROBLEM SOLVING

A. Mathematical model

In order to apply the random walk model to the project problem, we choose the direction of the particle movement with same probability P_{dir} in the cardinal directions, which is left, right, up, and down.

Initialize the lattice:

$$Area = L * L = 101 * 101 \tag{1}$$

Probability of the particle move to each direction:

$$P_{dir} = \frac{1}{D_{total}} = \frac{1}{4} = 25\% \tag{2}$$

More on this model:
Displacement R from the start point to end point after N steps random walk:

$$R^2 = (\Delta_{x1}+\Delta_{x2}+\cdots+\Delta_{xN})^2+(\Delta_{y_1}+\Delta_{y_2}+\cdots+\Delta_{y_N})^2 \tag{3}$$

Apply equation (3) to our model:

$$R^2 = {i_N}^2 + {j_N}^2 \tag{4}$$

Displacement is an important property of this model, it is the displacement from the start point to end point after N steps random walk. It is commonly represented as the equation (3), which uses the Pythagorean theorem. Apply this equation to our model, we will get the equation (4), then we can implement this equation in the program so that we can easily get the displacement R for each steps.

We can also do a lot of analysis to this model by making some modifications to the program. For example, calculate and display the displacement distribution of particles

after certain steps. There will be a detailed demonstration after the problem solving part.

B. Random walk

To solve the problem listed in the previous section, random walk process is adapted. It is a famous stochastic process which describes a path made by finite many random steps. In this project, we will use this math model to describe and implement the Brownian motion.

The figure 1 shows some examples of random walk, it demonstrates five motion path of particles start at the same point, which the particles can only move towards up, down, left, and right with same step size and probability.

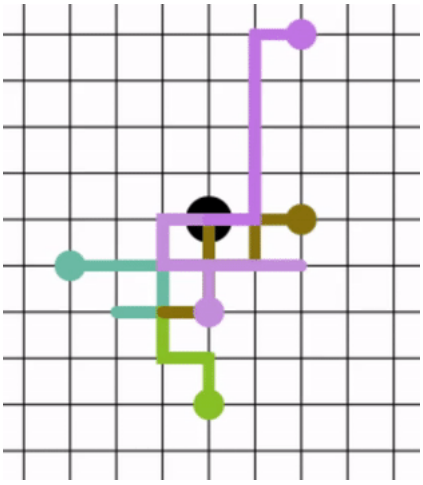

FIG. 1. Five 8-step random walk path from the same central point.

C. Initialization

Given the square lattice size should be $L * L$ with side length $L = 101$. In order to record the position of the moving particle, we create 2 variable i and j to hold the coordinates of the particle in the 2-D lattice. So, we can simply represent position of particle using i and j. By setting the initial value of i and j to 50, we successfully set the start position of the particle in the middle of the square lattice

Python code:

```
# Set up the square grid
L = 101 #Side length of the square grid

# Set the initial position of the particle
i = 51  # x-axis
j = 51  # y-axis
```

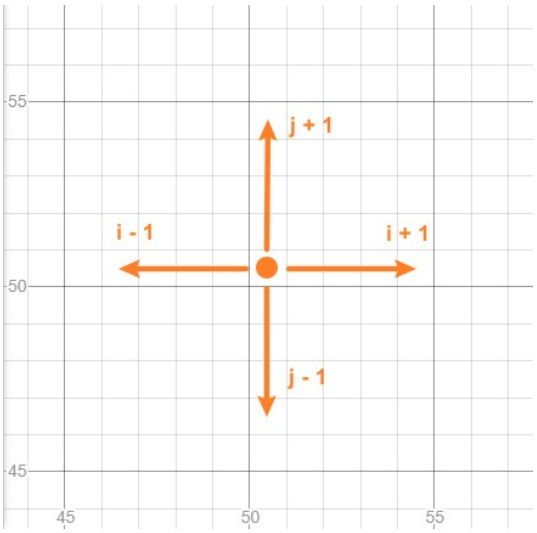

FIG. 2. Change i and j to represent the motion of the particle.

D. Prepare for visualization

In order to visualize the 2-D single particle Brownian motion, we firstly need to create an sphere object to represent the moving particle:

```
# Set up the square lattice
grid = canvas(title="Brownian Motion", width=200, height=200, center=vector(50,50,0),
              background=color.black)

# Set up the sphere object
particle = sphere(pos=vec(i, j, 0), radius=0.3, color=color.red, make_trail=True)
```

E. Computation

This part is the main body of the program. Here, we recursively compute and update the position of the particle (sphere object) using a for loop.

The loop structure will recursively compute and update the position of the particle for N times. For example, If we want to perform a 100 step Brownian motion, the for loop will run for a 100 times.

This loop will also stop the particle when the particle

is going to leave the lattice.

Inside the for loop there are three parts. The first part will randomly choose an integer among 4 different integers, each of the integer represents a direction. At the second part, the loop body will determine what direction is chosen in the last part and update the i or j value accordingly. In the end, it will update the position of sphere object we created before and visualize it.

Python code:

```
tpoints = arange(0, 1e6, 1)
for t in tpoints:
    rate(50) #Set the max animation rate
    # Use randint to generate random output
    go = rnd.randrange(4)
    if go==0:
        if i==L: continue
        i += 1  #Go right
    elif go==1:
        if i==0: continue
        i -= 1  #Go left
    elif go==2:
        if j==L: continue
        j += 1  #Go up
    elif go==3:
        if j==0: continue
        j -= 1  #Go down
    particle.pos=vec(i,j,0)
```

III. PYTHON PROGRAM MODIFICATION

In this subsection, the program is modified to print out the displacement distribution of the particles after certain amount of random walks.

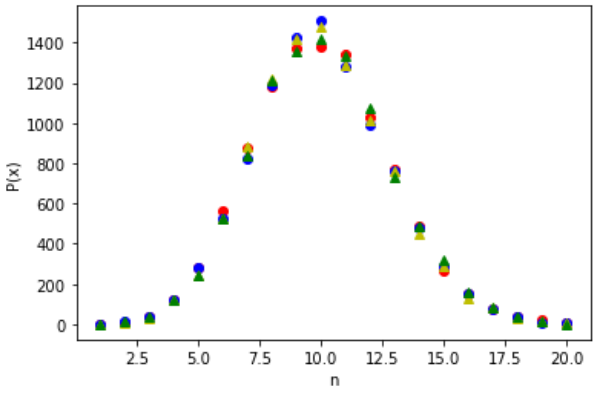

FIG. 3. It is the displacement distribution of the particles after 10000 times 40-step random walk try. Four different makes denotes four different directions respectively

It is clear that the position of particles are in a binomial distribution, we can also compare it to the data in the text book (FIG. 4.), which also in a binomial distribution:

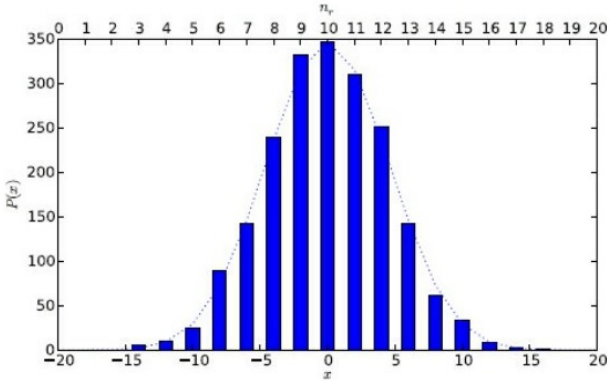

FIG. 4. The distribution of position for 2000 walkers moving 20 steps each ($p = \frac{1}{2}$). The dotted line is the (unnormalized) binomial distribution. The upper scale n_r is the number of right steps.

Python code:

```
from numpy import arange, zeros
import random as rnd
import matplotlib.pyplot as plt

# Use numpy arrays to hold displacement data
countr = zeros((1,20), dtype=int)
countl = zeros((1,20), dtype=int)
countu = zeros((1,20), dtype=int)
countd = zeros((1,20), dtype=int)
value = zeros((1,20), dtype=float)
for i in range(1,21,1):
    value[0,i-1] = i

def randomwalk(right, left, up, down):
    for t in arange(0, 40, 1):
        # Use ranint to pick random integers
        go = rnd.randint(1,4)
        if go==1:
            right += 1
        elif go==2:
            left += 1
        elif go==3:
            up += 1
        elif go==4:
            down += 1
    return right, left, up, down

def decider(num):
    for n in range(20):
        if num <= (n + 1):
            countr[0,n] = countr[0,n] + 1
            break
        else:
            continue

def decidel(num):
    for n in range(20):
        if num <= (n + 1):
```

```
            countl[0,n] = countl[0,n] + 1
            break
        else:
            continue

def decideu(num):
    for n in range(20):
        if num <= (n + 1):
            countu[0,n] = countu[0,n] + 1
            break
        else:
            continue

def decided(num):
    for n in range(20):
        if num <= (n + 1):
            countd[0,n] = countd[0,n] + 1
            break
        else:
            continue

for iter in arange(0, 10000):
    a,b,c,d = randomwalk(0,0,0,0)
    decider(a)
    decidel(b)
    decideu(c)
    decided(d)

plt.plot(value, countr, 'ro'
         ,value, countl, 'bo'
         ,value, countu, 'y^'
         ,value, countd, 'g^')
plt.xlabel('n')
plt.ylabel('P(x)')
plt.show()
```

IV. SAMPLE OUTPUT AND CONCLUSIONS

This part demonstrate the visualized Brownian motion and conclusions.

A. Sample output

Here, we display some frames of the animation:

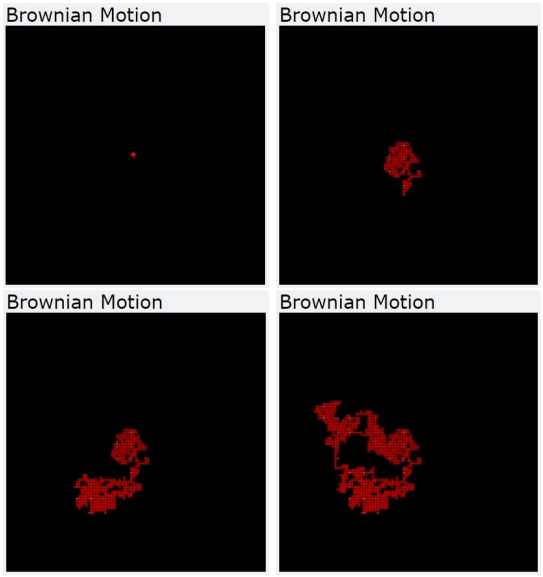

FIG. 5. Screenshots of the animated 2-D single particle Brownian motion.

B. Summary

To summarize this project, we adapted the random walk model to implement the 2-D Brownian motion. Although the way we implement the random walk is very simple, but we successfully modeled and visualized the Brownian motion with the support of python modules like vpython, numpy, and random. The main drawback of this program is the slow processing speed, we tried to use numpy module to improve its performance but the result is still not very good. I think it is because python is a relatively slow language.

C. Conclusion

As a conclusion, the program successfully implement the Brownian motion in a 2-D square lattice, but the processing time could be very long if we perform a Brownian motion with very high step number. In addition, this program has a lot of potential. We can keep modify it and get super detailed data of 2-D Brownian motion. What I learned from this project is the way to use programming

skills to solve physics problems and this project also gives me a better understanding about how to combine physics, mathematics, and computer science.

D. Further study opportunities

- 3-D simulation of the current model
- Use 3D creation suite like blender to make more intuitive animations
- Collect and analysis the movement data of the moving particle

Appendix A: Python module used

- numpy
- vpython
- random

[1] J. Wang, *Computational modeling and visualization of physical systems with Python* (wILEY, 2015).

[2] R. H. Landau, M. J. Páez, and C. C. Bordeianu, *Computational Physics: Problem Solving with Python, 3rd Edition* (wILEY, 2016).

[3] B. A. and S. Ewald, *Basic Concepts in Computational Physics* (Springer, 2015).