PHY407H F: Explanatory Notes For Lab 10

Idil Yaktubay and Souren Salehi

29 November 2022

1 Question 1 - By Idil Yaktubay

(a) In this question (Newman 10.3), we have created time and trajectory plots of a particle in two-dimensional Brownian motion using Python. Specifically, we have created a particle that starts at the center of a 101×101 square lattice and performs a "random-walk" for 5000 steps. The random-walk was generated using the `randrange()` function from the `random` module in Python. Here, the particle is confined to the limits of the lattice; the coordinates of the particle are given by $x, y = 0...100$. To ensure this, we have implemented code that moves the particle in possible random directions whenever it finds itself at the edges or at the corners of the lattice. For example, when the particle is at the top left corner of the lattice (i.e., $(x, y) = (0, 100)$), it randomly chooses to either move downwards or to the right. Or, when the particle is at a non-corner bottom edge of the lattice (i.e., $y = 0$, $x = 1, ..., 99$), it chooses to move left, right, or up. Note that such movements are not truly random due to the nature of Python's random number generators. Figure 1 gives x vs t and y vs t plots of the particle. Figure 2 gives the y vs x plots - one in the scale of the lattice and one zoomed in, respectively. The file named `LAB10_Q1.py` contains our pseudocode and code that generates these plots. The file `LAB10_Q1.py` is an edited and complete version of the sample file `Brownian-start.py`, which is also attached to this submission for reference.

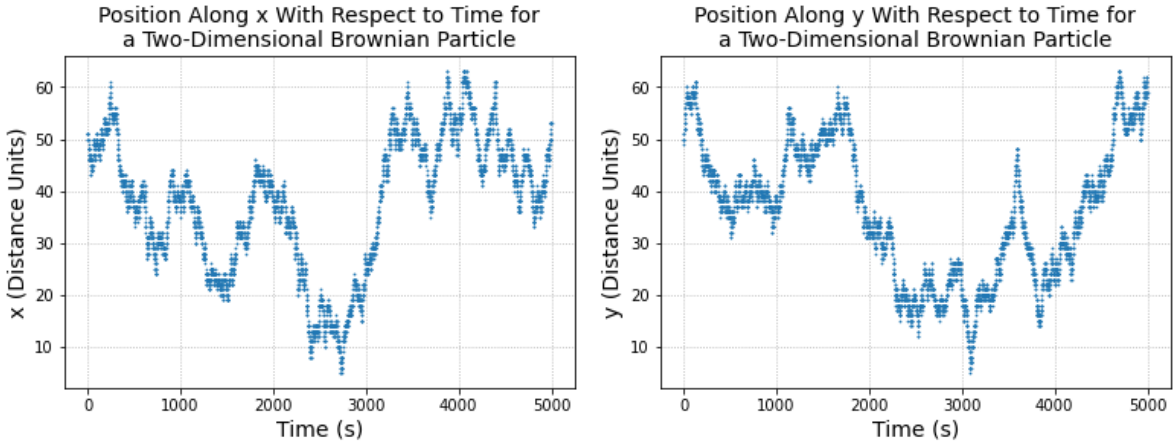

Figure 1: X-y coordinates of a particle in two-dimensional Brownian motion with respect to time. The particle performed a "random" walk generated by Python's `randrange()` function from the `random` package inside a 101×101 square lattice for 5000 steps. For each move, the particle was allowed to "randomly" move right, left, up, or down as long as it remained inside the lattice. It was assumed that the particle would make one move/second. The x-y coordinates have arbitrary distance units.

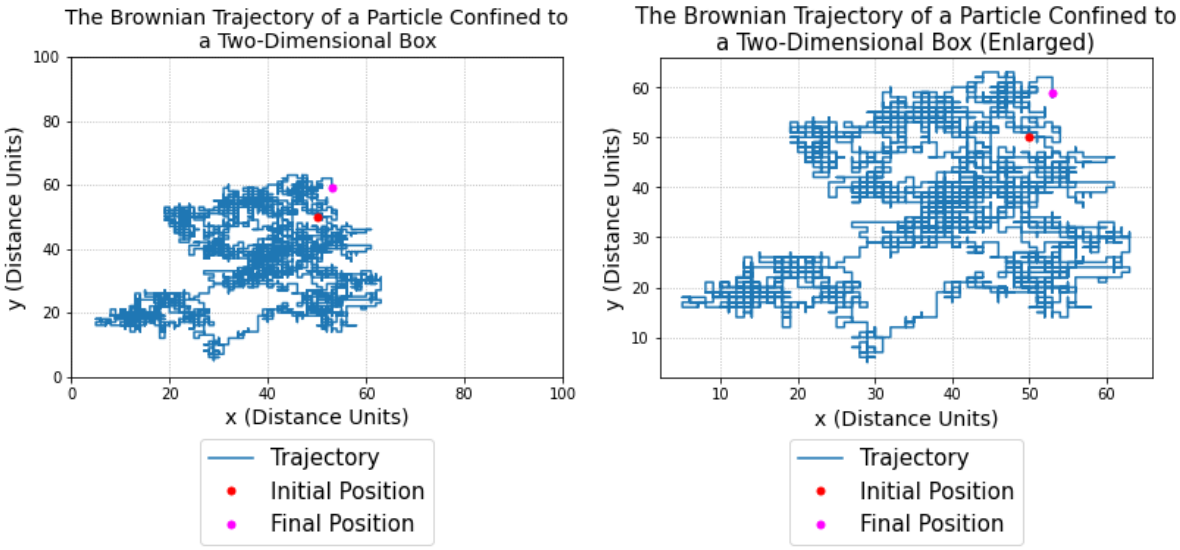

Figure 2: The x-y trajectory of a particle in two-dimensional Brownian motion. The plot on the right is the same plot as on the left, but enlarged for the viewer. Both of these plots correspond to the same random walk shown in Figure 1 and are described by the same set of conditions as in the Figure 1 caption.

(b) In this question (Newman 10.13a), we have further edited and developed our previous Python code to reproduce a *diffusion-limited aggregation* (DLA) model. The version of DLA that we have created is one where many Brownian particles take turns to perform random walks starting from the center of a 101×101 square lattice until they anchor to a lattice edge or to other anchored particles. New particles perform such random walks until the starting center point becomes occupied by an anchored particle. Evidently, this model differs from the previous one because instead of bouncing from the edges, they stick to them. Figure 3 depicts the DLA pattern that results from the final positions of particles in our set of random walks. The file named `LAB10_Q1.py` contains our pseudocode and code that generates this pattern. We have not used the sample file `DLA-start.py` to write our code. Instead, we have simply adapted our code from part (a) to the DLA model. Therefore, this submission does not include a copy of `DLA-start.py`.

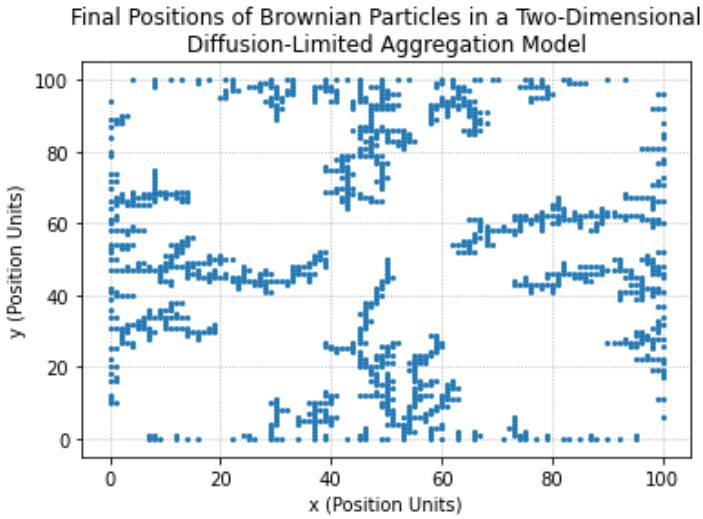

Figure 3: Final positions of particles that completed two-dimensional Brownian random walks to create a diffusion-limited aggregation pattern. This pattern contains 1231 particles whose motion was generated using Python's `randrange()` function from the `random` package. Once again, particle motion was limited to a 101×101 square lattice. The particles reached their final positions once they moved to the vicinity of another anchored particle or to a lattice edge.

2 Question 2 - By Idil Yaktubay

In this question (Newman 10.7), we have estimated the volume of a ten-dimensional unit hypersphere using Monte Carlo integration. Specifically, we have generated one million uniform random points from a ten-dimensional unit hypercube and evaluated the value of the unit hypersphere function given by equation 1, where $\mathbf{r}$ is the ten-dimensional position vector and x_i are the components of $\mathbf{r}$. Using the Monte Carlo integration method described by equation 2, we have found the volume of the hypersphere to be 2.56 ± 0.05. In equation 2, $V = 2^{10}$ is the volume of the unit hypercube and $N = 10^6$ is the number of random points. The standard deviation $\sigma = 0.05$ was calculated using equation 3, where $var\, f$ is the variance on a single random value of equation 1. Further, $var\, f$ is given by equation 4 where $\langle\rangle$ is the average value operator applied to the span of the unit hypercube. We found the exact value of the ten-dimensional unit hypersphere to be 2.55 using equation 5, where $R = 1$ is the radius and $n = 10$ is the dimension of the hypersphere, and Γ is the gamma function. This value falls within the uncertainty of our estimation, hence we can conclude that Monte Carlo integration was a sufficient method for our purposes. Figure 4 depicts our code output. Our pseudocode and code can be found in the file `LAB10_Q2`. Lastly, note that our results do not have units because they are arbitrary volume units.

$$f(\mathbf{r}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{10} x_i^2 \ \leq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

$$I = \frac{V}{N} \sum_{i=1}^{N} f(\mathbf{r}_i) \tag{2}$$

$$\sigma = V \frac{\sqrt{var\, f}}{\sqrt{N}} \tag{3}$$

$$var\, f = \langle f^2 \rangle - \langle f \rangle^2 \tag{4}$$

$$V = \frac{R^n \pi^{n/2}}{\Gamma(\frac{n}{2} + 1)} \tag{5}$$

```
The approximated volume of a 10D unit hypersphere is:  2.557952 with standard
deviation 0.05111555271701262
The exact volume of a 10D unit hypersphere is:  2.5501640398773345
```

Figure 4: Python code output of a computational estimation of a ten-dimensional unit hypersphere using the Monte Carlo integration method. The estimation was done by generating uniform random numbers, using Python's `unifrom()`, inside a ten-dimensional unit hypercube and calculating their values with equation 1 and applying these values to equation 2.

3 Question 3 - By Souren Salehi

In question 3 we used the mean value method and importance sampling method to find the integral of,

$$\int_0^1 \frac{x^{-\frac{1}{2}}}{1 + e^x} dx. \tag{6}$$

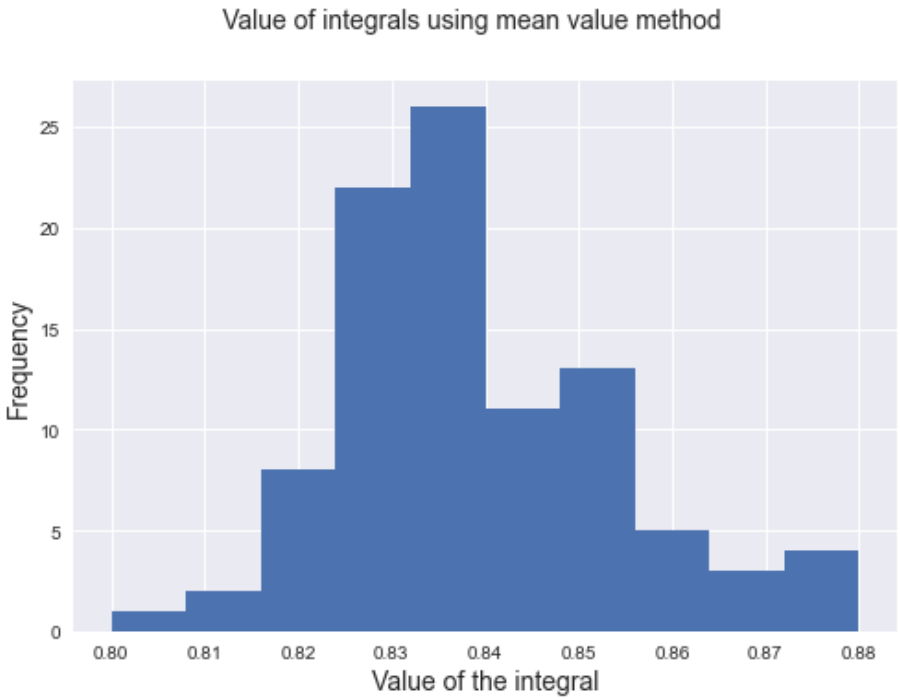

Figure 5: The distribution of the integral of equation 6 found using the mean value method using 10,000 sample points and repeated 100 times.

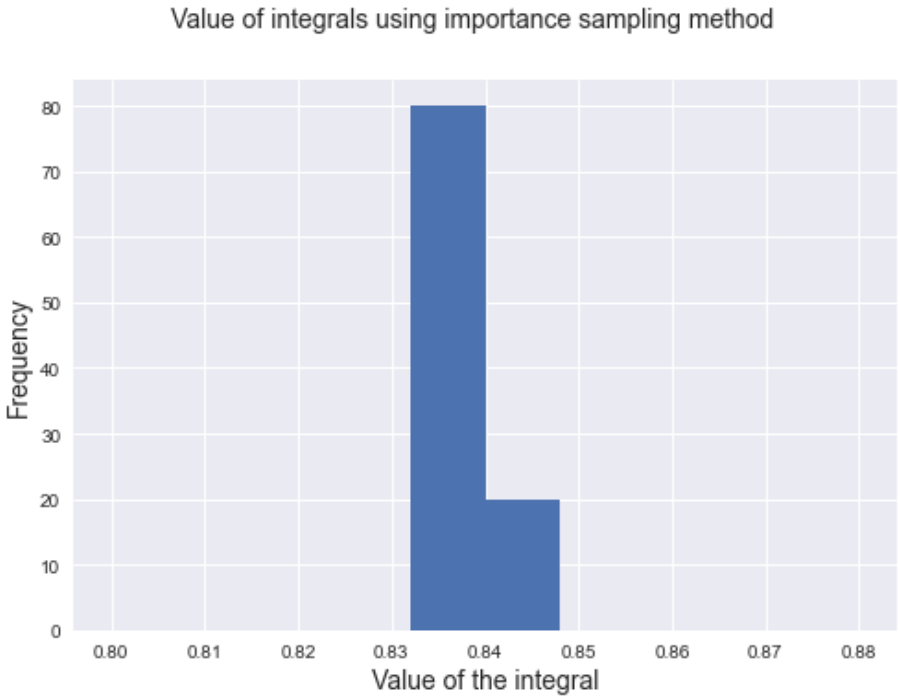

Figure 6: The distribution of the integral of equation 6 found using the importance sampling method using 10,000 sample points and repeated 100 times. The probability density function used was $p(x) = \frac{1}{2\sqrt{x}}$.

Please note that both the pseudocode and the Python code that generated these figures are contained in the file named `Lab10_Q3.py`. Comparing figure 5 and figure 6 we can see that both distributions are at their highest between 0.832 and 0.84 but the main difference between the two is the variance of the distributions.

Looking at figure 5 we see there is a distribution of values from 0.8 all the way to 0.88 whereas all the values for the importance sampling method are between 0.832 and 0.848. Therefor both methods are able to provide the correct value (if we average the values from from the mean value method) however importance sampling method is much more accurate with a smaller error.