Variable Step-Length Random Walks

Sahaana Vijay, Gayathri Thatavarthi

PHY-2610 | Thermal Physics

Abstract

This computational project explores the idea of a random walk, which is an important concept with a range of applications in vastly different fields, including physics, financial economics and the dynamics of biological systems. Using Python, this project undertakes the simulation of various types of random walks. Simulating complex physical systems computationally can be difficult, which is why we begin with very rudimentary models. First, simple cases of one and two dimensional random walks with fixed step lengths have been modelled and studied, after which the behaviour of one dimensional random walks with step lengths varying based on two different probability distributions were simulated. These computational models can be built on to simulate the behaviours of far more complex physical systems.

Table of Contents

4 | Introduction

1.1 One and Two-Dimensional Random Walks *4*
1.1.1 One-Dimensional Random Walks *4*
1.1.2 Two-Dimensional Random Walks *6*
1.2 Random Walks of Variable Step Length *8*

10 | Model and Methods

2.1 Model *10*
2.2 Methods *10*
2.2.1 Exponential Probability Distribution *10*
2.2.2 Lévy Distribution *11*
2.3 Algorithm *11*

14 | Final Remarks

3.1 Results *14*
3.2 Graphs *15*
3.3 Conclusion *19*

20 | Appendix

4.1 Code for a Single 1D Random Walk *20*
4.2 Code for N 1D Random Walks *21*
4.3 Code for Two-Dimensional Random Walks *22*
4.4 Code for Variable Step Length Random Walks *25*
4.4.1 Exponential Distribution *25*
4.4.2 Lévy Flights *29*
4.5 References *31*

Introduction

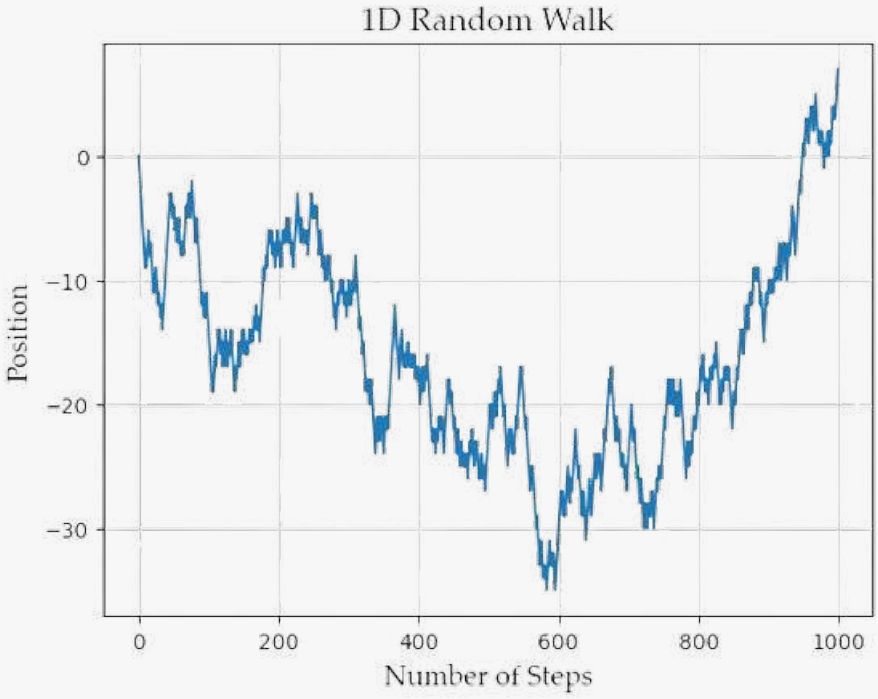
1.1 One and Two-Dimensional Random Walks

The concept of a random walk seems deceptively straightforward. Also known as a drunkard's walk, it represents the behavior of processes where every successive step is uncertain and random – much like a drunkard's choice of steps as they stumble around aimlessly. It serves as a mathematical model for studying the movement of any object undergoing random motion. In physics, random walks find application in describing various stochastic processes such as Brownian motion, gas diffusion, and even advanced domains like Quantum Field Theory.

1.1.1 One-Dimensional Random Walks

In the simplistic framework of a one-dimensional random walk, each step maintains a consistent length and operates autonomously from preceding steps. Typically presumed to be impartial or symmetric, wherein the walker has an equal likelihood of moving right or left at any juncture, the steps are inherently stochastic. While the simplest models assume random walks of fixed step lengths, slightly more realistic models can incorporate random walks of variable step lengths, for example.

Example. Here, we have simulated a random walk with 1000 steps where the walker moves left or right with an equal likelihood ($\mathrm{p} = 0.5$).



Additionally, we simulated 1000 such walks and plotted a histogram of the final positions, finding it to be the bell curve, a characteristic of Gaussian distributions. The mean of the histogram was close to zero, and the standard deviation was close to the square root of the number of steps, as theoretically determined.

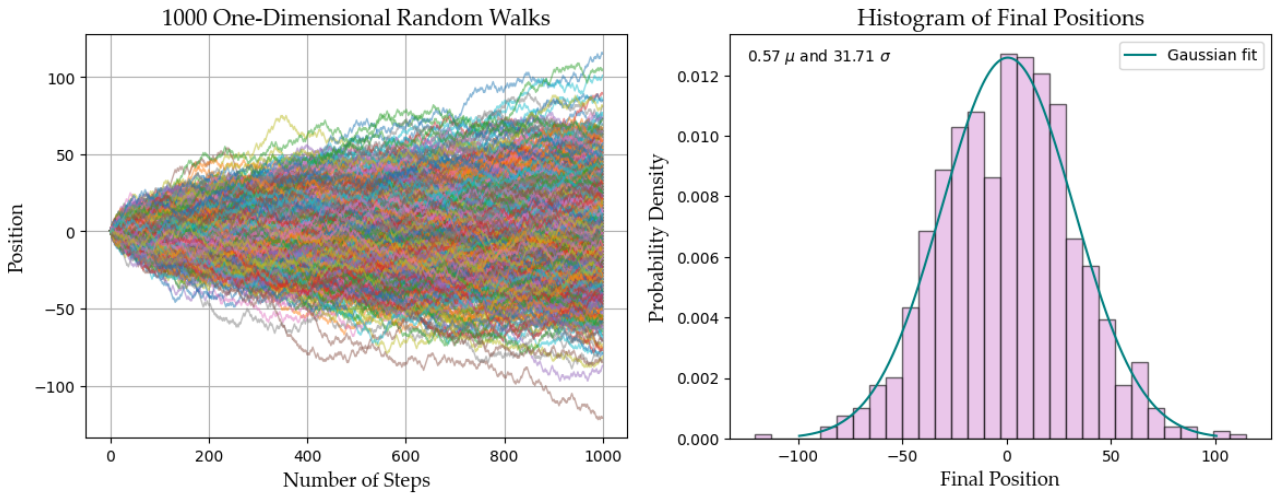


Figure 1.1: Plotting 1000 one-dimensional random walks of 1000 steps and plotting a histogram of the final positions for each of the walks. The mean here is 0.57, which is close to zero, and the standard deviation is 31.71, which is close to the square root of 1000, i.e., 31.62.

We also plotted the square of end-to-end distances against the number of steps for the 1000 random walks.

Figure 1.2: We calculated the distances travelled for all walks at each step, then averaged them out. Once we had these distances, we squared and plotted them against the number of steps. For 1D random walks, this relation is found to be linear with slope 1. However, in our case, we have a slope of 0.64 due to slight error.

Now, how is it that the slope is one, or in other words, how is it that the average of the

square of the end-to-end distances is equal to the number of steps N? Consider the end-to-end distance given by

$$R_d = |x_N - x_0|, \tag{1.1}$$

where x_N is the position after N steps and x_0 is the initial position. Squaring this, we get

$$\begin{aligned} R_d^2 &= (x_N - x_0)^2 \\ &= (x_N - x_{N-1} + x_{N-1} - x_{N-2} + x_{N+2} - \cdots + x_1 - x_0)^2 \\ &= (S_{N-1} + S_{N-2} + \cdots + S_0)^2 \end{aligned} \tag{1.2}$$

where S_i represents the difference between two consecutive positions. This can be further simplified to

$$R_d^2 = S_{N-1}^2 + S_{N-2}^2 + \cdots + S_0^2 + (\text{summation of some cross terms like } S_0S_1,\ S_1S_2, \ldots) \tag{1.3}$$

We know that as this algorithm has a fixed step-length of size 1, S_i could be either 1 or -1. This implies that $S_{N-1}^2 + S_{N-2}^2 + \cdots + S_0^2 = N$, as the square of each S_i would just be 1. So, we will have

$$R_d^2 = N + (\text{summation of some cross terms like } S_0S_1,\ S_1S_2, \ldots). \tag{1.4}$$

Now, let us consider one of the cross terms S_0S_1. The values this cross term can take are

S_0	S_1	S_0S_1
1	-1	-1
-1	1	-1
-1	-1	1
1	1	1

Table 1.1: This table shows all the possible values a cross term can take. From this, when we take the avaerage of all the prossible cross term values, we find that $\langle S_0S_1 \rangle = 0$.

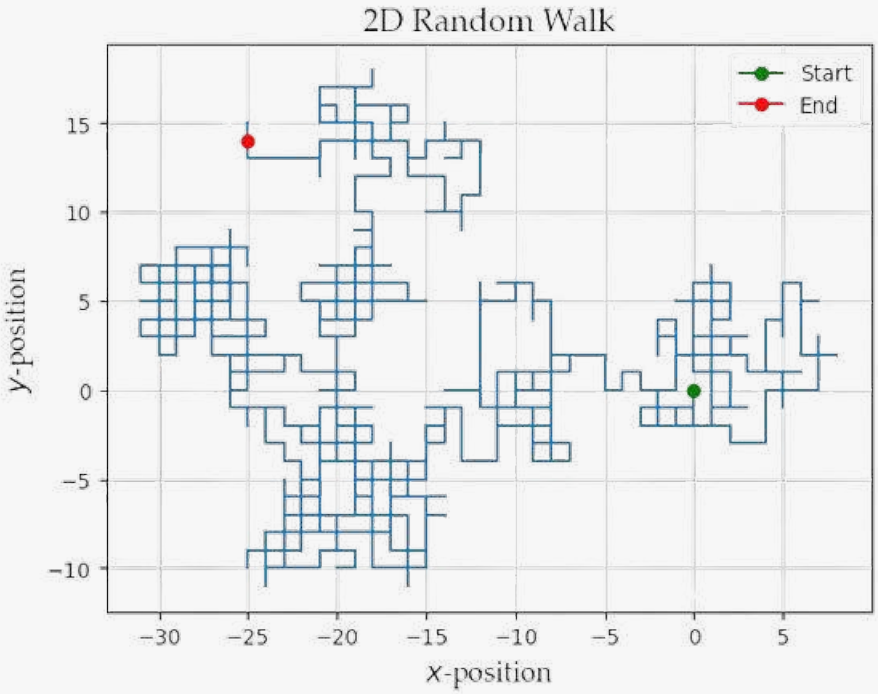
So, the average for all the cross terms is zero. Therefore,

$$\langle R_d^2 \rangle = N + (0) + (0) + \cdots = N \tag{1.5}$$

1.1.2 Two-Dimensional Random Walks

In a two-dimensional random walk, we explore the movement of a particle in two dimensions, much like navigating through a flat plane. The code we had written simulates such a walk by guiding the particle through a series of steps, each determined randomly. At each step, the particle has the option to move in one of four directions: up, down, left, or right. This decision is made randomly for both the horizontal (left and right) and vertical (up and down) dimensions. The code tracks the particle's position after each step, updating its coordinates accordingly. By repeating this process for a specified number of steps, the code generates a trajectory that showcases the particle's random exploration of the 2D space. This type of simulation is not only fascinating but also has practical applications in various fields, including physics, biology, and finance, where understanding the stochastic movement of particles or agents is essential for modeling real-world phenomena.

Example. Below is an example of a single two-dimensional random walk of 1000 steps simulated on Python.



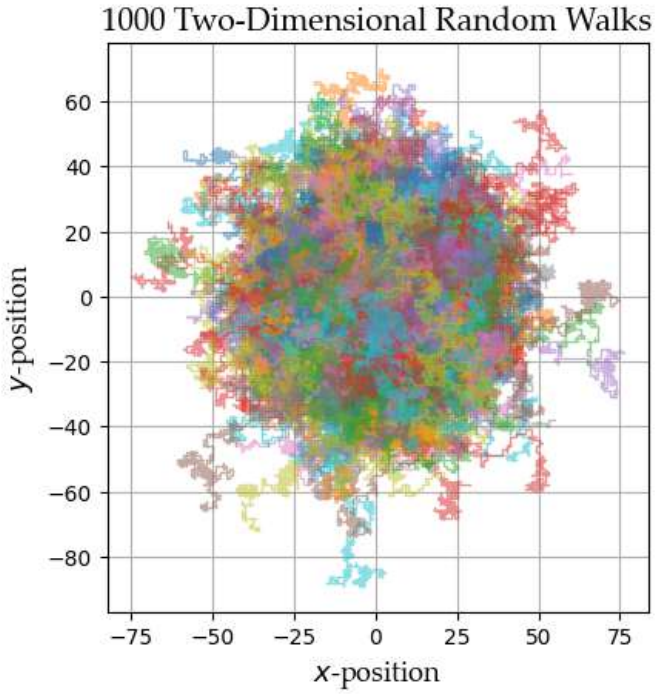


Figure 1.3: We generalised the problem to an N-step random walk on a two-dimensional grid. In such a walk, the walker starts off at the origin and, at every time step, moves a fixed amount in either the x or y directions. Thus, at every time-step, they randomly choose between one of four possibilities. This is then repeated N times.

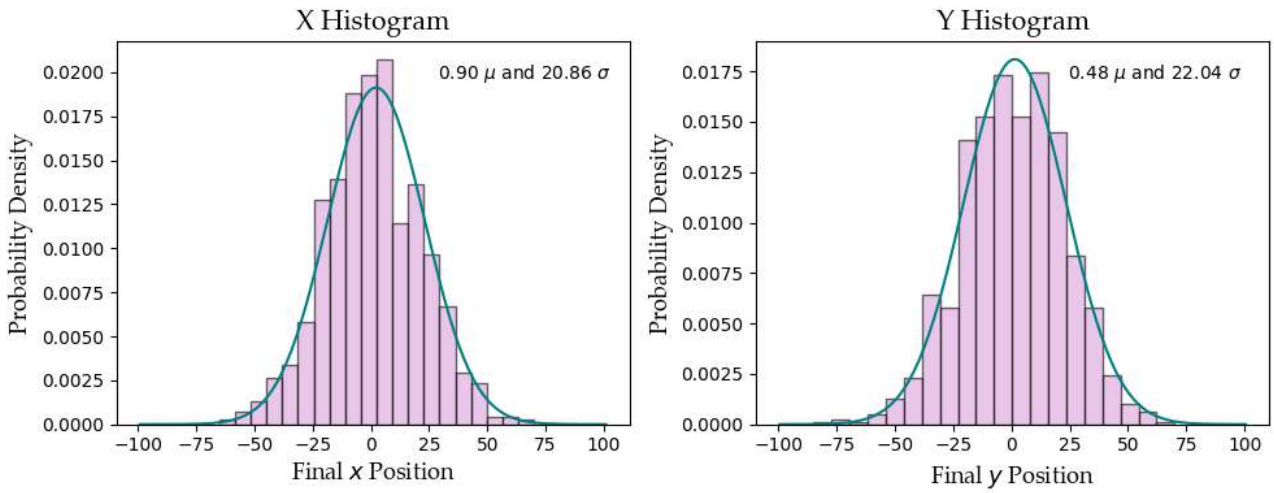


Figure 1.4: We then plotted histograms of the final x and y positions, which we found to be Gaussian distributions. The standard deviation and mean for the x-positions are 0.90 and 20.86 respectively, and 0.48 and 22.04 respectively for y-positions.

Figure 1.5: We found that the square of end-to-end distances $R_d^2(n)$ is modulus of the vector sum. That is, let $\vec{R}$ be the end-to-end distance vector of a random walk of fixed step length $|\vec{r}_i| = l$. So, $\vec{R_d} = \sum_{i=1}^{N} \vec{r}_i$, where $\vec{r}_i$ is the vector of the i-th step.

1.2 Random Walks of Variable Step Length

Imagine a small ant navigating through a park, taking steps of varying lengths with each move. Sometimes it takes a short step, other times a longer leap. This seemingly haphazard

movement mirrors the concept of a variable-length random walk. We start by considering a scenario where the length of each step follows a specific probability distribution. Picture this as the ant choosing step lengths from a list, with some lengths more probable than others.

To understand this better, we introduce the probability density function $f(a)$, representing the likelihood of a step falling within a range of lengths from a to $a + \Delta a$. For our first scenario, $f(a)$ adopts the form Ce^{-a} for $a > 0$, where C ensures that probabilities sum up to unity, creating a normalised distribution. This exponential decay captures the diminishing likelihood of longer steps, akin to the ant occasionally taking large strides but mostly opting for smaller ones. The probability distribution is fundamental to understanding random walks as it represents the likelihood of different outcomes or events occurring during the walk. In the context of random walks with variable step lengths, the probability distribution of step lengths directly influences the overall behaviour of the walk and the resulting probability distribution of displacements.

To simulate this scenario computationally, we employ an algorithm to generate step lengths according to $f(a)$. The resulting probability distribution $p(x)\Delta x$, showcasing the probability of displacement falling within x and $x + \Delta x$. But our exploration does not halt there. We venture into a more exotic territory where step lengths follow a power-law distribution $f(a) = C/a^2$ for $a \geq 1$. This is the case of *Lévy flights*. Through Monte Carlo simulations, we dissect the behaviour of these Lévy flights, uncovering their distinct signatures in $p(x)$ and assessing whether they conform to the Gaussian distribution or something else.

In short, the two following probability distributions have been chosen out of which we choose our step lengths:

1. $f(a) = Ce^{-a}$ with the normalisation condition $\int_0^\infty Ce^{-a} = 1$ The step length for this distribution is given by

```
step_length = - np.log(1 - np.random.uniform(0, 1))
```

2. $f(a) = \frac{C}{a^2}$, with the normalisation condition $\int_0^\infty \frac{C}{a^2} = 1$. The step length in the case of Lévy flights is given by

```
step_length = 1 / (1 - np.random.uniform(0, 1))
```

Model and Methods

2.1 Model

In this project, we simulated random walks with variable step lengths, exploring two probability density functions. We generated step lengths based on these probability distributions and verified if they conform to it for N walks. We also tried find $p(x)\Delta x$ which is the probability that the displacement is between x and $x + \Delta x$ after N steps. Here, $p(x)$ is the probability density function.

In both the Lévy and the exponential problems, we notice that our probability distribution is continuous. However, for practical purposes, we need to discretise the distributions so as to pick our step sizes from them. We can do this by introducing a bin width Δa. An assumption we are making here is that the walkers' steps are independent and unbiased, which means that at every step, the probability of moving left or right is the same, and the decision to do so does not depend on the previous step.

For simulating random walks with steps of variable length based on the given probability density functions, we employed a straightforward algorithm. First, we addressed the case where the probability density function is $f(a) = Ce^{-a}$ for $a > 0$, with the normalisation condition $\int_0^\infty f(a)\,da = 1$. To generate step lengths according to this distribution, we used stepLength $= -$np.log$(1 -$ np.random.uniform$(0, 1))$ in our code implementation.

Subsequently, for Lévy flights, we worked with $f(a) = C/a^2$ for $a \geq 1$, using the normalisation condition $\int_1^\infty f(a)\,da = 1$. Here, we generated values of a according to this distribution by employing stepLength $= 1.0/(1.0 -$ np.random.uniform$(0, 1))$. By calculating the integral, we found that in both cases, $C = 1$.

In conducting our simulations, we set the bin width Δa as one of the input parameters. Additionally, we ensured a minimum number of steps $N \geq 100$ for each walk to capture sufficient data. These parameters were chosen to strike a balance between computational efficiency and statistical accuracy in our simulations. Furthermore, we specified other relevant parameters such as the number of walks performed to generate statistical significance in our results. These parameters were crucial in determining the probability that the displacement is within a specified range after N steps, enabling us to analyse the distribution characteristics effectively.

2.2 Methods

2.2.1 Exponential Probability Distribution

Starting at the origin, the random walker chooses a particular step length randomly from the probability distribution function

$$f(a) = Ce^{-a}, \tag{2.1}$$

where a is the step length, after which the walker randomly moves towards the left or the right randomly. A step in either direction is equally possible. Each successive position is then recorded to plot the trajectory of the walker. A large number of steps is preferred to ensure that

the intrinsic stochastic nature of the process evens out to display any patterns in the behaviour of the walker. We chose to plot walks of a thousand steps.

After the trajectory for a single walk was modelled, we moved on to check for the behaviour of the random walker over a large number of walks, 1000 walks in this case, of 1000 steps each, to analyse the trends of the behaviour of the random walker over a large sample size for statistical accuracy, since doing so reduces the influence of outliers on our interpretation of the behaviour of the random walker. The displacement from the origin – simply modulus of the end point of the trajectory for each walk – was considered and then plotted over a histogram for a thousand walks.

2.2.2 Lévy Distribution

A Lévy flight, too, is a random walk in which the step lengths are drawn randomly, but instead of the exponential probability distribution function used in the first case, the step lengths are derived from the probability distribution function

$$f(a) = \frac{C}{a^2}, \tag{2.2}$$

where a is the step length. The probability of moving left or right, again, is 0.5 each. We then plotted the trajectory of a single Lévy flight and the displacements for a thousand walks of thousand steps with step lengths varying according to the power law probability distribution.

2.3 Algorithm

Exponential Distribution

Single 1D Walk

To simulate a random walk with exponential distribution of step lengths, we used the following algorithm:

```
1. Random Walk Function:
   - Initialise walker's position to 0.
   - Generate step length using -np.log(1 - np.random.random()).
   - Randomly determine step direction and update position.
   - Store positions and step lengths at each step in an array.
   - Return positions and step lengths.

2. Simulation:
   - Call random walk function with number of steps as a parameter in the function.
   - Plot positions to visualise walker's trajectory.
```

Multiple Random Walks

For simulating multiple random walks with exponential distribution, the following was our algorithm:

```
1. Multiple Walks Function:
   - Initialise n_walks and n_steps.
```

```
   - Loop over n_walks, call random walk function, and store positions.
   - Store all positions and step lengths in an array.
   - Plot positions of each walk.
```

Displacement Probability Distribution

To analyse the probability distribution of displacements for exponential random walks, we performed the following steps:

```
1. Monte Carlo Simulation:
   - Generate multiple walks using random walk function.
   - Calculate mean and variance of displacements for Gaussian fit.
   - Plot histogram of displacements with Gaussian fit.
```

Plotting End-to-End Distance Squared against Number of Steps

For analysing the end-to-end distance squared in exponential random walks, we followed these steps:

```
1. End-to-End Distance Calculation:
   - Define function to calculate squared distance.
   - Compute squared distance for each walk.
   - Plot end-to-end distance squared vs number of steps.
```

Lévy Distribution

Single Lévy Flight

To simulate a single Lévy flight with Lévy distribution, we implemented the following algorithm:

```
1. Lévy Flight Function:
   - Initialise walker's position to 0.
   - Generate step length using 1 / (1 - np.random.random()).
   - Randomly determine step direction and update position.
   - Return positions representing Lévy flight.

2. Simulation and Visualisation:
   - Call Lévy flight function with number of steps.
   - Plot positions to visualise Lévy flight trajectory.
```

Finding Probability Distribution for Final Displacement in Multiple Lévy Flights

For simulating multiple Lévy flights and analyzing the displacement distribution, we used the following approach:

```
1. Lévy Flight Function:
   - Generate step lengths using 1 / (1 - np.random.random()).
   - Compute cumulative sum to get displacement.

2. Monte Carlo Simulation:
   - Generate multiple flights using Lévy flight function.
   - Plot histogram of displacements for probability density.
```

Additionally, we also simulated a 2D Lévy Flight. The *Lévy_flight()* function in our code generates random step sizes and angles based on a Cauchy distribution and uniform distribution, respectively. These steps are then transformed into `x` and `y` coordinates using the angles. Using the Lévy index parameter *alpha*, the step lengths are calculated using a combination of the Cauchy and exponential distributions. The cumulative sum of these steps gave us the trajectory of the flight.

Note. We expect the 2D Lévy flight to look something like the below image.

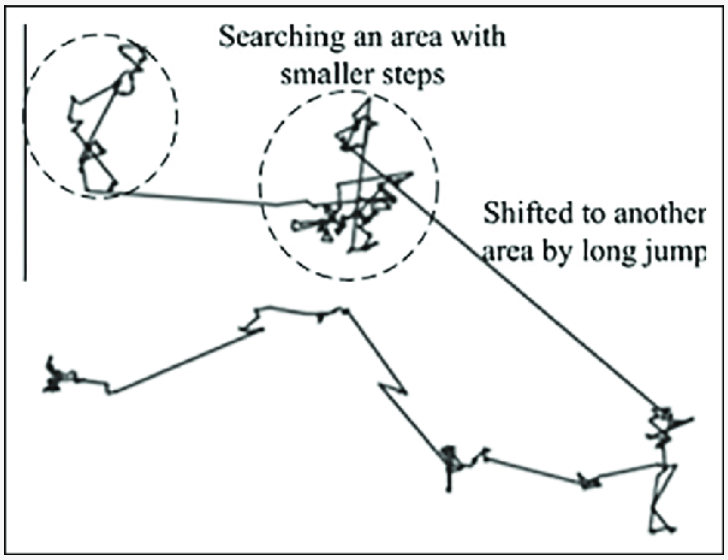


Figure 2.1: A two-dimensional Lévy flight. Image Source: *Research Gate*

In 1999, physicists from Brazil made a significant finding regarding the importance of Lévy walks in the foraging behaviour of birds. They observed that when food is scattered in sparse patches that regenerate rapidly, and animals lack sensory cues to locate the food, Lévy walks emerge as the mathematically optimal strategy for effectively finding food without guidance.

Additionally, Lévy flights have been important in cosmology as simplified models for studying how matter distributed in the universe due to gravitational effects.

Final Remarks

3.1 Results

The probability distribution functions for the two different step length distributions are displayed. It is evident from these plots that the functions for both probability distributions are distinct. The exponential probability distribution exhibits a decaying exponential decrease in the probability with an increase in the size of steps, whereas the heavy-tailed probability distribution follows an inverse square relationship, as is expected. These differences in functional forms directly influence the characteristics of the random walk trajectories and the resulting final position distributions.

The trajectory plot of a single walk of the exponential probability distribution illustrates the path taken by a walker on a single random walk. From the plot, one can see that the number of large jumps taken by the walker throughout the path is relatively low.

In a heavy tailed probability distribution or the Levy Flight, the probability of the walker taking much larger jumps is significantly higher than that in the exponential probability distribution. As a result, one can see a much higher number of instances of very large steps taken.

What this results in is evident in the probability density distributions for the displacements of the random walker over a thousand walks, one each for the two different probability distribution functions – the exponential probability distribution and the levy flight. In the case of the former, one can see that the the probability distribution is approximately a Gaussian distribution with a peak close to the origin, if not exactly at it. After a net displacement of around hundred units from the origin, the probability of the occurrence of displacements higher than hundred units is nearly zero, other than perhaps a few outliers.

In contrast, for the Levy Flight, in the histogram for the probability distributions, one can see that the probability of net displacements much higher than a hundred units from the origin is much higher than that of the exponential probability distribution, despite the fact that the distribution does still show a great bias toward the lower end of the displacements.

It is evident that the trajectory of the walker varies significantly depending on the probability distribution of step lengths. Walks generated with the exponential probability distribution tend to include shorter steps with smaller variations in position, resulting in a more localized trajectory. Conversely, walks generated with the heavy-tailed probability distribution exhibit longer steps with greater variability in position, leading to a more dispersed trajectory.

What this means for our random walker is that, if he takes steps following the exponential probability distribution, he would tend to meander, ending up where he started. However, if his step lengths are determined according to the power law distribution, he actually has a chance of getting somewhere.

3.2 Graphs

Exponential Distribution

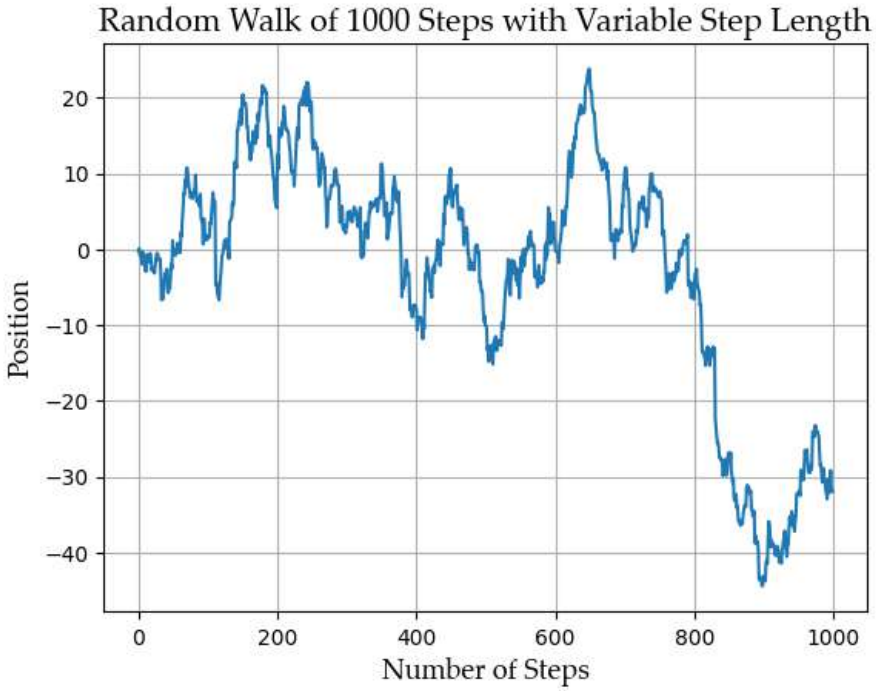


Figure 3.1: The graph plotting the trajectory of a random walker with step lengths following the exponential probability distribution.

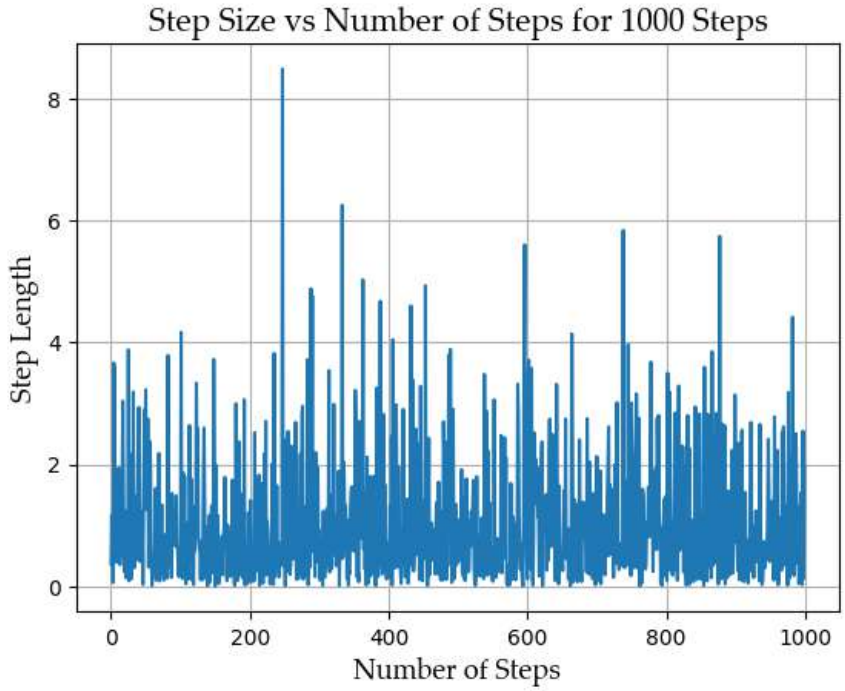


Figure 3.2: The plot of the step size against the number of steps taken for the exponential probability distribution.

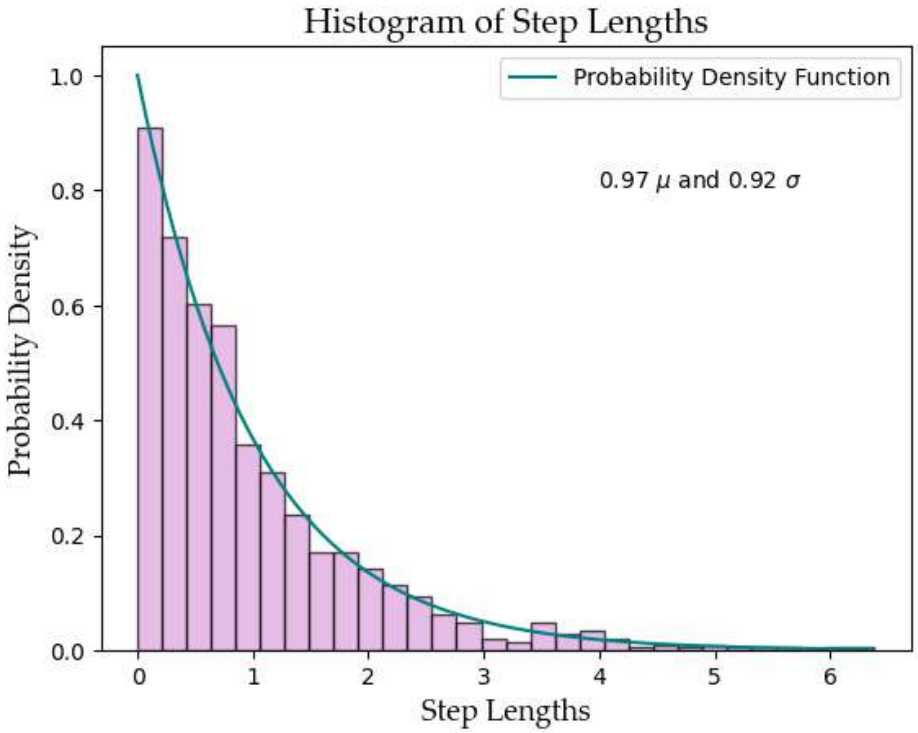


Figure 3.3: The histogram of the probability densities against step lengths, fitted to an exponential probability distribution. It has a mean of 0.97 and a standard deviation of 0.92.

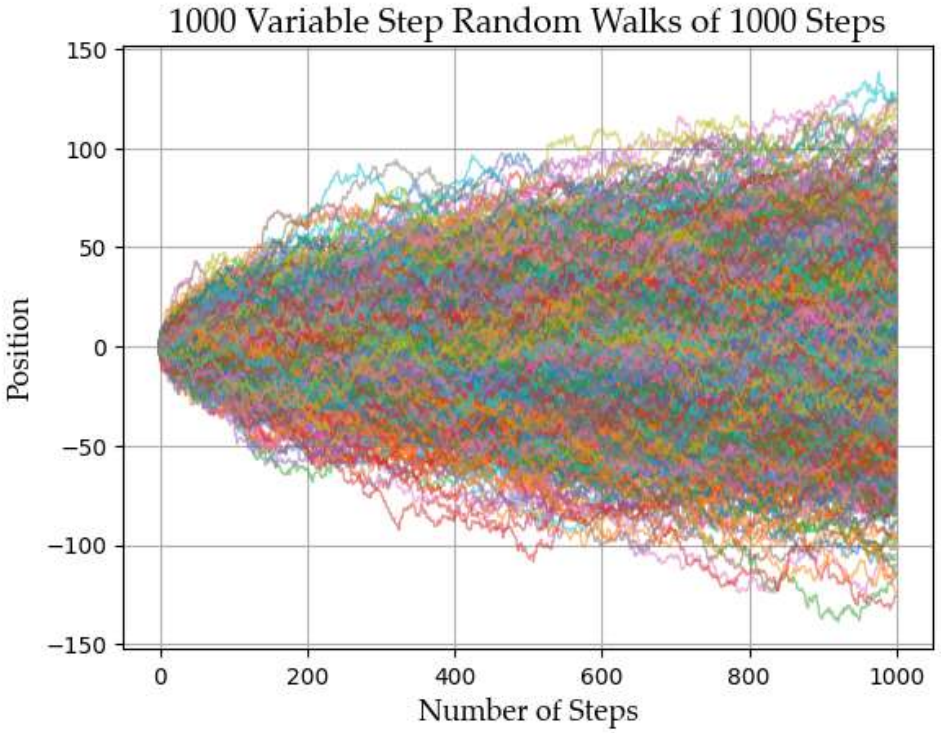


Figure 3.4: The trajectories of a thousand random walks following the exponential probability distribution.

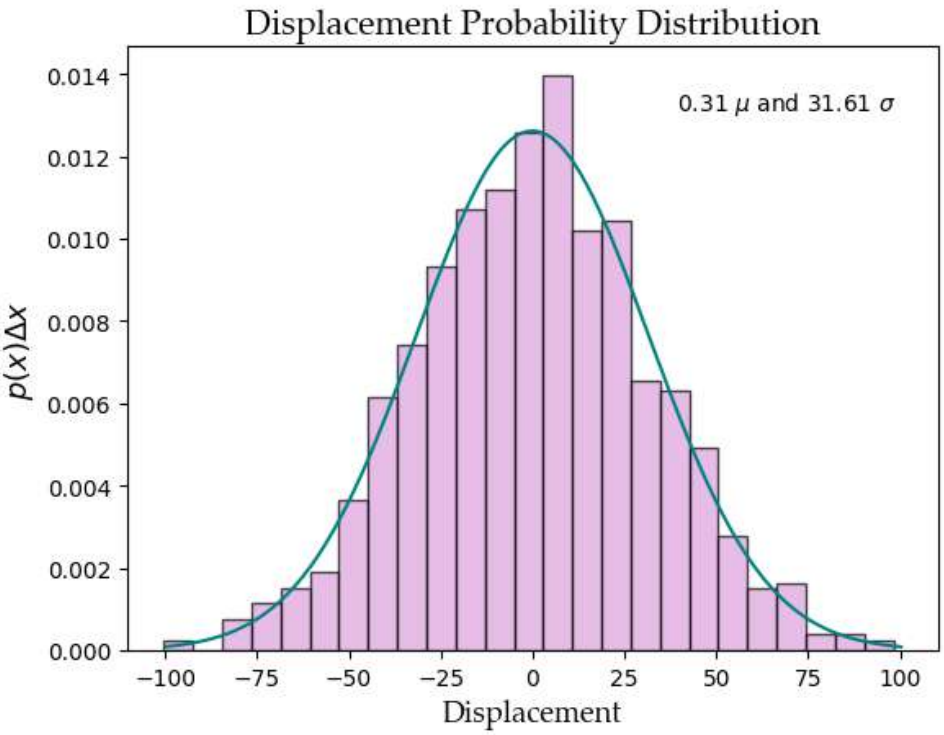


Figure 3.5: The probability distribution of the displacements of the random walker following the exponential probability distribution function. The distribution has a mean of 0.31 and a standard deviation of 31.51

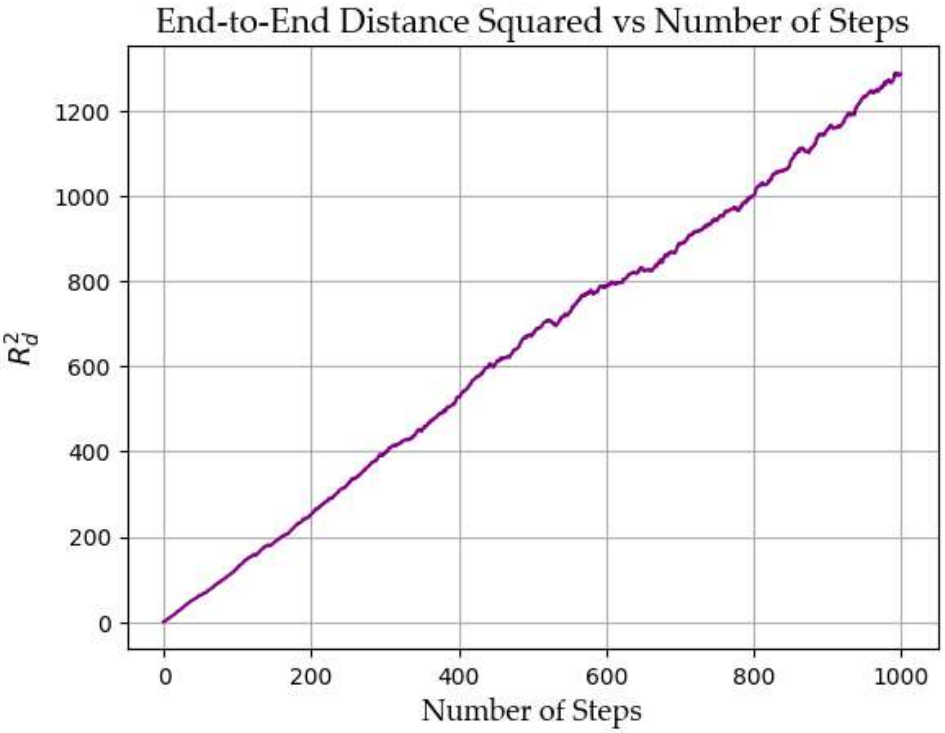


Figure 3.6: The square of the end to end distances plotted against the number of steps for a random walker following the exponential distribution.

Lévy Distribution

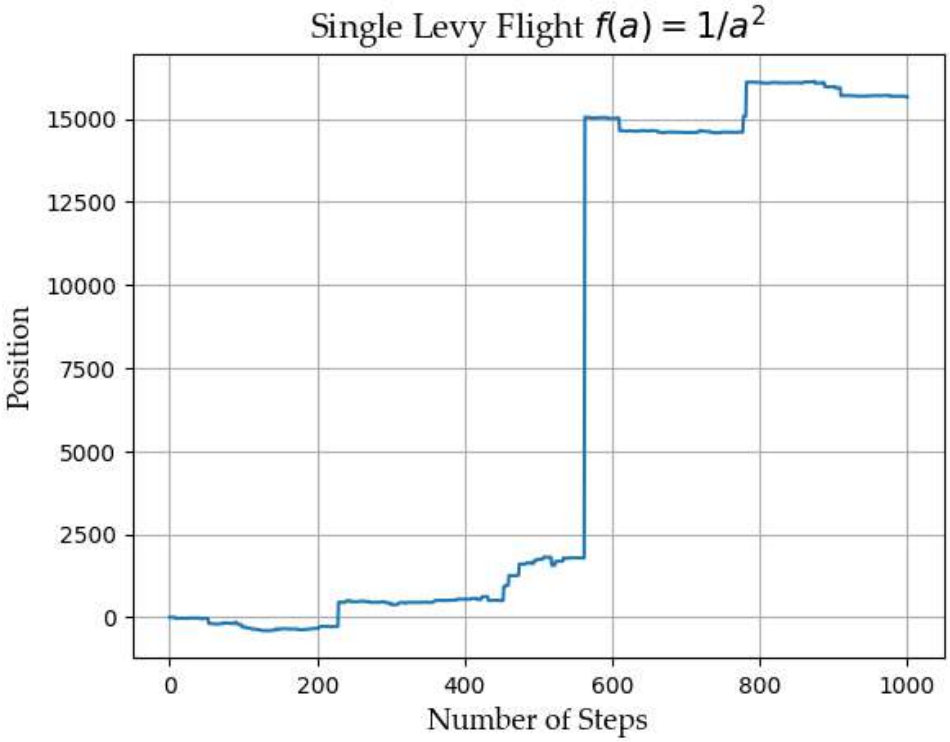


Figure 3.7: A single Lévy flight trajectory.

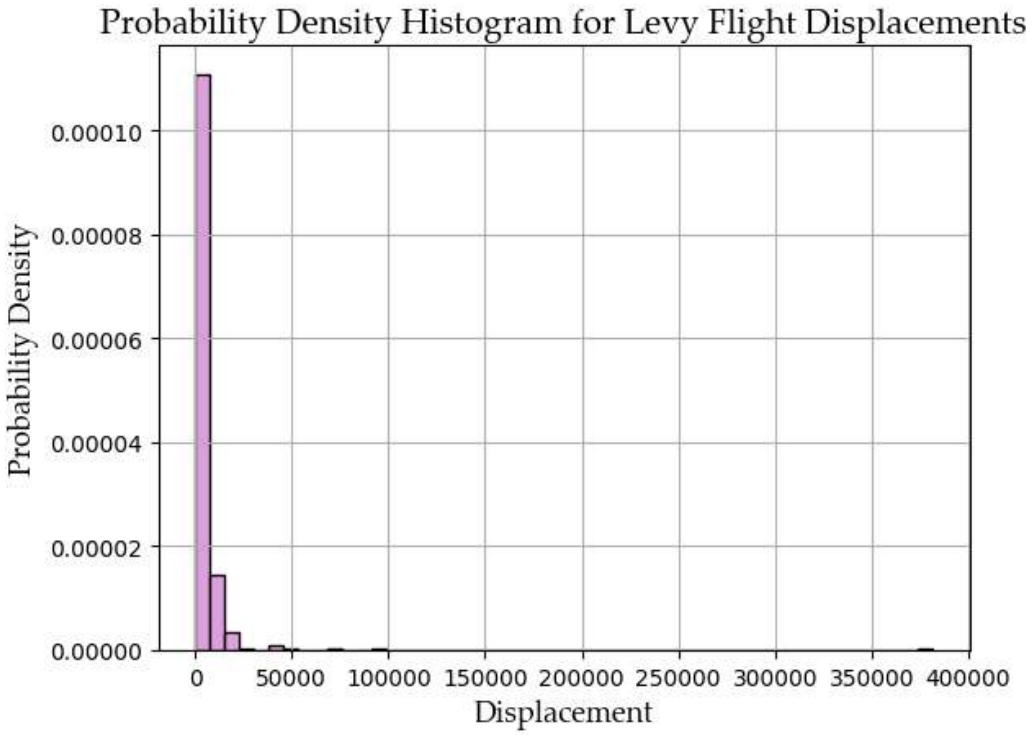


Figure 3.8: The probability density plotted for different displacements for Lévy flights, with a mean of 0.97 and a standard deviation of 0.92.

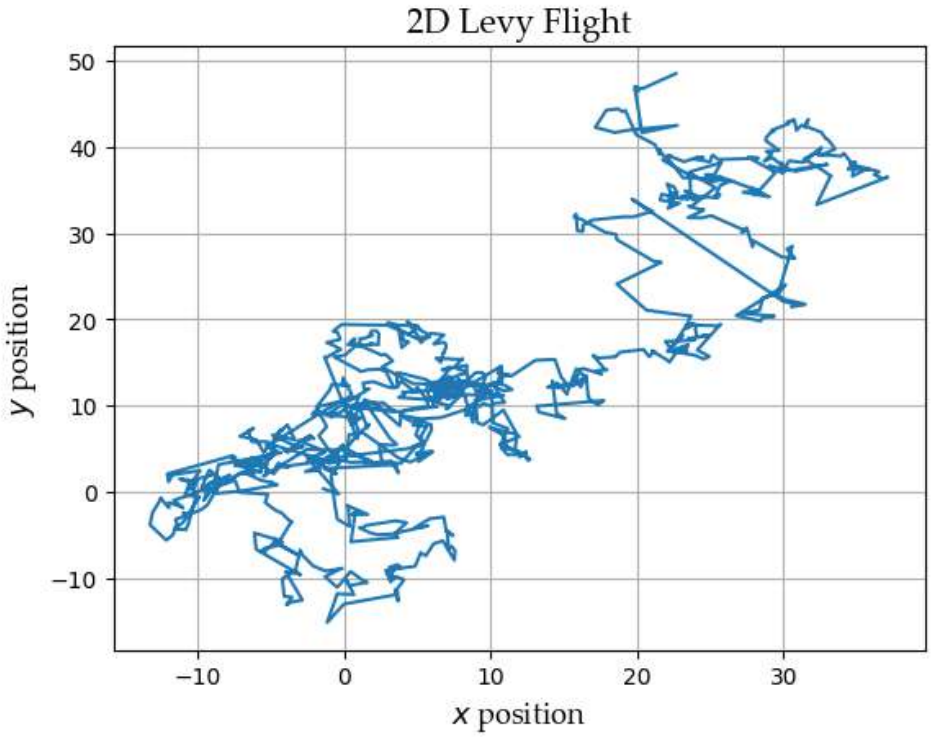


Figure 3.9: A two-dimensional Levy flight

3.3 Conclusion

The project delved into simulating random walks with variable step lengths, focusing on two distinct probability density functions: exponential and Lévy distributions. The aim was to generate step lengths based on these distributions, validate their conformity through multiple walks, and analyse the probability density functions' behaviour for displacements after a certain number of steps.

In both scenarios, the distributions were continuous, so we required discretisation for practical implementation. This was achieved by introducing a bin width parameter to choose step sizes from the distributions. It was assumed that the steps were unbiased and independent, meaning the decision to move left or right was not influenced by previous steps.

The algorithms for generating step lengths for exponential and Lévy distributions were straightforward. For the exponential case, step lengths were derived from $f(a) = Coe^{-a}$, while for Lévy flights, $f(a) = \frac{C}{a^2}$ was used. Monte Carlo simulations were employed to analyse displacements and plot probability distributions, ensuring statistical significance with parameters like bin width and number of steps.

The results highlighted distinct behaviours based on the probability distributions. Exponential distributions favoured localised trajectories with smaller variations, while Lévy flights exhibited longer steps and greater positional variability, leading to more dispersed trajectories. These findings underscored the significant impact of probability distributions on random walk trajectories and the resulting displacement probability distributions, offering insights into optimal search strategies and behaviour patterns in stochastic processes.

Appendix

4.1 Code for a Single 1D Random Walk

Here is a function *walk_1D()* that simulates a 1D random walk. The function accept two arguments – the probability of the walker moving right (p) and the number of steps (*n_steps*). Additionally, position vs number of steps has been for the single 1D random walk. First, we create an array, *pos*, to store the positions of the walker at each step. Subsequently, the function iterates through the specified number of steps, randomly determining the direction of each step based on the probability p. The walker moves left (-1) or right (1) with equal likelihood, and their position is updated accordingly in the *pos* array. The function returns the final position array. The code then generates a plot visualising the progression of the random walk.

```
    n_steps = 1000
    p = 0.5

    def walk_1D(n_steps, p):
        global pos
        pos = np.zeros(n_steps) # Stores positions
        for i in range(1, n_steps):
            if np.random.rand() <= p:
                step = -1
            else:
                step = 1
            pos[i] = pos[i - 1] + step
        return pos

    print('Array of Positions: ', walk_1D(n_steps, p)) # Array with
        positions

    steps = np.arange(0, n_steps, 1)

    # Plotting the walk
    plt.plot(steps, pos)
    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('Position', **hfont)
    plt.title('1D Random Walk', **csfont)
    plt.grid(True)
    plt.show()
```

4.2 Code for N 1D Random Walks

Here, we simulate a large number of walks, that is, 1000 walks for $p = 1/2$ and record the trajectory of each random walker by storing them in an array. This code is like running a thousand little experiments where a tiny dot takes random steps on a straight line. Imagine these dots are having a stroll, and we're watching where they end up after a certain number of steps. We want to see if there's any pattern in their wanderings. So, we let each dot take a walk, record its path, and then do this a thousand times! The code then plots all these paths on a single graph, so we can see the overall picture of where the dots tend to end up. This helps us understand how randomness works and what kind of patterns might emerge from it.

Simulation

```
    n_walks = 1000  # Number of walks to simulate
    pos2 = [walk_1D(n_steps, p) for _ in range(n_walks)]

    for i in range(n_walks):
        plt.plot(steps, pos2[i], linewidth=1, alpha=0.5)  # Each
           step in a walk --> [i, :]

    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('Position', **hfont)
    plt.title('1000 One-Dimensional Random Walks', **csfont)
    plt.grid(True)
    plt.show()
```

Histogram of Final Positions

We then plot a histogram of the final positions of all the walkers. We notice that it is a Gaussian with mean $\mu \approx 0$ and standard deviation $\sigma \approx \sqrt{\text{n\_steps}}$

```
    finpos = [i[-1] for i in pos2]

    mu = np.mean(finpos)
    std = np.std(finpos)
    var = np.var(finpos)
    x_array = np.linspace(mu-100, mu+100, 201)    # choosing sample
        points to draw the curve
    g_array = 1./np.sqrt(2*np.pi*var) * np.exp(-0.5*(x_array-mu)
       **2/var)    # Gaussian curve

    plt.hist(finpos, bins=30, density = True, alpha=0.6, color='
       plum', edgecolor='black')
    plt.plot(x_array, g_array, color='teal', label='Gaussian fit')

    plt.figtext(0.15, 0.83, "{:.2f} $\mu$ and {:.2f} $\sigma$".
       format(mu, std))
    plt.xlabel('Final Position', **hfont)
```

```
    plt.ylabel('Probability Density', **hfont)
    plt.title('Histogram of Final Positions', **csfont)
    plt.legend()
    plt.show()
```

Relation Between Square of End-to-End Distance and Number of Steps

Now, we are interested in not just about where each dot ends up, but also how far it has traveled overall. We define a function *Rd()* that calculates something called the "end-to-end distance" $R_d(n)$ for each step of the walk. It is like measuring the total distance traveled by the dot from its starting point for each step. The function *Rd()* takes *pos2*, which contains all the positions of each dot over each step. We calculate the distances traveled for all dots at each step, then average them out. To make things clearer, we square the average distances to emphasise the differences better. Once we have these distances, we plot them against the number of steps. Each point on the graph represents how far, on average, the dots have travelled after a certain number of steps.

```
    # End-to-end positions
    def Rd(pos2):
        n_steps = len(pos2[0]) - 1 # as indices start from 0.
        end_to_end_distances = np.zeros(n_steps + 1)
        for i in pos2:
            for j in range(1, n_steps + 1):
                end_to_end_distances[j] += abs(i[j] - i[0])
        return (end_to_end_distances/len(pos2))**2

    all_dis = Rd(pos2)
    plt.plot(range(len(all_dis)), all_dis, color = 'purple')
    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('$R_{d}^2$', **hfont)
    plt.title('End-to-End Distance Squared vs Number of Steps', **
       csfont)
    plt.gca().set_aspect("equal")
    plt.grid(True)
    plt.show()

    # Slope of Graph
    m, c = np.polyfit(range(len(Rd)), Rd, deg = 1)
    print('Slope:', m)
```

4.3 Code for Two-Dimensional Random Walks

We now generalise the above problem to an N-step random walk on a two-dimensional grid. In such a walk, the walker starts off at the origin and, at every time step, moves a fixed amount in either the x or y directions. Thus, at every time-step, they randomly choose between one

of four possibilities. This is then repeated N times. We also plot the histograms and find that the square of end-to-end distances $R_d^2(n)$ is modulus of the vector sum.

Single 2D Walk

This function *walk_2D(n_steps)* creates a two-dimensional random walk with a specified number of steps *n_steps*. It initialises arrays for the x and y coordinates, starting at (0, 0). The walk progresses by randomly choosing directions for left/right (LR) and up/down (UD) movements at each step. If LR is 1 (right), the x-coordinate increases by 1; if LR is 0 (left), it decreases by 1. Similarly, if UD is 1 (down), the y-coordinate decreases by 1; if UD is 0 (up), it increases by 1. The function returns arrays of x and y coordinates representing the random walk's path. The code then generates and stores a specific two-dimensional random walk with 1000 steps, storing its x and y coordinates in *x_steps* and *y_steps*.

```
    def walk_2D(n_steps):
        x, y = np.zeros(n_steps), np.zeros(n_steps)
        for i in range(1, n_steps):
            direction_lr = random.randint(0, 1)  # 0 for L, 1 for R
            direction_ud = random.randint(0, 1)  # 0 for U, 1 for D
            x[i] = x[i-1] + (1 if direction_lr == 1 else -1) * (1
               if direction_ud == 1 else 0)
            y[i] = y[i-1] + (1 if direction_lr == 0 else -1) * (1
               if direction_ud == 0 else 0)
        return x, y

    n_steps = 1000
    x_steps, y_steps = walk_2D(n_steps)

    plt.plot(x_steps, y_steps)
    plt.xlabel('$x$-position', **hfont)
    plt.ylabel('$y$-position', **hfont)
    plt.plot(x_steps[0], y_steps[0], marker='o', markersize=6,
       color='green', label='Start')
    plt.plot(x_steps[-1], y_steps[-1], marker='o', markersize=6,
       color='red', label='End')
    plt.title('2D Random Walk', **csfont)
    plt.gca().set_aspect("equal")
    plt.grid(True)
    plt.legend()
    plt.show()
```

Multiple 2D Random Walks

This code conducts a simulation of 1000 two-dimensional random walks and visually represents their trajectories on a plot. Each walk's x and y coordinates are generated using a function called *walk_2D()*, and these coordinates are plotted with a line connecting successive points. The resulting plot showcases the collective patterns of the random walks.

```
    # N walks
```

```
    n_walks = 1000
    x_walks = []
    y_walks = []

    for i in range(n_walks):
        x_walk, y_walk = walk_2D(n_steps)
        plt.plot(x_walk, y_walk, linewidth=1, alpha=0.5)
        x_walks.append(x_walk)
        y_walks.append(y_walk)

    plt.xlabel('$x$-position', **hfont)
    plt.ylabel('$y$-position', **hfont)
    plt.title('1000 Two-Dimensional Random Walks', **csfont)
    plt.gca().set_aspect("equal")
    plt.grid(True)
    plt.show()
```

Histogram for Final x and y Positions

```
    finxpos = [i[-1] for i in x_walks]
    finypos = [j[-1] for j in y_walks]

    plt.figure(figsize=(10, 4))

    mux, muy = np.mean(finxpos), np.mean(finypos)
    stdx, stdy = np.std(finxpos), np.std(finypos)
    varx, vary = np.var(finxpos), np.var(finypos) # For plotting
       the expected Gaussian

    x_array1, g_array1 = np.linspace(mux-100, mux+100, 201), 1./np.
       sqrt(2*np.pi*varx) * np.exp(-0.5*(x_array-mux)**2/varx)
    x_array2, g_array2 = np.linspace(muy-100, muy+100, 201), 1./np.
       sqrt(2*np.pi*vary) * np.exp(-0.5*(x_array-muy)**2/vary)

    # x Histogram
    plt.subplot(1, 2, 1)
    plt.hist(finxpos, bins = 20, density = True, alpha=0.6, color='
       plum', edgecolor='black')
    plt.plot(x_array1, g_array1, color='teal', label='Gaussian fit'
       )
    plt.figtext(0.345, 0.83, "{:.2f} $\mu$ and {:.2f} $\sigma$".
       format(mux, stdx))
    plt.xlabel('Final $x$ Position', **hfont)
    plt.ylabel('Probability Density', **hfont)
    plt.title('X Histogram', **csfont)

    # y Histogram
```

```
    plt.subplot(1, 2, 2)
    plt.hist(finypos, bins = 20, density = True, alpha=0.6, color='plum', edgecolor='black')
    plt.plot(x_array2, g_array2, color='teal', label='Gaussian fit')
    plt.figtext(0.830, 0.83, "{:.2f} $\mu$ and {:.2f} $\sigma$".format(muy, stdy))
    plt.xlabel('Final $y$ Position', **hfont)
    plt.ylabel('Probability Density', **hfont)
    plt.title('Y Histogram', **csfont)

    plt.tight_layout()
    plt.show()
```

Relation Between Square of End-to-End Distance and Number of Steps

```
    def Rd_2D(x_walks, y_walks):
        n_steps = len(x_walks[0]) - 1
        Rd = np.zeros(n_steps + 1)
        for x_walk, y_walk in zip(x_walks, y_walks):
            for i in range(1, n_steps + 1):
                Rd[i] += (x_walk[i] - x_walk[0])**2 + (y_walk[i] - y_walk[0])**2
        return (Rd/len(x_walks))**2

    Rd2 = Rd_2D(x_walks, y_walks) # End-to-end distance squared for each walk
    plt.plot(range(len(Rd2)), Rd2, color = 'purple')
    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('$R_{d}^2$', **hfont)
    plt.title('End-to-End Distance Squared vs Number of Steps', **csfont)
    plt.grid(True)
    plt.show()
```

4.4 Code for Variable Step Length Random Walks

4.4.1 Exponential Distribution

Single Walk

Imagine a stroll where every step is a surprise, with lengths ranging from tiny hops to giant strides. The likelihood of each step falling between a certain range, say a to $a + \Delta a$, follows a mathematical pattern called a probability density function $f(a)$. In our case, $f(a)$ takes the

form $f(a) = Ce^{-a}$ for $a > 0$, ensuring that all possible step lengths are covered and probabilities add up to 1 across the spectrum. The step sizes are given by this snippet:

step_length = $-\ln(1 - \text{np.random.uniform}())$

The width of each *bin* in our step length spectrum, denoted by Δa, is something we can tweak. Let's consider at least 100 walks and observe their journey unfold. By repeating these walks numerous times, we can piece together the probability $p(x)\Delta x$, showing how likely it is for the walker to end up between x and $x + \Delta x$ after completing the set number of steps N. Plotting $p(x)$ against x reveals a pattern that we hope resembles the familiar bell curve of a Gaussian distribution.

```
    def random_walk(n_steps):
        position = 0
        positions = []
        step_length = []
        for i in range(n_steps):
            step = -np.log(1 - np.random.random()) # step length
            if np.random.random() < 0.5:
                position += step
            else:
                position -= step
            positions.append(position)
            step_length.append(step)
        return positions, np.array(step_lengths)

    num_steps = 1000

    positions, step_lengths = random_walk(num_steps)

    plt.plot(np.array(positions))
    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('Position', **hfont)
    plt.grid(True)
    plt.title('Random Walk of 1000 Steps with Variable Step Length'
       , **csfont)
    plt.show()
```

Plotting Step Length for Each Step

```
    def random_walk2(N, delta_a): # An alternate way to get
       positions and step lengths
        step_lengths = -np.log(1 - np.random.random(N))
        displacement = np.sum(step_lengths) - N * delta_a / 2  #
           Correcting for bin width
        return step_lengths, displacement

    # Parameters
    N = 1000
    delta_a = 1  # Bin width
```

```
9
10      step_lengths, displacement = random_walk2(N, delta_a)
11
12      plt.plot(np.arange(N), step_lengths)
13      plt.xlabel('Number of Steps', **hfont)
14      plt.ylabel('Step Length', **hfont)
15      plt.grid(True)
16      plt.title('Step Size vs Number of Steps for 1000 Steps', **
            csfont)
17      plt.show()
```

Histogram of Step Lengths

```
1       # Histogram
2       plt.hist(step_lengths, bins=30, density=True, alpha=0.7, color=
            'plum', edgecolor='black')
3
4       mua = np.mean(step_lengths)
5       stda = np.var(step_lengths)
6
7       # Probability density function
8       a_values = np.linspace(0, max(step_lengths), 100)
9       plt.plot(a_values, np.exp(-a_values), 'teal', label='
            Probability Density Function')
10      plt.figtext(0.60, 0.70, "{:.2f} $\mu$ and {:.2f} $\sigma$".
            format(mua, stda))
11
12      plt.xlabel('Step Lengths', **hfont)
13      plt.ylabel('Probability Density', **hfont)
14      plt.title('Histogram of Step Lengths', **csfont)
15      plt.legend()
16      plt.show()
```

Multiple Variable Step Length Walks

```
1       n_walks = 1000  # Number of walks
2       n_steps = 1000  # Number of steps
3
4       all_positions = []
5       all_step_lengths = []
6
7       for i in range(n_walks):
8           positions, step_lengths = random_walk(n_steps)
9           all_positions.append(positions)
10          all_step_lengths.append(step_lengths)
11
12      for i in range(n_walks):
```

```
        plt.plot(all_positions[i], label=f'Walk {i+1}', alpha =
            0.5, linewidth = 1)

    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('Position', **hfont)
    plt.grid(True)
    plt.title(f'{n_walks} Variable Step Random Walks of {n_steps}
        Steps', **csfont)
    plt.show()
```

Histogram of Final Positions for N Variable Step Length Random Walks

```
    all_finpos = np.array([random_walk2(n_steps, 1)[1] for i in
        range(n_walks)]) # All final positions
    mua2, vara2 = np.mean(all_finpos - 500), np.var(all_finpos -
        500) # Shift of 500 due to a small error
    xv_array = np.linspace(mua2 - 100, mua2 + 100, 201)
    gv_array = 1. / np.sqrt(2 * np.pi * vara2) * np.exp(-0.5 * (
        x_array - mua2) ** 2 / vara2)

    plt.hist(all_finpos - 500, bins=25, density = True, color='plum
        ', edgecolor='black', alpha=0.7)
    plt.plot(xv_array, gv_array, color='teal', label='Gaussian fit'
        )
    plt.xlabel('Displacement', **hfont)
    plt.ylabel('$p(x) \Delta x$', **hfont)
    plt.title('Displacement Probability Distribution', **csfont)
    plt.figtext(0.65, 0.80, "{:.2f} $\mu$ and {:.2f} $\sigma$".
        format(np.mean(np.array(all_finpos - 500)), np.std(np.array(
        all_finpos - 500))))
    plt.show()
```

Relation Between Square of End-to-End Distance and Number of Steps

```
    all_disv = Rd(all_positions) # Using the old Rd() function
    plt.plot(range(len(all_disv)), all_disv, color = 'purple')
    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('$R_{d}^2$', **hfont)
    plt.title('End-to-End Distance Squared vs Number of Steps', **
        csfont)
    # plt.gca().set_aspect("equal")
    plt.grid(True)
    plt.show()
```

4.4.2 Lévy Flights

Single Lévy Flight

Imagine a scenario where each step's length obeys a different rule, following the probability density function $f(a) = C/a^2$ for $a \geq 1$. To ensure our probabilities stay balanced across all possible step lengths, we try to the normalisation constant C. This constant makes sure that when we integrate $f(a)$ from 1 to infinity, the result equals 1. With C in hand, generating step lengths according to this distribution becomes easy with the code snippet:

$$\text{step\_length} = 1/1 - \text{np.random.random}()$$

This gives us the step lengths that match with our power-law probability density function. We can now perform a Monte Carlo simulation, where we find the probabilities $p(x)\Delta x$ for the walker's displacement after a number of steps. This is essentially known as Lévy flights, where the probability density decreases following a power law $a^{-1-\alpha}$, introducing us to intriguing patterns when $\alpha \leq 2$.

To determine the normalisation constant C, we integrate the probability density function $f(a)$ over its valid range:

$$C = \int_1^\infty \frac{1}{a^2}\, da$$

$$C = \left[-\frac{1}{a}\right]_1^\infty$$

Since $\lim_{a\to\infty} \frac{1}{a} = 0$, the upper limit contributes zero, leaving us with:

$$C = -\left(-\frac{1}{1}\right) = 1$$

So,

$$C = 1$$

.

For $\alpha \leq 2$, the form of $p(x)$ is not Gaussian; it follows a power-law distribution due to the Levy flight behaviour.

```
    def levy(steps):
        position = 0
        positions = [position]
        for i in range(steps):
            step = 1/(1 - np.random.random())
            if np.random.random() < 0.5:
                position += step
            else:
                position -= step
            positions.append(position)
        return positions

    positions = levy(1000) # Levy flight of 1000 steps

```

```
    plt.plot(positions)
    plt.xlabel('Number of Steps', **hfont)
    plt.ylabel('Position', **hfont)
    plt.grid(True)
    plt.title('Single Levy Flight $f(a) = 1/a^{2}$', **csfont)
    plt.show()
```

Probability Distribution for Multiple Lévy Walks

```
    def levy_flight(n_steps):
        step_lengths = 1 / (1 - np.random.random(n_steps))
        return np.cumsum(step_lengths - 1) # Displacement

    n_walks = 200
    n_steps = 1000

    # 1000 Levy flights
    all_displacements = []
    for _ in range(num_walks):
        displacements = levy_flight(num_steps)
        all_displacements.extend(displacements)

    plt.hist(all_displacements, density = True, color = 'plum',
       bins=50, edgecolor='black')
    plt.xlabel('Displacement', **hfont)
    plt.ylabel('Probability Density', **hfont)
    plt.title('Probability Density Histogram for Levy Flight
       Displacements', **csfont)
    plt.grid(True)
    plt.show()
```

2D Levy Flight

```
    def levy_2D(alpha, size):
        # Generating random steps from a Levy distribution
        step_sizes = np.random.standard_cauchy(size=size)
        step_angles = np.random.uniform(0, 2 * np.pi, size=size)

        # Step Lengths
        step_lengths = step_sizes * (np.random.exponential(scale=1/
           np.abs(step_sizes)) ** (1/alpha))

        # Compute x and y coordinates
        x_steps = step_lengths * np.cos(step_angles)
        y_steps = step_lengths * np.sin(step_angles)

        return x_steps, y_steps
```

```
    x_steps, y_steps = levy_flight(1.5, 1000)

    # Calculating cumulative sum to get trajectory
    x = np.cumsum(x_steps)
    y = np.cumsum(y_steps)

    plt.plot(x, y)
    plt.title('2D Levy Flight', **csfont)
    plt.xlabel('$x$ position', **hfont)
    plt.ylabel('$y$ position', **hfont)
    plt.grid(True)
    plt.show()
```

4.5 References

- Quanta Magazine, *Random Search Wired Into Animals May Help Them Hunt*
- Harvey Gould, Jan Tobochnik, and Wolfgang Christian, *An Introduction to Computer Simulation Methods*