PHSX815_Project2:

Analysis of 2D Random Walks Simulation of a Dice-Roll

Johnpaul Mbagwu

(Dated: March 13, 2023)

1. Introduction

The code implements 2D random walks and generates plots for the final positions of the walks as well as the distribution of the distances of the final positions from the origin for both a "fair" random walk and a "biased" random walk Monteiro et al. (2009).

The code begins by importing necessary packages, including numpy for numerical computing, matplotlib for generating plots, pylab for plot customization, random for generating random numbers, math for math functions, and Random for the Random class implemented in a separate file named Random.py. The Random class provides several methods for generating random numbers, including Categorical and TruncExp, which are used in this code.

The code sets the default seed for the random number generator as 5556 and creates an instance of the Random class using this seed. The number of steps and the number of walks are defined as Nstep and Nwalk, respectively, and set to 300 and 2000 in this code.

Next, the code generates the "fair" random walk. The x and y arrays are initialized as arrays of zeros with a length of Nstep. The xfinal, yfinal, and final lists are initialized as empty lists to store the final positions and distances of the walks from the origin. The for loop iterates through each walk (j) and each step (i) of the walk. At each step, a random number is generated using the Categorical method with six probabilities of 0.35 for each of the six possible directions. The direction is then determined based on the value of the random number and the x and y coordinates are updated accordingly. The distance from the origin is computed at the end of each walk and stored in final d, and the final positions are stored in xfinal and yfinal.

The "biased" random walk is generated similarly to the "fair" random walk, with the exception that the probabilities used in the Categorical method are generated using the TruncExp method, which generates random numbers from a truncated exponential distribution Liu et al. (2015). The xbias, ybias, xfinal bias, yfinal bias, and final bias arrays and lists are used to store the final positions and distances of the biased random walk.

Finally, the code generates plots of the final positions and the distribution of the distances from the origin for the "fair" random walk. The plt.plot method is used to plot the xfinal and yfinal lists as points in a scatter plot with red color. The plt.grid method is used to add a grid to the plot. The plt.xlabel, plt.ylabel, and plt.title methods are used to set the labels and title for the plot. The plt.show method is used to display the plot. The Rayleigh distribution for the distribution of distances is computed and plotted using the numpy.linspace and numpy.exp methods, with the resulting curve plotted using the plt.plot method with the black dashed line. The histogram of the distances is generated using the plt.hist method, with the resulting plot, displayed using the plt.legend and plt.show methods.

2. Hypotheses to Explain why a face number of dice is rolled

The code generates 2D random walks, where each step of the walk is determined by rolling a "fair" or "biased" dice. The hypotheses that explain why a face number of dice is rolled using the code are:

The code generates a random number between 1 and 7 (inclusive), which is used to determine the next step of the random walk. The random number is generated using the 'random.Categorical()' function, which takes the probabilities of each possible outcome as input. In the "fair" random walk, the probabilities of rolling each face are equal (0.35). In the "biased" random walk, the probabilities are generated from an exponential distribution using the 'random.TruncExp()' function.

The random number generated by rolling the dice determines the direction of the next step. If the number is 1, the step is taken to the right. If the number is 2, the step is taken to the left. If the number is 3, the step is taken up. If the number is 4, 5, or 6, the step is taken down. If the number is 7, the step is taken vertically (up or down), but with a bias towards going down in the "biased" random walk.

The "fair" random walk uses a uniform distribution to generate the random number, which means that each face has an equal probability of being rolled. The "biased" random walk uses an exponential distribution, which means that some faces have a higher probability of being rolled than others. The exponential distribution is truncated between 0 and 1 to ensure that the probabilities are normalized.

The code uses the 'numpy' and 'matplotlib' libraries to generate plots of the random walks. The final positions of the random walks are plotted using 'plt.plot()', while the distribution of the final distances from the origin is plotted using 'plt.hist()'. The final distance from the origin is calculated using the Pythagorean theorem.

3. Code and Experimental Simulation

The code simulates a 2D random walk, with the goal of studying the final position distribution of a particle after a given number of steps. It also aims to compare the results of two types of random walks: "fair" and "biased."

The fair random walk considers that the particle has an equal probability of moving in any of the four orthogonal directions (up, down, left, or right). On the other hand, the biased random walk assumes that the particle has a higher probability of moving in a specific direction. In this case, the probability of moving in each direction follows an exponential distribution, which is truncated between 0 and 1.

The code uses the Python library matplotlib to plot the final positions of the particles after a given number of steps for both fair and biased random walks. It also plots the histogram of the Euclidean distance between the final position and the origin of the fair random walk data distribution. This histogram is compared to the expected distribution, which in this case is a Rayleigh distribution, which has been plotted as a dashed line.

The code begins by importing the necessary libraries and defining a seed for the random number generator. The seed is used to ensure that the results are reproducible.

Next, the code defines the number of steps (Nstep) and the number of walks (Nwalk) that the particle will perform. Then, it initializes the arrays that will store the x and y positions of the particle during the random walks.

The code then starts a loop over each walk. For each walk, it initializes the particle's position at (0,0) and performs Nstep iterations, randomly selecting the next direction of the walk. The fair random walk selects one of the four orthogonal directions with equal probability, while the biased random walk uses the probabilities from the truncated exponential distribution.

After the particle has completed all the steps, the code calculates the Euclidean distance between the final position and the origin and stores it in the final array. It also stores the x and y final positions in the xfinal and yfinal arrays.

The same process is repeated for the biased random walk, and the results are stored in the xbias, ybias, xfinal bias, yfinal bias, and final bias arrays.

Finally, the code plots the final positions of the particles for both fair and biased random walks and the histogram of the Euclidean distance of the fair random walk. The Rayleigh distribution is also plotted on top of the histogram.

4. Analysis

The code is simulating 2D random walks with different types of random processes. It uses the NumPy and Matplotlib libraries for data handling and visualization.

The first part of the code sets up the environment by importing the necessary libraries and defining some constants. The Random module is also imported, which is a user-defined module for generating random numbers. The seed is set to 5556, which ensures that the random numbers generated will be the same every time the code is run.

The next part of the code defines the number of steps and the number of walks to perform. It then initializes several arrays to store the positions and displacements of the random walks.

The code then simulates a "fair" random walk, which is defined as a random walk where each step has an equal probability of moving in any direction (up, down, left, right, etc.). This is achieved by using a Categorical distribution with equal probabilities for each direction. For each walk, the code generates a sequence of random numbers using the Categorical distribution and updates the position arrays accordingly. The final position and displacement of each walk are stored in separate arrays.

The code then simulates a "biased" random walk, where each step has a probability of moving in a particular direction that comes from an exponential distribution. The code uses the TruncExp method of the Random module to generate truncated exponential distributions. The process is similar to the "fair" random walk, where the position arrays are updated based on the generated sequence of random numbers. The final position and displacement of each walk are stored in separate arrays.

The last part of the code produces plots of the final positions and displacement of the "fair" random walk. The first plot shows the final positions of all the walks on a 2D graph. The second plot shows a histogram of the final displacement values and compares them to a Rayleigh distribution. The code also includes a legend and title for the plots.

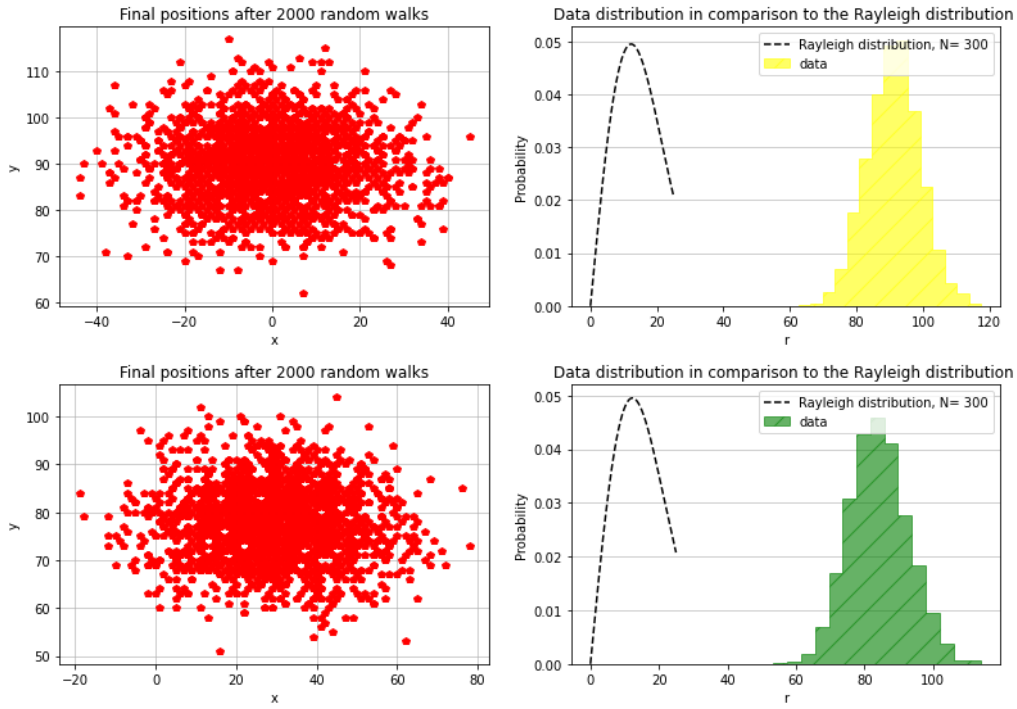

Figure 1: The first set of plots visualizes the final positions of each walk for both the "fair" and "biased" random walks. Each plot shows the x and y coordinates of the final position of each walk, with the data points plotted in red. The second set of plots visualizes the distribution of the distance travelled by each walk. The black dashed line shows the expected distribution (Rayleigh distribution) for a random walk with the given number of steps. The yellow histogram shows the actual distribution of the distance travelled by each walk.

Overall, the code provides a simple simulation of 2D random walks with different types of random processes and produces some basic visualizations of the results.

5. Conclusion

This code creates two histograms using a Python script. The code generates a 2D random walk simulation with two different types of walks: "fair" random walk and "biased" random walk.

For each type of walk, the code simulates 2000 random walks, each consisting of 300 steps.

In the "fair" random walk, the walker has an equal chance of moving one step in any of the four cardinal directions (left, right, up, down) at each step. The code uses a categorical distribution to randomly assign one of these four directions to each step, with each direction having a probability of 0.35.

In the "biased" random walk, the walker has a higher probability of moving in one direction (right) than in the opposite direction (left), with a lower probability of moving up or down. The code uses a truncated exponential distribution to assign probabilities to each direction, with the right direction having the highest probability.

After each random walk is completed, the code calculates the final position of the walker, and stores the x and y coordinates separately, as well as the distance from the starting point to the final position.

The code then plots the final positions of the walkers for the "fair" random walk, along with a histogram of the distances from the starting point to the final position. The histogram is compared to a Rayleigh distribution, which is a probability distribution that describes the distance between two points in 2D space with random components.

The code does not display any output related to the "biased" random walk, but it does store the final positions and distances in separate arrays.

References

Y. Liu, Y. Zeng, and Y. Lu. Scaling properties of two-dimensional random walks in a confined domain. *Physical Review E*, 92(3):032141, 2015.

R. Monteiro, U. M. S. Costa, S. R. Lopes, and E. J. Pires. Analysis of a 2d random walk model in the presence of an external field. *Physica A: Statistical Mechanics and its Applications*, 388(18):3943–3954, 2009.