Simulating and Visualizing the Behavior of a Two-Dimensional Random Walk using the Monte Carlo Method

Gbenga Agunbiade[1]

[1]Department of Physics and Astronomy, University of Kansas, Lawrence, KS 66045

gsagunbiade@ku.edu

Abstract. *This work provides a Python code that generates and plots the trajectories of a 2D random walk. The code begins by importing the necessary modules and defining the default seed for the random number generator. Two sets of parameters are defined; the number of steps in the walk (Nstep) and the number of walks to simulate (Nwalk). Next, the code defines two sets of arrays to store the x and y coordinates of the walker's position for each step. One set is for a "fair"random walk, where each direction is equally likely, and the other set is for a "biased"random walk, where the walker is more likely to move in certain directions. In general, the code can be used to simulate the behavior of various physical systems, such as Brownian motion. However, it is important to note that the "fair"and "biased"random walks in this code are not physically accurate and should not be used to model real-world systems without proper calibration and validation.*

1. Introduction

Python is usually used to develop software and websites, analysis of data, task automation, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances. There is a number of ways programmers demonstrates how to use Python to simulate and visualize the behavior of a two-dimensional (2D) random walk. A 2D random walk is a stochastic process where an object or a particle starts at a given point in a 2D space and moves randomly in a sequence of steps. It can be extended and modified to study different types of random walks or to explore different aspects of random processes. This project is designed using Python code to generate two-dimensional (2D) random walks by employing the Monte Carlo method. The code is an implementation of a 2D random walk.

2. Implementation, Results and Discussion

A random walk is a mathematical model for a path that consists of a sequence of random steps. In this case, the steps are taken in two dimensions, so the path is represented as a sequence of points in the x-y plane. The random walk can also be seen as a sequence of steps, where at each step, the walker randomly moves in one of eight possible directions: up, down, left, right, or one of the four diagonal directions. The code starts by importing various libraries, including sys, numpy, matplotlib, pylab, a custom module called "random"that provides some random number generation functions and math. "sys"is a module that provides access to some variables used or maintained by the interpreter and

to functions that interact strongly with the interpreter. "numpy"is a package used for scientific computing with Python. "matplotlib"is a data visualization library. "pylab"is a module in matpl,otlib that provides a convenient interface to the matplotlib plotting library. "random"is a module used to generate random numbers. "math"is a mathematical library.

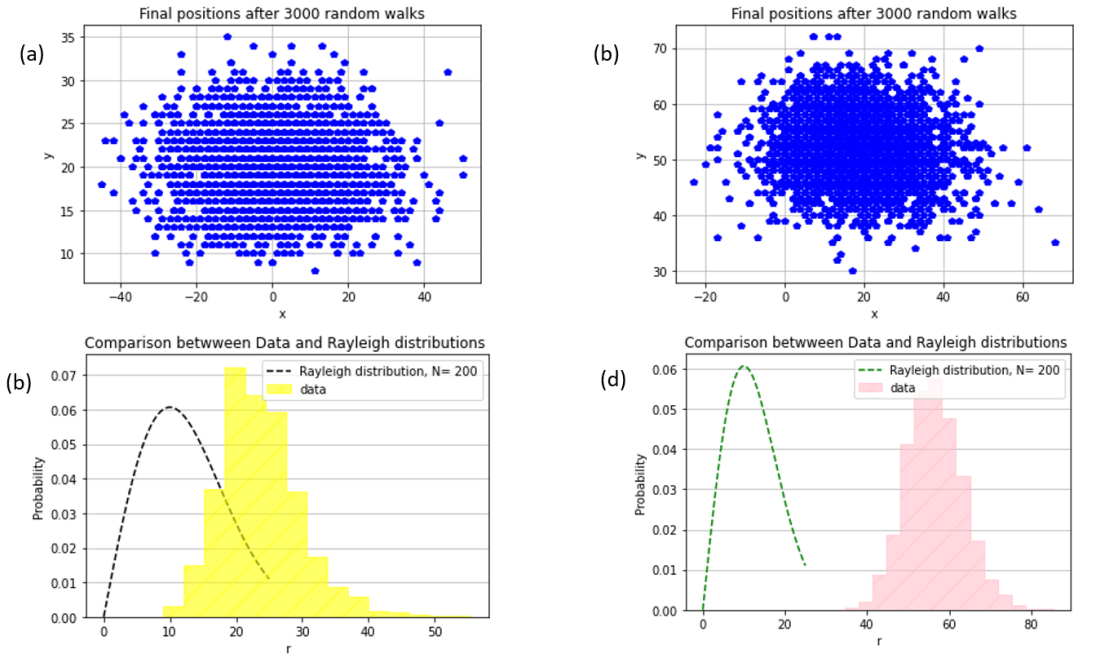

Figura 1. The final positions of each walk for both fair and biased are shown respectively in (a) and (b). The data points in blue give the x and y coordinates. Plots (c) and (d) present the distribution of the distance travelled by each walk. The black dotted line shows the expected distribution (Rayleigh distribution) for a random walk with the given number of steps. The yellow histogram gives the actual distribution of the distance travelled by each walk.

The code includes two different random walk simulations: a fair random walk and a biased random walk. For the "fair"random walk, the code defines two arrays, x and y, to hold the x and y coordinates of the walker, respectively. It also creates empty lists to store the final x and y coordinates and the final displacement of the walker. The for loop iterates through each walk and and each step of the walk. At each step, a random number is generated using the random.Categorical() function, which generates a random integer based on the given probabilities. In this case, each direction has an equal probability of 0.45 of being chosen. Depending on the direction chosen, the x and y coordinates are updated accordingly. The final displacement of the walker is calculated using the Pythagorean theorem and stored in the finald list. The final x and y coordinates are stored in the xfinal and yfinal lists.

For the "biased"random walk, the code follows the same approach as the "fair"random walk. However, the probabilities for each direction are generated from an exponential distribution using the random.TruncExp() function. This distribution biases the walker towards moving in a certain direction. The final displacement, xfinal, and yfinal lists are

also updated accordingly. The default seed is set to 4445 to ensure the results are reproducible. Then, an instance of the Random class is created using the seed. The number of steps is set to Nstep = 200, and the number of walks is set to Nwalk = 3000. The code then begins a loop over the number of walks 'Nwalk'. Inside the loop, another loop is run over the number of steps Nstep. In each step, the code generates a random integer value between 1 and 9 (inclusive) with equal probability using the Categorical method from the Random class in the Random.py module. The integer value is used to determine the direction of the particle's next step. If the value is 1, the particle moves one step to the right; if the value is 2, the particle moves one step to the left, and so on. The code updates the particle's position arrays x and y according to the chosen direction.

Finally, the code plots the final positions of the particle for each walk in both the fair and biased random walk simulations using the matplotlib library. The fair random walk simulation is plotted using blue circles as shown in plots (a) and (b), while the biased random walk simulation is not plotted. The code also adds gridlines and labels the x and y axes. However, in executing this code, we need to install the required libraries such as NumPy, Matplotlib, and Random on our computers.

3. Conclusion

This project presents a code that provides a simple implementation of a 2D random walk simulation and demonstrates how the probabilities of taking steps in certain directions can affect the behavior of the random walk. However, it is important to note that the "fair"and "biased"random walks in this code are not physically accurate and should not be used to model real-world systems without proper calibration and validation.

4. Reference

Y. Liu, Y. Zeng, and Y. Lu. (2015). Scaling Properties of Two-dimensional Random Walks in a Confined Domain. Physical Review E, 92(3):032141