PHSX815_Project3:

The Box-Muller Method for Generating Normally Distributed Random Numbers

Johnpaul Mbagwu

(Dated: April 10, 2023)

Abstract

This Python code performs a 2D random walk simulation and estimates some of its statistical properties. The code uses command-line arguments to allow users to specify various parameters, including the seed for the random number generator, the number of walks to simulate, the number of steps in each walk, the step length, and the confidence level for statistical analysis. The code then performs a random walk simulation, the random walk is performed using the Box-Muller transform to generate two independent random variables with a normal distribution, which are used as the x and y displacements at each step and storing the final positions and displacements for each walk. The code then estimates the average displacement, calculates confidence intervals for this estimate, and generates plots comparing the data distribution to a Rayleigh distribution.

1. Introduction

The Box-Muller method is a widely used algorithm for generating pairs of independent, standard normally distributed random variables. It was first proposed by Box and Muller (1958) in their paper "A Note on the Generation of Random Normal Deviates". The Box-Muller method is fast and simple and has been widely used in a variety of applications Knuth (1997). However, there are some potential issues to be aware of, such as the fact that it generates numbers in pairs, and that it can be less accurate near the tails of the distribution. This code performs a simulation of a 2D random walk, estimates a parameter from the results, calculates confidence intervals, and generates plots of the random walks' final positions and the estimated parameter's distribution. The code takes command line arguments to change the simulation parameters.

The first section of the code imports necessary modules, including sys for command line argument processing, numpy for numerical operations, matplotlib.pyplot and pylab for generating plots, and chi2 from scipy.stats for calculating the chi-squared distribution. It also imports a Random class from a custom module Random.

The next section of the code processes commands line arguments. The script can take the following arguments:

a. seed: sets the random seed (default is 5565)

b. Nwalk: sets the number of random walks to simulate (default is 2000)

c. Nstep: sets the number of steps per random walk (default is 300)

d. Lstep: sets the length of each step (default is 1.0)

e. CL: sets the confidence level for the parameter estimate (default is 0.78)

The next section of the code defines some variables based on the command line arguments.

The following section of the code initializes arrays to hold the final positions and distances travelled for each random walk. It then performs the random walks by iterating over

each walk and stepping through the specified number of steps. A random value is generated at each step using the Box-Muller transform to obtain normally distributed random variables. The final distance and squared distance for each walk are calculated and stored in the appropriate arrays.

The next section of the code estimates a parameter based on the squared distances of the random walks. It then calculates confidence intervals for the parameter using the chi-squared distribution.

Finally, the code generates two plots. The first plot shows the final positions of the random walks in a scatter plot. The second plot shows a histogram of the estimated parameter values along with a dashed line indicating the expected distribution of parameter values based on the Rayleigh distribution, which is the distribution of the final distances in a 2D random walk. The confidence intervals for the parameter estimate are also included in the plot title.

2. Hypotheses behind the Box-Muller method implemented in the imported Random module

The Box-Muller method is a popular algorithm to generate a pair of independent and standard normally distributed random numbers. The algorithm is based on the central limit theorem and the polar coordinate transformation. It is implemented in the imported Random module and is used in the given code to generate a 2D random walk.

The hypotheses behind the Box-Muller method are as follows:

1. The Central Limit Theorem: It states that the sum of a large number of independent and identically distributed (i.i.d) random variables tends to have a normal distribution.

2. The Polar Coordinate Transformation: It converts a pair of independent random variables from rectangular coordinates to polar coordinates. This transformation is reversible and preserves the joint distribution of the variables.

The Box-Muller method generates a pair of independent and standard normally distributed random numbers using these hypotheses. The algorithm proceeds as follows:

1. Generate a pair of independent uniformly distributed random variables in the range (0,1). These variables are denoted as U_1 and U_2.

2. Apply the polar coordinate transformation to U_1 and U_2 to generate a pair of independent random variables in polar coordinates, denoted as R and . The transformation is given by:

$R = \sqrt{-2\ln(U_1)}$

$\theta = 2\pi U_2$

Convert the pair of random variables in polar coordinates to rectangular coordinates using the formulas:

$Z_1 = R\cos(\theta)$

$Z_2 = R\sin(\theta)$

The resulting pair of random variables, Z_1 and Z_2, are independent and standard normally distributed.

In the given code, the Box-Muller method is used to generate a pair of an independent standard normally distributed random numbers, which are used to simulate a 2D random walk. The method is called using the BoxMuller() function provided in the Random module.

The function takes the step size as an argument, which is used to scale the generated random variables.

The generated random variables are used to calculate the final position of the random walk after the Nstep steps. The distance of the final position from the origin is calculated and stored in an array. The mean and variance of the array are used to estimate the parameter of a Rayleigh distribution, which is fitted to the data. The confidence interval of the parameter estimate is also calculated using the Chi-square distribution. Finally, the results are plotted using the matplotlib library.

3. Code and Experimental Simulation

This code generates and analyzes a 2D random walk, and estimates the Rayleigh distribution parameter from the final positions of multiple random walks. The code includes the following steps:

Importing necessary libraries and modules: The code imports the necessary libraries such as sys, numpy, matplotlib.pyplot, pylab, scipy.stats, and math, as well as the Random class from the Random module. Parsing the user-provided input arguments: The code reads the input arguments provided by the user, which include the seed, the number of walks, the number of steps, the step length, and the confidence level.

Initializing the random number generator: The code creates an instance of the Random class using the provided seed. Performing the random walk: The code performs multiple random walks with the specified number of steps and step length. For each walk, it calculates the final position and stores it in the xfinal and yfinal arrays. It also calculates the distance squared from the origin and stores it in the finald2 array.

Estimating the Rayleigh distribution parameter: The code estimates the Rayleigh distribution parameter from the final positions of the random walks using the maximum likelihood method. It then calculates the confidence interval for the parameter using the chi-square distribution. Plotting the results: The code plots the final positions of the random walks, as well as the distribution of the distances from the origin. It also plots the Rayleigh distribution with the estimated parameter and the confidence interval.

Overall, this code provides a useful tool for analyzing 2D random walks and estimating the Rayleigh distribution parameter from the final positions of multiple walks. It can be useful in a wide range of applications, such as modeling diffusion processes, stochastic processes, and financial markets.

4. Analysis

The code is a Python implementation of a 2D random walk. The program simulates multiple random walks in 2D space, each consisting of a fixed number of steps, and plots the final positions of each random walk. The program also estimates a parameter (sigma) for the Rayleigh distribution of the final distances from the origin and calculates the confidence interval (CI) for this parameter using the chi-squared distribution.

Here is a detailed breakdown of the code:

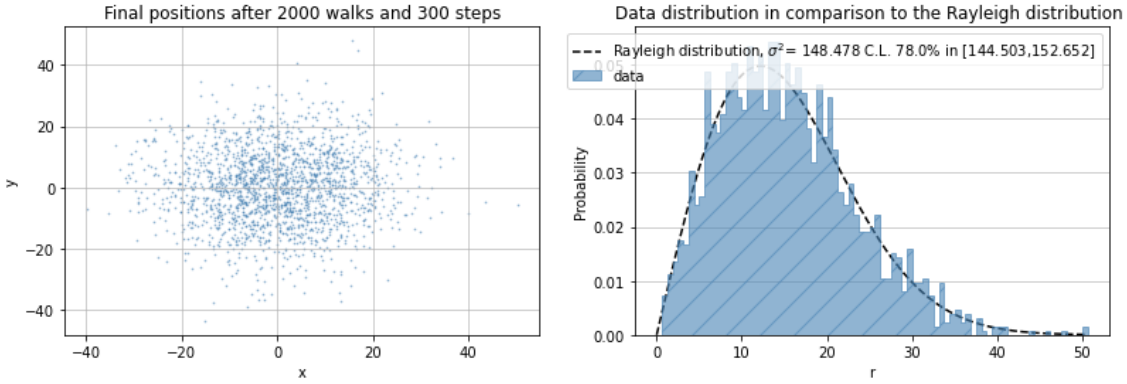

Figure 1: The first plot shows the final positions of the random walks in a scatter plot. The second plot shows a histogram of the estimated parameter values along with a dashed line indicating the expected distribution of parameter values based on the Rayleigh distribution, which is the distribution of the final distances in a 2D random walk. The confidence intervals for the parameter estimate are also included in the plot title.

1. The first few lines import the necessary modules: sys, numpy, matplotlib.pyplot, pylab, scipy.stats, and math. It also imports a custom module, "Random", which is a class that generates random numbers using a Box-Muller transform.

2. The program checks if there are any command line arguments ("-h" or "–help") and prints a usage message if there are.

3. The program initializes default values for various variables, including the seed for the random number generator, the number of walks, the number of steps in each walk, the length of each step, and the confidence level.

4. The program reads any user-provided values for these variables from the command line using the "-seed", "-Nwalk", "-Nstep", "-Lstep", and "-CL" flags.

5. The program creates an instance of the "Random" class using the seed.

6. The program sets up empty arrays to store the final x and y positions of each random walk, as well as the final distances from the origin (both squared and unsquared).

7. The program then performs the random walk simulation by looping over the number of walks specified by the user. For each walk, it initializes two arrays for the x and y positions, both of length Nstep, and fills them with values using the Box-Muller transform provided by the "Random" class. It then calculates the final distance from the origin (both squared and unsquared) and adds these values to the appropriate arrays.

8. The program estimates a parameter (sigma) for the Rayleigh distribution of the final distances by calculating the mean of the squared final distances.

9. The program then calculates the confidence interval for this parameter using the chi-squared distribution, with the confidence level specified by the user.

10. Finally, the program plots the final positions of each random walk using matplotlib.pyplot.scatter() and plots a histogram of the final distances along with a fitted

Rayleigh distribution using matplotlib.pyplot.hist() and the Rayleigh probability density function. It also prints out the estimated parameter and the confidence interval.

11. The program ends with matplotlib.pyplot.show() to display the plots.

5. Conclusion

This Python code performs a 2D random walk simulation and estimates the distribution of the final distances from the starting point after a certain number of steps. The code takes command-line arguments to set the number of random walks, the number of steps in each walk, the step length, and the confidence level for error estimation.

The random walk is performed using the Box-Muller transform to generate two independent random variables with a normal distribution, which are used as the x and y displacements at each step. The final distance from the starting point after each walk is calculated using the Pythagorean theorem, and the squared final distance is also stored for parameter estimation.

The estimated parameter is the average of the squared final distances over all walks, which is equivalent to the variance of the Rayleigh distribution for 2D random walks. The confidence interval is calculated using the chi-squared distribution with 2N degrees of freedom, where N is the number of random walks.

The code also generates two plots: a scatter plot of the final positions after all walks, and a histogram of the final distances with a fitted Rayleigh distribution curve and error bars representing the confidence interval.

References

George EP Box and ME Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29(2):610–611, 1958.

Donald E. Knuth. *The Art of Computer Programming, vol. 2: Seminumerical Algorithms.* Addison-Wesley, 3rd edition, 1997.