Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230814

An exploration to random walk

Tongtong Liu

Drexel University, Philadelphia, United States of America

tl873@drexel.edu

Abstract. There is a lot of uncertainty in many things that happen in real life, which leads to many events occurring randomly, and the quantitative description of the relationship between this sequence of random events is the stochastic process. Stochastic process is an important part of Probability. The stochastic process is a set of random variables, which is {Xt} t belongs to the T, The variable t is the parameter known as time, The parameter in a stochastic process are uncertain, each t corresponding to a random variable. T is the parameter set, and is a discrete parameter process when T is a finite set or a countable set, otherwise it is a continuous parameter process. The stochastic process can be divided into discrete state and continuous state according to the state space. If the random process x(t) takes countable values, then it is a discrete state process, otherwise it is not countable which is a continuous state process. The values of random variables change according to the results of random trials, and the pattern of change is also known as probability distribution. Stochastic process, which belonged to the field of physics in the early days of research, have developed into a powerful tool for research in many fields such as natural sciences, engineering sciences, and social sciences. Stochastic process use formulas to derive the probability of future events, find the inner laws from the apparent chance of events and describe the laws through probability. It can also be used as a statistical model to predict and deal with the development of some complex things in nature.

Keywords: random walk, stochastic process.

1. Basic Knowledge of Random Walk

Random walk is a kind of stochastic process, and the random walk algorithm can be applied in many fields of life, such as simulate stock prices, describe the statistical properties of genetic drift, simulate the cascade of neuronal firing in the brain, simulate the movement of gas molecules during diffusion, simulate the search path of foraging animals, etc. In many complex real-world situations, it is very difficult to draw direct conclusions about the occurrence of random events [1-18]. Probabilistic programming is a technique for developing system that assist humans in making decisions in the face of uncertainty. Many daily choices involve using judgment to identify pertinent, non-observable aspects. Probabilistic programming identifies those unseen aspects that are crucial to a decision by fusing our understanding of a situation with the rules of probability. Probabilistic reasoning system is becoming simpler to design and more often utilized thanks to a unique technique called probabilistic programming. For random events, historical data from past occurrences cannot be used to predict the exact probability of future occurrences. By modeling the random walk of events through a programming language, the process of event occurrence can be fully simulated to approximate the most realistic outcome. At the same time, it is possible to determine the direction of things in the simulation process and get the

DOI: 10.54254/2755-2721/6/20230814

corresponding probability of the development of things to help in decision making. In essence, the implementation of random walk algorithm through programming language can simulate the development of random events and achieve better decision making by finding the development pattern of simulated events. Therefore, in order to facilitate the simulation and observation of the random walk event to determine its trend, it is necessary to research how to implement random walk algorithm and modeling in the programming language. Random walk is often used as a simplified model for Brownian motion. Brownian motion is a classical example of a continuous stochastic process, it is the motion of flowing molecules colliding with particles at random. These liquid molecules keep doing irregular motion, in this irregular motion process, they will be randomly collision particles, these particles will randomly move to different directions due to the impact of molecules from different directions and force imbalance. In this whole movement process, x(t) is composed of the total number of particles, each one of them represents a particular particle sample. Also, this is a canonical process. However, random walk is a classical discrete stochastic process. The concept of random walk is very close to Brownian motion. A simple one-dimensional random walk can be thought of as a random step to the left or right on the number axis with the same probability moving forward to each direction after one unit of time. Each step of the move is an independent random variable, and the set consisting of these independent random variables is the simple random walk. The probability of each move in both directions is equal, so as the step of moves increases, the average of all moves converges to 0, so the expected value is 0. The expected distance after moving n steps is n^(1/2). In a random walk, the position of each move depends only on the position of the previous step and is not affected by the direction of the move of the other previous steps. Thus, a one-dimensional random walk can also be referred to as a Markov chain. When there exists a random variable Xt (t = 0, 1, 2，⋯), if Xt depends only on Xt-1 and not on {X0, X1, ⋯, Xt-2} is called Markov property, and a random sequence with Markov property is called a Markov chain, which is also a description of the state sequence. The Markov chain's transfer probability distribution determines its characteristics. In discrete time and states, a discrete Markov chain such as a dice throw, where the result of each throw is uncorrelated with the previous result and the sum of any previous results, at the same time, the probability of each result occurring is the same. Higher dimensional random walk's set have geometric properties and can be thought of as discrete fractals. Fractals have self - similarity. Classical fractals such as Koch snowflake, Peano Curve, Sierpinski triangle, Sierpinski carpet, etc. The two-dimensional random walk can be thought of as a random movement towards four different directions which are up, down, left and right on the coordinate axis, and will definitely return to the origin after some random steps. However, for higher dimensions, the probability of returning to the origin will gradually decrease. The step length of the random walk is constant for each step. The three-dimensional random walk can be thought of as a random movement in six directions: up, down, left, right, front and back. Other random walks such as self-avoiding random walks, which can be used to model chain-like entities such as solvents and polymers. In a self-avoiding random walk, each access point in a path can only be visited once and cannot be repeated so the paths will not cross. A Gaussian random walk whose step size varies according to a normal distribution can be thought of as the sum of a series of independent and identically distributed random variables and has a wide range of applications, such as the Black-Scholes formula used as a basic assumption to model option prices. Other random walks such as Branching random walk, Loop-erased random walk, Maximal Entropy Random Walk, etc. are also applied in various fields.

2. Introduction to Programming Languages That Can Implement The Random Walk Algorithm

When applying random walks to real life using a programming language, it is important to first focus on the uncertainty factor involved in real life problems. Since it is impossible to predict whether things will happen with 100% probability, probability data can be considered as a key factor when making decisions using random walk model in the present. When faced with different decisions, the probability data obtained after different simulations can be used as a reference at the same time. Using the probabilities obtained after modeling random walk to simulate real problems to analyze and make

DOI: 10.54254/2755-2721/6/20230814

decisions about things that have not yet happened in the future can lead to clearer answers to decision problems. These decisions are made through a combination of knowledge of the current situation and logic. In other words, when making difficult decisions, it is also important to understand the problem in depth and then get reliable decisions through approximate data obtained by logical analysis. In analyzing the actual problem, not only the probability data of the random walk is used as one of the factors influencing the decision, but also observe the whole simulation process of the random walk and then the data can be applied to the problem. Knowledge about the problem and all the relevant factors affecting the outcome of the problem are put into the model, and the final judgment will still be generated in probabilistic form after the model shows the entire process of the random walk. In this process of modeling through the random walk algorithm, the entire knowledge about the problem and the conditions affecting the outcome will be replaced in quantitative form and the model will be applied to the actual factors to find the answer to the decision by continuously observing the random walk process and probabilistic data. Its essence is calculated by mathematical rules. In current computer applications, the mathematical operations in this process can be easily implemented in several common programming languages. Programming languages like MATLAB, Python, SAS, Lingo, Java, C, C++, etc. can be used for modeling random walks[1-18]. The implementation of random walk algorithms using programming languages is essentially a simulation, a delineation of the actual problem using mathematical symbols, graphs, etc. Therefore, it is necessary to use a programming language that facilitates the application of various mathematical notations to perform mathematics and present mathematical logic more efficiently. Matlab is not a free software with powerful built-in functions and various toolkits to efficiently build complex models, and it comes with a complete set of toolkits with algorithms written by experts that can be used directly. Python as a free open source language, contains a lot of open source code that can be used, and its simple syntax makes it easy to build models, with a large number of related libraries available for model building (e.g., NumPy, SciPy, scikit-learn, SageMath, etc.). There is also a rich ecosystem of real-life relevant data and problems that can be updated more quickly, which facilitates solving real-world problems with random walk algorithms. Other languages such as SAS are used for professional statistical analysis, but again the full functionality is not available for free. LINGO is also very good for modeling and is mainly used for solving planning type optimization problems. Java, C, C++ are also powerful open source mainstream programming languages for modeling random walk algorithms, but the syntax is more complex than Python. Therefore, using Python as the programming language for modeling the random walk algorithm is the more common choice.

3. Steps of Implement The Random Walk Algorithm Using a Programming Language

Programming is a process from finding a problem to finding a solution, then express the solution to the problem on computer, and finally the computer outputs the answer to the user. The algorithm is the core of this process. Good algorithms allow computers to consume less time and space, a good algorithm determined by its time and space complexity. Implementing a simple random walk algorithm requires an initial starting point, typically the origin, and then moving from one or a series of starting points until the entire graph is traversed. At any starting point, the probability of a random move in a different direction is fixed and the same each time, and a probability distribution is derived after each wander that reflects the probability of each vertex in the graph being visited. This probability distribution is used as input for the next wander and iterated over. When this process satisfies certain preconditions, this probability distribution will converge. After convergence, a smooth probability distribution can be obtained for observation. It is important to set some variables during the iterative process. The first thing is to set the iteration start point and the step size of each walk according to different dimensions. Then the number of iterations is determined, followed by randomly generating the vector under the specific dimension, and then continuing the execution until the end after showing the position where the current point is located. The Python code for a simple random walk 3D model is as follows:

```
import random

t = int(input("Input maximum time : "))
```

DOI: 10.54254/2755-2721/6/20230814

```
def RandomWalk3D():
    global x,y,z
    x = 0
    y = 0
    z = 0
    OriginPoint = [x,y,z]
    print("Origin point is :", OriginPoint)
    direction = ["left","right","up","down","forward","back"]
    for i in range(0,t):
        step = random.choice(direction)
        if step == "left":
            x -= 1
            print([x,y,z])
        elif step == "right":
            x += 1
            print([x,y,z])
        elif step == "up":
            y += 1
            print([x,y,z])
        elif step == "down":
            y -= 1
            print([x,y,z])
        elif step == "forward":
            z += 1
            print([x,y,z])
        elif step == "back":
            z -= 1
            print([x,y,z])


loop = int(input("How many independent trajectories you wanna: "))
j = 0
xSum = 0
ySum = 0
zSum = 0

while j < loop:
    RandomWalk3D()
    xSum += x
    ySum += y
    zSum += z
    j += 1
print("The average of the distance is:", [xSum/j,ySum/j,zSum/j])
```

The basic random walk simulates a simple random walk process, and random walks with different characteristics can be implemented by gradually adding the corresponding code to the basic random walk model. For example, the self-avoiding random walk model can be implemented on the basic simple random walk model. The code to implement a 3D self-avoiding random walk model in Python is as follows.

```
import random
```

DOI: 10.54254/2755-2721/6/20230814

```
t = int(input("Input maximum time : "))
direction = ["left","right","up","down","forward","back"]

def Direction(a,b,c):
    step = random.choice(direction)
    if step == "left":
        a -= 1
    elif step == "right":
        a += 1
    elif step == "up":
        b += 1
    elif step == "down":
        b -= 1
    elif step == "forward":
        c += 1
    elif step == "back":
        c -= 1

def RandomWalk3D():
    global x,y,z
    x = 0
    y = 0
    z = 0
    OriginPoint = [x,y,z]
    HistoryPosition = [[x,y,z]]
    print("Origin point is :", OriginPoint)
    for i in range(0,t):
        step = random.choice(direction)
        if step == "left":
            x -= 1
            while [x,y,z] in HistoryPosition:
                x += 1
                Direction(x,y,z)
            print([x,y,z])
        elif step == "right":
            x += 1
            while [x,y,z] in HistoryPosition:
                x -= 1
                Direction(x,y,z)
            print([x,y,z])
        elif step == "up":
            y += 1
            while [x,y,z] in HistoryPosition:
                y -= 1
                Direction(x,y,z)
            print([x,y,z])
        elif step == "down":
            y -= 1
            while [x,y,z] in HistoryPosition:
                y += 1
```

DOI: 10.54254/2755-2721/6/20230814

```
            Direction(x,y,z)
        print([x,y,z])
    elif step == "forward":
        z += 1
        while [x,y,z] in HistoryPosition:
            z -= 1
            Direction(x,y,z)
        print([x,y,z])
    elif step == "back":
        z -= 1
        while [x,y,z] in HistoryPosition:
            z += 1
            Direction(x,y,z)
        print([x,y,z])
    HistoryPosition.append([x,y,z])
print("HistoryPosition ",HistoryPosition)


loop = int(input("How many independent trajectories you wanna: "))
j = 0
xSum = 0
ySum = 0
zSum = 0

while j < loop:
    RandomWalk3D()
    xSum += x
    ySum += y
    zSum += z
    j += 1
print("The average of the distance is:", [xSum/j,ySum/j,zSum/j])
```

The random walk algorithm enables the use of a programming language to build many different random walk models. When there is a demand for a particular probability data, the data generated by the random walk can be obtained by adding different instructions to the code of the model which can be effectively applied in various fields.

4. Summary

Probabilistic programming is a technique for developing system that assist humans in making decisions in the face of uncertainty. Many daily choices involve using judgment to identify pertinent, non-observable aspects. Probabilistic programming identifies those unseen aspects that are crucial to a decision by fusing our understanding of a situation with the rules of probability. Probabilistic reasoning system is becoming simpler to design and more often utilized thanks to a unique technique called probabilistic programming. Modeling of random walks can be applied in real life and used to solve many problems. The simulation process results in intuitive data close to the real outcome, and the data obtained through analysis can be used to confirm and solve problems. There are different kinds of random walks, but they can all be modeled by programming languages to achieve the algorithm, and different programming languages have different advantages and disadvantages, which do not have much impact on the implementation of simple algorithms, but complex models prefer to use more professional modeling software. What still needs to be addressed is how to use the random walk algorithm in greater depth to simulate possible future scenarios more accurately and to obtain more accurate probabilities.

DOI: 10.54254/2755-2721/6/20230814

References

[1] Anderson, W. J. (1991). Continuous-Time Markov Chains. Springer-Verlag, New York.
[2] Adler, R. J. (1981). The Geometry of Random Fields., Wiley, New York.
[3] Baldi, P., Mazliak, L. and P. Priouret. (2002). Martingales and Markov Chains: Solved Exercises and Elements of Theory. Chapman-Hall/CRC, Boca Raton.
[4] Basawa, I. V. and B. L. S. Prakasa Rao. Statistical Inference for Stochastic Processes. Academic Press, London.
[5] Billingsley, P. (2000). Convergence of Probability Measures, 2nd Ed., Wiley, New York.
[6] Chung, K. L. (1967). Markov Chains with Stationary Transition Probabilities, 2nd Ed. Springer-Verlag, New York.
[7] Chung, K. L. and R. J. Williams (1990). Introduction to Stochastic Integration, 2nd Ed. Birkhauser, Boston.
[8] Embrechts, P. and Maejima, M. (2002). Selfsimilar Processes. Princeton University Press, Princeton.
[9] Hida, T. and M. Hitsuda. (1991). Gaussian Processes. American Mathematical Society, Providence, R.I.
[10] Kallenberg, O. (1976). Random Measures. Academic Press, London.
[11] Karatzas, I. and S. E. Shreve (1991). Brownian Motion and Stochastic Calculus. Springer-Verlag, New York.
[12] Lukacs, E. (1970). Characteristic Functions, 2nd Ed. Griffin.
[13] Meyn, S. P. and Tweedie, R. L. (1993). Markov Chains and Stochastic Stability. Springer-Verlag, New York.
[14] Petrov, V. V. (1995). Limit Theorems of Probability Theory. Oxford University Press, Oxford.
[15] Samorodnitsky, G. and M. S. Taqqu. (1994). Stable Non-Gaussian Random Processes. Chapman-Hall/CRC, Boca Raton.
[16] Shorack, G. R. (2000). Probability for Statisticians. Springer-Verlag, New York.
[17] Durrett, R. (2019). Probability: theory and examples (Vol. 49). Cambridge university press.
[18] Pfeffer, A. (2016). Practical probabilistic programming. Simon and Schuster.