Bernoulli Distribution Simulator and Random Function

Yuzhen Jiang

Beijing Guangqumen Middle School, Beijing, China
molson52492@student.napavalley.edu

Abstract. This study explored the development and application of a random number generator following the Bernoulli distribution, emphasizing its significance in various fields such as simulation experiments, probability analysis, optimization algorithms, and machine learning. The generator's versatility in generating random outcomes aligning with specific probability distributions makes it a valuable tool in computer programming. Through the simulation of 10,000 games, the study confirmed the generator's accuracy in producing results consistent with the expected Bernoulli distribution parameter. The sample mean and sample variance closely matched theoretical expectations, highlighting the generator's reliability. The results demonstrated that the generated random variable adhered to the Bernoulli distribution properties, with outcomes mainly concentrated around 0 and 1. This aligns with the two possible values and equal probabilities inherent to the Bernoulli distribution. Looking ahead, the study suggests potential enhancements for the generator, including parameter tuning for different distributions, encapsulation as a versatile function or class, performance optimization for large-scale simulations, and statistical analysis of generated data using libraries like NumPy and SciPy. In conclusion, this research underscores the foundational role of random number generators in computer programming and their adaptability to meet evolving needs and complex scenarios in random number generation and analysis.

Keywords: Bernoulli Distribution, Random Function, Random Number Generator

1 Introduction

The random number generator stands as a frequently employed tool in computer programming, enabling the creation of random numbers that adhere to designated probability distributions [1-4]. In this example, a random number generator was written to generate random variables that conform to the Bernoulli distribution of parameter p. This generator can be used in multiple application fields such as simulation experiments, probability analysis, and statistical inference.

This random number generator has a wide range of uses. For instance, there are some possible applications: 1) Simulation experiments: In many experiments, generating random experimental results are needed. For example, in sports competitions, gambling games, or financial market simulations, this random number generator is able to simulate different experimental results and analyze their

B. H. Ahmad (ed.), *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, Advances in Intelligent Systems Research 180,
https://doi.org/10.2991/978-94-6463-370-2_59

probability distribution in order to make corresponding decisions. 2) Probability analysis: Random number generators can be used for probability analysis. It could used to generate a large number of random samples and calculate their mean, variance, standard deviation, and other statistics to understand the properties of the distribution. This is very useful for understanding the characteristics of random variables and conducting probability inference. 3) Optimization algorithms: In many optimization algorithms, requires randomly generate initial solutions or perform random perturbations to explore the solution space. This random number generator can provide randomness and help us better explore the solution space during the search process, thereby finding better solutions. 4) Machine learning and deep learning: When training machine learning models or deep learning models, randomize the data to avoid the model's dependence on a specific order is usually needed. This random number generator can help to generate random training, validation, and testing sets, as well as optimize algorithms such as random batch gradient descent.

In short, random number generators are one of the most important tools in computer programming. It can facilitate simulating experiments, conduct probability analysis, optimize algorithms, and machine learning in multiple application fields. The random number generator in this example can generate random variables that conform to the Bernoulli distribution of parameter p, providing a method for generating random numbers that conform to a specific probability distribution. By using this random number generator, individuals can better understand the characteristics of random variables, conduct probability inference, and conduct simulation experiments and optimize algorithm design in various applications.

Based on the effectiveness of it, this study defines a game function that takes the names of two teams and their corresponding level p as parameters, and determines the winning team based on their level. Then, this study defined sequence function takes team name, number of games, and level parameters, and uses game function generates a series of random games. The function returns the number of games, winners of the series, and game records.

2 Method

This study used the random module to generate the results of 10, 000 games and draws a histogram of the total number of championships won by the Golden State Warriors. This study started by importing the random module, which provides various functions for generating random numbers. The code starts by importing the random module, which provides various functions for generating random numbers. A function named simulate_game is defined with two parameters, teamA and teamB. This function simulates a game between the two teams and returns the number of games played, the list of winners, and the list of game results. Inside the simulate_game function, two empty lists are initialized: winners and game_results. These lists will store the winners of each game and the results of each game, respectively. A for loop is used to simulate a series of games. The loop iterates n times, representing the number of game series to simulate. In each iteration, a single game is simulated.

Within the loop, the random.choices function is used to randomly choose a winner for each game. The two teams, teamA and teamB, are passed as choices, with equal weights of 0.5 assigned to each team. The function returns a list of choices, and the first element is selected and assigned to the result variable. The result is appended to the game_results list to keep track of the outcome of each game. The result is also appended to the winners list to keep track of the winners of each game. After the loop finishes, the function returns the number of games played (n), the list of winners (winners), and the list of game results (game_results). Outside the function, the value of n is set to 10000, representing the number of game series to simulate. The team names teamA and teamB are defined as strings. Finally, the simulate_game function is called with the teamA and teamB parameters, and the returned values are assigned to n, winners, and game_results variables. This code snippet sets up a simulation of a series of games between two teams (teamA and teamB), simulates the games using the simulate_game function, and stores the results in the n, winners, and game_results variables. When writing code for a random number generator, it is important to be aware of common mistakes that can occur. Here are some common errors to watch out for:

Incorrect probability parameters: When using the random.choices function to generate random outcomes, it is crucial to ensure that the probability parameters sum up to 1. In the provided example, the probabilities for both teams are 0.5, resulting in a total sum of 1. Forgetting to normalize the probabilities can lead to unexpected results.

Incorrect parameter naming: Make sure to use the same parameter names in both the function definition and function call. In the provided code, the parameter names in the function definition are teamA and teamB, and these names should be used consistently when calling the function.

Incorrect loop count: When generating a series of games within a loop, ensure that the correct loop count is used. In the provided code, this study used range(n) to generate a loop for n game series, so make sure to pass the correct number of game series when calling the function.

Incorrect order of return values: Ensure that the order of return values matches the order specified in the function definition. In the provided code, the return values are defined as n, winners, game_results, so make sure to return these values in the same order.

Incorrect variable naming: Use meaningful variable names and avoid using names that conflict with existing variables or functions. In the provided code, game_results and winners are used as variable names to store the game results and winners' lists.

By being mindful of these common errors, you can minimize potential issues when writing a random number generator.

3 Results and Discussion

After running the simulator 10,000 times with a given parameter p=0.6, this study has obtained the sample mean and sample variance.

The sample mean is an estimate of the central tendency of the random variable. In the experiment, the sample mean is 0.6002, which is very close to the expected value of the parameter p=0.6. This indicates that in the simulation, the sample mean of the random variable is quite close to the expected value of the parameter p. This is consistent with the properties of the Bernoulli distribution, where the probability of success of the random variable is equal to the parameter p.

The sample variance is an estimate of the dispersion of the random variable. In the experiment, the sample variance is 0.2402. This suggests that the simulated results have relatively low variability, and the values of the random variable are concentrated around the parameter p. This is also in line with the properties of the Bernoulli distribution, where the random variable only has two possible values, resulting in relatively low variability.

By plotting a histogram, it can be visually observed the distribution of the random variable. From the shape of the histogram, it can be observed that the values of the random variable are mainly concentrated around 0 and 1, with relatively similar frequencies for both values. This is consistent with the properties of the Bernoulli distribution, where the random variable only has two possible values with equal probabilities.

In conclusion, through the analysis of the code results, it can be concluded that the generated random variable follows the Bernoulli distribution with the given parameter p. The simulation results are consistent with the theoretical expectations. This suggests that the random number generator exhibits a certain level of accuracy and reliability in generating random variables that meet the specified requirements.

4 Suggestion

In the process of further utilizing this random number generator in the future, the following aspects can be considered:

Parameter tuning: According to specific needs, random variables with different distributions can be generated by adjusting the value of parameter p. For example, different probability values can be attempted to generate random variables that are closer to a normal or uniform distribution.

Extension function: The random number generator can be encapsulated as a function or class, and additional functions can be added, such as generating multiple random variables, generating random numbers within a specific range, and generating fixed length random sequences.

Performance optimization: When dealing with large-scale simulations or generating large numbers of random numbers, it is possible to consider optimizing the performance of the code to improve computational efficiency. For example, the random number generation function provided by the NumPy library can be used to accelerate the calculation process through its vectorization operation.

Statistical analysis: After generating random numbers, statistical analysis can be performed on the generated sample data, such as calculating sample mean, sample

variance, frequency distribution, etc. You can use libraries such as NumPy and SciPy for statistical analysis.

All in all, writing a random number generator is just the beginning and can be further explored and expanded according to specific needs and application scenarios to meet more complex random number generation and analysis needs. In addition, artificial intelligence methods may be also considered in this case due to their excellent performance [5-10].

5 Conclusion

In conclusion, this study delved into the development and application of a Bernoulli distribution-based random number generator. The generator's versatility and utility were highlighted across various fields, including simulation experiments, probability analysis, optimization algorithms, and machine learning applications. Through the simulation of 10,000 games, this study demonstrated the generator's effectiveness in producing random outcomes that closely align with the expected Bernoulli distribution parameter.

The study's results showcased that the sample mean and sample variance closely matched the theoretical expectations, reaffirming the generator's accuracy and reliability. Furthermore, this study provided insights into potential future directions for improving and extending the generator's capabilities. These include parameter tuning for different distributions, encapsulation as a function or class with additional features, performance optimization, and statistical analysis of generated data.

In essence, this research underscores the foundational role of random number generators in computer programming and their widespread applicability in diverse domains. It emphasizes the potential for further exploration and adaptation to meet evolving needs and complex scenarios in the realm of random number generation and analysis.

References

1. Herrero-Collantes, M., Garcia-Escartin, J. C.: Quantum random number generators. Reviews of Modern Physics, 89(1), 015004 (2017).
2. Marsaglia, G., Zaman, A., & Tsang, W. W.: Toward a universal random number generator. Statistics & Probability Letters, 9(1), 35-39 (1990).
3. Blum, L., Blum, M., & Shub, M. A simple unpredictable pseudo-random number generator. SIAM Journal on computing, 15(2), 364-383 (1986).
4. Jun, B., & Kocher, P. The Intel random number generator. Cryptography Research Inc. white paper, 27, 1-8 (1999).
5. Qiu, Y., Wang, J., Jin, Z., Chen, H., Zhang, M., & Guo, L. Pose-guided matching based on deep learning for assessing quality of action on rehabilitation training. Biomedical Signal Processing and Control, 72, 103323 (2022).
6. Monil, P., Darshan, P., Jecky, R., Vimarsh, C., & Bhatt, B. R.: Customer segmentation using machine learning. International Journal for Research in Applied Science and Engineering Technology (IJRASET), 8(6), 2104-2108 (2020).

7. Ozan, Ş.: A case study on customer segmentation by using machine learning methods. In 2018 International Conference on Artificial Intelligence and Data Processing (IDAP) (pp. 1-6). IEEE (2018).
8. Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T. L. Machine learning for medical imaging. Radiographics, 37(2), 505-515 (2017).
9. Kononenko, I.: Machine learning for medical diagnosis: history, state of the art and perspective. Artificial Intelligence in medicine, 23(1), 89-109 (2001).
10. Dwyer, D. B., Falkai, P., & Koutsouleris, N.: Machine learning approaches for clinical psychology and psychiatry. Annual review of clinical psychology, 14, 91-118 (2018)