Islamic University of Technology

Simulation and Modelling Lab

CSE 4550

Lab 01: Random Number Generation (RNG)

Author:
Ahmed M. S. Albreem 210041258

Sep 24, 2024

Contents

1 Introduction

In this lab, we will delve into the mechanics of the LCG, exploring how it generates uniformly distributed random numbers. Building upon this foundation, we will employ these uniform random numbers to derive various probability distributions, including exponential, normal, and triangular distributions. Each distribution serves distinct purposes in modeling different types of random phenomena. For instance, the exponential distribution is pivotal in modeling time between independent events, the normal distribution is ubiquitous in natural and social sciences due to the Central Limit Theorem, and the triangular distribution is valuable in scenarios where limited information about the distribution is available but bounds and a mode are known.

2 RNG Algorithm Description

Random Number Generators (RNGs) are algorithms or devices designed to produce sequences of numbers that lack any discernible pattern, thereby simulating the properties of true randomness. RNGs are classified into two primary categories: True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs). TRNGs derive randomness from physical phenomena, such as radioactive decay or thermal noise, which are inherently unpredictable. In contrast, PRNGs use deterministic mathematical algorithms to produce sequences that appear random, relying on an initial value known as a seed.

The Linear Congruential Generator (LCG) is a widely used PRNG due to its simplicity and computational efficiency. The LCG generates a sequence of pseudo-random numbers based on the recurrence relation:

$$X_{n+1} = (a \cdot X_n + c) \bmod m$$

Here:

- X_n is the current seed or state.
- a is the multiplier.
- c is the increment.
- m is the modulus.

The choice of parameters a, c, and m is crucial for ensuring the quality of the generated random numbers.

In this lab, we have selected the parameters:

- $a = 1664525$
- $c = 1013904223$
- $m = 2^{32}$

These parameters are chosen based on empirical evidence and recommendations from numerical analysis literature, ensuring a long period and decent randomness quality for many practical purposes. Specifically, $m = 2^{32}$ allows the generator to produce $4,294,967,296$ unique numbers before repeating the sequence, which is generally sufficient for simulations requiring a large volume of random numbers.

3 Uniform Distribution

The uniform distribution is the simplest probability distribution, characterized by equal probability across its entire range. In the context of random number generation, a uniform distribution implies that every number within a specified interval has an identical likelihood of being selected. This property is foundational because many other, more complex distributions can be derived from a uniform base through transformation methods.

In this lab, the generation of uniform random numbers is implemented using the LCG. The core function, 'lcg(seed)', updates the seed based on the LCG formula and normalizes the output to produce a floating-point number within $[0, 1)$.

```
# LCG function
def lcg(seed):
    next_seed = (a * seed + c) % m
```

```
    return next_seed / float(m), next_seed

# Generate uniform samples
def generate_uniform_samples(seed, number):
    random_numbers = []
    for _ in range(number):
        random_number, seed = lcg(seed)
        random_numbers.append(random_number)
    return random_numbers
```

Listing 1: Uniform Distribution Generation

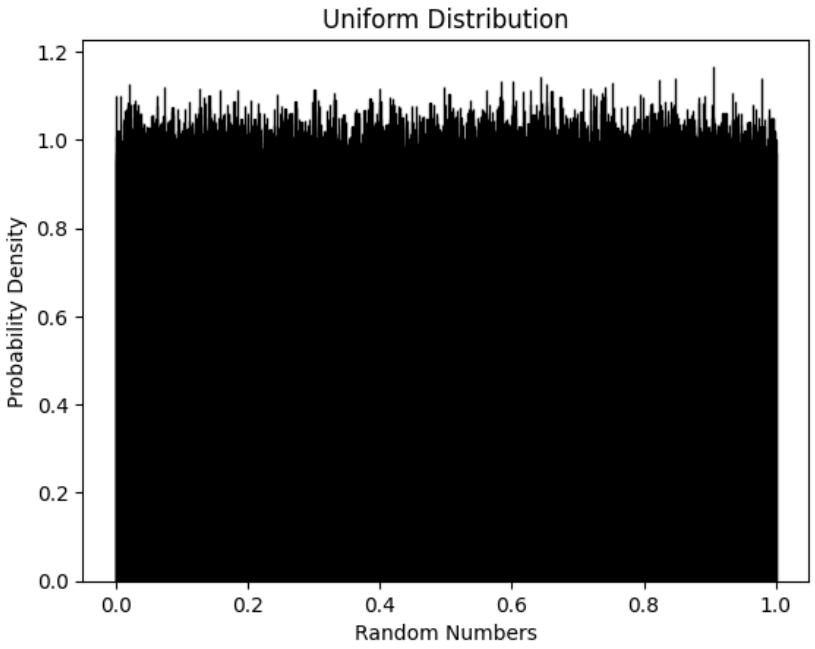

Figure 1: Histogram of Uniformly Distributed Random Numbers

4 Exponential Distribution

The exponential distribution is a continuous probability distribution characterized by its memoryless property, which implies that the probability of an event occurring in the next instant is independent of how much time has already elapsed. This distribution is extensively used to model the time between independent events that occur at a constant average rate, making it

particularly relevant in fields such as reliability engineering, queuing theory, and survival analysis.

Mathematically, the exponential distribution with rate parameter λ is defined by the probability density function (PDF):

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The cumulative distribution function (CDF) is given by:

$$F_X(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

To generate exponentially distributed random variables from uniformly distributed random numbers, we employ the inverse transform sampling method. This technique involves applying the inverse of the CDF to a uniform random variable U, transforming it into a variable X that follows the desired distribution.

Starting with a uniform random variable $U \in [0, 1)$, the transformation is as follows:

$$X = F_X^{-1}(U) = -\frac{1}{\lambda} \ln(1 - U)$$

Given that $1 - U$ is also uniformly distributed over $[0, 1)$, we can simplify the transformation to:

$$X = -\frac{1}{\lambda} \ln(U)$$

The Python implementation of this transformation is encapsulated in the ‘exponential‘ function:

```
# Exponential distribution function
def exponential(lambd, seed):
    u, seed = lcg(seed)
    x = -1/lambd * np.log(1 - u)
    return x, seed
```

Listing 2: Exponential Distribution Generation

This function takes the rate parameter λ and the current seed, generates a uniform random number u using the LCG, and applies the inverse transform to obtain an exponentially distributed random variable x.

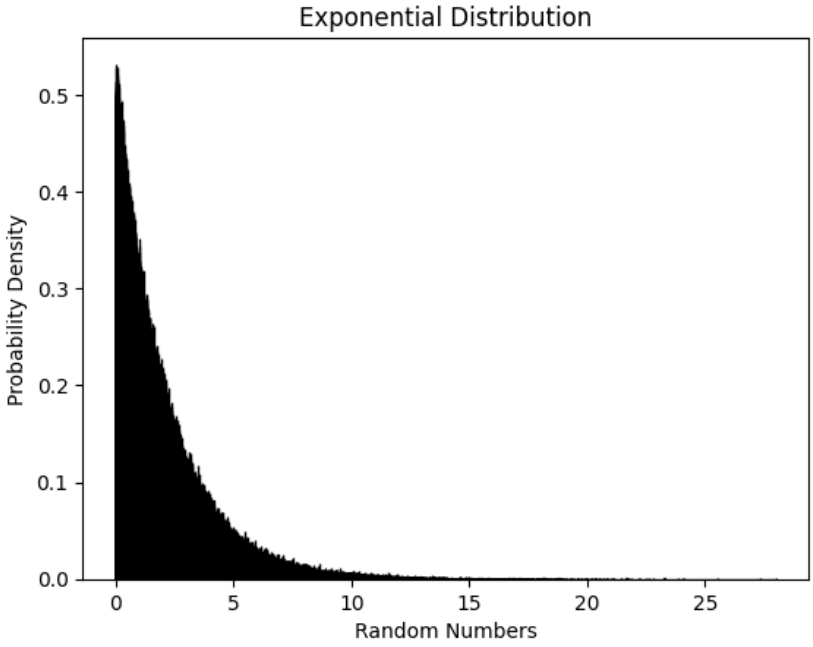

Figure 2: Histogram of Exponentially Distributed Random Numbers

5 Normal Distribution

The normal distribution, also known as the Gaussian distribution, is one of the most significant and widely used probability distributions in statistics and natural sciences. Characterized by its symmetric, bell-shaped curve, the normal distribution arises naturally in various contexts due to the Central Limit Theorem (CLT). The CLT states that the sum of a large number of independent, identically distributed random variables tends toward a normal distribution, regardless of the original distribution of the variables. This property explains the prevalence of the normal distribution in phenomena such as measurement errors, biological traits, and financial returns.

The probability density function (PDF) of a normal distribution with mean μ and standard deviation σ is given by:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The cumulative distribution function (CDF) does not have a closed-form

expression but is integral to understanding the distribution's properties.

The transformation is defined as follows:

$$Z_0 = \sqrt{-2\ln(U_1)} \cdot \cos(2\pi U_2)$$
$$Z_1 = \sqrt{-2\ln(U_1)} \cdot \sin(2\pi U_2)$$

Here, U_1 and U_2 are independent uniform random variables in $[0, 1)$, and Z_0 and Z_1 are independent standard normal random variables (mean 0, standard deviation 1).

```
# Normal distribution using Box-Muller transform
def normal(mu, sigma, seed):
    u1, seed = lcg(seed)
    u2, seed = lcg(seed)
    z0 = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    x = mu + sigma * z0
    return x, seed
```

Listing 3: Normal Distribution Generation

This function generates a single normally distributed random variable with specified mean μ and standard deviation σ. By repeatedly calling this function, a sequence of normal random variables can be obtained.

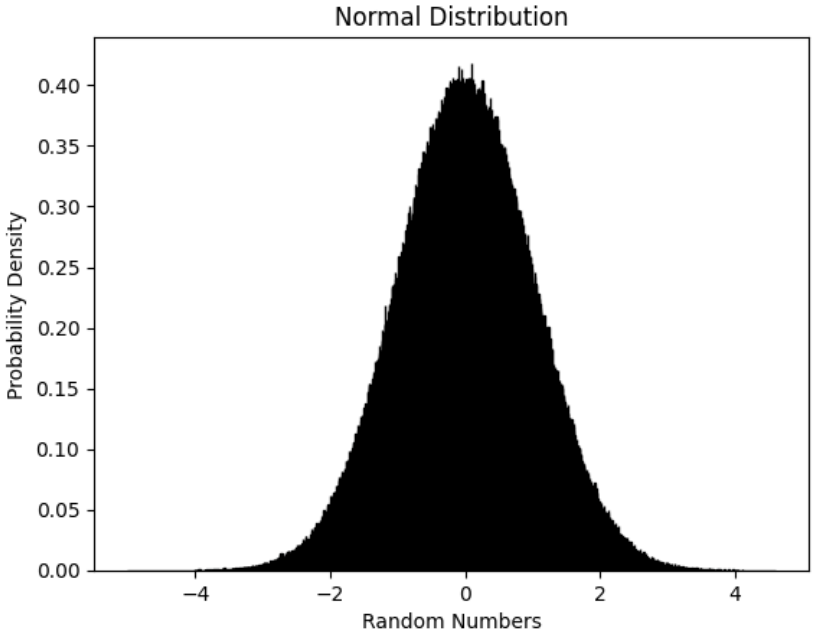

Figure 3: Histogram of Normally Distributed Random Numbers

6 Triangular Distribution

The triangular distribution is a continuous probability distribution shaped like a triangle, defined by a lower limit a, an upper limit b, and a mode c (the peak of the triangle). This distribution is particularly useful in scenarios where limited information is available about the underlying distribution of a variable, but bounds and a most probable value are known. The triangular distribution is commonly employed in project management for risk analysis, in simulation models where simplicity and boundedness are desired, and in situations where the exact distribution is difficult to ascertain.

The probability density function (PDF) of the triangular distribution is defined as:

$$f_X(x) = \begin{cases} 0 & x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \leq b \\ 0 & x > b \end{cases}$$

The cumulative distribution function (CDF) is piecewise and consists of two linear segments, increasing from 0 to 1 as x moves from a to b.

The transformation is as follows:

$$X = \begin{cases} a + \sqrt{U(b-a)(c-a)} & \text{if } U \leq \frac{c-a}{b-a} \\ b - \sqrt{(1-U)(b-a)(b-c)} & \text{otherwise} \end{cases}$$

Here, U is a uniform random variable in $[0, 1)$. The condition $U \leq \frac{c-a}{b-a}$ determines whether the generated X falls on the increasing or decreasing side of the triangular PDF.

The Python implementation of this transformation is encapsulated in the 'triangular' function:

```python
# Triangular distribution function
def triangular(a, b, c, seed):
    u, seed = lcg(seed)
    if u <= (c - a) / (b - a):
        return a + np.sqrt(u * (b - a) * (c - a)), seed
    else:
        return b - np.sqrt((1 - u) * (b - a) * (b - c)), seed
```

Listing 4: Triangular Distribution Generation

This function generates a single triangularly distributed random variable based on the provided parameters a, b, and c, along with the current seed.

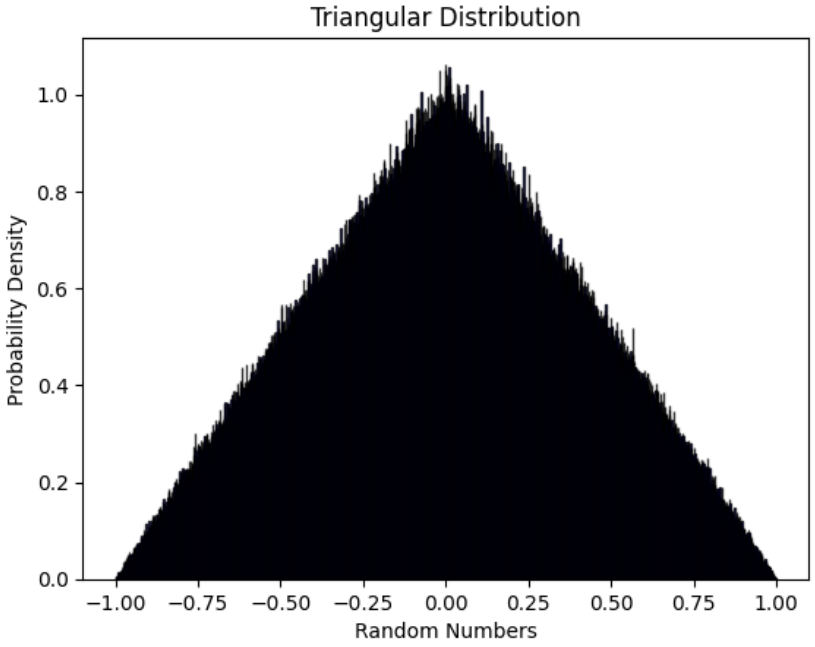

Figure 4: Histogram of Triangularly Distributed Random Numbers

Additionally, statistical measures such as the mean, median, and mode can be calculated to compare the generated data with theoretical expectations. The mean of the triangular distribution is given by:

$$\text{Mean} = \frac{a + b + c}{3}$$

7 Conclusion

In this lab, we embarked on an in-depth exploration of Random Number Generation (RNG) and its application in simulating various probability distributions. The journey began with the implementation of the Linear Congruential Generator (LCG), a fundamental pseudo-random number generator, which served as the backbone for generating uniformly distributed random numbers. By meticulously selecting parameters $a = 1664525$, $c = 1013904223$, and $m = 2^{32}$, we ensured that the LCG produced a robust sequence of pseudo-random numbers with a long period and satisfactory uniformity.

Each distribution was meticulously analyzed through visualizations such as histograms, which were juxtaposed with theoretical probability density functions (PDFs) to validate the accuracy of the generation process. These visual assessments confirmed the fidelity of the RNG and transformation methods, demonstrating their effectiveness in producing random variables that adhere to the expected statistical properties.

The lab underscored the critical role of RNGs in simulations and modeling, highlighting how deterministic algorithms like the LCG can approximate the stochastic nature required for various applications. Moreover, the exploration of different transformation techniques illuminated the versatility of uniform random numbers in generating a spectrum of probability distributions, each tailored to specific modeling needs.

However, the lab brought to light the inherent limitations of the LCG, such as its susceptibility to short periodicity and potential correlations in higher dimensions. While sufficient for educational purposes and basic simulations, more advanced applications may necessitate the use of higher-quality RNGs or more sophisticated algorithms to ensure true randomness and eliminate biases.

In conclusion, this lab provided a comprehensive understanding of RNGs and their pivotal role in statistical modeling and simulation. By mastering the LCG and transformation techniques, we have laid a solid foundation for tackling more complex stochastic models and simulations in future endeavors. The insights gained from this lab not only enhance our technical proficiency but also deepen our appreciation for the interplay between algorithm design and statistical theory in the realm of computational science.

Tools Used:

- **Python**: A versatile programming language utilized for implementing the RNG algorithms and performing statistical analyses.
- **NumPy**: A fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices.
- **Matplotlib**: A plotting library for creating static, animated, and interactive visualizations in Python.
- **Linear Congruential Generator (LCG)**: The core algorithm for generating uniform pseudo-random numbers, serving as the foundation for generating other distributions.

Concepts Covered:

- **Random Number Generation (RNG)**: Understanding the principles and implementation of algorithms to produce sequences of numbers that mimic true randomness.
- **Linear Congruential Generator (LCG)**: Detailed exploration of the LCG algorithm, including parameter selection and its impact on randomness quality.
- **Probability Distributions**: In-depth analysis of uniform, exponential, normal, and triangular distributions, including their properties, applications, and generation techniques.
- **Statistical Visualization**: Techniques for visualizing and validating random distributions using histograms and overlaying theoretical PDFs.

References:

- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley.
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press.
- L'Ecuyer, P. (1999). *Random Number Generation*. In *Handbook of Applied Cryptography*, vol. 1, pp. 1–54.
- Fishman, G. S. (1996). *Monte Carlo: Concepts, Algorithms, and Applications*. Springer.
- Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer.