Testing and Comparison of True and Pseudo Random Number Generators

Nazım Can Araz
Electrical and Electronics Engineering
Modelling and Simulation
Ankara, Turkey
nzmcnarz@gmail.com

***Abstract*—In this project pseudo random numbers is generated by using some pseudo random number generator algorithms. Some randomness tests is applied to generated pseudo random numbers and true random numbers. True random numbers are compared with pseudo random numbers.**

Keywords—random number generator, middle-square generator, linear congruential generator, inverse transform sampling, Mersenne twister

I. Introduction (*Heading 1*)

This document is a report of final assignment of my Modelling and Simulation course. A random number generator generates a sequence of numbers. These random numbers cannot be reasonably predicted. There are many ways to generate pseudo and true random numbers. Pseudo random numbers look like a true random numbers but they are actually deterministic. John von Neumann joked about pseudo random numbers "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

II. Importance of Random Number Generators

Random numbers are important because many things in real life are so complicated that they appear random. So to simulate those processes we need random numbers. Therefore we use random number generators to generate these random numbers.

Random number generators have applications in gambling, statistical sampling, computer simulation, cryptography, completely randomized design, and other areas where producing an unpredictable result is desirable. Monte Carlo method requires a lot of random numbers.

The generation of pseudo random numbers is an important and common task in computer programming. They are used in cryptography. Sender and receiver can generate the same set of numbers automatically to use as keys. Weaker forms of randomness are used in hash algorithms and in creating amortized searching and sorting algorithms. Quicksort is a familiar, commonly used algorithm in which randomness can be useful[1].

There are two main approaches to generating random numbers; true random number generators (TRNGs) and pseudo random number generators (PRNGs).

III. True Random Number Generators

True random number generator (TRNG) is a device that generates random numbers from a physical process, rather than by means of an algorithm. True or hardware random number generators which generate genuinely random numbers. There are many ways to generate true random numbers. Some TRNG examples we can say; rolling of dice, coin flipping, the shuffling of playing cards, roulette wheels. Using these techniques generating large numbers of sufficiently random numbers required a lot of work and/or time [2]. A really good physical phenomenon to use is a radioactive source. The points in time at which a radioactive source decays are completely unpredictable. Another physical phenomenon is atmospheric noise, which is quite easy to pick up with a normal radio. TRNGs have poor efficiency, taking considerably longer time to generate numbers. They are also nondeterministic, meaning that a given sequence of numbers cannot be reproduced. TRNGs have no period [3].

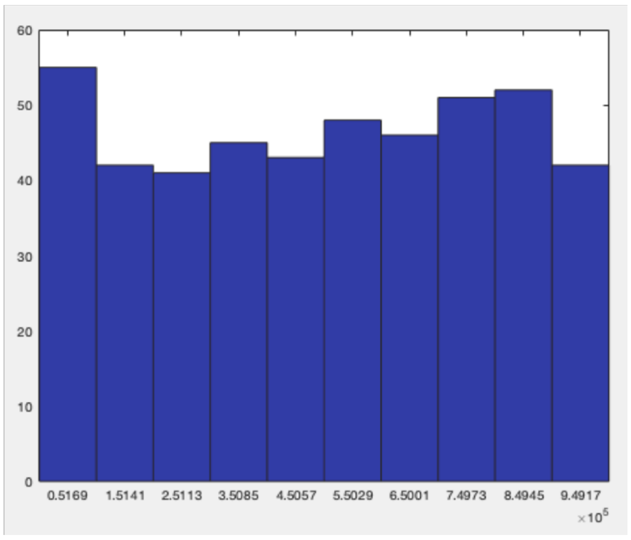
IV. Pseudo Random Number Generators

A pseudorandom number generator (PRNG), also known as a deterministic random bit generator (DRBG), is an algorithm for generating a sequence of numbers. The PRNG generated sequence is not truly random, because it is completely determined by an initial value. The goal of the PRNG is true randomness. There are many ways to generate pseudo random numbers. Some PRNG examples we can say; Linear Congruential Generator, Middle Square Generator, Inverse Transform Sampler. PRNGs have good efficiency. They can generate too many pseudo random numbers in a short time. Although they are deterministic so they have a period, given sequence of numbers can be reproduced [4].

A. Middle Square Generator

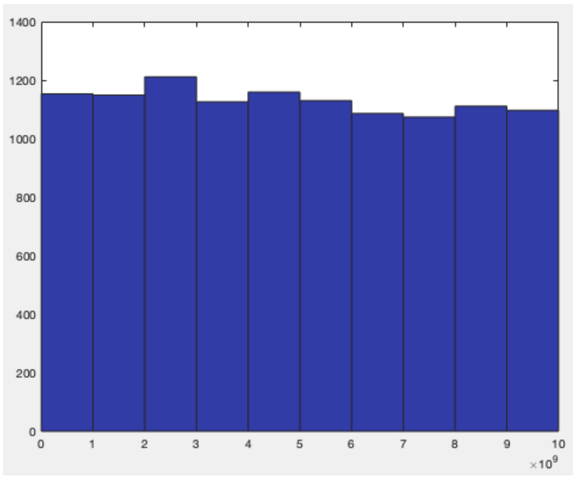
The method was invented by famous mathematician John von Neumann, and was described at a conference in 1949. Middle square generator is simple pseudo random number generator but in practice it is not a good method since its period is usually very short. To generate a sequence of n digit pseudo random numbers, an n digit starting value (seed) is created and squared. The middle of n digits of the result would be next number in the sequence and returned as the result. This process is then repeated to generate more numbers [5].

Figure. 1 Histogram of Middle Square Generator (seed: 735973)



Histogram of generated pseudo random numbers by the middle square generator entered as 735973 as the seed value is shown in the Fig. 1. Its period is 465. Fig 1. seems to be uniformly distributed with value of seed 735973, but still cannot say properly distributed. If we increase the seed value more we can expect the period to increase. When the value of seed is 761462387 selected, the new period is 11296. The histogram of generated pseudo random numbers entered as this seed value is shown in the Fig. 2. Fig 2. seems like uniformly distributed. As the number of generated random numbers increases in other words if the period increases, it approaches to uniform distribution. However, with middle square method it is necessary to enter large seed value to generate large period random number sequence and more processing power is required in large seed value processing and it takes more time to do these operations.

Figure. 2 Histogram of Middle Square Generator (seed: 761462387)



B. *Linear Congruential Generator*

Linear congruential generator (LCG) which is the frequently referred to as a Lehmer random number generator introduced by Derrick Henry Lehmer in 1951. This method represents one of the oldest and best known pseudo random number generator algorithms.

Understanding the underlying theory is quite simple. All linear congruential generators can have different parameters, but all LCGs use (1).

$$X_{n+1} = (aX_n + c) \bmod m \tag{1}$$

X_0	$0 \le X_0 < m$	Seed or start value
X_n	$0 \le X_n < m$	Generated random numbers
a	$0 < a < m$	multiplier
c	$0 \le c < m$	increment
m	$0 < m$	modulus

Parameters a, c and m are constant integers. X is sequence of generated pseudo random numbers. With the selection of appropriate parameters, the period becomes longer and the number of periods is known. To do this, the Hull-Dobell Theorem must be implemented:

1. m and the offset c are relatively prime,
2. a – 1 is divisible by all prime factors of m,
3. a – 1 is divisible by 4 if m is divisible by 4.

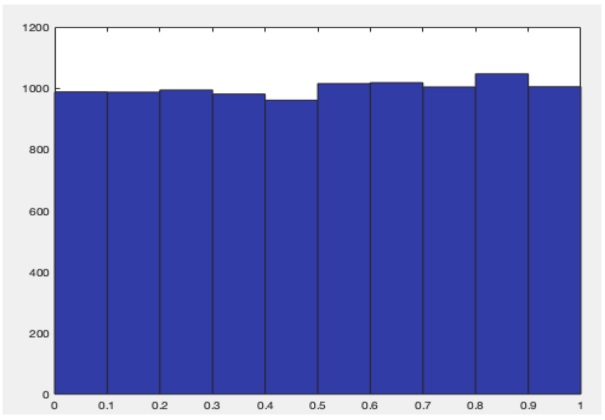
If Hull-Dobell Theorem is applied, the period becomes m.

LCGs in common use in runtime libraries of various compilers. For example; Microsoft Visual, C, C++, Turbo Pascal, Borland Delphi, Java, etc [6].

Histogram of 10000 generated pseudo random numbers by the linear congruential generator is shown in the Fig. 3. It seems to be uniformly distributed with these parameters;

$m = 2^{32}$, $a = 214013$, $c = 2531011$

Figure. 3 Histogram of Linear Congruential Generator



C. *Inverse Transform Sampling*

Inverse transform sampling actually does not generate pseudo random numbers. It transforms pseudo random numbers which are uniformly distributed as the name of method suggests.

This method involves computing the quantile function of the distribution. In other words, computing the cumulative distribution function (CDF) of the distribution and then inverting function.

$$F(x) = \int_{-\infty}^{x} f(x)\,dx \tag{2}$$

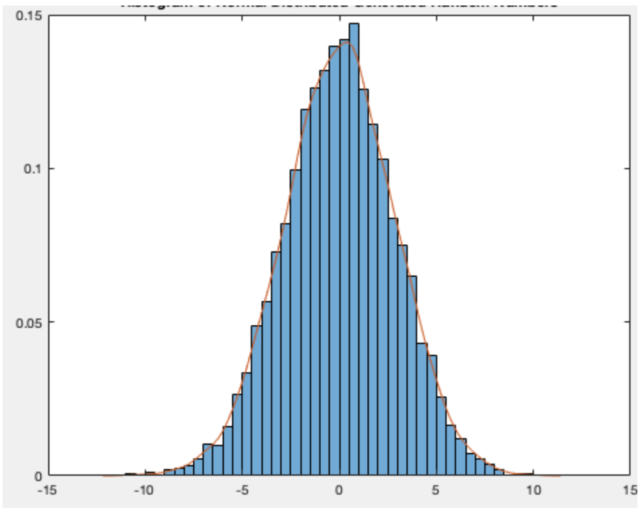
$$x = F^{-1}(u) \tag{3}$$

f(x)	Probability distribution function (PDF)	
F(x)	$0 \le F(x) < 1$	CDF
u	$0 \le u \le 1$	Uniformly distributed random numbers
x	Quantile Function	

This method involves computing the quantile function of the distribution. In other words, computing the cumulative distribution function (CDF) of the distribution and then inverting function. If F(x) CDF of normal distribution, then x has normal distribution as f(x). x represents the random numbers generated with desired distribution. Fig. 4 shows that the numbers generated have a normal distribution because f(x) has a normal distribution [7].

Using inverse transform sampling, uniformly distributed random numbers can be transformed to random numbers with desired distribution.

Figure. 4 Histogram of Inverse Transform Sampling



D. *Mersenne Twister*

The Mersenne Twister is the most widely used in pseudo random number generators. Its name derives from the fact that its period length is chosen to be a Mersenne prime. The Mersenne Twister was developed in 1997 by Makoto Matsumoto and Takuji Nishimura. The most commonly used version of the Mersenne Twister algorithm is based on the Mersenne prime $2^{19937-1}$. The Mersenne Twister is the default PRNG for the following software systems, languages and libraries; Microsoft Excel, GAUSS, MATLAB, Pascal, PHP, Python, Ruby, R, Scilab, C, C++, Boost and more [8].

V. Randomness Tests

Oddly enough, it is theoretically impossible to prove that a random number generator is really random. In stochastic modeling, as in some computer simulations, the hoped for randomness of generated random numbers can be verified, by a formal test for randomness. As the sequences pass more of the tests, the confidence in the randomness of the numbers increases. There are many practical measures of randomness for a binary sequence [9].

A. *Frequency (Monobit) Test*

This test measures whether the number of 0s and 1s produced by the generator are approximately the same as would be expected for a truly random sequence. The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence [10].

n:	The length of the number sequence.
ε:	Number sequence.

When we apply the equations (4), (5), (6), and (7) respectively, we compute the P-value. P-value is between 0 and 1. If the computed P-value is less than 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

$$X_i = 2\varepsilon_i - 1 \quad (4)$$

$$S_n = \sum_{i=1}^{n} X_i \quad (5)$$

$$S_{obs} = \frac{|S_n|}{\sqrt{n}} \quad (6)$$

$$P - value = erfc(\frac{S_{obs}}{\sqrt{2}}) \quad (7)$$

B. *Runs Test*

The focus of this test is the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence.

$$\pi = \frac{\sum_i^n \varepsilon_i}{n} \quad (8)$$

$$\tau = \frac{2}{\sqrt{n}} \quad (9)$$

$$r(k) = \begin{cases} 0, & \varepsilon_k = \varepsilon_{k+1} \\ 1, & otherwise \end{cases} \quad (10)$$

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1 \quad (11)$$

$$P - value = erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right) \quad (12)$$

If it can be shown that $|\pi - 1/2| \geq \tau$, then the Runs test need not to be performed. When we apply the equations (8), (11), and (12) respectively, we compute the P-value. P-value is between 0 and 1. If the computed P-value is less than 0.01, then conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

C. *Simple Visual Analysis*

One way to examine a random number generator is to create a visualisation of the numbers it produces. Humans are really good at spotting patterns, and visualisation allows you to use your eyes and brain directly for this purpose. Bitmap can be used to visualize numbers.

Fig. 5 is not clear from any pattern or model appearance but in Fig. 6 seems clearly appears to be a pattern or model. Because Fig. 6 is bitmap of generated random numbers using middle square generator and its period is 57. This period value is not large, so we can see pattern or model.

Figure. 5 Bitmap of True Random Numbers

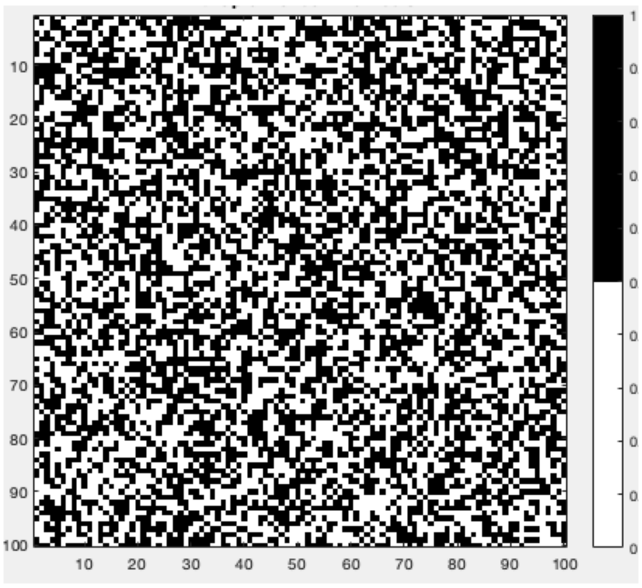
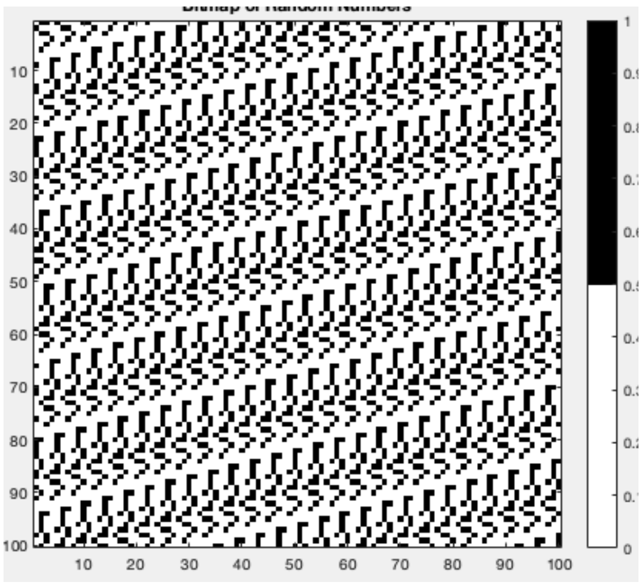


Figure. 6 Bitmap of Generated Random Numbers using Middle Square Generator



VI. Conclusion

True random number generators generate genuinely random numbers but costly. Also generation of true random numbers take more time than pseudo random number generation. Pseudo random number generators generate look random, but they are actually deterministic. Pseudo random number generation more easy than true random number generation.

References

[1] *https://en.wikipedi.org/wiki/Random_number_generation*

[2] *https://www.random.org/randomness*

[3] *http://www.fourmilab.ch/hotbits/hardware3.html*

[4] *https://en.wikipedi.org/wiki/Pseudorandom_number_generator*

[5] *https://en.wikipedi.org/wiki/Middle-square_method*

[6] *https://en.wikipedi.org/wiki/Linear_congruential_generator*

[7] *https://en.wikipedi.org/wiki/Inverse_transform_sampling*

[8] *https://en.wikipedi.org/wiki/Mersenne_Twister*

[9] *https://en.wikipedi.org/wiki/Randomness*

[10] *https://www.random.org/statistics/frequency-monobit/*