P346 project report

Title: Random Number Generator

Name: Dibya Bharati Pradhan
Roll No.: 1911067

School of Physical Sciences, National Institute of Science Education and Research, HBNI, Jatni-752050, India
dibyabharati.pradhan@niser.ac.in

November 21, 2021

Abstract

This report covers the theory behind the problem statement of using a specific linear congruential generator (LCG) to create random variates from the uniform(0,1) distribution and testing the uniformity by the roll of a symmetric die . The LCG is used to generate multiple samples of pseudo-random numbers and statistical computation techniques are used to assess whether those samples could have resulted from a uniform(0,1) distribution.

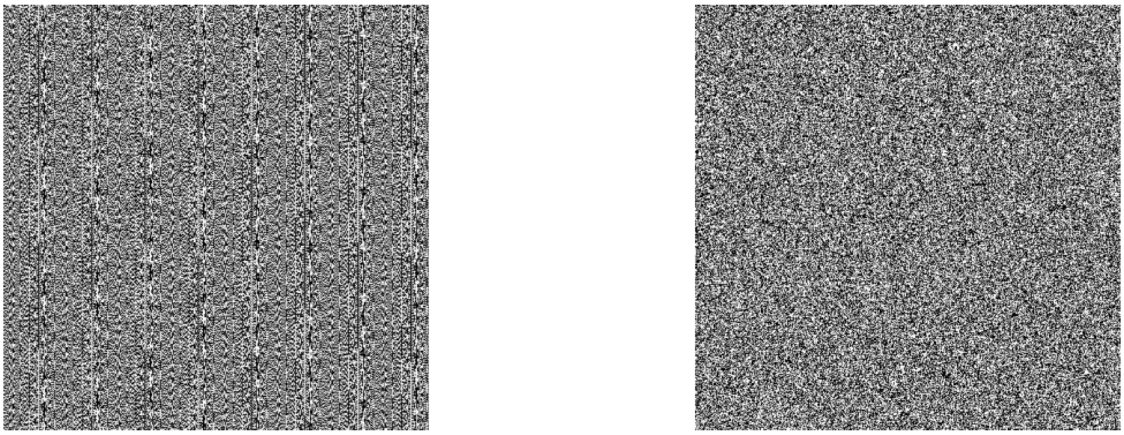
I Theory

The Linear Congruential Generator is one of the most common methods of generating pseudo-random numbers. It is a fast and efficient tool well-suited for computer implementations and It is defined by a recursion as follows:

$$Z_{n+1} = (aZ_n + c)\, mod\ m \quad \text{where, } n \geq 0$$
$$U_n = Z_n/m$$

here, $0 < a < m,\ 0 \leq c < m$ are constant integers, and mod m means modulo m which means you divide by m and leave the remainder. All the Z_n fall between 0 and $m-1$; the U_n are thus between 0 and 1 .

$0 \leq Z_0 < c$ is called the seed. m is chosen to be very large, usually of the form $m = 2^{32}$ or $m = 2^{64}$ because the computer architecture is based on 32 or 64 bits per word. The modulo computation would barely involve truncation by the computer, hence is immediate.



bit 1 bit 32

Figure 1: *Lower bits will be "less random" than the upper bits.*

A 'good' random-number generator should satisfy the following properties:

- Uniformity: The numbers generated appear to be distributed uniformly on (0.1);
- Independence: The numbers generated show no correlation with each other;
- Replication: The numbers should be replicable (e.g., for debugging or comparison of different systems).
- Cycle length: It should take long before numbers start to repeat;
- Speed: The generator should be fast;
- Memory usage: The generator should not require a lot of storage.

The numbers a, c, m must be carefully chosen to get a "good" random number generator, in particular we would want all c values $0, 1, \ldots c-1$ to be generated in which case we say that the LCG has full period of length c. Such generators would cyclically run thru the numbers over and over again.

To illustrate, consider

$$Z_{n+1} = (5Z_n + 1) \mod 8, \quad n \geq 0,$$

with $Z_0 = 0$. Then

$$(Z_0, Z_1, \ldots, Z_7) = (0, 1, 6, 7, 4, 5, 2, 3)$$

and $Z_8 = 16 \mod 8 = 0$, hence causing the sequence to repeat.
If we increase m to $m = 16$,

$$Z_{n+1} = (5Z_n + 1) \bmod 16, \quad n \geq 0,$$

with $Z_0 = 0$, then

$$(Z_0, Z_1, \ldots, Z_{15}) = (0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3)$$

and $Z_{16} = 16 \bmod 16 = 0$, hence causing the sequence to repeat.

Choosing good numbers a, c, m involves the sophisticated use of number theory: prime numbers and such, and has been extensivaly researched/studied by computer scientists and mathematicians for many years.

A linear congruential generator has full period (cycle length is m) if and only if the following conditions hold:

1. The only positive integer that exactly divides both m and c is 1 ;
2. If q is a prime number that divides m, then q divides $a-1$;
3. If 4 divides m, then 4 divides $a-1$.

Note : The numbers generated are entirely deterministic: If you know the values a, c, m, then once you know one value (Z_0, say) you can predict them all. But if you are handed a long sequence of the U_m, they certainly appear random, and this is the reason they are called pseudo random numbers.

The advantages of using a LCG:

1. Very fast to implement
2. Requires no storage of the numbers, only the most recent value.
3. Replication: Using the same seed, you can generate exactly the same sequence again and again which is extremely useful when comparing alternative systems or models: By using the same numbers you are reducing the variability of differences that would be caused by using different numbers; any difference in the comparisons are thus due to the inherent difference in the models themselves.

More sophisticated generators :
Python currently uses the Mersenne Twister as its core random number generator:
U - random.random()
It produces at double precision (64 bit), 53 -bit precision (floating), and has a period of 2^{19937} - 1 (a Mersenne prime number) The Mersenne Twister is one of the most extensively tested random number generators in existence. (There is both a 32 bit and 64 bit implementation. It is not a LCG ; it is far more complex, but yet again is deterministic and recursive.

II Problem Statement

Implement the linear congruential generator for different values of Z_0(initial seed) with the choice $m = 7, a = 5, c = 9$. Next, implement it for the choice $m = 2^{31} - 1$, $a = 7^5$, $c = 12345$. Generate $n = 100, 500, 1000, 10000$ random number and draw a histogram.
Simulate the roll of a symmetric die and draw a histogram for several values of n. Repeat the same with the generators *random.random* and *numpy.random.uniform*. Finally, Compare the time efficiency of these generators by using *time.time*.

References :

- Linear congruential generator
`https://en.wikipedia.org/wiki/Linear_congruential_generator`
- Random Number Generation, Freie Universität ,Berlin
`https://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2012_SS/L_19540_Modeling_and_Performance_Analysis_with_Simulation/06.pdf`
- Random Number Generators,Professor Karl Sigman,Columbia University,Department of IEOR,New York
`http://www.columbia.edu/~ks20/4106-18-Fall/Simulation-LCG.pdf`
- 2WB05 Simulation Lecture 5: Random-number generators by Marko Boon
`https://www.win.tue.nl/~marko/2WB05/lecture5.pdf`