Experiment No. 1

Title: Random Number Generator

Batch: B–1 **Roll No.: 16010422234** **Experiment No.: 1**

Aim: To study and implement a PseudoRandom Number Generator (PRNG) using Linear Congruential Method

---

Resources needed: Turbo C / Java / python

Theory

Problem Definition:
Write a Program for generating random numbers using Linear Congruential method such that
i) Period of the numbers generated is >= 100
ii) Density of the numbers generated is maximum (average gap between random numbers is < 0.1).

Concepts:

Random Numbers: Random numbers are a necessary basic ingredient in simulation of almost all discrete systems. Most computer languages have a subroutine, object or function that will generate a random number. A simulation language generates random numbers that are used to generate event times and other random variables.

Properties of random Numbers:
A sequence of random numbers R1, R2 … must have two important statistical properties, uniformity and independence.

Uniformity:
If the interval (0, 1) is divided into "n" classes or subintervals of equal length , the expected number of observations in each interval is N/n, where N is the total number of observations.

Independence:
The probability of observing a value in a particular interval is independent of the previous drawn value.

Problems faced in generating random numbers:
1. The generated number may not be uniformly distributed.
2. The number may be discrete valued instead of continuous values.
3. The mean of the numbers may be too high or low
4. The variance of the number may be too high or low.
5. The numbers may not be independent

e.g.

a. Autocorrelation between numbers
b. Numbers successively higher or lower than adjacent numbers.

Criteria for random no. generator:
1. The routine should be fast.
2. The routine should be portable.
3. The routine should have a sufficient long cycle. The cycle length or period represents the length of the random number sequence before previous numbers begin to repeat themselves in an earlier order. A special case of cycling is degenerating. A routine degenerates when some number appears repeatedly which is unacceptable.
4. The random number should be replicable.
5. Most importantly, the generated random numbers should closely approximate the ideal statistical properties of uniformity and independence.

Procedure / Approach / Algorithm / Activity Diagram:

Linear Congruential Method:

The Linear Congruential method produces a sequence of integers $X1, X2, \ldots$ between 0 and m -1 according to the following recursive relationship.

$X_{i+1} = (a X_i + c) \bmod m$, $i = 0, 1, 2\ldots$

The initial value X0 is called the seed, a is constant multiplier, c is the increment and m is the modulus. Maximal period can be achieved by a, c, m, X0 satisfying one of the following conditions

1. For m, a power of 2 ($m = 2b$) and $c \neq 0$ period $p = 2b$ is achieved provided c is relatively prime to m and $a = 1+4k$, $k = 0,1,2,\ldots$
2. For $m = 2b$ and $c = 0$, period $p = 2b-2$ is achieved provided X0 is odd and multiplier a $= 3+8k$ or $a = 5+8k$, $k = 0,1,2,\ldots$
3. For m a prime number and $c = 0$, period $p = m-1$ is achieved provided a has the property that the smallest integer is such that a k-1 is divisible by m is $k = m-1$.

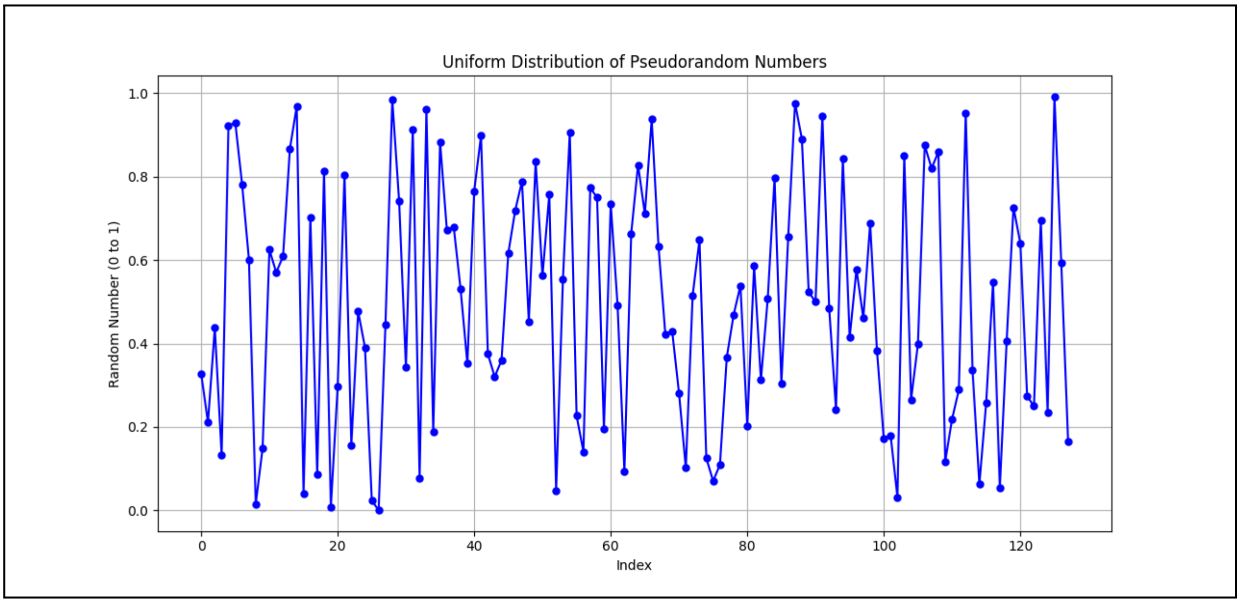
Results: (Program printout with output / Document printout as per the format)

```
import matplotlib.pyplot as plt

def linear_congruential_generator(seed, a, c, m, n):
    x = seed
    random_numbers = []
    seen = {}
    period = 0

    for i in range(n):
        x = (a * x + c) % m
        if x in seen:
            period = i - seen[x]
            break
        seen[x] = i
        random_numbers.append(x / m)

    if period < 100:
        print("Period is less than 100 or zero. Adjusting parameters to ensure a longer period.")
        increment_step = max(1000, m // 1000)
        while period < 100:
            a = (a + increment_step) % m
            x = seed
            random_numbers = []
```

```
            seen = {}
            period = 0

            for i in range(n * 2):
                x = (a * x + c) % m
                if x in seen:
                    period = i - seen[x]
                    break
                seen[x] = i
                random_numbers.append(x / m)

    return random_numbers, period

def find_period(random_numbers):
    seen = {}
    for i, num in enumerate(random_numbers):
        if num in seen:
            return i - seen[num]
        seen[num] = i
    return 0

print("Linear Congruential Method Parameters")
seed = int(input("Enter the seed (X0): "))
a = int(input("Enter the multiplier (a): "))
c = int(input("Enter the increment (c): "))
m = int(input("Enter the modulus (m): "))
n = int(input("Enter the number of random numbers to generate (n): "))

random_numbers, period = linear_congruential_generator(seed, a, c, m, n)

print("\nGenerated Random Numbers (Scaled to [0, 1]):")
print(random_numbers)

print(f"\nPeriod of the sequence: {period}\n")

plt.figure(figsize=(10, 6))
plt.plot(range(len(random_numbers)), random_numbers, marker="o",
linestyle="-", color="blue", markersize=5)
plt.title("Uniform Distribution of Pseudorandom Numbers")
plt.xlabel("Index")
plt.ylabel("Random Number (0 to 1)")
plt.grid(True)
plt.show()
```

Output:

```
PS C:\Users\Dell\Downloads\VI SEM\MS\EXP01> & C:/Users/Dell/AppData/Local/Programs/Python/Python313/python.exe
"c:/Users/Dell/Downloads/VI SEM/MS/EXP01/EXP01.py"
Linear Congruential Method Parameters
Enter the seed (X0): 123456789
Enter the multiplier (a): 1103515245
Enter the increment (c): 12345
Enter the modulus (m): 128
Enter the number of random numbers to generate (n): 500
0.5546875, 0.90625, 0.2265625, 0.140625, 0.7734375, 0.75, 0.1953125, 0.734375, 0.4921875, 0.09375, 0.6640625, 0
.828125, 0.71090.5546875, 0.90625, 0.2265625, 0.140625, 0.7734375, 0.75, 0.1953125, 0.734375, 0.4921875, 0.0937
5, 0.6640625, 0.828125, 0.0.5546875, 0.90625, 0.2265625, 0.140625, 0.7734375, 0.75, 0.1953125, 0.734375, 0.4921
0.5546875, 0.90625, 0.2265625, 0.140625, 0.7734375, 0.75, 0.1953125, 0.734375, 0.4921875, 0.09375, 0.6640625, 0
.828125, 0.7109375, 0.9375, 0.6328125, 0.421875, 0.4296875, 0.28125, 0.1015625, 0.515625, 0.6484375, 0.125, 0.0
703125, 0.109375, 0.3671875, 0.46875, 0.5390625, 0.203125, 0.5859375, 0.3125, 0.5078125, 0.796875, 0.3046875, 0
.65625, 0.9765625, 0.890625, 0.5234375, 0.5, 0.9453125, 0.484375, 0.2421875, 0.84375, 0.4140625, 0.578125, 0.46
09375, 0.6875, 0.3828125, 0.171875, 0.1796875, 0.03125, 0.8515625, 0.265625, 0.3984375, 0.875, 0.8203125, 0.859
375, 0.1171875, 0.21875, 0.2890625, 0.953125, 0.3359375, 0.0625, 0.2578125, 0.546875, 0.0546875, 0.40625, 0.726
5625, 0.640625, 0.2734375, 0.25, 0.6953125, 0.234375, 0.9921875, 0.59375, 0.1640625]

Period of the sequence: 128
```



1. For m, a power of 2 (m = 2b) and c≠0 period p = 2b is achieved provided c is relatively prime to m and a = 1+4k , k = 0,1,2,…

Linear Congruential Method Parameters
Enter the seed (X0): 1
Enter the multiplier (a): 5
Enter the increment (c): 3
Enter the modulus (m): 8
Enter the number of random numbers to generate (n): 10

Generated Random Numbers (Scaled to [0, 1]):
[0.0, 0.375, 0.25, 0.625, 0.5, 0.875, 0.75, 0.125, 0.0, 0.375]

Period of the sequence: 8

2. For m = 2b and c = 0 , period p = 2b-2 is achieved provided X0 is odd and multiplier a = 3+8k or a = 5+8k , k = 0,1,2,…

Linear Congruential Method Parameters
Enter the seed (X0): 1
Enter the multiplier (a): 3
Enter the increment (c): 0
Enter the modulus (m): 8
Enter the number of random numbers to generate (n): 10

Generated Random Numbers (Scaled to [0, 1]):
[0.375, 0.125, 0.375, 0.125, 0.375, 0.125, 0.375, 0.125, 0.375, 0.125]

Period of the sequence: 2

3. For m a prime number and c = 0, period p = m-1 is achieved provided a has the property that the smallest integer is such that a k-1 is divisible by m is k = m-1.

Linear Congruential Method Parameters
Enter the seed (X0): 1
Enter the multiplier (a): 3
Enter the increment (c): 0
Enter the modulus (m): 7
Enter the number of random numbers to generate (n): 10

Generated Random Numbers (Scaled to [0, 1]):
[0.42857142857142855, 0.2857142857142857, 0.8571428571428571, 0.5714285714285714, 0.7142857142857143, 0.14285714285714285, 0.42857142857142855, 0.2857142857142857, 0.8571428571428571, 0.5714285714285714]

Period of the sequence: 6

Questions:

1. **List down a few real life applications using random numbers as input.**
   - Cryptography (e.g., key generation).
   - Simulations (e.g., Monte Carlo simulations).
   - Gaming (e.g., dice rolls, shuffling cards).
   - Sampling in statistical experiments.
   - Machine learning (e.g., initializing weights).
   - Procedural content generation in games (e.g., terrain or level design).

2. **List the methods for generating random numbers.**
   - Linear Congruential Method (LCM).
   - Middle-Square Method.
   - Mersenne Twister.
   - Additive or Multiplicative Lagged Fibonacci Generator.
   - Hardware-based true random number generators.
   - Cryptographic PRNGs (e.g., Blum Blum Shub).

Outcomes: Generate pseudorandom numbers and perform empirical tests to measure the quality of a pseudorandom number generator

---

Conclusion: (Conclusion to be based on outcomes)
Using the Linear Congruential Method, we successfully generated a sequence of pseudorandom numbers with a period and uniformity depending on the chosen parameters (a, c, m, and seed). The numbers were scaled to the range [0, 1] and visualized to verify their distribution. This approach demonstrates the fundamental principles of pseudorandom number generation and how parameters influence the statistical properties of the generated numbers.

Grade: AA / AB / BB / BC / CC / CD /D

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites: Text Book:
Banks J., Carson J. S., Nelson B. L., and Nicol D. M., “Discrete Event System Simulation”, Pearson Education.

Websites:
[1] http://en.wikipedia.org/wiki/Pseudorandom_number_generator
[2] http://en.wikipedia.org/wiki/Linear_congruential_generator

---

.