INSTITUTE OF ENGINEERING

ADVANCED COLLEGE OF ENGINEERING AND MANAGEMENT

KALANKI, KATHMANDU

(AFFILIATED TO TRIBHUVAN UNIVERSITY)

LAB REPORT

SUBJECT: SIMULATION & MODELING

LAB NO: 02

SUBMITTED BY:

NAME: DIPESH DHUNGANA

ROLL NO: ACE077BCT035

DATE: December 27, 2024

SUBMITTED TO:

DEPARTMENT OF COMPUTER AND ELECTRONICS

TITLE: GENERATION OF RANDOM NUMBER USING PSEUDO RANDOM AND CHI-SQUARE TEST

OBJECTIVE:

- TO GENERATE SEQUENCE OF RANDOM NUMBERS USING PSEUDO RANDOM ALGORITHM AND PLOT THE SEQUENCE.
- TO EVALUATE UNIFORMITY OF THE GENERATED SEQUENCE USING CHI-SQUARE TEST.

THEORY:

Random numbers refer to a sequence of numbers chosen unpredictably from a defined set, ensuring that each number has an equal probability of being selected. For a sequence to be truly random, it must satisfy two key conditions:

1. The values should be uniformly distributed over a specified range.
2. Future values should be independent of past or present values, making them unpredictable.

Despite their seemingly arbitrary nature, random numbers play a crucial role in various real-world applications. In cryptography, they are used for secure encryption of data. Simulations, such as Monte Carlo methods, rely on random numbers to model complex systems. In machine learning, they help initialize model parameters before training. Additionally, in scientific research, random sampling techniques ensure unbiased statistical analysis and accurate survey results. The generation of random numbers is essential across multiple fields, making them a fundamental aspect of computational and statistical processes.

The search for a perfect source of randomness has been ongoing for years. In nature, unpredictable phenomena like electron movement and charge carrier fluctuations are considered among the most random processes observed. Today, random numbers can be produced using hardware-based or software-based methods. Hardware methods rely on natural physical processes, while software methods generate sequences known as pseudorandom numbers, which follow deterministic rules but appear random. Some commonly used algorithms for generating random numbers include:

- **Linear Congruential Generator (LCG)**: This method generates random numbers using modular arithmetic, making it simple and efficient for many applications.
- **Mersenne Twister**: This algorithm is widely used because it produces high-quality random numbers with a very long period, ensuring minimal repetition.
- **XOR Shift Generator**: This approach generates random sequences by repeatedly applying bitwise XOR and shift operations, making it both fast and lightweight.

- **Cryptographically Secure Pseudorandom Number Generator (CSPRNG)**: This type of generator is designed for security applications, ensuring that the numbers it produces are unpredictable and suitable for encryption.

Linear Congruential Generator

Linear Congruential Generator (LCG) is one of the oldest and best-known pseudorandom number generator algorithms. LCGs are fast and require minimal memory to retain state. It makes the algorithm valuable for simulating multiple independent streams. However, LCGs should not be intended for cryptographic applications due to it having too small a state. The theory behind is easy to understand and easy to implement especially on a computer hardware which can provide modular arithmetic by storage-bit truncation. The generator is defined by the recurrence relation:

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

Where X is the sequence of pseudo random values and the constants are m,

$0 < m$ – is the "modulus" a,
$0 < a < m$ – is the "multiplier"
c, $0 \leq c < m$ – the "increment"
X_0, $0 \leq X_0 < m$ – the "seed" or "start value"

Chi-Square Test (χ^2)

Any generated sequence of numbers should be tested for "randomness" as it ensures to meet the requirement of the application they're used for. If randomness of the generated numbers isn't of great quality, then many problems can occur where such numbers are used. For instance, simulation results may be inaccurate, security can be compromised in case of cryptography, outcomes in games may not appear fair leading to dissatisfaction or financial loss (gambling games). To test the randomness of the sequence generated and the algorithm in general, chi-square test (χ^2) is used. Chi square test uses the sample statistic:

$$\chi^2 = \sum_{i=0}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where,
O_i is the observed number in the i^{th} instance
E_i is the expected number in the i^{th} instance
n is the number of bins of chi-square test

Here the Ei is calculated as:

$E_i = N / n$

where, N is the total number of random numbers generated.

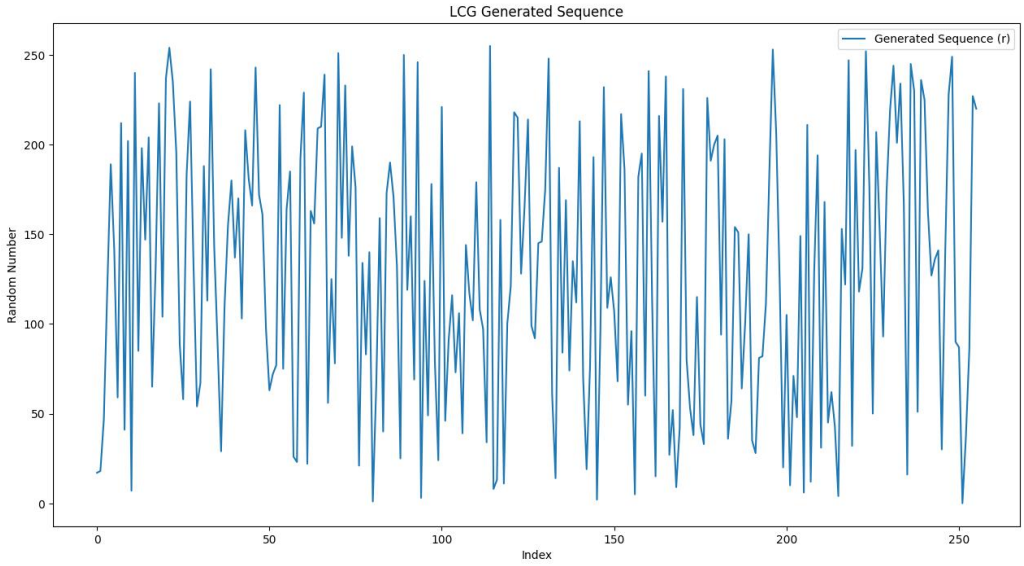
The obtained chi square value is then compared to the critical value of chi square value determined by significance level (α) and degree of freedom (n – 1). If the obtained value is less than or equal to the critical value then the generated sample are “random” else the sample is not seen as “random” enough

SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2
# Constants
iterator = 256  # Number of random numbers to generate
a = 29  # Multiplier
b = 37  # Increment
p = 256  # Modulus
n = 8  # Number of bins for Chi-Square test
x02 = 0  # Chi-Square statistic initialization
critical_value = chi2.ppf(1 - 0.05, df=n - 1)  # Corrected critical value
# Initialize arrays
r = np.ones(iterator, dtype=int)  # Array to store generated random numbers
ei = iterator / n  # Expected frequency for each bin
c = np.zeros(n, dtype=int)  # Bin counts
# Initial seed
r[0] = 17
# Generate random numbers using LCG
for i in range(1, iterator):
    r[i] = (r[i - 1] * a + b) % p
# Bin the generated numbers using numpy's histogram
bin_edges = np.linspace(0, p, n + 1)  # Create bin edges from 0 to 256 with 8
bins
c, _ = np.histogram(r, bins=bin_edges)  # Count occurrences in each bin
# Compute Chi-Square statistic
x02 = np.sum(((c - ei) ** 2) / ei)
# Print results
print("Values of r:", r)
print("Values of c (bin frequencies):", c)
print("Value of Chi-Square statistic (x02):", x02)
print("The critical value is", critical_value)
print("Chi square test passed" if x02 <= critical_value else "Chi square test
failed")
```

```
# Visualize the generated random numbers
plt.figure()
plt.plot(r, label="Generated Sequence (r)")
plt.xlabel("Index")
plt.ylabel("Random Number")
plt.title("LCG Generated Sequence")
plt.legend()
plt.show()
```

OUTPUT



(LCG GENERATED SEQUENCE)

Values of r: [17 18 47 120 189 142 59 212 41 202   7 240 85 198 147 204 65 130 223 104 237 254 235 196 89 58 183 224 133 54 67 188 113 242 143 88 29 110 155 180 137 170 103 208 181 166 243 172 161 98 63 72 77 222 75 164 185 26 23 192 229 22 163 156 209 210 239 56 125 78 251 148 233 138 199 176 21 134 83 140   1 66 159 40 173 190 171 132 25 250 119 160 69 246   3 124 49 178 79 24 221 46 91 116 73 106 39 144 117 102 179 108 97 34 255   8 13 158 11 100 121 218 215 128 165 214 99 92 145 146 175 248 61 14 187 84 169 74 135 112 213 70 19 76 193   2 95 232 109 126 107 68 217 186 55 96   5 182 195 60 241 114 15 216 157 238 27 52   9 42 231 80 53 38 115 44 33 226 191 200 205 94 203 36 57 154 151 64 101 150 35 28 81 82 111 184 253 206 123 20 105 10 71 48 149   6 211 12 129 194 31 168 45 62 43   4 153 122 247 32 197 118 131 252 177 50 207 152 93 174 219 244 201 234 167 16 245 230 51 236 225 162 127 136 141 30 139 228 249 90 87   0 37 86 227 220]

Values of c (bin frequencies): [32 32 32 32 32 32 32 32]

Value of Chi-Square statistic (x02): 0.0

The critical value is 0.014067140449340169

Chi square test passed

DISCUSSION AND CONCLUSION:

In this lab we simulated the LCG algorithm to generate 256 random numbers and the random numbers are tested using Chi-squared test. We considered the following constants:

iterator = 256 (Number of random numbers to generate)
a = 29 (Multiplier)
b = 37 (Increment)
p = 256 (Modulus)
n = 8 (Number of bins for Chi-Square test)
x02 = 0 (Chi-Square statistic initialization)
critical_value = chi2.pdf(1 - 0.05, df = n - 1) (Chi-Square critical value)

We learned about random numbers and their importance of simulation and modeling along with its functional applications in the fields like cryptography, machine learning, scientific studies, etc. In the lab, we used python libraries numpy and matplotlib for coding the LCG algorithm to generate the random numbers and plotted the generated sequence respectively. Likewise, we also used chi2 module from scipy.stats to get the critical value for the chi-square test. After performing the chi-square test, we found the obtained value of chi-square to be 0.0 which is less than the critical value of 0.014549 with significance level (α) equals to 0.05 and degree of freedom to be 7. This infers that the numbers generated by LCG in our case is random as it fails to reject the null hypothesis. Thus, the lab objectives of generating random numbers and checking their uniformity were met and the lab was concluded successfully