Experiment No.: 4

Title: Implementation of Independence test

Batch: B–1 **Roll No.: 16010422234** **Experiment No.: 4**

Aim: To implement Autocorrelation test / Runs test to perform Independence test of generated random numbers.

Resources needed: Turbo C / Java / python

Theory

Problem Statement:
Write function in C / C++ / java / python or macros in MS-excel to implement Autocorrelation / Runs test.

Concepts:
Random Numbers generated using a known process or algorithm is called Pseudo random Number.The random numbers generates must possess the property of :

1. Uniformity
2. Independence

Tests for Independence:
These tests are done to check the independence of sequences of random numbers.

1. **Runs Test**
   This test analyses an orderly grouping of numbers in a sequence to test the hypothesis of independence. A Run is defined as a succession of similar events preceded and followed by a different. The length of the run is the number of events that occur in the run.
   In all cases, actual values are compared with expected values using the chi square test.
   The Runs test used re:
   1. Runs Up and Down
   2. Runs above and below the mean
   3. Runs test for testing length of runs

   Runs Up and Down:
   In a sequence of numbers, if a number is followed by a larger number, this is an upward run. Likewise, a number followed by a smaller number is a downstream run. The numbers are given + and – depending on whether they are followed by larger or smaller numbers. The last number is followed by no event. Eg. 10 numbers there will be 9 + or -. If the numbers are truly random, one would expect to find a certain number of runs up and down.
   In a sequence of N numbers, a is the total no of runs, the mean and variance is given by the following equation:

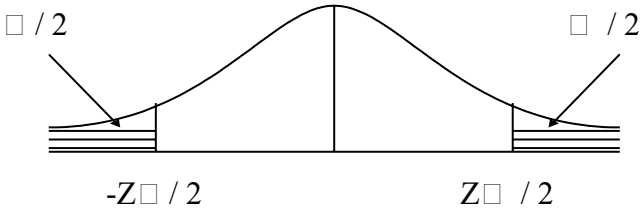
$$\mu = \frac{(2N-1)}{3} \qquad \sigma^2 = \frac{(16N-29)}{90}$$

For N > 20, the distribution of "a" is approximated by a normal distribution, N(0,1). This approximation can be used to test the independence of numbers from a generator. Finally, the standardised normal test statistics ,Zo is developed and compared with critical value

$Z_0 = (a - \mu) / \sigma$

Where a is total no of runs.

Acceptance region for hypothesis of independence $-Z_{a/2} \leq Z_0 \leq Z_{a/2}$



2. Autocorrelation Test: The test for autocorrelation is concerned with dependence between numbers in a sequence. The test computes auto correlation between every m numbers starting with the ith number. Thus the autocorrelation limit between following numbers would be of interest.

Ri , Ri+m , Ri+2m , Ri+(M+1)m

where M is the largest integer such that $i+(M+1)m \leq N$ where N is the total number of values in the sequence.

Since the nonzero autocorrelation implies a lack of independence, the following test is appropriate:

$H_0: \rho_{im} = 0,$ if numbers are independent

$H_1: \rho_{im} \neq 0,$ if numbers are dependent

For large values of M, the distribution of the estimator of ρim, denoted $\hat{\rho}_{im}$ is approximately normal, if the values Ri , Ri+m , Ri+2m , Ri+(M+1)m are uncorrelated.

The test statistics is

$$Z_0 = \frac{\hat{\rho}_{im}}{\hat{\sigma}_{\hat{\rho}_{im}}}$$

with a mean of 0 and variance of 1,under the assumption of independence , for large M.

If $-Z_{\alpha/2} \leq Z_0 \leq Z_{\alpha/2}$, H0 is not rejected for the significance level α .

3. Gap Test: The gap test is used to determine the significance of the interval between reoccurrence of the same digit. A gap of length x occurs between recurrence of same digit.

4. Poker Test: The poker test for independence is based on frequency with which certain digits are repeated in a series of numbers in each case a pair of like digits appear in the numbers that were generated. In 3 digit sample of numbers there are three possibilities which are as follows:

1. The individual numbers can all be different
2. The individual numbers can all be same

3. There can be one pair of like digits.

Procedure: (Write the algorithm for the test to be implemented and follow the steps given below)

Steps:

- Implement either Autocorrelation Test or Runs test using C / C++ / java or macros in MS-excel
- Generate 5 sample sets (Each set consisting of 100 random numbers) of Pseudo
- random numbers using Linear Congruential Method.
- Execute the test using all the five sample sets of random numbers as input and using α=0.05.
- Draw conclusions on the acceptance or rejection of the null hypothesis of independence.

Results: (Program printout with output)

```
from math import sqrt

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 runs_test(random_numbers):
    n = len(random_numbers)
    runs_up_down = 1
    runs_above_below = 1
    plus_minus_up_down = []
    plus_minus_above_below = []

    for i in range(1, n):
        if random_numbers[i] > random_numbers[i - 1]:
            plus_minus_up_down.append("+")
        else:
            plus_minus_up_down.append("-")

        if random_numbers[i] > random_numbers[i - 1] and random_numbers[i
- 1] <= random_numbers[i - 2]:
            runs_up_down += 1
        elif random_numbers[i] < random_numbers[i - 1] and
random_numbers[i - 1] >= random_numbers[i - 2]:
            runs_up_down += 1

    mean_value = sum(random_numbers) / n
    for i in range(n):
        if random_numbers[i] > mean_value:
            plus_minus_above_below.append("+")
        else:
            plus_minus_above_below.append("-")

        if i > 0 and ((random_numbers[i] > mean_value and
random_numbers[i - 1] <= mean_value) or \
                      (random_numbers[i] <= mean_value and
random_numbers[i - 1] > mean_value)):
```

```
            runs_above_below += 1

    mean_up_down = (n + 1) / 2
    variance_up_down = (16 * n - 29) / 90
    z0_up_down = (runs_up_down - mean_up_down) / sqrt(variance_up_down)

    mean_above_below = (n + 1) / 2
    variance_above_below = (16 * n - 29) / 90
    z0_above_below = (runs_above_below - mean_above_below) /
sqrt(variance_above_below)

    return (runs_up_down, runs_above_below, mean_up_down,
variance_up_down, z0_up_down,
            mean_above_below, variance_above_below, z0_above_below,
            plus_minus_up_down, plus_minus_above_below)

def main():
    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")

    (runs_up_down, runs_above_below, mean_up_down, variance_up_down,
z0_up_down,
     mean_above_below, variance_above_below, z0_above_below,
     plus_minus_up_down, plus_minus_above_below) =
runs_test(random_numbers)

    print("\nRuns Test Results:")

    print("\nRuns Up and Down:")
    print(f"Total Runs: {runs_up_down}")
    print(f"Expected Runs (Mean): {mean_up_down:.2f}")
```

```
    print(f"Variance: {variance_up_down:.2f}")
    print(f"Z Statistic: {z0_up_down:.2f}")
    print(f"Number of '+': {plus_minus_up_down.count('+')} \nNumber of
'-': {plus_minus_up_down.count('-')}")

    print("\nRuns Above and Below the Mean:")
    print(f"Total Runs: {runs_above_below}")
    print(f"Expected Runs (Mean): {mean_above_below:.2f}")
    print(f"Variance: {variance_above_below:.2f}")
    print(f"Z Statistic: {z0_above_below:.2f}")
    print(f"Number of '+': {plus_minus_above_below.count('+')} \nNumber
of '-': {plus_minus_above_below.count('-')}")

    z_alpha = 1.96

    print("\nHypothesis Testing for Up and Down:")
    if -z_alpha <= z0_up_down <= z_alpha:
        print("The null hypothesis of independence is not rejected
(Accept H0).")
    else:
        print("The null hypothesis of independence is rejected (Reject
H0).")

    print("\nHypothesis Testing for Above and Below the Mean:")
    if -z_alpha <= z0_above_below <= z_alpha:
        print("The null hypothesis of independence is not rejected
(Accept H0).")
    else:
        print("The null hypothesis of independence is rejected (Reject
H0).")

if __name__ == "__main__":
    main()
```

```
PS C:\Users\Admin\Desktop\CHANDANA> & C:/Users/Admin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Admin/Desktop/CHANDANA/EXP4.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

Generated Random Numbers (Scaled to [0, 1]):
[0.328125, 0.2109375, 0.4375, 0.1328125, 0.921875, 0.9296875, 0.78125, 0.6015625, 0.015625, 0.1484375, 0.625, 0.5703125, 0.609375, 0.8671875, 0.96875, 0.0390625, 0.703125, 0.0859375, 0.8125, 0.0078125, 0.296875, 0.8046875, 0.15625,
 0.4765625, 0.390625, 0.0234375, 0.0, 0.4453125, 0.984375, 0.7421875, 0.34375, 0.9140625, 0.078125, 0.9609375, 0.1875, 0.8828125, 0.671875, 0.6796875, 0.53125, 0.3515625, 0.765625, 0.8984375, 0.375, 0.3203125, 0.359375, 0.6171875,
0.71875, 0.7890625, 0.453125, 0.8359375, 0.5625, 0.7578125, 0.046875, 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, 0.0703125, 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, 0.4
84375, 0.2421875, 0.84375, 0.4140625, 0.578125, 0.4609375, 0.6875, 0.3828125, 0.171875, 0.1796875, 0.03125, 0.8515625, 0.265625, 0.3984375, 0.875, 0.8203125, 0.859375, 0.1171875, 0.21875, 0.2890625, 0.953125, 0.3359375, 0.0625, 0.2
578125, 0.546875, 0.0546875, 0.40625, 0.7265625, 0.640625, 0.2734375, 0.25, 0.6953125, 0.234375, 0.9921875, 0.59375, 0.1640625]

Period of the sequence: 128


Runs Test Results:

Runs Up and Down:
Total Runs: 84
Expected Runs (Mean): 64.50
Variance: 22.43
Z Statistic: 4.12
Number of '+': 64
Number of '-': 63

Runs Above and Below the Mean:
Total Runs: 63
Expected Runs (Mean): 64.50
Variance: 22.43
Z Statistic: -0.32
Number of '+': 64
Number of '-': 64

Hypothesis Testing for Up and Down:
The null hypothesis of independence is rejected (Reject H0).

Hypothesis Testing for Above and Below the Mean:
The null hypothesis of independence is not rejected (Accept H0).
PS C:\Users\Admin\Desktop\CHANDANA>
```

Questions:

1) Give an example and interpret the need for the Independence test.

Ans: Example – Consider a random number generator used in a simulation of customer arrivals at a bank. If the numbers are not independent, it may create patterns that don't reflect real-world randomness, skewing the simulation results.

Interpretation – The independence test ensures that random numbers do not show any predictable patterns, making them suitable for simulations, cryptography, or statistical sampling.

2) What is Type 1 and Type 2 error?

Ans: Type 1 Error – Rejecting the null hypothesis (H_0) when it is actually true (false positive).

Type 2 Error – Failing to reject the null hypothesis (H_0) when it is actually false (false negative).

3) What independence tests make use of the Chi square test?

Ans: The Runs Test and the Poker Test make use of the Chi-square test to compare observed values with expected values, validating the independence of the random numbers.

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

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Jerry Banks, Jerry Banks, John Carson, Barry Nelson, and David M. Nicol; “Discrete EventSystem System Simulation”, Third Edition, Pearson Education
2. "Linear Congruential Generators" by Joe Bolte, Wolfram Demonstrations Project.
3. "A collection of selected pseudorandom number generators with linear structures, K. Entacher, 1997". Retrieved 16 June 2012.
4. GNU Scientific Library: Other random number generators
5. S.K. Park and K.W. Miller (1988). "Random Number Generators: Good Ones Are Hard To Find". Communications of the ACM 31 (10): 1192–1201. doi:10.1145/63039.63042.
6. D. E. Knuth. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89684-2. Section 3.2.1: The Linear Congruential Method, pp. 10–26.
7. P. L'Ecuyer (1999). "Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure". Mathematics of Computation 68 (225): 249–260. doi:10.1090/S0025-5718-99-00996-5.
8. Gentle, James E., (2003). Random Number Generation and Monte Carlo Methods, 2nd edition, Springer, ISBN 0-387-00178-6.
9. Joan Boyar (1989). "Inferring sequences produced by pseudo-random number generators". Journal of the ACM 36 (1): 129–141. doi:10.1145/58562.59305. (in this paper, efficient algorithms are given for inferring sequences produced by certain pseudo-random number generator