Lab 7

Objectives:

The objective of this lab is to implement and understand various statistical tests for assessing the quality of random number generators. The Auto-Correlation Test is used to evaluate the correlation between a sequence of numbers and a shifted version of itself, helping to identify patterns or dependencies in random sequences. The Poker Test checks for uniformity in the distribution of random numbers by grouping them and comparing the frequency of each group, which is crucial for testing the randomness of number generators. The Chi-Square Test is implemented to measure the goodness-of-fit between observed and expected frequencies, helping to determine if a random sequence follows a particular distribution. By implementing these tests, the lab aims to develop skills in validating the randomness and uniformity of generated sequences, which are essential for various applications in statistical analysis and simulations.

Q.1. WAP to implement auto correlation test.

Source Code:

```
#include <stdio.h>
#define MAX 50  // Maximum data points
int main() {
    int n, d, i;
    float data[MAX], sum = 0.0, R_d;
    printf("Enter number of data points (max %d): ", MAX);
    scanf("%d", &n);
    printf("Enter %d random numbers (0 to 1):\n", n);
    for (i = 0; i < n; i++)
        scanf("%f", &data[i]);
    printf("Enter lag value (d): ");
    scanf("%d", &d);
    if (d >= n) {
        printf("Invalid lag value!\n");
        return 1;
    }
    for (i = 0; i < n - d; i++)
        sum += data[i] * data[i + d];
    R_d = sum / (n - d);
    printf("Auto-Correlation Coefficient (R_%d) = %.4f\n", d, R_d);
    return 0;
}
```

Output:

```
Enter number of data points (max 50): 5
Enter 5 random numbers (0 to 1):
0.1 0.4 0.7 0.2 0.9
Enter lag value (d): 1
Auto-Correlation Coefficient (R_1) = 0.1600
[1] + Done                       "/usr/bin/gdb" --interpreter=mi
>"/tmp/Microsoft-MIEngine-Out-32eu1lga.j0q"
● → 23081024 git:(main) ✗ pwd
/home/d33pan/docs/Studies/5th sem/simulationAndModeling/23081024
```

Q.2. WAP to implement Poker test.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 50  // Maximum random numbers

// Function to count unique digits in a 5-digit group

int count_unique_digits(int num) {

    int digits[10] = {0}, count = 0, digit;

    while (num > 0) {

        digit = num % 10;

        if (digits[digit] == 0) {

            digits[digit] = 1;

            count++;

        }

        num /= 10;

    }

    return count;
```

```
}
int main() {
    int n, i, groups, num, unique_count[6] = {0};  // To store frequency of unique digits
   printf("Enter the number of random numbers (max %d): ", MAX);
   scanf("%d", &n);
   if (n < 5 || n > MAX) {
      printf("Invalid input! Enter at least 5 numbers.\n");
      return 1;
   }
   groups = n / 5;  // Number of 5-digit groups
   printf("Enter %d numbers (each 5-digit long):\n", groups);
   for (i = 0; i < groups; i++) {
      scanf("%d", &num);
      if (num < 10000 || num > 99999) {
         printf("Invalid input! Enter a 5-digit number.\n");
         return 1;
      }
      unique_count[count_unique_digits(num)]++;
   }
   // Calculate Poker Test Statistic
   float X2 = 0.0;
   for (i = 1; i <= 5; i++)
      X2 += (unique_count[i] * unique_count[i]);

   X2 = ((10.0 / groups) * X2) - groups;
```

```
    // Output results
    printf("\nPoker Test Statistic (X^2) = %.4f\n", X2);
    printf(X2 > 9.49 ? "Sequence is NOT random!\n" : "Sequence appears random!\
n");
    return 0;
}
```

Output:

```
Enter the number of random numbers (max 50): 5
Enter 1 numbers (each 5-digit long):
12345 23456 34567 45678 56789

Poker Test Statistic (X^2) = 9.0000
Sequence appears random!
[1] + Done                       "/usr/bin/gdb" --interpreter=mi -
>"/tmp/Microsoft-MIEngine-Out-wbhjgby2.b2r"
➜  23081024 git:(main) ✗ pwd
/home/d33pan/docs/Studies/5th sem/simulationAndModeling/23081024
```

Q.3. WAP to implement chi-square test.

Source Code:

```
#include <stdio.h>
#define MAX 100  // Maximum number of random values
#define INTERVALS 10  // Number of frequency bins
int main() {
    int n, i, observed[INTERVALS] = {0};
    float expected, chi_square = 0.0, random_numbers[MAX];
    // Input: Number of random values
    printf("Enter number of random values (max %d): ", MAX);
    scanf("%d", &n);
```

```
    if (n <= 0 || n > MAX) {
        printf("Invalid input!\n");
        return 1;
    }
    // Input: Random numbers (0 to 1)
    printf("Enter %d random numbers (0 to 1):\n", n);
    for (i = 0; i < n; i++) {
        scanf("%f", &random_numbers[i]);
       observed[(int)(random_numbers[i] * INTERVALS)]++; // Categorizing into bins
    }
    // Expected frequency for uniform distribution
    expected = (float)n / INTERVALS;
    // Compute Chi-Square statistic
    for (i = 0; i < INTERVALS; i++) {
        chi_square += ((observed[i] - expected) * (observed[i] - expected)) / expected;
    }
    // Output results
    printf("\nChi-Square Statistic  = %.4f\n", chi_square);
    printf(chi_square > 16.92 ? "Sequence is NOT random!\n" : "Sequence appears
random!\n");
    return 0;
}
```

Output:

```
Enter number of random values (max 100): 5
Enter 5 random numbers (0 to 1):
0.05 0.23 0.45 0.67 0.88

Chi-Square Statistic= 5.0000
Sequence appears random!
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --
>"/tmp/Microsoft-MIEngine-Out-lzqlygo0.vrd"
➜  23081024 git:(main) ✗ pwd
/home/d33pan/docs/Studies/5th sem/simulationAndModeling/23081024
```

Conclusion:

In conclusion, this lab provides hands-on experience in implementing key statistical tests—Auto-Correlation, Poker, and Chi-Square—used to assess the randomness and uniformity of generated sequences. These tests play a crucial role in validating the effectiveness of random number generators, ensuring they produce unbiased and unpredictable results. By applying these tests, students gain a deeper understanding of how to analyze and improve the quality of random sequences for use in simulations, cryptography, and statistical modeling. The lab reinforces the importance of rigorous testing in computational experiments to ensure the reliability and accuracy of random data generation methods.