Lab 6

Objectives:

The use of this lab is to understand and implement fundamental random number generation methods, such as the Mixed and Multiplicative Congruential Methods, which are crucial for simulations, statistical modeling, and cryptographic applications. It also demonstrates how to test the goodness-of-fit of empirical data to a theoretical distribution using the Kolmogorov-Smirnov test. By exploring these methods, students can gain practical insights into the generation of pseudo-random numbers and the validation of statistical hypotheses. The lab enhances knowledge of algorithmic approaches for randomness and their applications in real-world problem-solving.

Q.1. WAP to generate 50 random numbers using Mixed Congruential Method where X0=11, m=100, a = 5 and c = 13.

Source Code:

```
#include <stdio.h>

int main() {
    int X0 = 11, a = 5, c = 13, m = 100;
    int Xn = X0;
    int i; // Declare variables outside loops
    printf("Generated 50 random numbers using Mixed Congruential Method:\n");
    for (i = 0; i < 50; i++) {
        Xn = (a * Xn + c) % m;
        printf("%d ", Xn);
        if ((i + 1) % 10 == 0)  // Print 10 numbers per line
            printf("\n");
    }
    return 0;
}
```

Output:

```
Generated 50 random numbers using Mixed Congruential Method:
68 53 78 3 28 53 78 3 28 53
78 3 28 53 78 3 28 53 78 3
28 53 78 3 28 53 78 3 28 53
78 3 28 53 78 3 28 53 78 3
28 53 78 3 28 53 78 3 28 53
[1] + Done                       "/usr/bin/gdb" --interpreter=mi
>"/tmp/Microsoft-MIEngine-Out-kg5knn2y.p4k"
➜  23081024 git:(main) ✗ pwd
/home/d33pan/docs/Studies/5th sem/simulationAndModeling/23081024
➜  23081024 git:(main) ✗
```

Q.2. WAP to generate 50 random numbers using Multiplicative Congruential Method where X0=13, m =1000, a = 15 and c = 7.

Source Code:

```
#include <stdio.h>

int main() {

   int X0 = 13, a = 15, m = 1000;

   int Xn = X0;

   int i; // Declare loop variable outside (for old compilers)

      printf("Generated 50 random numbers using Multiplicative Congruential Method:\n");

   for (i = 0; i < 50; i++) {

      Xn = (a * Xn) % m; // Multiplicative congruential formula

      printf("%d ", Xn);

      if ((i + 1) % 10 == 0) // Print 10 numbers per line

         printf("\n");

   }

   return 0;

}
```

Output:

```
Generated 50 random numbers using Multiplicative Congruential Method:
195 925 875 125 875 125 875 125 875 125
875 125 875 125 875 125 875 125 875 125
875 125 875 125 875 125 875 125 875 125
875 125 875 125 875 125 875 125 875 125
875 125 875 125 875 125 875 125 875 125
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty
>"/tmp/Microsoft-MIEngine-Out-spfpq024.bwi"
➜  23081024 git:(main) ✗ pwd
/home/d33pan/docs/Studies/5th sem/simulationAndModeling/23081024
```

Q.3. WAP to implement Kolmogorov – Smirnov test.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50  // Max data points
// Function to sort the array (Bubble Sort for compatibility)
void sort(float arr[], int n) {
    int i, j;
    float temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
// Kolmogorov-Smirnov Test Function
void kolmogorov_smirnov(float data[], int n) {
    float D_plus[MAX], D_minus[MAX], D, max_D_plus = 0, max_D_minus = 0;
    int i;
    // Sorting the data
    sort(data, n);
```

```
    printf("\nSorted Data:\n");
    for (i = 0; i < n; i++) {
        printf("%.4f ", data[i]);
    }
    printf("\n\n");
    // Computing D+ and D-
    for (i = 0; i < n; i++) {
        float F_empirical = (float)(i + 1) / n;  // Empirical CDF
            float F_theoretical = data[i];  // Theoretical CDF (Uniform Distribution assumption)
        D_plus[i] = F_empirical - F_theoretical;
        D_minus[i] = F_theoretical - (float)i / n;
        if (D_plus[i] > max_D_plus) max_D_plus = D_plus[i];
        if (D_minus[i] > max_D_minus) max_D_minus = D_minus[i];
        printf("i: %d | X: %.4f | F_emp: %.4f | F_theo: %.4f | D+: %.4f | D-: %.4f\n",
            i + 1, data[i], F_empirical, F_theoretical, D_plus[i], D_minus[i]);
    }
    // Maximum D-value
    D = (max_D_plus > max_D_minus) ? max_D_plus : max_D_minus;
    printf("\nMax D+ = %.4f, Max D- = %.4f\n", max_D_plus, max_D_minus);
    printf("Kolmogorov-Smirnov D-value = %.4f\n", D);
}

int main() {
    int n, i;
    float data[MAX];
```

```
    printf("Enter the number of data points (max %d): ", MAX);
    scanf("%d", &n);
    if (n <= 0 || n > MAX) {
        printf("Invalid input. Exiting program.\n");
        return 1;
    }
    printf("Enter %d data values (between 0 and 1):\n", n);
    for (i = 0; i < n; i++) {
        scanf("%f", &data[i]);
    }
    kolmogorov_smirnov(data, n);
    return 0;
}
```

Output:

```
Enter the number of data points (max 50): 5
Enter 5 data values (between 0 and 1):
0.1 0.4 0.7 0.2 0.9

Sorted Data:
0.1000 0.2000 0.4000 0.7000 0.9000

i: 1 | X: 0.1000 | F_emp: 0.2000 | F_theo: 0.1000 | D+: 0.1000 | D-: 0.1000
i: 2 | X: 0.2000 | F_emp: 0.4000 | F_theo: 0.2000 | D+: 0.2000 | D-: 0.0000
i: 3 | X: 0.4000 | F_emp: 0.6000 | F_theo: 0.4000 | D+: 0.2000 | D-: 0.0000
i: 4 | X: 0.7000 | F_emp: 0.8000 | F_theo: 0.7000 | D+: 0.1000 | D-: 0.1000
i: 5 | X: 0.9000 | F_emp: 1.0000 | F_theo: 0.9000 | D+: 0.1000 | D-: 0.1000

Max D+ = 0.2000, Max D- = 0.1000
Kolmogorov-Smirnov D-value = 0.2000
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${Dbg
>"/tmp/Microsoft-MIEngine-Out-jq5ko5e5.sg0"
➜  23081024 git:(main) ✗ pwd
/home/d33pan/docs/Studies/5th sem/simulationAndModeling/23081024
```

Conclusion:

In conclusion, this lab provides a practical understanding of generating pseudo-random numbers using the Mixed and Multiplicative Congruential Methods, which are foundational for many computational applications, including simulations and cryptography. It also highlights the importance of the Kolmogorov-Smirnov test in validating the distribution of data and testing hypotheses about statistical models. By applying these methods, we gain valuable insights into how randomness is generated and how it can be assessed for accuracy. Overall, the lab enhances skills in random number generation and statistical analysis, which are essential for data-driven decision-making and problem-solving in various fields.