Experiment No.: 5
Title: Implementation of Uniformity test

Batch: B2 **Roll No.: 16010421059** **Experiment No.: 5**

Aim: To implement Kolmogorov –Smirnov (K S) test / Chi-square test on the random number generator implemented in experiment no 1 for uniformity testing.

Resources needed: Turbo C / Java / python

Theory

Problem Statement:
Write function in C / C++ / java / python or macros in MS-excel to implement Kolmogorov-Smirnov (KS) / Chi-square 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

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 total number of observations.

Tests for Random numbers

1) Uniformity Test

A basic test that is to be performed to validate a new generator is the test of uniformity. Two different testing methods are available, they are

a. Kolmogorov- Smirnov Test
b. Chi-square Test

Both of these measure the degree of agreement between distance of sample of generated random numbers and the theoretical uniform distributions.

a) Kolmogorov-Smirnov Test: This test compares the continuous cdf F(x) of the uniform distribution to the empirical cdf $S_N(x)$ of sample of N distribution

By definition,

$$F(x) = x \qquad 0 \le x \le 1$$

If the sample from random no. generated is $R_1, R_2, \ldots, R_N$ then the empirical cdf $S_N(x)$ is defined as

$$S_N(x) = \frac{\text{No. of } R_1, R_2, \ldots, R_N \text{ which are } x}{N}$$

As N becomes larger $S_N(x)$ should become better approximation to F(x) provided the null hypothesis is true. The Kolmogorov-Smirnov distance test is best on largest absolute deviation between F(x) & $S_N(x)$ over range of random variable.

b)) Chi square test: The Chi square test sample test statistics is:

$$\boxed{X_0^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}}$$

Where, O_i = Observed frequency in ith class E_i = Expected frequency in ith class n = is the no. ofclasses

---

Procedure:

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

Steps:

- Make a null hypothesis for uniformity
- Generate 5 sample sets (Each set consisting of 100 random numbers) of Pseudo random numbers using Linear Congruential Method implemented in expt 1
- Implement either Kolmogorov-Smirnov Test or Chi-square Test
- Execute the test usingall the fivesample sets of random numbers as input and using α=0.05.
- Draw conclusions on the acceptance or rejection of the null hypothesis of uniformity.

Results: (Program printout with output)

Main.java

```
package com.sm;

import java.io.*;

public class Main {
    public static void main(String[] args){
        try {
            KSTest ksTest = new KSTest();
            float[] ris = ksTest.randomNumbers();
            float dPlus = ksTest.dPlus(ris);
            float dMinus = ksTest.dMinus(ris);
            String H0 = ksTest.kSTest(dPlus, dMinus, ris.length);
            System.out.println("H0 is: " + H0);
        }
        catch(Exception e){}

    }
```

```
}
```

RandomNumberGenerator.java

```
package com.sm;

public class RandomNumberGenerator {
    private int X0=12,a=13,m=256,c=115;

    public float[] random(int len){
        float[] output=new float[len];
        output[0]=X0;
        for(int i=1;i<len;i++) {
            output[i] = (a * output[i - 1] + c) % m;
        }

        return output;
    }

    public void printOutput(float[] out,int len){
        StringBuilder finalOutput = new StringBuilder();
        for(int i=0;i<len;i++){
            finalOutput.append(out[i]);
            finalOutput.append(" ");
            if((i+1)%25==0){
                finalOutput.append("\n");
            }
        }
        System.out.println(finalOutput);
    }

    public float period(float[] out,int len){
        int i;
        for(i=1;i<len;i++){
            if(out[i]==X0){
                break;
            }
        }
        return i;
    }

    public float density(float[] out,int len) {
        float[] denArr = new float[len];
        for (int i = 0; i < len; i++) {
            denArr[i] = (float) out[i] / m;
        }
        float sum = 0;
        for (int i = 1; i < m; i++) {
            sum = sum + Math.abs(denArr[i-1] - denArr[i]);
        }
        return sum / m ;
    }
}
```

KSTest.java

```
package com.sm;
import java.util.*;
```

```
public class KSTest {
    float[] randomNumbers(){
        RandomNumberGenerator randomNumber=new RandomNumberGenerator();
        float[] random=randomNumber.random(10);
        for(int i = 0; i < random.length; i++){
            random[i] = random[i]/100;
        }
        Arrays.sort(random);
        System.out.println("rando numbers: ");
        for(int i=0;i<random.length;i++){
            System.out.print(random[i]+" ");
        }
        System.out.println("\n");
        return random;
    }

    float dPlus(float[] random){
        float[] d=new float[random.length];
        for (float i=1;i<=random.length;i++){
            float temp=(i/random.length)-random[(int) (i-1)];
            if(temp<0)
                d[(int) (i-1)]=0;
            else
                d[(int) (i-1)]=temp;
        }
        System.out.println("DPlus: "+Arrays.toString(d) );

        float dPlus = d[0];
        for(int i = 1; i < d.length; i++){
            if(d[i] > dPlus)
                dPlus = d[i];
        }
        return dPlus;
    }

    float dMinus(float[] random){
        float[] d=new float[random.length];
        for (float i=1;i<random.length;i++){
            float temp=random[(int) (i-1)]-((i-1)/random.length);
//              System.out.println(temp);
            if(temp<0)
                d[(int) (i-1)]=0;
            else
                d[(int) (i-1)]=temp;
        }
        System.out.println("DMinus: "+Arrays.toString(d));

        float dMinus = d[0];
        for(int i = 1; i < d.length; i++){
            if(d[i] > dMinus)
                dMinus = d[i];
        }
        return dMinus;
    }

    String kSTest(float dPlus,float dMinus,int N){
        float max=Math.max(dPlus,dMinus);
        System.out.println("D value: "+max + "\n");
        float dAlpha= (float) ((float) 1.35810/Math.sqrt(N));
        if(max>dAlpha)
            return "rejected";
        else
            return "accepted";
    }
```

```
}
```

Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:D:\java ecclipse\new intellij\IntelliJ IDEA Commu
rando numbers:
0.1 0.15 0.23 0.29 0.34 0.51 0.88 0.89 1.0 1.18

DPlus: [0.0, 0.049999997, 0.07000001, 0.110000014, 0.16, 0.09000003, 0.0, 0.0, 0.0, 0.0]
DMinus: [0.1, 0.050000004, 0.030000001, 0.0, 0.0, 0.00999999, 0.27999997, 0.19, 0.19999999, 0.0]
D value: 0.27999997

H0 is: accepted

Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:D:\java ecclipse\new intellij\IntelliJ
rando numbers:
0.12 0.15 0.2 0.49 0.54 1.19 1.63 1.86 2.29 2.4

DPlus: [0.0, 0.049999997, 0.10000001, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
DMinus: [0.12, 0.050000004, 0.0, 0.19, 0.14000002, 0.69000006, 1.03, 1.1600001, 1.49, 0.0]
D value: 1.49

H0 is: rejected

Process finished with exit code 0
```

Questions:

1. List down the pros and cons of the Kolmogorov - Smirnov test and Chi- Square test.

Answer

Kolmogorov-Smirnov Test:

Pros:

1. Non-parametric and distribution-free.
2. Sensitivity to differences in location and shape.
3. Applicable to both continuous and discrete data.

Cons:

1. Sensitivity to sample size, especially for small datasets.
2. Global comparison might miss local differences.
3. Less suitable for discrete data or data with ties.

Chi-Square Test:

Pros:

1. Versatile for categorical data and counts.
2. Suitable for goodness-of-fit and testing independence.
3. Less sensitive to sample size than the Kolmogorov-Smirnov test.

Cons:

1. Assumes independence of observations.
2. Sensitive to small expected cell sizes.

3. Limited to categorical data; not ideal for continuous data or small sample sizes.

2. What is the minimum sample size to apply each of the uniformity and independencetests?

Answer

Uniformity Test - Chi-Square Test for Goodness-of-Fit:

- Minimum sample size depends on the number of categories and expected frequencies.
- At least five expected observations per category for reliability.
- Calculate sample size based on the expected distribution.

Independence Test - Chi-Square Test for Independence:

- Minimum sample size depends on the number of categories in variables and expected frequencies in the contingency table.
- Aim for at least five expected observations per cell for reliability.
- Calculate sample size considering categories in both variables and the expected distribution.

3. Why is it essential to test the random number generator?

Answer

- **Statistical Properties:** Ensure the RNG produces numbers with statistical properties akin to true randomness.
- **Validity of Simulations:** Validate the reliability of simulations and models relying on random numbers.
- **Security Concerns:** Guarantee the randomness quality in cryptographic applications to prevent vulnerabilities.
- **Reproducibility:** Testing ensures consistent and reproducible results across different environments.
- **Identifying Biases:** Detect and rectify biases or flaws in the RNG to avoid non-random patterns.
- **Compliance with Standards:** Ensure the RNG meets industry or application-specific standards for randomness.
- **Trust in Applications:** Build user and developer trust in the reliability and randomness of the generated numbers.
- **Preventing Predictability:** Avoid predictability issues, especially in applications where unpredictability is crucial.

Outcomes:

CO2: Generate pseudorandom numbers and perform empirical tests to measure the quality of a pseudo random number generator.

Conclusion:

Learned how to run the KS test on pseudo random numbers.

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

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. "LinearCongruential Generators" by Joe Bolte, Wolfram Demonstrations Project.
2. Severance, Frank (2001). *System Modeling and Simulation*. John Wiley & Sons, Ltd. p. 86. ISBN 0-471-49694-4.
3. The GNU C library's *rand()* in stdlib.h uses a simple (single state) linear congruential generator only in case that the state is declared as 8 bytes. If the state is larger (an array), the generator becomes an additive feedback generator and the period increases. See the simplified code that reproduces the random sequence from this library.
4. "A collection of selected pseudorandom number generators with linear structures, K. Entacher, 1997". Retrieved 16 June 2012.
5. "How Visual Basic Generates Pseudo-Random Numbers for the RND Function". *Microsoft Support*. Microsoft. Retrieved 17 June 2011.
6. In spite of documentation on MSDN, RtlUniform uses LCG, and not Lehmer's algorithm, implementations before Windows Vista are flawed, because the result of multiplication is cut to 32 bits, before modulo is applied

7. GNU Scientific Library: Other random number generators
8. Novice Forth library
9. Matsumoto, Makoto, and Takuji Nishimura (1998) ACM Transactions on Modeling and Computer Simulation
10. 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.
11. 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.
12. 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.
13. Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007), "Section 7.1.1. Some History", *Numerical Recipes: The Art of Scientific Computing* (3rd ed.), New York: Cambridge University Press, ISBN 978-0-521-88068-8
14. Gentle, James E., (2003). *Random Number Generation and Monte Carlo Methods*, 2nd edition, Springer, ISBN 0-387-00178-6.
15. 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 generators).