

**Experiment No.: 5**

**Title: Implementation of Uniformity test**

# Batch: A2 Roll No.:16010421073 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.

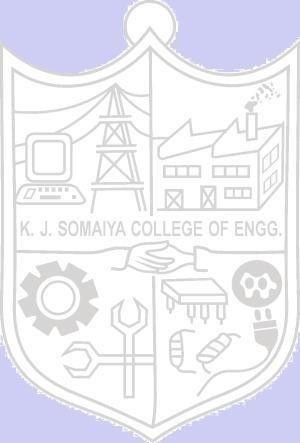
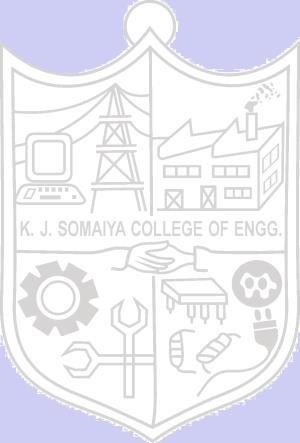
**Resources needed:** Turbo C / Java / python

# 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

* 1. Kolmogorov- Smirnov Test
  2. Chi-square Test

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

1. **Kolmogorov-Smirnov Test:** This test compares the continuous cdf F(x) of the uniform distribution to the empirical cdf SN(x) of sample of N distribution

By definition,

F(x) = x 0 ≤ x ≤ 1

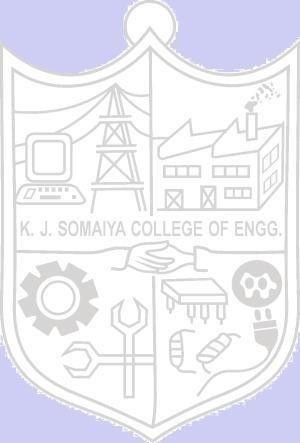
If the sample from random no. generated is R1, R2, … ,RN then the empirical cdf SN(x) is defined as

SN(x) =

No. of R1, R2, … ,RN which are x N

As N becomes larger SN(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) & SN(x) over range of random variable.

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



*n*

2 

*i i*

(*O*  *E* )2

0

*i*1

*E*

*i*

Where, Oi = Observed frequency in ith class Ei = 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)

# Code :

import numpy as np

import scipy.stats as stats

class LinearCongruentialGenerator: def init (self, seed, a, c, m):

self.seed = seed self.a = a self.c = c self.m = m

def generate\_uniform(self, n): random\_numbers = []

xn = self.seed

for \_ in range(n):

xn = (self.a \* xn + self.c) % self.m random\_numbers.append(round(xn / self.m, 2))

return random\_numbers

KJSCE/IT/TY BTECH/SEMVI/MS/2023-24

def generate\_non\_uniform(self, n): random\_numbers = []

xn = self.seed

for \_ in range(n):

xn = (self.a \* xn + self.c) % self.m # Introduce non-uniformity

if xn % 2 == 0:

random\_numbers.append(round((xn / self.m) \* 0.2, 2)) else:

random\_numbers.append(round(xn / self.m, 2)) return random\_numbers

def perform\_chi\_square\_test(data):

observed, \_ = np.histogram(data, bins=10, range=(0, 1)) expected = np.full\_like(observed, fill\_value=10)

chi2\_stat, p\_val = stats.chisquare(observed, f\_exp=expected) return observed, expected, chi2\_stat, p\_val

def display\_matrix(data):

matrix = np.array(data).reshape(10, 10) print("Random Numbers Matrix:") print(matrix)

def main():

seed = 12345

a = 1103515245

c = 12345

m = 2\*\*31

generator = LinearCongruentialGenerator(seed, a, c, m) alpha = 0.05

for i in range(5): if i == 3:

random\_set = generator.generate\_non\_uniform(100) else:

random\_set = generator.generate\_uniform(100)

print(f"Sample Set {i+1}:") display\_matrix(random\_set) # Display 10x10 matrix print("Interval\tOi\t\tEi\t\t(Oi - Ei)^2 / Ei") observed, expected, chi2\_stat, p\_val =

perform\_chi\_square\_test(random\_set)

for interval, oi, ei in zip(range(1, 11), observed, expected): oi\_ei\_squared\_over\_ei = ((oi - ei) \*\* 2) / ei print(f"{interval}\t\t{oi}\t\t{ei}\t\t{oi\_ei\_squared\_over\_ei}")

# Perform hypothesis test dof = len(observed) - 1

critical\_value = stats.chi2.ppf(1 - alpha, dof) print("\nHypothesis Test:")

print(f"Chi-square statistic: {chi2\_stat}") print(f"Critical value (alpha = {alpha}): {critical\_value}")

if chi2\_stat < critical\_value:

KJSCE/IT/TY BTECH/SEMVI/MS/2023-24

print("Null hypothesis accepted. The numbers are likely generated from a uniform distribution.")

else:

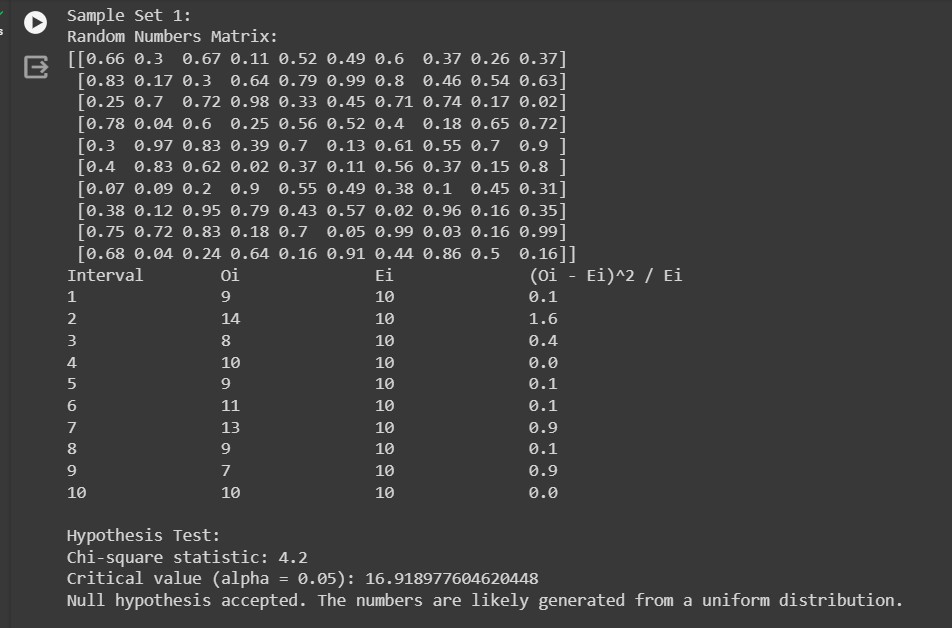
print("Null hypothesis rejected. The numbers are not generated from a uniform distribution.")

print()

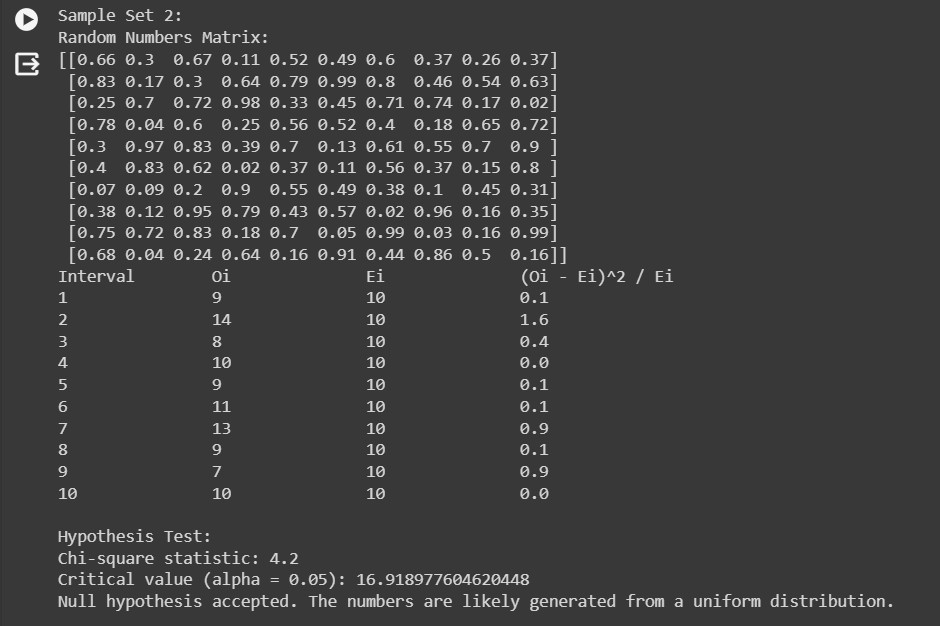
if name == " main ": main()

**Output:**

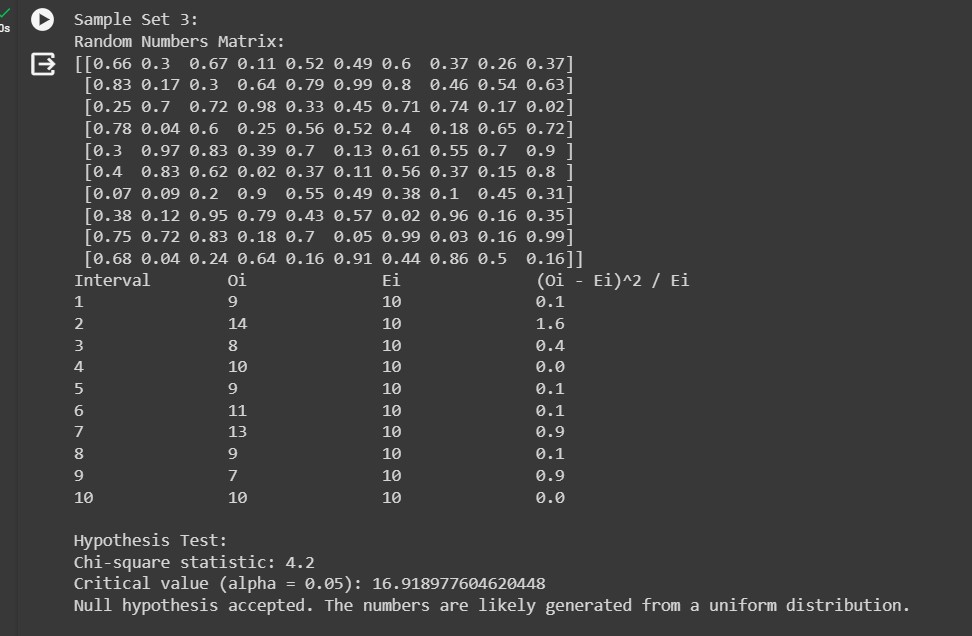
**Sample-1**



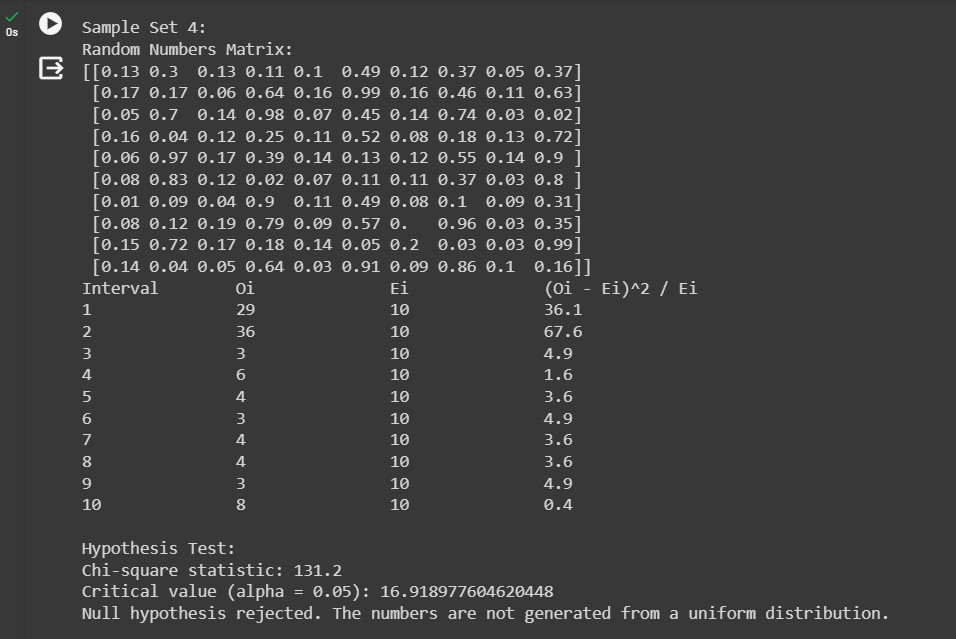
**Sample-2**



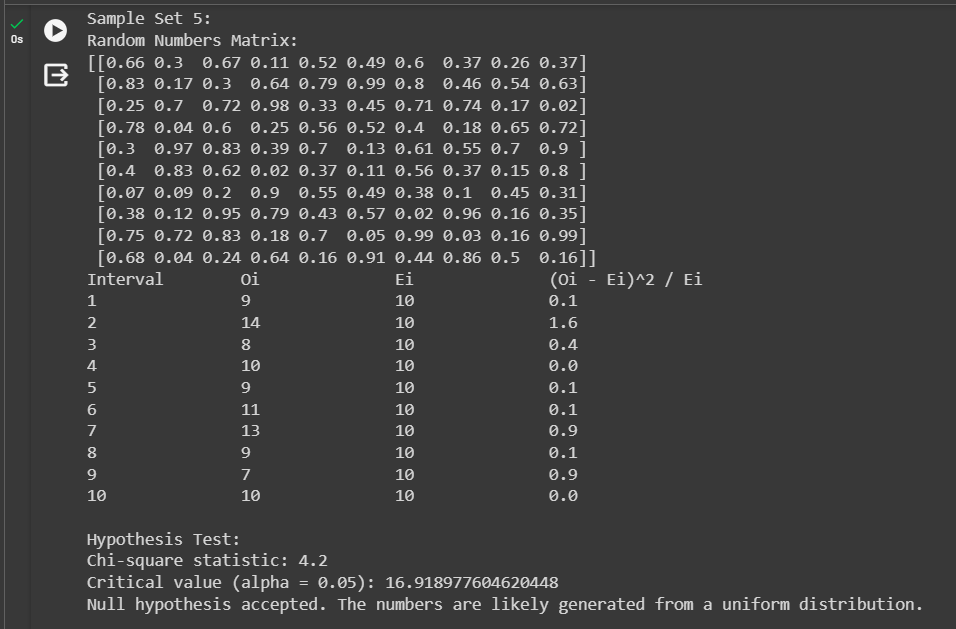
**Sample-3**

KJSCE/IT/TY BTECH/SEMVI/MS/2023-24

**Sample-4**



**Sample-5**

KJSCE/IT/TY BTECH/SEMVI/MS/2023-24

# Questions:

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

# Ans: Pros and Cons of Kolmogorov-Smirnov Test and Chi-Square Test: Kolmogorov-Smirnov Test:

*Pros:*

* + - Suitable for continuous and discrete data.
    - No assumptions about the distribution of data are needed.
    - Can be used to compare any two distributions, not just normal distributions.
    - Sensitive to differences in both location and shape of distributions.

*Cons:*

* + - * Less powerful than the chi-square test for detecting differences in the tails of distributions.
      * More sensitive to differences in the center of distributions than in the tails.
      * Requires larger sample sizes for small effect sizes compared to other tests.

# Chi-Square Test:

*Pros:*

* + - * Suitable for categorical data analysis.
      * Easy to understand and interpret.
      * Provides a single statistic and p-value to summarize the degree of discrepancy between observed and expected frequencies.
      * Powerful for detecting differences in the tails of distributions.

*Cons:*

* + - * Assumes that observed frequencies are independent.
      * Assumes that the sample size is large enough for the chi-square distribution to approximate the sampling distribution of the test statistic.
      * Less suitable for small sample sizes or when expected frequencies are low.

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

**Ans:** Minimum Sample Size for Uniformity and Independence Tests:

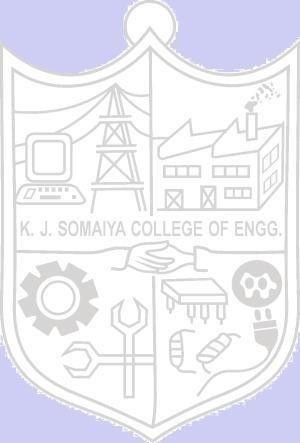
* The minimum sample size required depends on several factors including the desired level of significance, the effect size, and the specific test being used. However, generally:
* For the Kolmogorov-Smirnov test, there is no strict minimum sample size requirement, but it becomes more reliable with larger sample sizes, especially for detecting small effect sizes.
* For the Chi-Square test, a common guideline is to have expected frequencies greater than 5 in each cell of the contingency table. So, the minimum sample size depends on the number of categories and the expected frequencies.
  1. **Why is it essential to test the random number generator? Ans:** Importance of Testing Random Number Generators:

Testing random number generators (RNGs) is essential for several reasons:

* **Ensuring randomness:** RNGs are used in various applications such as simulations, cryptography, and statistical sampling. It's crucial to ensure that the numbers generated are statistically indistinguishable from true random numbers.
* **Detecting biases:** RNGs may have biases or patterns that can lead to inaccurate results or vulnerabilities in cryptographic systems.
* **Verifying compliance:** Many industries and standards require the use of validated RNGs, especially in fields such as finance, gambling, and cryptography.
* **Improving reliability:** Testing RNGs helps identify flaws and improve the quality and reliability of random number generation algorithms.
  + **Establishing trust:** By testing RNGs rigorously and transparently, users can have confidence in the randomness and integrity of the generated numbers.

**Autonomous College Affiliated to University of Mumbai**

**Outcomes:**



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

**Conclusion:** Learnt about chi square test and KS test and their implementations using Linear Congruential method.

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

# Signature of faculty in-charge with date

**References:**

# Books/ Journals/ Websites:

1. "[LinearCongruential Generators"](http://demonstrations.wolfram.com/LinearCongruentialGenerators/) by Joe Bolte[, Wolfram Demonstrations Project.](http://en.wikipedia.org/wiki/Wolfram_Demonstrations_Project)
2. Severance, Frank (2001). *System Modeling and Simulation*. John Wiley & Sons, Ltd. p. 86. [ISBN 0-471-49694-4.](http://en.wikipedia.org/wiki/International_Standard_Book_Number)
3. The GNU C library's *rand()* [in stdlib.h use](http://en.wikipedia.org/wiki/Stdlib.h)s 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 t](http://www.mscs.dal.ca/~selinger/random/)hat reproduces the random sequence from this library.
4. ["A collection of selected pseudorandom number generators with linear structures, K.](http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3686&rep=rep1&type=pdf) [Entacher, 1997".](http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3686&rep=rep1&type=pdf) Retrieved 16 June 2012.
5. ["How Visual Basic Generates Pseudo-Random Numbers for the RND Function".](http://support.microsoft.com/kb/231847)

*Microsoft Support*. Microsoft. Retrieved 17 June 2011.

1. In spite of documentation on [MSDN](http://msdn.microsoft.com/en-us/library/bb432429%28VS.85%29.aspx), RtlUniform uses LCG, and not Lehmer's algorithm, implementations before [Windows Vista a](http://en.wikipedia.org/wiki/Windows_Vista)re flawed, because the result of multiplication is cut to 32 bits, before modulo is applied
2. [GNU Scientific Library: Other random number generators](http://www.gnu.org/software/gsl/manual/html_node/Other-random-number-generators.html)
3. [Novice Forth library](http://www.forth.org/novice.html)
4. Matsumoto, Makoto, and Takuji Nishimura (1998) ACM Transactions on Modeling and Computer Simulation
5. S.K. Park and K.W. Miller (1988)[. "Random Number Generators: Good Ones Are](http://portal.acm.org/citation.cfm?id=63042) [Hard To Find". *Communications of the ACM***31**](http://portal.acm.org/citation.cfm?id=63042)(10): 1192–1201.

[doi:10.1145/63039.63042.](http://en.wikipedia.org/wiki/Digital_object_identifier)

1. [D. E. Knuth.](http://en.wikipedia.org/wiki/Donald_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.
2. P. L'Ecuyer (1999). ["Tables of Linear Congruential Generators of Different Sizes and](http://citeseer.ist.psu.edu/132363.html) [Good Lattice Structure". *Mathematics of Computation***68**](http://citeseer.ist.psu.edu/132363.html) (225): 249–260. [doi:10.1090/S0025-5718-99-00996-5.](http://en.wikipedia.org/wiki/Digital_object_identifier)



1. Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007), ["Section 7.1.1.](http://apps.nrbook.com/empanel/index.html#pg%3D343) [Some History",](http://apps.nrbook.com/empanel/index.html#pg%3D343) *Numerical Recipes: The Art of Scientific Computing* (3rd ed.), New York: Cambridge University Press[, ISBN 978-0-521-88068-8](http://en.wikipedia.org/wiki/International_Standard_Book_Number)
2. Gentle, James E., (2003). *Random Number Generation and Monte Carlo Methods*, 2nd edition, Springer, ISBN 0-387-00178-6.
3. Joan Boyar (1989). ["Inferring sequences produced by pseudo-random number](http://portal.acm.org/citation.cfm?id=59305&dl=ACM&coll=portal) [generators". *Journal of the ACM***36**](http://portal.acm.org/citation.cfm?id=59305&dl=ACM&coll=portal) (1): 129–141. [doi:10.1145/58562.59305.](http://en.wikipedia.org/wiki/Digital_object_identifier) (in this paper, efficient algorithms are given for inferring sequences produced by certain pseudo-random number generators).