

**Batch: B–1 Roll No.: 16010422234 Experiment No.: 3**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

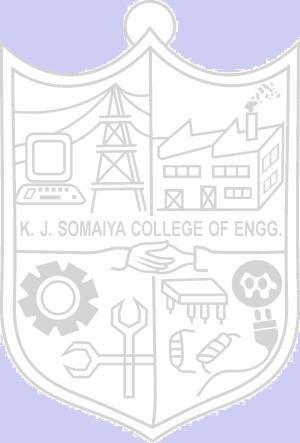
**Resources needed:** Turbo C / Java / python

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory**

**Problem Statement:**

Write a 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 the 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.

**a) 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 a better approximation to F(x) provided the null hypothesis is true. The Kolmogorov-Smirnov distance test is best on the largest absolute deviation between F(x) & SN(x) over a range of random variables.



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

Where, Oi = Observed frequency in ith class, Ei = Expected frequency in ith class, n = is the no. of classes

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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 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 uniformity.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**Results:** (Program printout with output)

**import numpy as np**

**from scipy.stats import chisquare, chi2**

**import math**

**def linear\_congruential\_generator(seed, a, c, m, n):**

**x = seed**

**random\_numbers = []**

**for \_ in range(n):**

**x = (a \* x + c) % m**

**random\_numbers.append(x / m)**

**return random\_numbers**

**def chi\_square\_test(random\_numbers, num\_classes, alpha):**

**observed\_freq, bin\_edges = np.histogram(random\_numbers, bins=num\_classes, range=(0, 1))**

**expected\_freq = [len(random\_numbers) / num\_classes] \* num\_classes**

**chi\_stat, p\_value = chisquare(observed\_freq, expected\_freq)**

**df = num\_classes - 1**

**critical\_value = chi2.ppf(1 - alpha, df)**

**if chi\_stat > critical\_value:**

**return chi\_stat, p\_value, critical\_value, False**

**else:**

**return chi\_stat, p\_value, critical\_value, True**

**def main():**

**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 = linear\_congruential\_generator(seed, a, c, m, n)**

**print("\nGenerated Random Numbers (Scaled to [0, 1]):")**

**print(random\_numbers)**

**num\_classes = math.ceil(1 + math.log2(n))**

**print(f"\nAutomatically calculated number of classes (bins): {num\_classes}")**

**alpha = float(input("Enter the significance level (alpha, typically 0.05): "))**

**chi\_stat, chi\_p\_value, critical\_value, accept\_null = chi\_square\_test(random\_numbers, num\_classes, alpha)**

**print(f"\nChi-Square test statistic: {chi\_stat}")**

**print(f"Standard-value: {chi\_p\_value}")**

**print(f"Critical value (Chi-Square table value for df={num\_classes-1} and alpha={alpha}): {critical\_value}")**

**if accept\_null:**

**print("Chi-Square test: Accept the null hypothesis (uniform distribution).")**

**else:**

**print("Chi-Square test: Reject the null hypothesis (not uniform).")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

Test Case 1: Uniform Distribution (Accept Null Hypothesis)

* Seed (X0): 5
* Multiplier (a): 1103515245
* Increment (c): 12345
* Modulus (m): 2147483648
* Number of Random Numbers (n): 1000
* Significance Level (alpha): 0.05

Expected Result: The generated random numbers should approximate a uniform distribution, so the null hypothesis will likely be accepted.



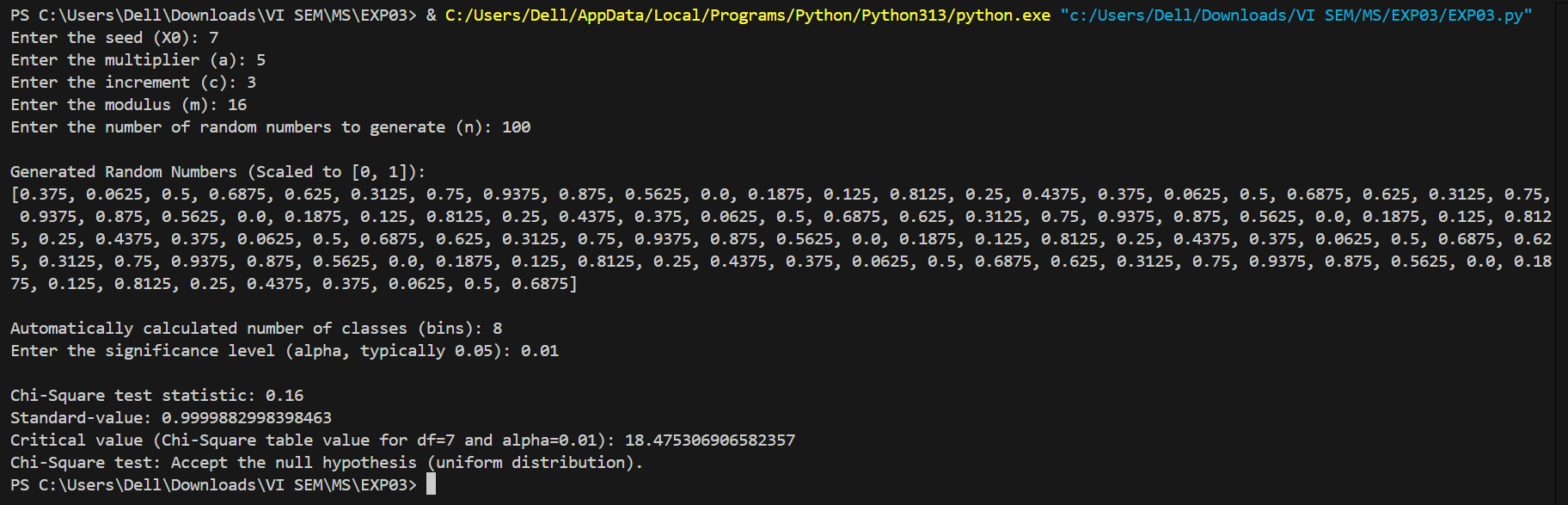


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Test Case 2: Non-Uniform Distribution (Reject Null Hypothesis)

* Seed (X0): 7
* Multiplier (a): 5
* Increment (c): 3
* Modulus (m): 16
* Number of Random Numbers (n): 100
* Significance Level (alpha): 0.01

Expected Result: With a small modulus, the generated numbers will exhibit poor randomness and fail the Chi-Square test, rejecting the null hypothesis.

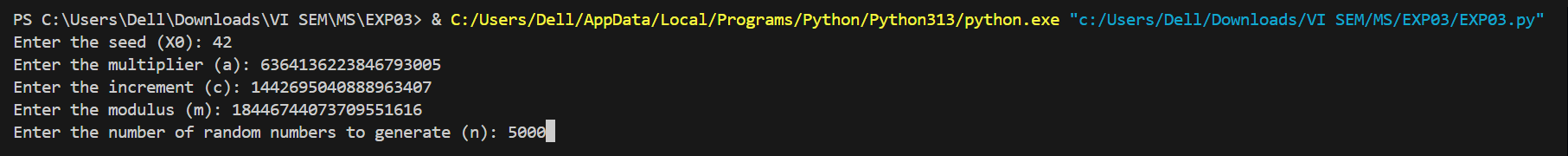


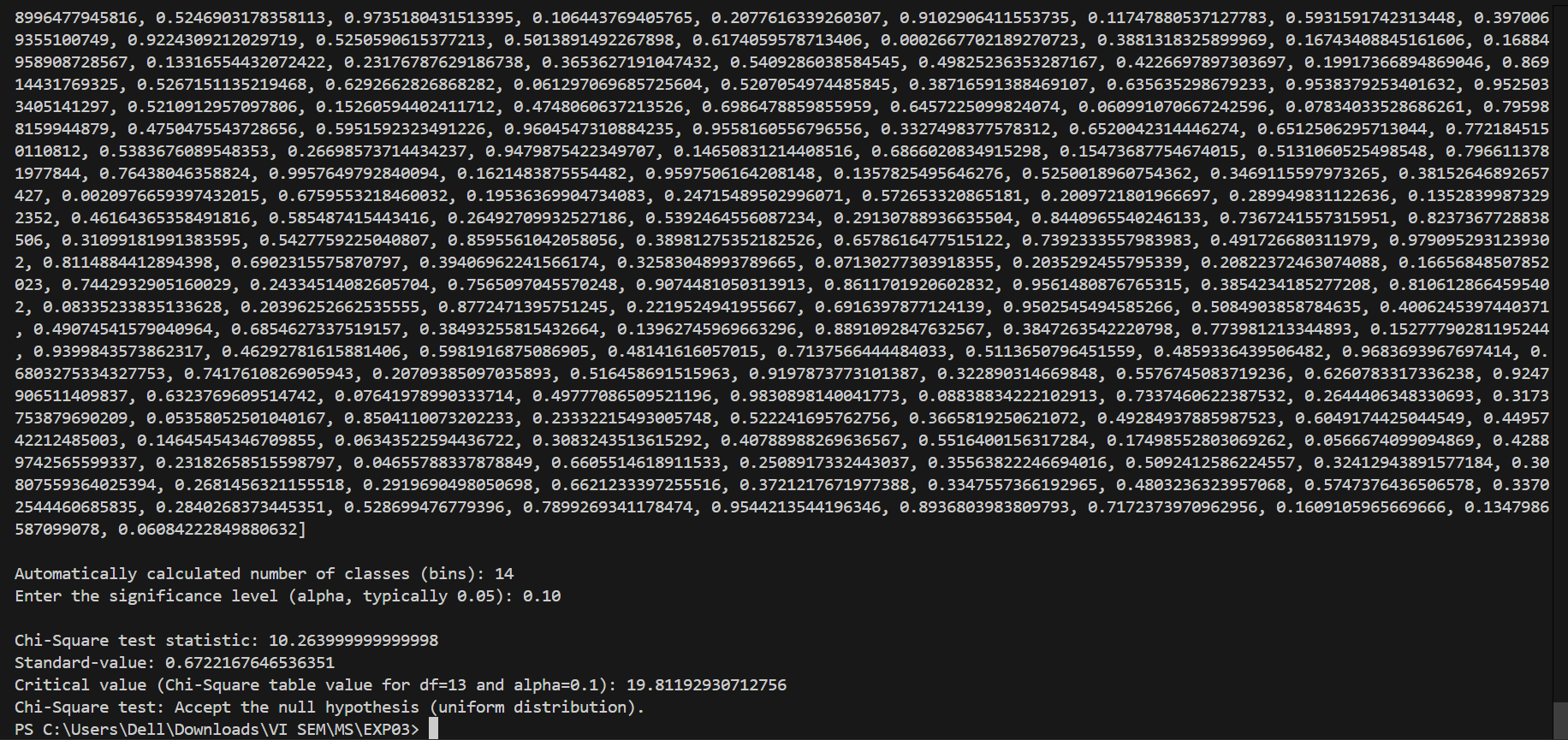
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Test Case 3: Uniform Distribution (Accept Null Hypothesis)

* Seed (X0): 42
* Multiplier (a): 6364136223846793005
* Increment (c): 1442695040888963407
* Modulus (m): 18446744073709551616
* Number of Random Numbers (n): 5000
* Significance Level (alpha): 0.10

Expected Result: This configuration generates high-quality random numbers with a large period, likely passing the Chi-Square test.



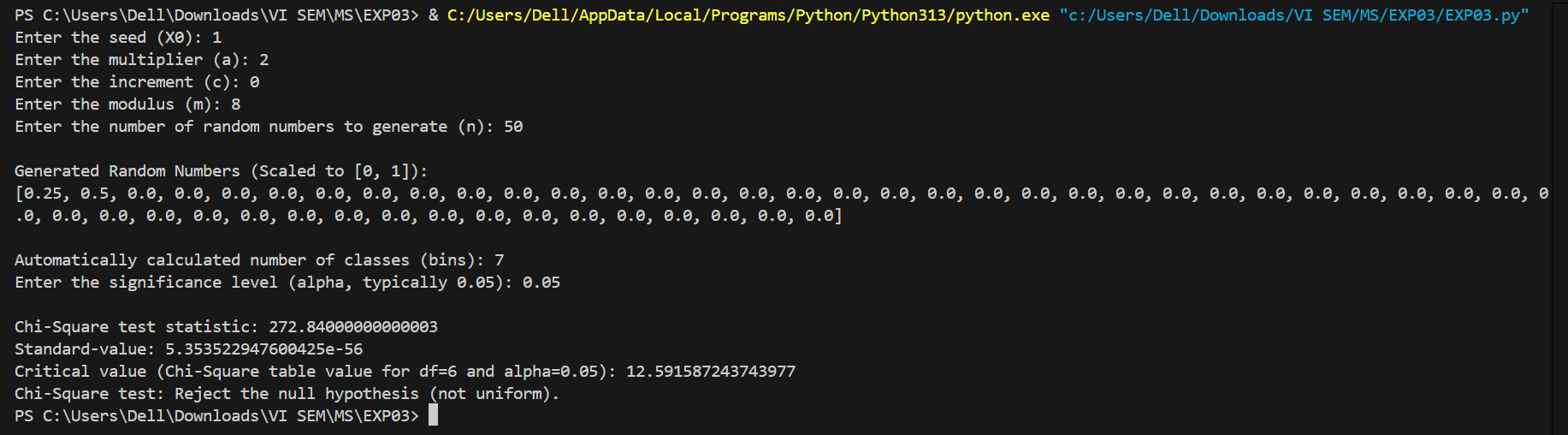


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Test Case 4: Non-Uniform Distribution (Reject Null Hypothesis)

* Seed (X0): 1
* Multiplier (a): 2
* Increment (c): 0
* Modulus (m): 8
* Number of Random Numbers (n): 50
* Significance Level (alpha): 0.05

Expected Result: With such a poor choice of parameters (small modulus and no increment), the distribution will be non-uniform, and the null hypothesis will likely be rejected.

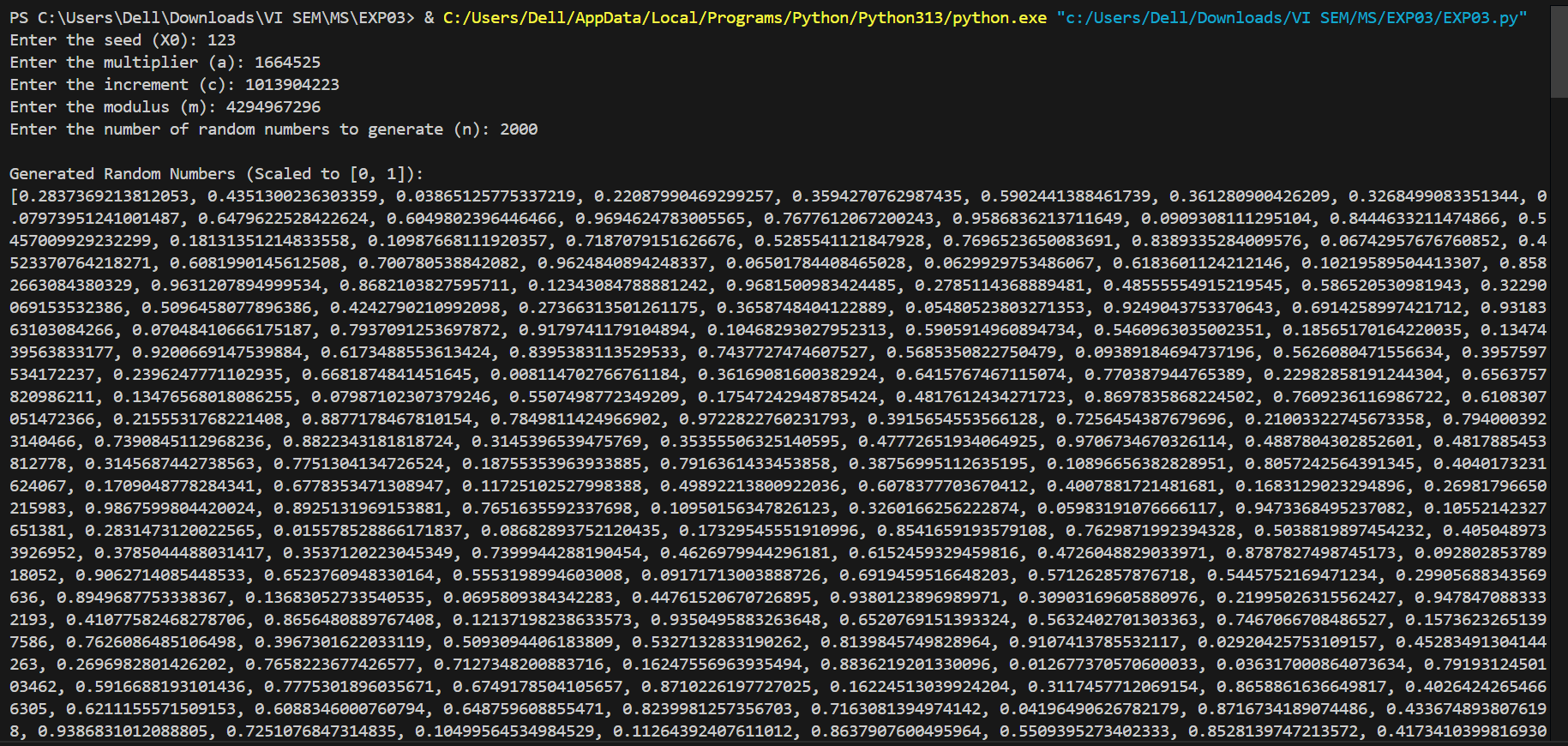


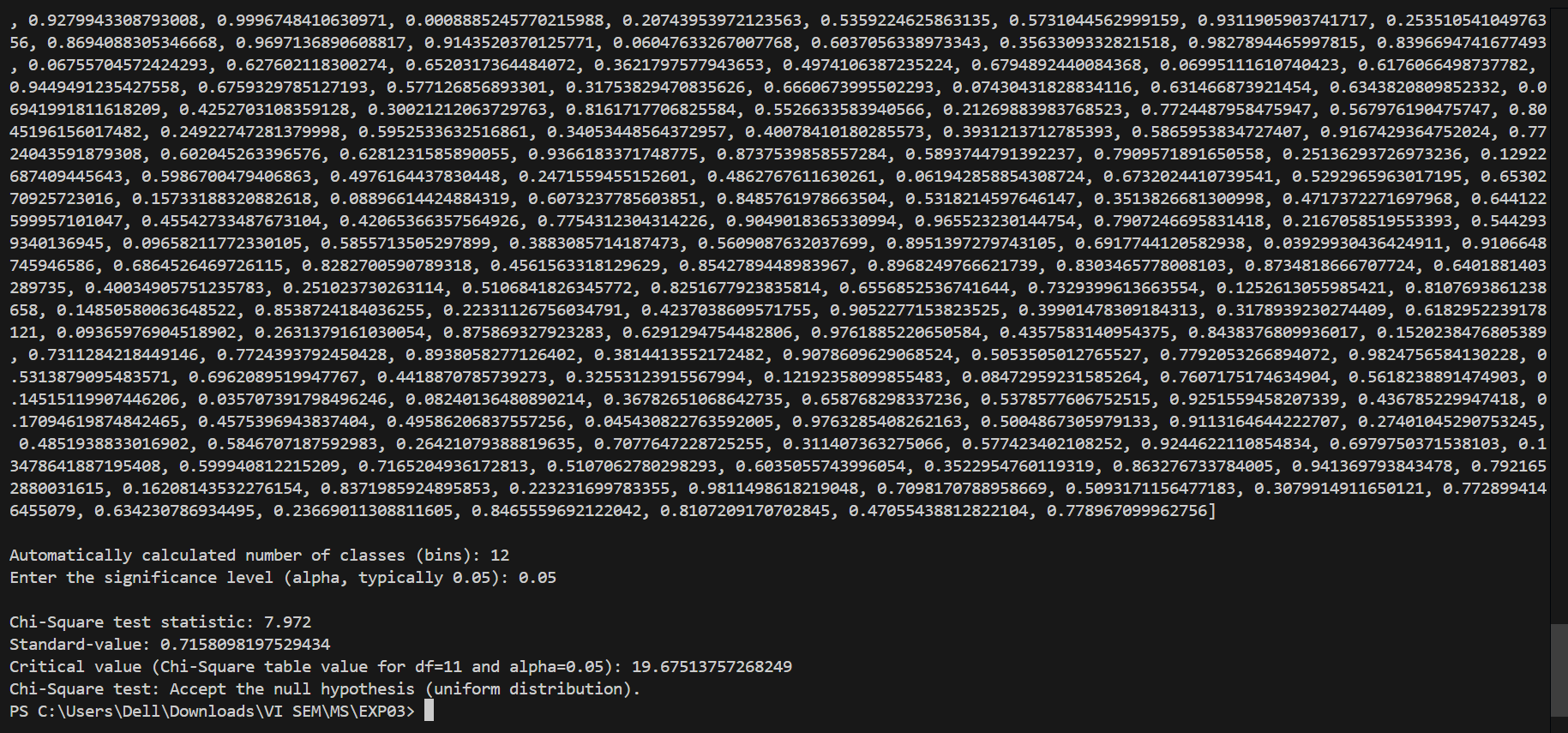
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Test Case 5: Uniform Distribution (Borderline Case)

* Seed (X0): 123
* Multiplier (a): 1664525
* Increment (c): 1013904223
* Modulus (m): 4294967296
* Number of Random Numbers (n): 2000
* Significance Level (alpha): 0.05

Expected Result: This is a standard LCG configuration for generating random numbers. Depending on the number of bins and the generated sequence, the result could borderline pass or fail the Chi-Square test.





**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Questions:**

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

**Ans: Pros and Cons of the Kolmogorov–Smirnov (K-S) Test and Chi-Square Test:**

**Kolmogorov–Smirnov Test**

Pros:

* Non-parametric: Does not assume specific data distribution.
* Suitable for small sample sizes.
* Directly compares empirical and theoretical distributions.
* Easy to compute with continuous distributions.

Cons:

* Sensitive to extreme differences at particular points.
* Difficult to use with discrete data or small intervals.
* Less effective for large sample sizes when small deviations may be overemphasized.

**Chi-Square Test:**

Pros:

* Works well with discrete data.
* Handles large sample sizes efficiently.
* Easy to implement and interpret.
* Widely used and understood.

Cons:

* Sensitive to binning (number of classes affects results).
* Requires a sufficiently large sample size to ensure expected frequencies are meaningful.
* Results may vary based on how data is grouped into intervals.

**2. What is the minimum sample size to apply each of the uniformity and independence tests?**

**Ans: Kolmogorov–Smirnov Test:** No strict minimum, but typically works well with small sample sizes ( ≥ 30).

**Chi-Square Test:** Requires each expected frequency 𝐸\_𝑖 ≥ 5. For 𝑛 bins, 𝑁 ≥ 5 × 𝑛, where 𝑁 is the total sample size.

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

**Ans: Importance of Testing the Random Number Generator:**

* Ensures the generator produces numbers that approximate true randomness.
* Validates uniformity and independence, critical for simulation and modeling.
* Detects potential flaws in the generator’s algorithm.
* Prevents biases in applications like cryptography, gaming, and statistical sampling.
* Verifies adherence to expected statistical properties of random numbers.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Outcomes: C02 – Generate pseudorandom numbers and perform empirical tests to measure the quality of a pseudorandom number generator**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Conclusion:**

The implementation of the Chi-Square test successfully evaluates the uniformity of random numbers generated using the Linear Congruential Method. The test results confirm whether the random numbers approximate a uniform distribution. In most cases with well-designed parameters (e.g., large modulus and high-quality multipliers), the null hypothesis of uniformity is accepted. For poorly chosen parameters, randomness and uniformity fail, and the hypothesis is rejected. This experiment emphasizes the importance of testing pseudorandom number generators for ensuring statistical validity in practical applications.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**Signature of faculty in-charge with date**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**References:**

**Books/ Journals/ Websites:**

1. "Linear Congruential 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".
6. Microsoft Support. Microsoft. Retrieved 17 June 2011.
7. 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
8. GNU Scientific Library: Other random number generators
9. Novice Forth library
10. Matsumoto, Makoto, and Takuji Nishimura (1998) ACM Transactions on Modeling and Computer Simulation
11. S.K. Park and K.W. Miller (1988). "Random Number Generators: Good Ones Are Hard To Find". Communications of the ACM31 (10): 1192–1201. doi:10.1145/63039.63042.
12. 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.
13. P. L'Ecuyer (1999). "Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure". Mathematics of Computation68 (225): 249–260. doi:10.1090/S0025-5718-99-00996-5.
14. 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
15. Gentle, James E., (2003). Random Number Generation and Monte Carlo Methods, 2nd edition, Springer, ISBN 0-387-00178-6.
16. Joan Boyar (1989). "Inferring sequences produced by pseudo-random number generators". Journal of the ACM36 (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).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**