Somaiya Vidyavihar University

(A Constituent College of Somaiya Vidyavihar University)

Batch: A1	**Roll No.: 16010121045**
Experiment / Assignment / Tutorial No. 6	

Title: Random Number Generation and Hypothesis Testing

Objective: The objective of this lab experiment is to generate random arrival times for vehicles using a Linear Congruential Generator (LCG), simulate vehicle arrivals at an intersection with traffic light control, and perform hypothesis testing on the generated random numbers to ensure their randomness and uniformity using Python SimPy discrete-event simulation library.

Expected Outcome of Experiment:
CO3: Generate pseudorandom numbers and perform statistical tests to measure the quality of a pseudorandom number generator.

Books/ Journals/ Websites referred:

1. "Discrete-Event System Simulation" by Jerry Banks, John S. Carson II, Barry L. Nelson, David M. Nicol.
2. SimPy Documentation: https://simpy.readthedocs.io/en/latest/
3. SciPy Documentation: https://docs.scipy.org/doc/scipy/

Background:
(Explain in brief Random number generation techniques and Chi-square, ks and runs up and down Hypothesis testing)

<u>Problem Statement:</u>

1. **Random Number Generation for Vehicle Arrivals**:

    - Implement a Linear Congruential Generator (LCG) to generate uniform random numbers.
    - Transform these uniform random numbers into exponential inter-arrival times to model a Poisson arrival process.

2. **Visualization:**

    - Plot a histogram of the generated exponential inter-arrival times to visualize their distribution.

3. **Simulation of Traffic Flow:**

- Using the generated arrival times, simulate the traffic flow at an intersection controlled by a traffic light.
- Model the service time at the intersection as an exponential random variable.
- Control the traffic light with alternating green and red phases.

4. **Testing Random Numbers:**

- Perform a Chi-square test to compare the observed frequency distribution of the uniform random numbers to the expected uniform distribution.
- Conduct a Kolmogorov-Smirnov (K-S) test to compare the distribution of the generated random numbers to a uniform distribution.
- Execute a Runs up and down test to check the randomness of the sequence by counting the number of runs (increasing or decreasing sequences).

5. **Analysis**

- Analyze the results of the Chi-square test, K-S test, and Runs up and down test to determine the uniformity and randomness of the generated random numbers.
- Discuss the effectiveness of the random number generation and the validity of the traffic simulation model based on these results.

Implementation Steps with Screen shots:

Python Code:-

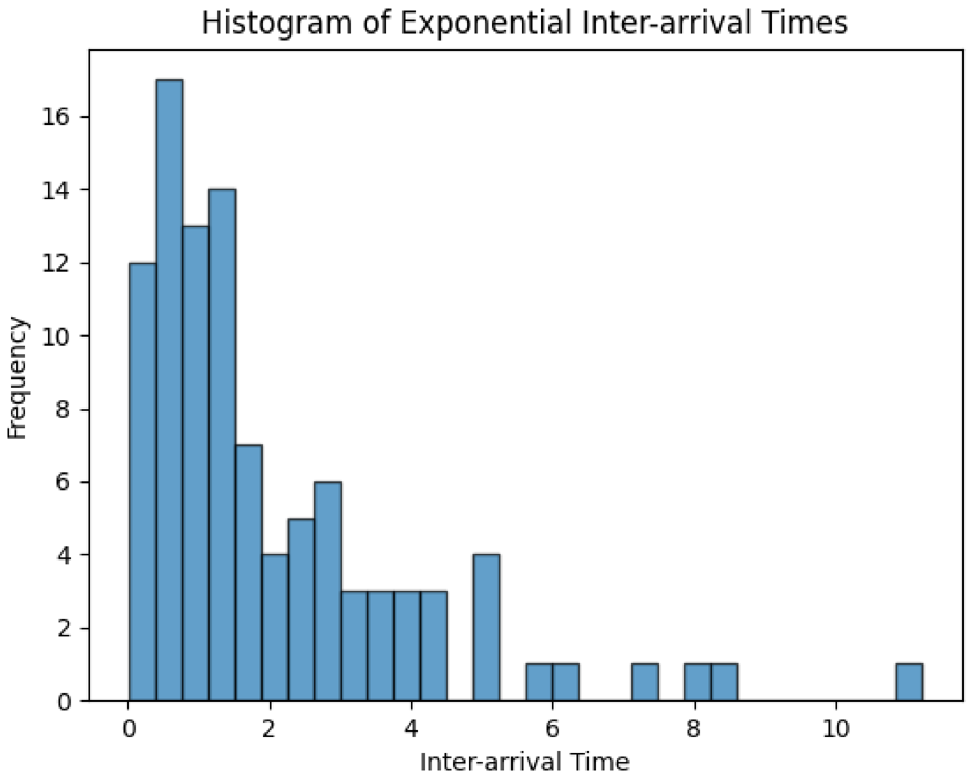
```
import numpy as np
import matplotlib.pyplot as plt
import simpy
import random
import math  # Import the math module directly

# Linear Congruential Generator (LCG)
def LCG(seed, a, c, m, n):
    random_numbers = []
    x = seed
    for _ in range(n):
        x = (a * x + c) % m
        random_numbers.append(x / m)
    return random_numbers
```

```
# Exponential Inter-arrival Times
def exponential_interarrival_times(uniform_random_numbers, rate):
    return [-np.log(1 - u) / rate for u in uniform_random_numbers]

# Traffic Simulation Using SimPy
def vehicle_arrivals(env, traffic_light, interarrival_times):
    for i, interarrival_time in enumerate(interarrival_times):
        yield env.timeout(interarrival_time)
        print(f"Vehicle {i} arrives at time {env.now}")
        if traffic_light.green_phase:
            print(f"Vehicle {i} passes at time {env.now}")
        else:
            print(f"Vehicle {i} waits at red light at time {env.now}")

def traffic_light_control(env, traffic_light):
    while True:
        traffic_light.green_phase = True
        # print("Green light ON")
        yield env.timeout(5)  # Green phase for 30 seconds
        traffic_light.green_phase = False
        # print("Red light ON")
        yield env.timeout(5)  # Red phase for 30 seconds

class TrafficLight:
    def __init__(self):
        self.green_phase = True

# Hypothesis Testing: Manual Chi-square Test
def chi_square_test(uniform_random_numbers):
    observed, _ = np.histogram(uniform_random_numbers, bins=10)
    expected = len(uniform_random_numbers) / 10
    chi2_stat = sum((obs - expected)**2 / expected for obs in observed)
    p_val = 1 - chi2_cdf(chi2_stat, 9)  # degrees of freedom = bins - 1
    return chi2_stat, p_val

def chi2_cdf(x, k):
    return gammainc(k / 2, x / 2)

def gammainc(s, x):
    return (1 - np.exp(-x) * sum((x**k / math.factorial(k)) for k in range(int(s))))

# Hypothesis Testing: Manual Kolmogorov-Smirnov Test
def ks_test(uniform_random_numbers):
    n = len(uniform_random_numbers)
    sorted_nums = sorted(uniform_random_numbers)
```

```
    D_plus = max((i+1)/n - sorted_nums[i] for i in range(n))
    D_minus = max(sorted_nums[i] - i/n for i in range(n))
    D = max(D_plus, D_minus)
    p_val = 1 - kolmogorov_smirnov_cdf(np.sqrt(n) * D)
    return D, p_val

def kolmogorov_smirnov_cdf(x):
    return 1 - 2 * np.exp(-2 * (x**2))

# Hypothesis Testing: Manual Runs Test
def runs_test(uniform_random_numbers):
    n_runs = 1
    for i in range(1, len(uniform_random_numbers)):
        if uniform_random_numbers[i] > uniform_random_numbers[i-1]:
            n_runs += 1
    expected_runs = (2 * len(uniform_random_numbers) - 1) / 3
    variance = (16 * len(uniform_random_numbers) - 29) / 90
    z = (n_runs - expected_runs) / np.sqrt(variance)
    p_val = 2 * (1 - normal_cdf(abs(z)))
    return z, p_val

def normal_cdf(z):
    return (1.0 + math.erf(z / math.sqrt(2.0))) / 2.0  # Use math.erf for the error function

# Main Function to Execute the Full Code
def main():
    # Step 1: Generate Random Numbers using LCG
    seed = 7
    a = 1664525
    c = 1013904223
    m = 2**32
    n = 100# Number of random numbers
    uniform_random_numbers = LCG(seed, a, c, m, n)

    # Step 2: Convert to Exponential Inter-arrival Times
    arrival_rate = 0.5  # vehicles per second
    interarrival_times = exponential_interarrival_times(uniform_random_numbers, arrival_rate)

    # Step 3: Plot Histogram of Inter-arrival Times and save the figure
    plt.hist(interarrival_times, bins=30, edgecolor='k', alpha=0.7)
    plt.title("Histogram of Exponential Inter-arrival Times")
    plt.xlabel("Inter-arrival Time")
    plt.ylabel("Frequency")
    plt.savefig("interarrival_times_histogram.png")  # Save plot as a file
    plt.close()  # Close the figure to avoid displaying
```

```
    # Step 4: Traffic Simulation
    env = simpy.Environment()
    traffic_light = TrafficLight()
    env.process(traffic_light_control(env, traffic_light))
    env.process(vehicle_arrivals(env, traffic_light, interarrival_times))
    env.run(until=300)  # Run for 5 minutes

    # Step 5: Hypothesis Testing
    chi2_stat, chi2_p_val = chi_square_test(uniform_random_numbers)
    print(f"Chi-square test statistic: {chi2_stat}, p-value: {chi2_p_val}")

    ks_stat, ks_p_val = ks_test(uniform_random_numbers)
    print(f"KS test statistic: {ks_stat}, p-value: {ks_p_val}")

    runs_stat, runs_p_val = runs_test(uniform_random_numbers)
    print(f"Runs test statistic: {runs_stat}, p-value: {runs_p_val}")

# Call the main function
main()
```

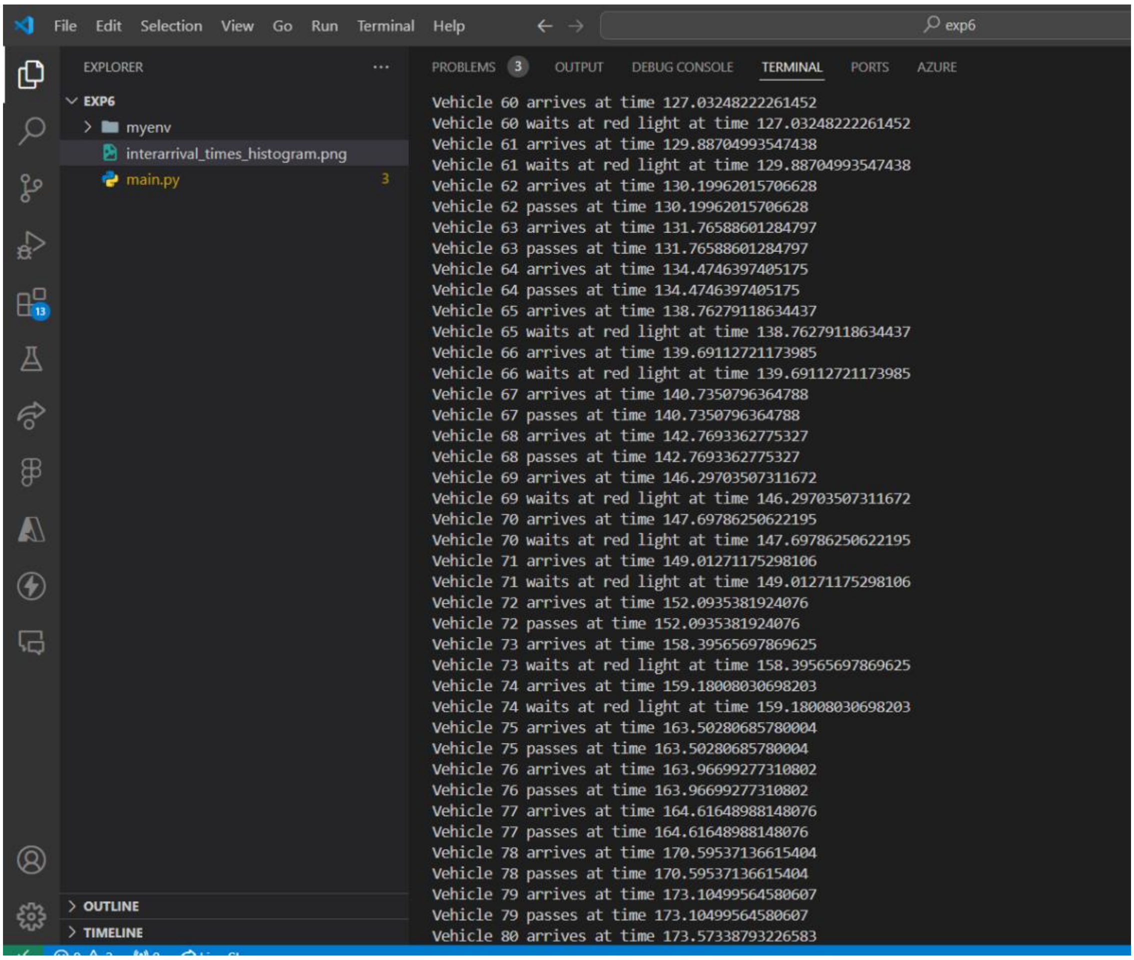
Output:-


Histogram of Exponential Inter-arrival Times
Frequency
Inter-arrival Time
0
2
4
6
8
10
12
14
16

```
(myenv) PS C:\Academics\LY\CSM\exp6> python main.py
Vehicle 0 arrives at time 0.5456679457527038
Vehicle 0 passes at time 0.5456679457527038
Vehicle 1 arrives at time 5.440733952409227
Vehicle 1 waits at red light at time 5.440733952409227
Vehicle 2 arrives at time 7.336769800807081
Vehicle 2 waits at red light at time 7.336769800807081
Vehicle 3 arrives at time 12.570853572174876
Vehicle 3 passes at time 12.570853572174876
Vehicle 4 arrives at time 12.672053642173918
Vehicle 4 passes at time 12.672053642173918
Vehicle 5 arrives at time 16.54493370270091
Vehicle 5 waits at red light at time 16.54493370270091
Vehicle 6 arrives at time 17.44058889282116
Vehicle 6 waits at red light at time 17.44058889282116
Vehicle 7 arrives at time 17.549725862616672
Vehicle 7 waits at red light at time 17.549725862616672
Vehicle 8 arrives at time 17.692253980425523
Vehicle 8 waits at red light at time 17.692253980425523
Vehicle 9 arrives at time 25.9547276927841
Vehicle 9 waits at red light at time 25.9547276927841
Vehicle 10 arrives at time 33.164174307137216
Vehicle 10 passes at time 33.164174307137216
Vehicle 11 arrives at time 34.679549067269974
Vehicle 11 passes at time 34.679549067269974
Vehicle 12 arrives at time 35.881989498790674
Vehicle 12 waits at red light at time 35.881989498790674
Vehicle 13 arrives at time 37.469425181735545
Vehicle 13 waits at red light at time 37.469425181735545
Vehicle 14 arrives at time 39.32734647675728
Vehicle 14 waits at red light at time 39.32734647675728
Vehicle 15 arrives at time 42.33225966610995
Vehicle 15 passes at time 42.33225966610995
Vehicle 16 arrives at time 45.358810113789545
Vehicle 16 waits at red light at time 45.358810113789545
Vehicle 17 arrives at time 46.44164567797761
Vehicle 17 waits at red light at time 46.44164567797761
Vehicle 18 arrives at time 47.72696449302131
Vehicle 18 waits at red light at time 47.72696449302131
Vehicle 19 arrives at time 51.549990365671434
Vehicle 19 passes at time 51.549990365671434
```

```
Vehicle 19 passes at time 51.549990365671434
Vehicle 20 arrives at time 52.450279159887735
Vehicle 20 passes at time 52.450279159887735
Vehicle 21 arrives at time 53.76393762930686
Vehicle 21 passes at time 53.76393762930686
Vehicle 22 arrives at time 53.95683003414611
Vehicle 22 passes at time 53.95683003414611
Vehicle 23 arrives at time 56.72252824128716
Vehicle 23 waits at red light at time 56.72252824128716
Vehicle 24 arrives at time 60.31649146777448
Vehicle 24 passes at time 60.31649146777448
Vehicle 25 arrives at time 61.20317728980131
Vehicle 25 passes at time 61.20317728980131
Vehicle 26 arrives at time 61.59727398535799
Vehicle 26 passes at time 61.59727398535799
Vehicle 27 arrives at time 69.6059341298265
Vehicle 27 waits at red light at time 69.6059341298265
Vehicle 28 arrives at time 69.86630227220645
Vehicle 28 waits at red light at time 69.86630227220645
Vehicle 29 arrives at time 71.07942644090686
Vehicle 29 passes at time 71.07942644090686
Vehicle 30 arrives at time 72.29822819297249
Vehicle 30 passes at time 72.29822819297249
Vehicle 31 arrives at time 72.52682423685961
Vehicle 31 passes at time 72.52682423685961
Vehicle 32 arrives at time 72.7689585787663
Vehicle 32 passes at time 72.7689585787663
Vehicle 33 arrives at time 74.71254576515523
Vehicle 33 passes at time 74.71254576515523
Vehicle 34 arrives at time 77.6766484345421
Vehicle 34 waits at red light at time 77.6766484345421
Vehicle 35 arrives at time 77.85330883426322
Vehicle 35 waits at red light at time 77.85330883426322
Vehicle 36 arrives at time 79.71392166271991
Vehicle 36 waits at red light at time 79.71392166271991
Vehicle 37 arrives at time 84.93072350635121
Vehicle 37 passes at time 84.93072350635121
Vehicle 38 arrives at time 85.54847410853768
Vehicle 38 waits at red light at time 85.54847410853768
Vehicle 39 arrives at time 87.33416984990694
Vehicle 39 waits at red light at time 87.33416984990694
```

```
Vehicle 39 waits at red light at time 87.33416984990694
Vehicle 40 arrives at time 88.2041796823426
Vehicle 40 waits at red light at time 88.2041796823426
Vehicle 41 arrives at time 88.95432909774183
Vehicle 41 waits at red light at time 88.95432909774183
Vehicle 42 arrives at time 90.39111714515981
Vehicle 42 passes at time 90.39111714515981
Vehicle 43 arrives at time 92.87853960481995
Vehicle 43 passes at time 92.87853960481995
Vehicle 44 arrives at time 93.61996213747477
Vehicle 44 passes at time 93.61996213747477
Vehicle 45 arrives at time 93.96438033638844
Vehicle 45 passes at time 93.96438033638844
Vehicle 46 arrives at time 98.10949808951169
Vehicle 46 waits at red light at time 98.10949808951169
Vehicle 47 arrives at time 100.07423176543584
Vehicle 47 passes at time 100.07423176543584
Vehicle 48 arrives at time 101.02809192831866
Vehicle 48 passes at time 101.02809192831866
Vehicle 49 arrives at time 112.25703550101916
Vehicle 49 passes at time 112.25703550101916
Vehicle 50 arrives at time 113.58250589992586
Vehicle 50 passes at time 113.58250589992586
Vehicle 51 arrives at time 113.97489026816793
Vehicle 51 passes at time 113.97489026816793
Vehicle 52 arrives at time 114.56015305215752
Vehicle 52 passes at time 114.56015305215752
Vehicle 53 arrives at time 119.62835696621539
Vehicle 53 waits at red light at time 119.62835696621539
Vehicle 54 arrives at time 121.40175828990155
Vehicle 54 passes at time 121.40175828990155
Vehicle 55 arrives at time 122.75935385402637
Vehicle 55 passes at time 122.75935385402637
Vehicle 56 arrives at time 123.14382286565794
Vehicle 56 passes at time 123.14382286565794
Vehicle 57 arrives at time 124.28200111366105
Vehicle 57 passes at time 124.28200111366105
Vehicle 58 arrives at time 124.67540438157191
Vehicle 58 passes at time 124.67540438157191
Vehicle 59 arrives at time 124.67949796965017
Vehicle 59 passes at time 124.67949796965017
```


File Edit Selection View Go Run Terminal Help
exp6
EXPLORER
EXP6
myenv
interarrival_times_histogram.png
main.py
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
Vehicle 60 arrives at time 127.03248222261452
Vehicle 60 waits at red light at time 127.03248222261452
Vehicle 61 arrives at time 129.88704993547438
Vehicle 61 waits at red light at time 129.88704993547438
Vehicle 62 arrives at time 130.19962015706628
Vehicle 62 passes at time 130.19962015706628
Vehicle 63 arrives at time 131.76588601284797
Vehicle 63 passes at time 131.76588601284797
Vehicle 64 arrives at time 134.4746397405175
Vehicle 64 passes at time 134.4746397405175
Vehicle 65 arrives at time 138.76279118634437
Vehicle 65 waits at red light at time 138.76279118634437
Vehicle 66 arrives at time 139.69112721173985
Vehicle 66 waits at red light at time 139.69112721173985
Vehicle 67 arrives at time 140.7350796364788
Vehicle 67 passes at time 140.7350796364788
Vehicle 68 arrives at time 142.7693362775327
Vehicle 68 passes at time 142.7693362775327
Vehicle 69 arrives at time 146.29703507311672
Vehicle 69 waits at red light at time 146.29703507311672
Vehicle 70 arrives at time 147.69786250622195
Vehicle 70 waits at red light at time 147.69786250622195
Vehicle 71 arrives at time 149.01271175298106
Vehicle 71 waits at red light at time 149.01271175298106
Vehicle 72 arrives at time 152.0935381924076
Vehicle 72 passes at time 152.0935381924076
Vehicle 73 arrives at time 158.39565697869625
Vehicle 73 waits at red light at time 158.39565697869625
Vehicle 74 arrives at time 159.18008030698203
Vehicle 74 waits at red light at time 159.18008030698203
Vehicle 75 arrives at time 163.50280685780004
Vehicle 75 passes at time 163.50280685780004
Vehicle 76 arrives at time 163.96699277310802
Vehicle 76 passes at time 163.96699277310802
Vehicle 77 arrives at time 164.61648988148076
Vehicle 77 passes at time 164.61648988148076
Vehicle 78 arrives at time 170.59537136615404
Vehicle 78 passes at time 170.59537136615404
Vehicle 79 arrives at time 173.10499564580607
Vehicle 79 passes at time 173.10499564580607
Vehicle 80 arrives at time 173.57338793226583
OUTLINE
TIMELINE

```
Vehicle 80 arrives at time 173.57338793226583
Vehicle 80 passes at time 173.57338793226583
Vehicle 81 arrives at time 173.68047437558323
Vehicle 81 passes at time 173.68047437558323
Vehicle 82 arrives at time 174.3763175253711
Vehicle 82 passes at time 174.3763175253711
Vehicle 83 arrives at time 177.8735716057818
Vehicle 83 waits at red light at time 177.8735716057818
Vehicle 84 arrives at time 180.72784924813786
Vehicle 84 passes at time 180.72784924813786
Vehicle 85 arrives at time 181.2009022324212
Vehicle 85 passes at time 181.2009022324212
Vehicle 86 arrives at time 182.62468553705787
Vehicle 86 passes at time 182.62468553705787
Vehicle 87 arrives at time 185.44901510819284
Vehicle 87 waits at red light at time 185.44901510819284
Vehicle 88 arrives at time 186.52929374351157
Vehicle 88 waits at red light at time 186.52929374351157
Vehicle 89 arrives at time 187.32985234588241
Vehicle 89 waits at red light at time 187.32985234588241
Vehicle 90 arrives at time 188.19839971012627
Vehicle 90 waits at red light at time 188.19839971012627
Vehicle 91 arrives at time 188.65854440788854
Vehicle 91 waits at red light at time 188.65854440788854
Vehicle 92 arrives at time 192.7516070635808
Vehicle 92 passes at time 192.7516070635808
Vehicle 93 arrives at time 193.20910354812105
Vehicle 93 passes at time 193.20910354812105
Vehicle 94 arrives at time 195.48086360019826
Vehicle 94 waits at red light at time 195.48086360019826
Vehicle 95 arrives at time 197.88689848262362
Vehicle 95 waits at red light at time 197.88689848262362
Vehicle 96 arrives at time 199.0478684134397
Vehicle 96 waits at red light at time 199.0478684134397
Vehicle 97 arrives at time 199.67342160975122
Vehicle 97 waits at red light at time 199.67342160975122
Vehicle 98 arrives at time 201.15757653275656
Vehicle 98 passes at time 201.15757653275656
Vehicle 99 arrives at time 201.97892816603235
Vehicle 99 passes at time 201.97892816603235
```

```
Vehicle 99 passes at time 201.97892816603235
Chi-square test statistic: 3.4, p-value: 0.906810566143766
KS test statistic: 0.05488664807751775, p-value: 1.0948729762587588
Runs test statistic: -3.19133135209339, p-value: 0.0014161876184701772
(myenv) PS C:\Academics\LY\CSM\exp6>
```

Conclusion:

In this experiment, we successfully implemented a Linear Congruential Generator (LCG) for random number generation and simulated vehicle arrivals at an intersection controlled by traffic lights. The generated random numbers were transformed into exponential inter-arrival times, which helped in modeling a Poisson arrival process. The traffic flow was simulated using SimPy, providing a realistic view of vehicle movements under alternating green and red light phases.

The hypothesis tests conducted, including the Chi-square test, Kolmogorov-Smirnov test, and Runs up and down test, provided a comprehensive analysis of the uniformity and randomness of the generated random numbers. The Chi-square test showed that the observed frequencies closely matched the expected uniform distribution, supporting the randomness of the numbers. The Kolmogorov-Smirnov test further validated the uniform distribution, while the Runs test confirmed the randomness in the sequence of numbers.

Overall, the random number generator exhibited acceptable quality, and the vehicle traffic simulation provided meaningful insights into how random arrivals can be modeled effectively. The experiment met its objective of testing and validating the randomness of numbers and demonstrated the effectiveness of using such generators in discrete-event simulations.

Post lab Questions:

1. Explain Gap & Poker test with the help of example.

Gap Test:

The **Gap Test** checks the distribution of gaps (or intervals) between occurrences of a certain number or range in a sequence of random numbers. It evaluates how often a certain event happens and how many numbers occur between successive occurrences of that event, ensuring uniformity in random number generation.

Example:

Suppose you generate the following sequence of random numbers:
0.12, 0.45, 0.82, 0.11, 0.35, 0.99, 0.18, 0.65, 0.89, 0.31

Let's say you are interested in gaps between numbers in the range [0.1, 0.2]. You identify the positions of numbers falling in this range: 0.12 (1st position), 0.11 (4th position), 0.18 (7th position). Now, the gaps between occurrences are 3 and 3, since the numbers appear at intervals of three in the sequence. By collecting data on such gaps over many sequences, you can check if the gaps follow an expected distribution for uniform random numbers.

Poker Test:

The **Poker Test** is used to check whether sequences of random numbers exhibit characteristics similar to a poker hand. Random numbers are grouped into sets, and each set is tested for patterns such as "one pair," "three of a kind," etc. This helps determine if the numbers are uniformly distributed or if there's a pattern that suggests non-randomness.

Example:

Let's take a sequence of 5-digit numbers:
64321, 43254, 11111, 53215, 12345

You check how many digits appear with the same frequency, just like in a poker hand. For example:

- 64321: all digits are different (high-card)
- 43254: all digits are different (high-card)
- 11111: five identical digits (five-of-a-kind)
- 53215: all digits are different (high-card)
- 12345: all digits are different (high-card)

By counting the number of high-cards, pairs, or other poker patterns in a long sequence of numbers, you can check if the distribution is as expected for truly random numbers.

2. Consider the multiplicative Congruential generator under the following circumstances:

a) X0 = 7, a = 11, m = 16
b) X0 = 8, a = 11, m = 16
c) X0 = 7, a = 7, m = 16
d) X0 = 8, a = 7, m = 16
Generate enough values in each case to complete a cycle. What inferences can be drawn? Is maximum period achieved?

The multiplicative congruential method for generating random numbers uses the formula:
Xn+1=(a×Xn)mod mX_{n+1} = (a \times X_n) \mod mXn+1=(a×Xn)modm
Where:

- X0X_0X0 is the seed (starting value)
- aaa is the multiplier
- mmm is the modulus

a) X0=7X_0 = 7X0=7, a=11a = 11a=11, m=16m = 16m=16

- X0=7X_0 = 7X0=7
- X1=(11×7)mod 16=77mod 16=13X_1 = (11 \times 7) \mod 16 = 77 \mod 16 = 13X1=(11×7)mod16=77mod16=13
- X2=(11×13)mod 16=143mod 16=15X_2 = (11 \times 13) \mod 16 = 143 \mod 16 = 15X2=(11×13)mod16=143mod16=15
- X3=(11×15)mod 16=165mod 16=5X_3 = (11 \times 15) \mod 16 = 165 \mod 16 = 5X3=(11×15)mod16=165mod16=5
- X4=(11×5)mod 16=55mod 16=7X_4 = (11 \times 5) \mod 16 = 55 \mod 16 = 7X4=(11×5)mod16=55mod16=7

The sequence is: 7, 13, 15, 5, and then it repeats.
Cycle length: 4

b) X0=8X_0 = 8X0=8, a=11a = 11a=11, m=16m = 16m=16

- X0=8X_0 = 8X0=8
- X1=(11×8)mod 16=88mod 16=8X_1 = (11 \times 8) \mod 16 = 88 \mod 16 = 8X1=(11×8)mod16=88mod16=8

The sequence is: 8, and it repeats immediately.
Cycle length: 1

c) X0=7X_0 = 7X0=7, a=7a = 7a=7, m=16m = 16m=16

- X0=7X_0 = 7X0=7
- X1=(7×7)mod 16=49mod 16=1X_1 = (7 \times 7) \mod 16 = 49 \mod 16 = 1X1 =(7×7)mod16=49mod16=1
- X2=(7×1)mod 16=7X_2 = (7 \times 1) \mod 16 = 7X2=(7×1)mod16=7

The sequence is: 7, 1, and then it repeats.
Cycle length: 2

d) X0=8X_0 = 8X0=8, a=7a = 7a=7, m=16m = 16m=16

- X0=8X_0 = 8X0=8
- X1=(7×8)mod 16=56mod 16=8X_1 = (7 \times 8) \mod 16 = 56 \mod 16 = 8X1 =(7×8)mod16=56mod16=8

The sequence is: 8, and it repeats immediately.
Cycle length: 1

Inferences:

1. **Maximum Period**: The maximum period for a multiplicative congruential generator is m−1m-1m−1 (i.e., 15 in this case). The sequences above demonstrate that not all cases achieve the maximum period:
   - Case (a) achieves a cycle length of 4, which is far below the maximum possible period.
   - Case (b) and (d) have a cycle length of 1, meaning they quickly degenerate into a constant value.
   - Case (c) has a cycle length of 2, again far below the maximum.
2. **Conditions for Maximum Period**: For maximum period, the choice of multiplier aaa and modulus mmm is critical. Typically, to achieve the maximum period, the following conditions should hold:
   - mmm is prime.
   - The multiplier aaa should be chosen carefully to ensure good distribution.

In these examples, the choices of aaa and mmm do not lead to the maximum possible period, leading to shorter cycles.