Computerized Simulation
Exercise No. 2

name : Seyed Mohammad Ghoreishy
teacher : Seyed Amirhossein Tabatabaei

Date: 1403.08.29

Contents

1 Exercise 1

Under which condition, the LCG and Multiplicative Congruential methods achieve their maximum period?

Solution: To achieve the maximum period in the Linear Congruential Generator (LCG) method, the following conditions must be met: 1. The multiplier a and the modulus m should be coprime. 2. The increment c must be coprime with m. 3. The number $a-1$ should be a multiple of all prime factors of m. 4. If m is divisible by 4, then $a-1$ must also be a multiple of 4.

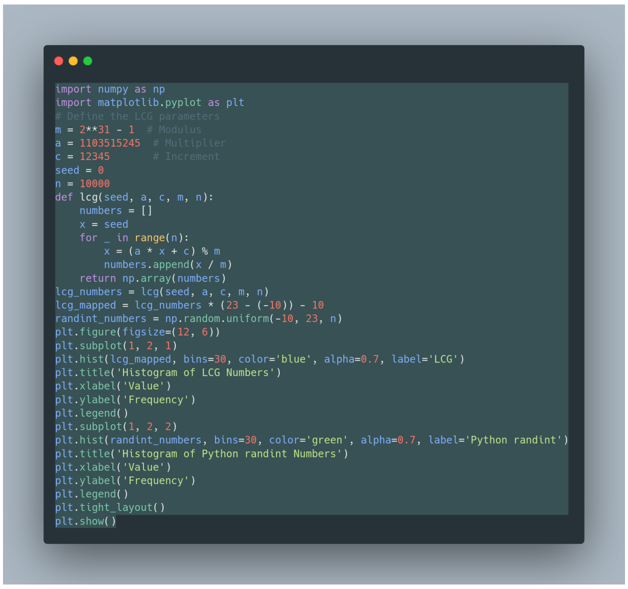
These conditions ensure that the LCG has the longest possible period before repeating values.

2 Exercise 2

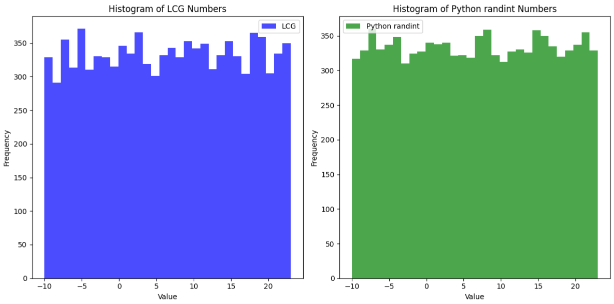
Describe a physical process for generating a sequence of random numbers with 2-digit accuracy.

Solution: One approach to generate a sequence of random numbers using a physical process is by leveraging thermal noise in an electronic circuit. Thermal noise, caused by the random motion of electrons, is inherently random and can be digitized to produce random numbers. By carefully sampling this noise and rounding it to 2-digit precision, we can achieve a sequence of random numbers with the desired accuracy.

3 Exercise 3

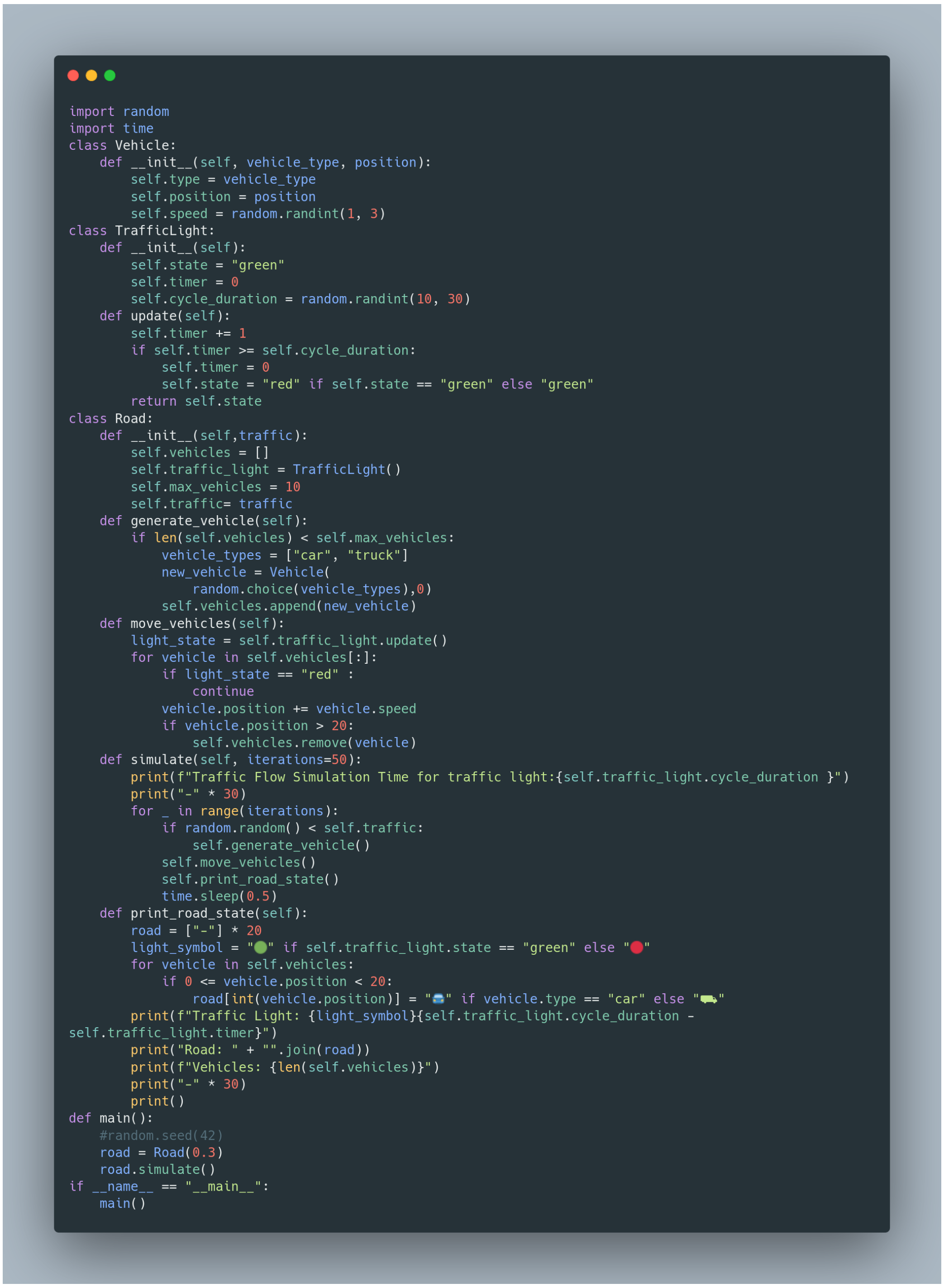


```python
import numpy as np
import matplotlib.pyplot as plt
# Define the LCG parameters
m = 2**31 - 1  # Modulus
a = 1103515245  # Multiplier
c = 12345       # Increment
seed = 0
n = 10000
def lcg(seed, a, c, m, n):
    numbers = []
    x = seed
    for _ in range(n):
        x = (a * x + c) % m
        numbers.append(x / m)
    return np.array(numbers)
lcg_numbers = lcg(seed, a, c, m, n)
lcg_mapped = lcg_numbers * (23 - (-10)) - 10
randint_numbers = np.random.uniform(-10, 23, n)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(lcg_mapped, bins=30, color='blue', alpha=0.7, label='LCG')
plt.title('Histogram of LCG Numbers')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.subplot(1, 2, 2)
plt.hist(randint_numbers, bins=30, color='green', alpha=0.7, label='Python randint')
plt.title('Histogram of Python randint Numbers')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.tight_layout()
plt.show()
```



4 Exercise 4

just run 4.py and enjoy the output.



```
import random
import time
class Vehicle:
    def __init__(self, vehicle_type, position):
        self.type = vehicle_type
        self.position = position
        self.speed = random.randint(1, 3)
class TrafficLight:
    def __init__(self):
        self.state = "green"
        self.timer = 0
        self.cycle_duration = random.randint(10, 30)
    def update(self):
        self.timer += 1
        if self.timer >= self.cycle_duration:
            self.timer = 0
            self.state = "red" if self.state == "green" else "green"
        return self.state
class Road:
    def __init__(self,traffic):
        self.vehicles = []
        self.traffic_light = TrafficLight()
        self.max_vehicles = 10
        self.traffic= traffic
    def generate_vehicle(self):
        if len(self.vehicles) < self.max_vehicles:
            vehicle_types = ["car", "truck"]
            new_vehicle = Vehicle(
                random.choice(vehicle_types),0)
            self.vehicles.append(new_vehicle)
    def move_vehicles(self):
        light_state = self.traffic_light.update()
        for vehicle in self.vehicles[:]:
            if light_state == "red" :
                continue
            vehicle.position += vehicle.speed
            if vehicle.position > 20:
                self.vehicles.remove(vehicle)
    def simulate(self, iterations=50):
        print(f"Traffic Flow Simulation Time for traffic light:{self.traffic_light.cycle_duration }")
        print("-" * 30)
        for _ in range(iterations):
            if random.random() < self.traffic:
                self.generate_vehicle()
            self.move_vehicles()
            self.print_road_state()
            time.sleep(0.5)
    def print_road_state(self):
        road = ["-"] * 20
        light_symbol = "🟢" if self.traffic_light.state == "green" else "🔴"
        for vehicle in self.vehicles:
            if 0 <= vehicle.position < 20:
                road[int(vehicle.position)] = "🚘" if vehicle.type == "car" else "🚚"
        print(f"Traffic Light: {light_symbol}{self.traffic_light.cycle_duration -
self.traffic_light.timer}")
        print("Road: " + "".join(road))
        print(f"Vehicles: {len(self.vehicles)}")
        print("-" * 30)
        print()
def main():
    #random.seed(42)
    road = Road(0.3)
    road.simulate()
if __name__ == "__main__":
    main()
```