Report on Lehmer Pseudo-Random Number Generator and Randomness Testing

The task is to create a custom pseudo-random number generator using the Lehmer algorithm and then validate its randomness. The Lehmer generator follows a simple recurrence formula and is widely used because of its efficiency and relatively long period. However, the quality of randomness can vary, so statistical tests (like the chi-squared test) are necessary to check if the numbers produced are close to a truly random sequence.

Specifically, we need:

1. **A Pseudo-Random Generator**: Implemented using the Lehmer algorithm with chosen constants.
2. **Randomness Testing**: We test the generator's output using statistical tests:
    - The **chi-squared test** compares the distribution of generated numbers against an expected uniform distribution.
    - **Higher-order tests** (order 3 to 7) evaluate the probability of seeing unique sequences of numbers, with expected probabilities decreasing as 1/k! for each order k

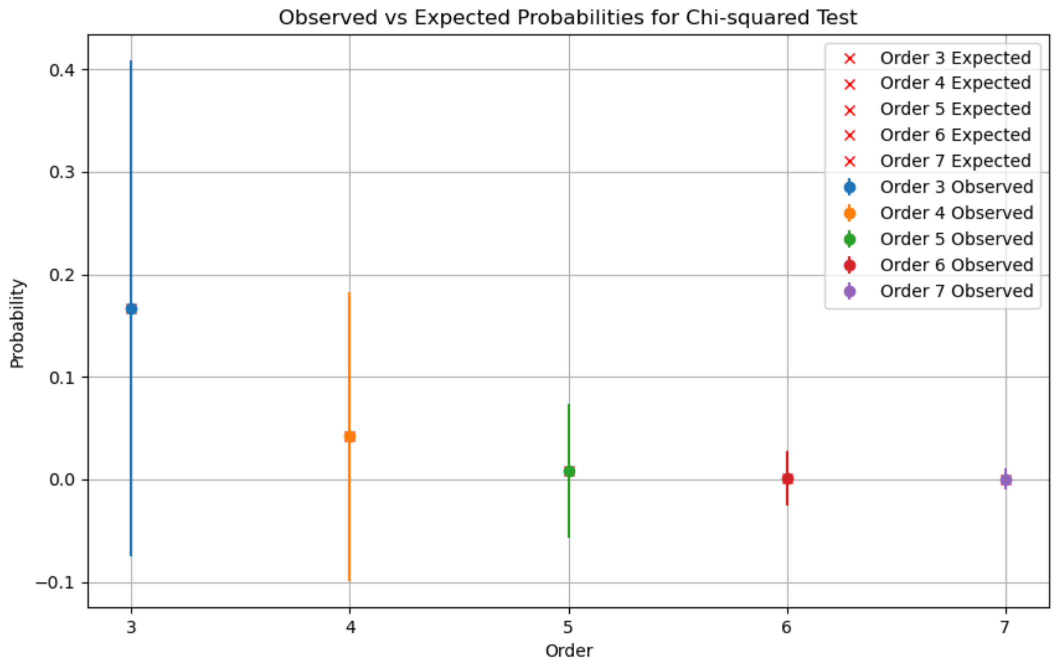
Code

Plots

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chisquare
import math

# Constants for Lehmer Algorithm
N1 = 48271
N2 = 0
N3 = 10**6
X0 = 12345
n_numbers = 126000  # Size for k=7

# Lehmer Algorithm Pseudo-Random Number Generator
def lehmer_generator(n, x0, n1, n2, n3):
    x = np.zeros(n)
    x[0] = x0
    for i in range(1, n):
        x[i] = (n1 * x[i - 1] + n2) % n3
    return x

# Generate the random numbers
random_numbers = lehmer_generator(n_numbers, X0, N1, N2, N3)

# Function to perform Chi-squared tests for different orders
def chi_squared_test(random_numbers, max_order):
    observed_probs = []
    expected_probs = []
    for order in range(3, max_order + 1):
        k_factorial = math.factorial(order)
        bin_counts = np.zeros(k_factorial)

        # Count occurrences of each tuple of length 'order'
        counts_dict = {}
        for i in range(len(random_numbers) - order):
            tuple_value = tuple(random_numbers[i:i + order])
            if tuple_value in counts_dict:
                counts_dict[tuple_value] += 1
            else:
                counts_dict[tuple_value] = 1

        # Convert counts to bin counts
        for count in counts_dict.values():
            index = count % k_factorial  # Adjust to fit within bin counts
            bin_counts[index] += count

        # Calculate observed probabilities
        observed_probs.append(bin_counts / np.sum(bin_counts))
        expected_probs.append(np.full(k_factorial, 1 / k_factorial))  # Uniform distribution

        # Perform Chi-squared test
        chisq_stat, p_value = chisquare(bin_counts + 1e-10)  # Adding small constant to avoid zero counts
        print(f"Order: {order}, Chi-squared Statistic: {chisq_stat}, P-value: {p_value}")

    return observed_probs, expected_probs

# Perform tests and get probabilities
observed_probs, expected_probs = chi_squared_test(random_numbers, 7)

# Plot observed vs expected probabilities
orders = range(3, 8)
plt.figure(figsize=(10, 6))
for order in orders:
    plt.errorbar(order, np.mean(observed_probs[order-3]), yerr=np.std(observed_probs[order-3]), label=f'Order {order} Observed', fmt='o')
    plt.plot(order, np.mean(expected_probs[order-3]), 'rx', label=f'Order {order} Expected')


plt.title('Observed vs Expected Probabilities for Chi-squared Test')
plt.xlabel('Order')
plt.ylabel('Probability')
plt.xticks(orders)
plt.legend()
plt.grid()
#plt.show()
plt.savefig(f"plot.png")
```

.

Plots


Observed vs Expected Probabilities for Chi-squared Test
Order 3 Expected
Order 4 Expected
Order 5 Expected
Order 6 Expected
Order 7 Expected
Order 3 Observed
Order 4 Observed
Order 5 Observed
Order 6 Observed
Order 7 Observed
Probability
0.4
0.3
0.2
0.1
0.0
−0.1
3
4
5
6
7
Order