НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт з комп'ютерного практикуму №1

«Перевірка генератора випадкових чисел на відповідність закону розподілу.»

роботи з дисципліни: « Моделювання систем »

Студент: Мєшков Андрій Ігорович

Група: ІП-15

Викладач: асистент Дифучин А. Ю.

Київ, 2024

Завдання

1. Згенерувати 10000 випадкових чисел трьома вказаними нижче способами. **45 балів.**

- Згенерувати випадкове число за формулою $x_i = -\frac{1}{\lambda} ln\xi_i$, де $\xi_i -$ випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξ_i можна створювати за допомогою вбудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність експоненційному закону розподілу $F(x) = 1 - e^{-\lambda x}$. Перевірку зробити при різних значеннях λ.

- Згенерувати випадкове число за формулами:

$$x_i = \sigma\mu_i + a$$

$$\mu_i = \sum_{i=1}^{12} \xi_i - 6$$

де $\xi_i -$ випадкове число, рівномірно розподілене в інтервалі (0;1). Числа можна створювати за допомогою вбудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність нормальному закону розподілу:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-a)^2}{2\sigma^2})$$

Перевірку зробити при різних значеннях a і σ.

- Згенерувати випадкове число за формулою $z_{i+1} = az_i(\mathrm{mod}c)$, $x_{i+1} = \frac{z_{i+1}}{c}$, де a=5^{13}, c=2^{31}. Перевірити на відповідність рівномірному закону розподілу в інтервалі (0;1). Перевірку зробити при різних значеннях параметрів a і c.

2. Для кожного побудованого генератора випадкових чисел побудувати гістограму частот, знайти середнє і дисперсію цих випадкових чисел. По виду гістограми частот визначити вид закону розподілу. **20 балів.**

3. Відповідність заданому закону розподілу перевірити за допомогою критерію згоди χ^2 . **30 балів**

4. Зробити висновки щодо запропонованих способів генерування випадкових величин. **5 балів**

Хід роботи

Генератори для кожного типу розподілу:

1. Експоненційний розподіл

```
class FirstGenerator:
    def __init__(self, lam):
        self.lam = lam

    def generate_number(self):
        return (-1 / self.lam) * math.log(random.random())

    def calculate_distribution(self, x):
        return 1 - math.exp(-self.lam * x)
```

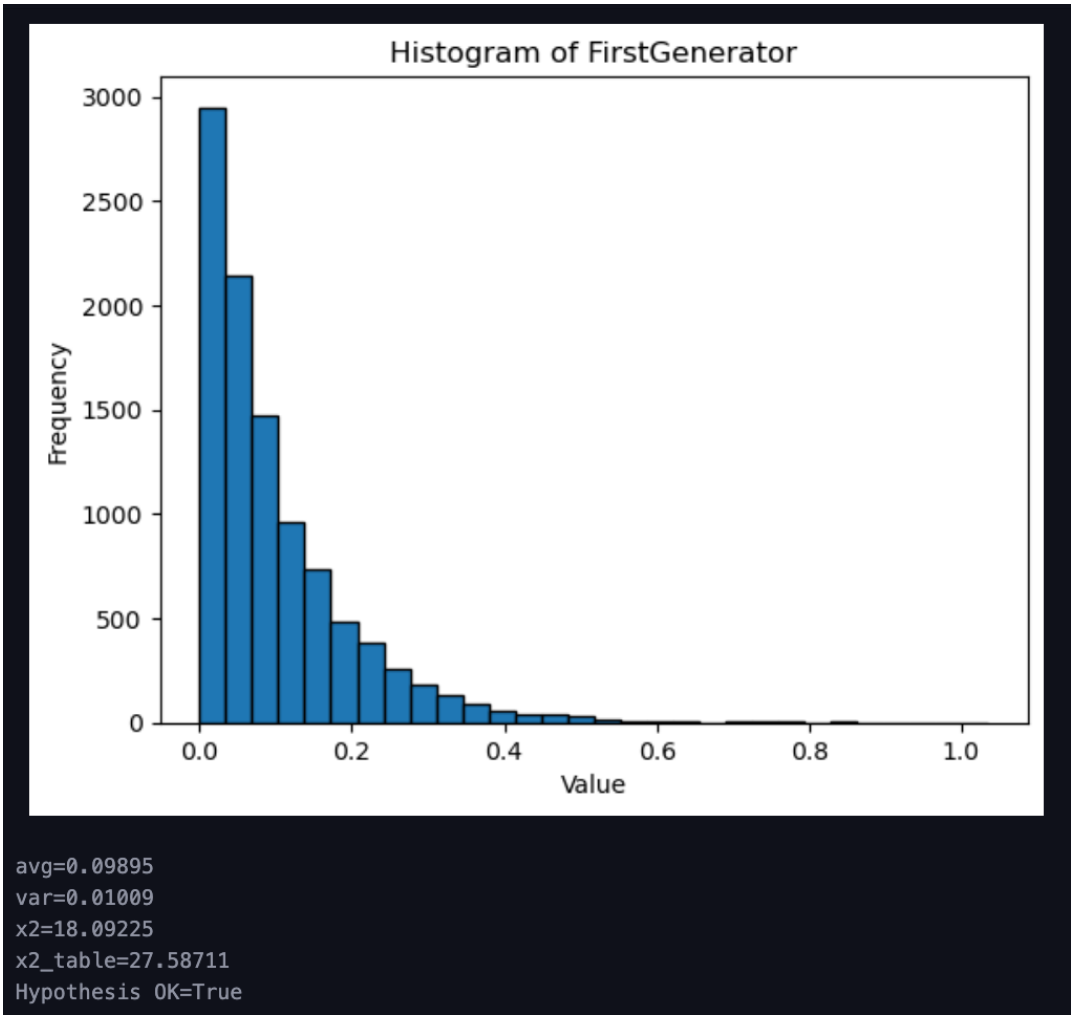


Рисунок 1.1. Перший генератор, $\lambda = 10$

За видом гістограми можемо визначити, що маємо експоненційний розподіл чисел.

χ^2 менше табличного значення при кількості ступенів свободи 17. Можна стверджувати, що знайдений закон розподілу відповідає спостережуваним значенням випадкової величини.

Перевіримо відповідність закону розподілу для різних значень λ:

λ	1	5	10	25	50	75	100
χ^2	19.02694	15.91125	14.74020	11.15002	15.96321	13.82769	10.06101
χ^2 критичне	27.58711	27.58711	27.58711	27.58711	27.58711	27.58711	27.58711

2. Нормальний розподіл

```
class SecondGenerator:
    def __init__(self, a, sigma):
        self.a = a
        self.sigma = sigma

    def generate_number(self):
        u = sum(random.random() for _ in range(12)) - 6
        return self.sigma * u + self.a

    def calculate_distribution(self, x):
        return (1 + math.erf((x - self.a) / (math.sqrt(2) * self.sigma))) / 2
```

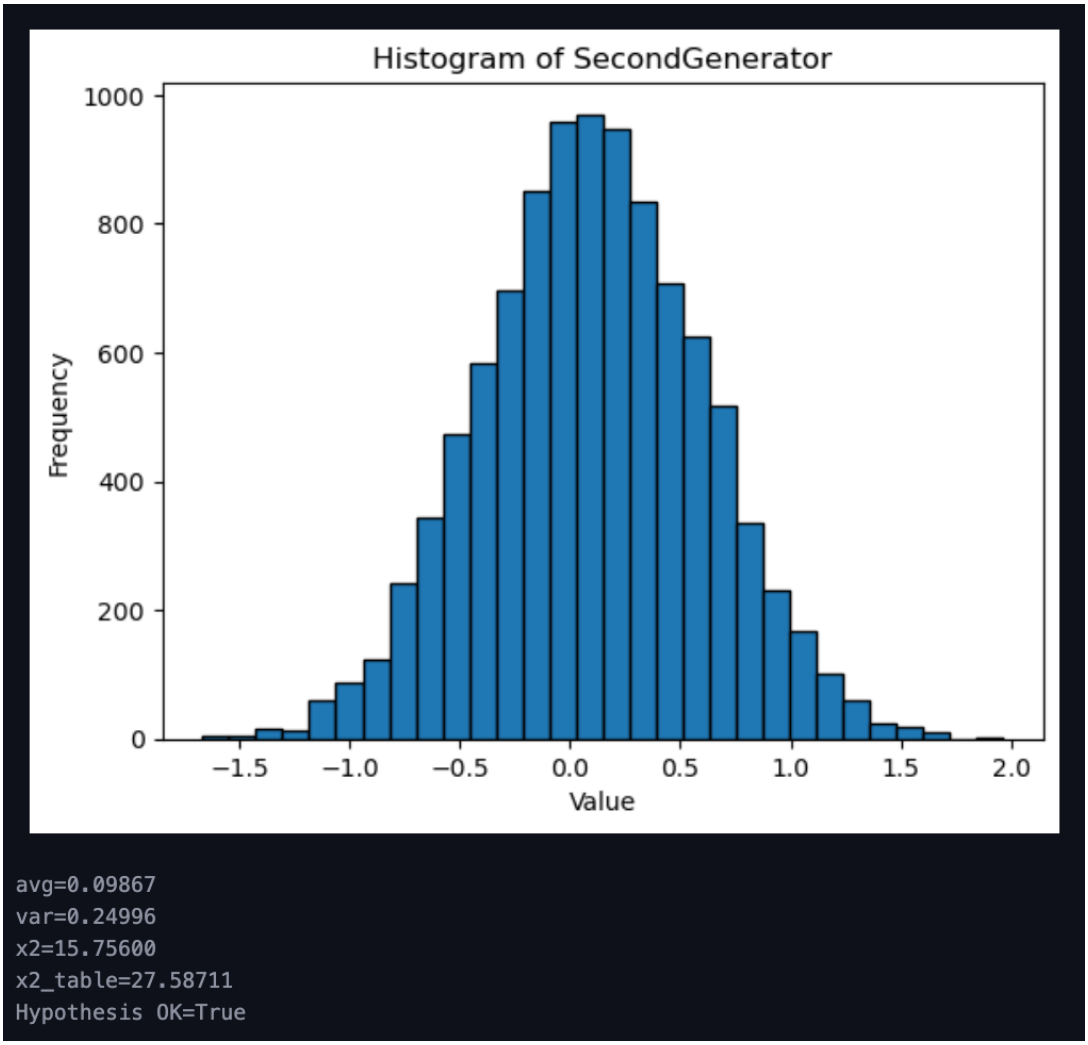


Рисунок 1.2. Другий генератор, $a = 0.1$ і $\sigma = 0.5$

За видом гістограми можемо визначити, що маємо нормальний розподіл чисел.

χ^2 менше табличного значення при кількості ступенів свободи 17. Можна стверджувати, що знайдений закон розподілу відповідає спостережуваним значенням випадкової величини.

Перевіримо відповідність закону розподілу для різних значень a і σ:

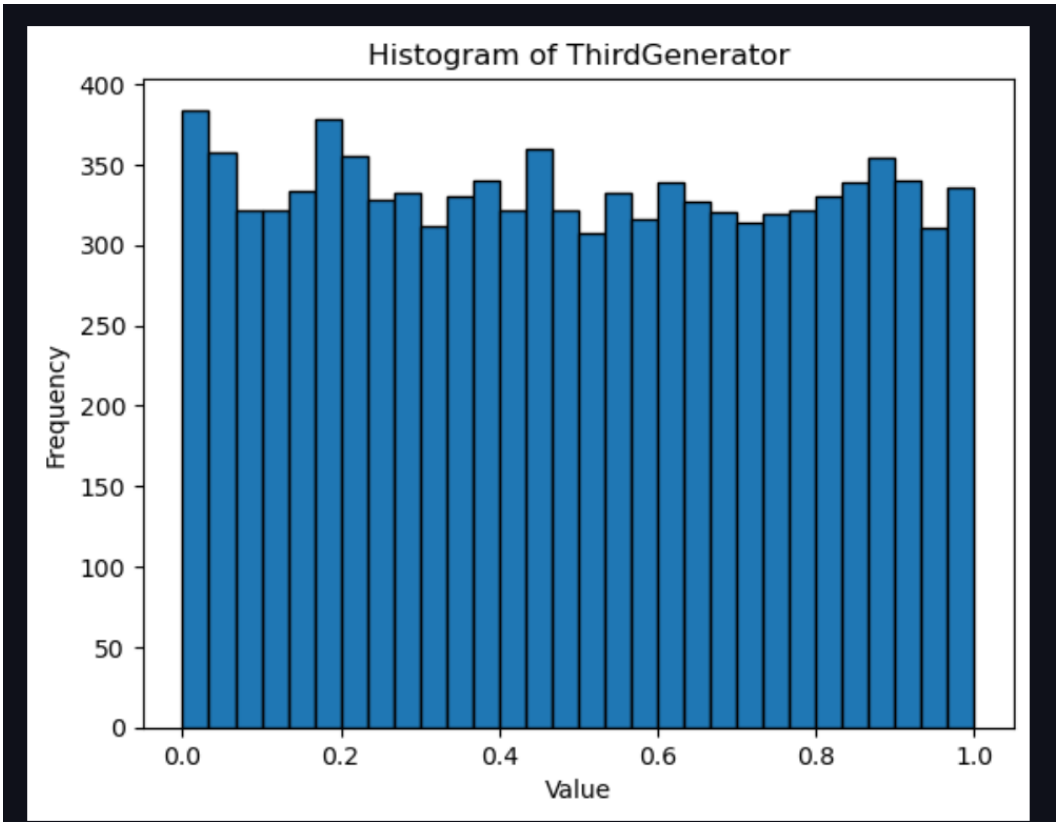
a	0.1	0.1	0.1	0.1	-100	0	100
σ	0.5	-200	1	200	0.5	0.5	0.5
χ^2	16.02644	-40031	15.16234	25.64442	22.33101	13.45594	14.36999
χ^2 критичне	27.58711	27.58711	27.58711	27.58711	27.58711	27.58711	27.58711

3. Рівномірний розподіл

```
class ThirdGenerator:
    def __init__(self, a, c):
        self.a = a
        self.c = c
        self.z = random.random()

    def generate_number(self):
        self.z = math.fmod(self.a * self.z, self.c)
        return self.z / self.c

    def calculate_distribution(self, x):
        return max(0, min(1, x))
```




```
avg=0.49451
var=0.08464
x2=21.45111
x2_table=27.58711
Hypothesis OK=True
```

Рисунок 1.3. Третій генератор, $a = 5^{13}, c = 2^{31}$

За видом гістограми можемо визначити, що маємо рівномірний розподіл чисел.

χ^2 менше табличного значення при кількості ступенів свободи 17. Можна стверджувати, що знайдений закон розподілу відповідає спостережуваним значенням випадкової величини.

Перевіримо відповідність закону розподілу для різних значень a і c:

a	5^{13}	5^{13}	5^{13}	5^{13}	5^{5}	5^{8}	5^{10}
c	2^{31}	2^{15}	2^{20}	2^{25}	2^{31}	2^{31}	2^{31}
χ^2	2.00839	11.77642	23.50469	15.24471	18.21681	14.60140	24.07789
χ^2 критичне	27.58711	27.58711	27.58711	27.58711	27.58711	27.58711	27.58711

ВИСНОВКИ

Під час виконання лабораторної роботи було здійснено перевірку трьох різних методів генерації випадкових чисел та оцінено їх відповідність теоретичним законам розподілу.

1. **Експоненційний розподіл**: За допомогою першого генератора було згенеровано випадкові числа, що повинні відповідати експоненційному розподілу. Гістограма показала, що згенеровані числа дійсно відповідають експоненційному розподілу. Критерій χ^2 підтвердив відповідність теоретичному закону розподілу при різних значеннях параметра λ, оскільки χ^2 був меншим за табличне критичне значення.
2. **Нормальний розподіл**: Другий генератор, який використовує суму рівномірно розподілених чисел для створення нормального розподілу, також дав коректні результати. Гістограма демонструє характерний для нормального розподілу вигляд, а перевірка за допомогою критерію χ^2 показала, що результат відповідає теоретичному нормальному розподілу при різних значеннях параметрів a та σ.
3. **Рівномірний розподіл**: Третій генератор, що використовує метод лінійного конгруентного генератора, показав рівномірний розподіл. Гістограма підтвердила рівномірний характер розподілу, а перевірка χ^2 також продемонструвала відповідність рівномірному розподілу при різних значеннях параметрів a та c.

Загалом, усі три генератори успішно продемонстрували свою здатність генерувати випадкові числа, які відповідають теоретичним законам розподілу. Кожен із методів пройшов перевірку за критерієм χ^2, що свідчить про їх коректність та відповідність задекларованим законам.

На основі отриманих результатів можна зробити висновок, що запропоновані способи генерування випадкових величин є ефективними і можуть бути використані для подальших моделювань та статистичних аналізів.

ЛІСТИНГ КОДУ

```
import math
import random
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
class FirstGenerator:
    def __init__(self, lam):
        self.lam = lam

    def generate_number(self):
        return (-1 / self.lam) * math.log(random.random())

    def calculate_distribution(self, x):
        return 1 - math.exp(-self.lam * x)
class SecondGenerator:
    def __init__(self, a, sigma):
        self.a = a
        self.sigma = sigma

    def generate_number(self):
        u = sum(random.random() for _ in range(12)) - 6
        return self.sigma * u + self.a

    def calculate_distribution(self, x):
        return (1 + math.erf((x - self.a) / (math.sqrt(2) * self.sigma))) / 2
        # return ((1 / (self.sigma * math.sqrt(2 * math.pi))) * math.exp(-((x -
self.alpha) ** 2) / (2 * self.sigma ** 2)))
class ThirdGenerator:
    def __init__(self, a, c):
        self.a = a
        self.c = c
        self.z = random.random()

    def generate_number(self):
        self.z = math.fmod(self.a * self.z, self.c)
        return self.z / self.c

    def calculate_distribution(self, x):
        return max(0, min(1, x))
def calc_counts_in_interval(numbers, interval_count):
    min_val, max_val = min(numbers), max(numbers)
    interval_size = (max_val - min_val) / interval_count
    counts = [0] * interval_count

    for num in numbers:
        index = int((num - min_val) / interval_size)
        if num == max_val:
            index -= 1
        counts[index] += 1
```

```
    return counts, interval_size
def calc_chi_value(calculate_distribution, numbers, interval_count):
    counts, interval_size = calc_counts_in_interval(numbers, interval_count)
    min_val = min(numbers)
    x2 = 0
    left_index = 0
    count_in_interval = 0

    for i in range(interval_count):
        count_in_interval += counts[i]
        if count_in_interval < 5 and i != interval_count - 1:
            continue

        left = min_val + interval_size * left_index
        right = min_val + interval_size * (i + 1)
        expected = len(numbers) * (calculate_distribution(right) -
calculate_distribution(left))

        x2 += (count_in_interval - expected) ** 2 / expected
        left_index = i + 1
        count_in_interval = 0

    return x2
def check_chi_value(calculate_distribution, numbers, interval_count,
significance_level):
    x2 = calc_chi_value(calculate_distribution, numbers, interval_count)
    degrees_of_freedom = interval_count - 3
    x2_table = stats.chi2.ppf(1 - significance_level, degrees_of_freedom)
    return x2, x2_table, x2 < x2_table

def analyze_generator(generator, number_count, interval_count, significance_level):
    numbers = [generator.generate_number() for _ in range(number_count)]
    avg = sum(numbers) / len(numbers)
    var = sum((x - avg) ** 2 for x in numbers) / len(numbers)
    x2, x2_table, hypothesis_ok = check_chi_value(generator.calculate_distribution,
numbers, interval_count, significance_level)

    plt.hist(numbers, bins=30, edgecolor='black')
    plt.title(f'Histogram of {generator.__class__.__name__}')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.show()
    print(f'avg={avg:.5f}')
    print(f'var={var:.5f}')
    print(f'x2={x2:.5f}')
    print(f'x2_table={x2_table:.5f}')
    print(f'Hypothesis OK={hypothesis_ok}')
number_count = 10000
interval_count = 20
significance_level = 0.05
```

```
generators = [
    FirstGenerator(1),
    SecondGenerator(0.1, 0.5),
    ThirdGenerator(5 ** 13, 2 ** 31),
]

for gen in generators:
    analyze_generator(gen, number_count, interval_count, significance_level)
```