**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 , де випадкове число, рівномірно розподілене в інтервалі (0;1). Числа можна створювати за допомогою вбудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність експоненційному закону розподілу . Перевірку зробити при різних значеннях 𝜆.
* Згенерувати випадкове число за формулами:

де 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:

Перевірку зробити при різних значеннях і .

* Згенерувати випадкове число за формулою , де =513, =231. Перевірити на відповідність рівномірному закону розподілу в інтервалі (0;1). Перевірку зробити при різних значеннях параметрів і .

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 . **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)

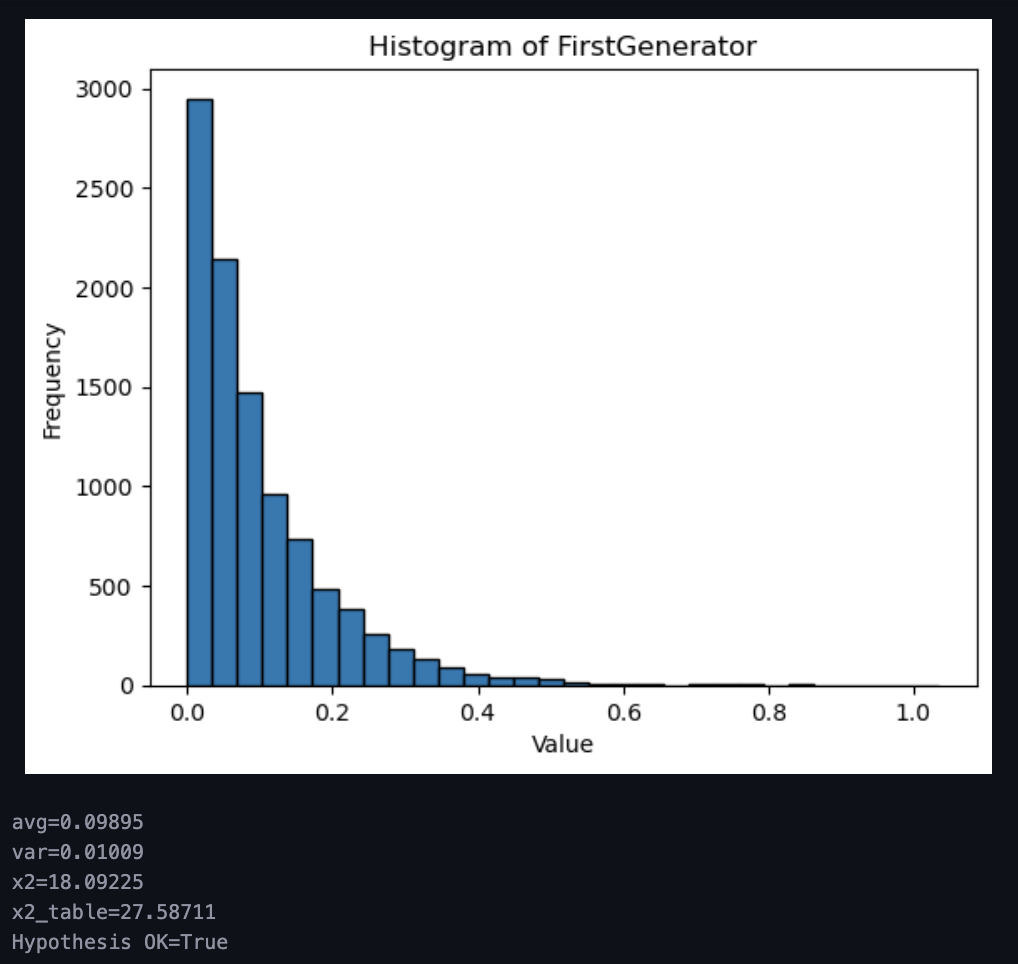


Рисунок 1.1. Перший генератор,

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

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 |
|  | 19.02694 | 15.91125 | 14.74020 | 11.15002 | 15.96321 | 13.82769 | 10.06101 |
| критичне | 27.58711 | 27.58711 | 27.58711 | 27.58711 | 27.58711 | 27.58711 | 27.58711 |

1. 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

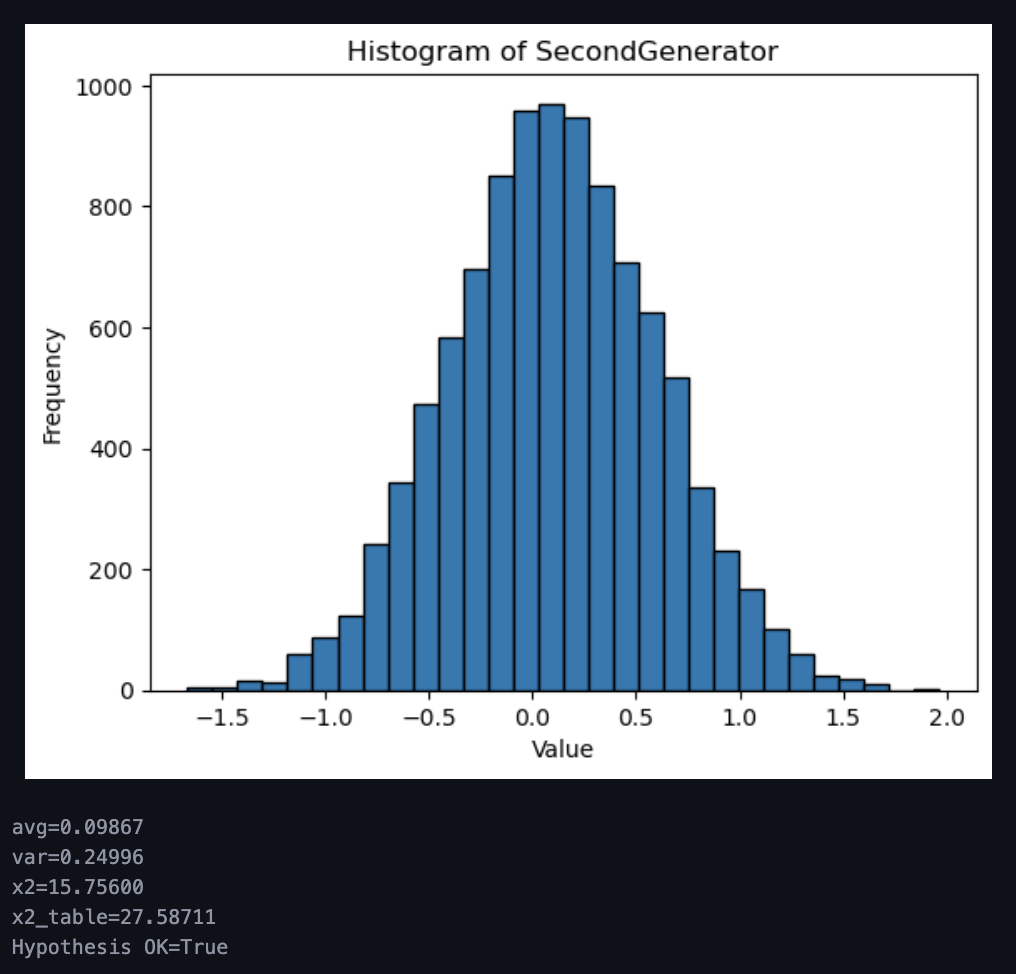


Рисунок 1.2. Другий генератор, 0.1 і

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

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 і :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.1 | 0.1 | 0.1 | 0.1 | -100 | 0 | 100 |
|  | 0.5 | -200 | 1 | 200 | 0.5 | 0.5 | 0.5 |
|  | 16.02644 | -40031 | 15.16234 | 25.64442 | 22.33101 | 13.45594 | 14.36999 |
| критичне | 27.58711 | 27.58711 | 27.58711 | 27.58711 | 27.58711 | 27.58711 | 27.58711 |

1. 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))

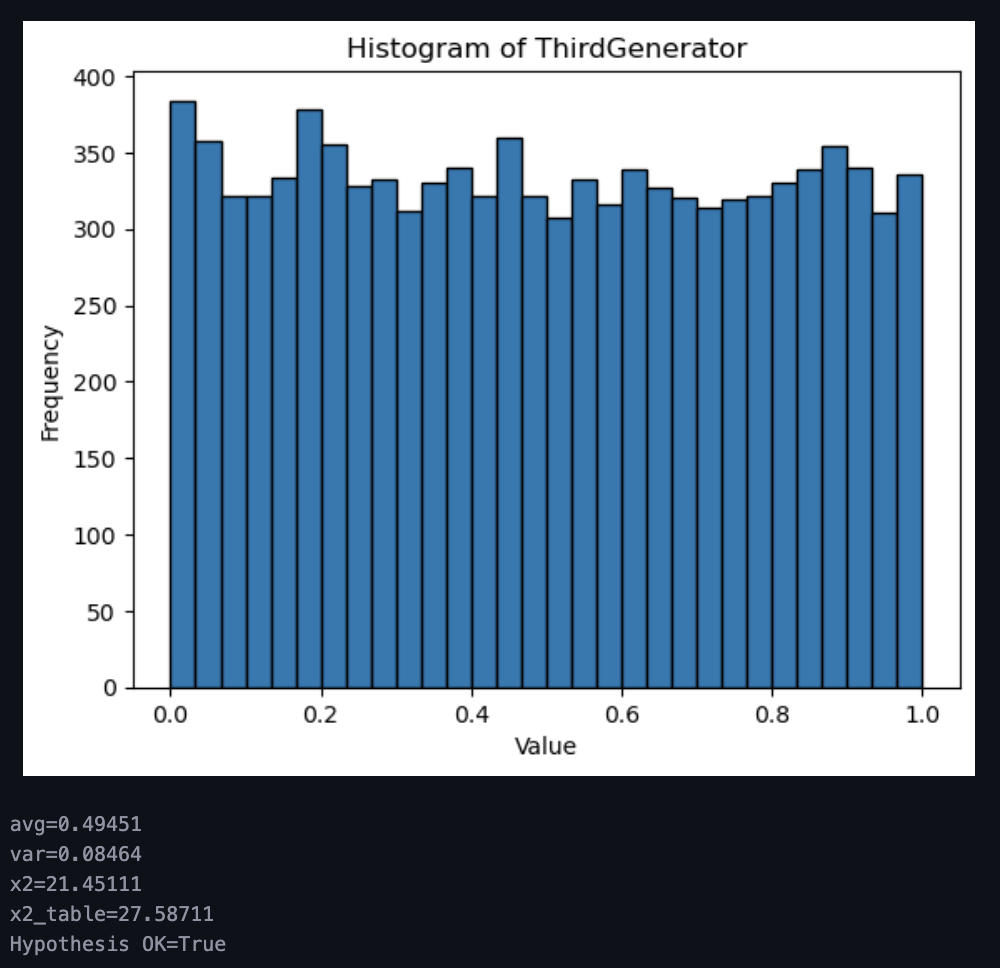


Рисунок 1.3. Третій генератор,

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

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 і :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 2.00839 | 11.77642 | 23.50469 | 15.24471 | 18.21681 | 14.60140 | 24.07789 |
| 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 χ² підтвердив відповідність теоретичному закону розподілу при різних значеннях параметра λ, оскільки χ² 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 χ² 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 χ² 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 χ², 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)