МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

**Звіт**

З лабораторної роботи № 1 з дисципліни

«Моделювання систем»

Тема: «Перевірка генератора випадкових чисел на відповідність закону розподілу»

| **Виконав(ла)** | *ІП-13 Бабіч Денис* |  |  |
| --- | --- | --- | --- |
|  | (шифр, прізвище, ім'я, по батькові) |  |  |

| **Перевірив(ла)** | *Дифучин А. Ю.* |  |  |
| --- | --- | --- | --- |
|  | (посада, прізвище, ім'я, по батькові) |  |  |

Київ 2024

# ОСНОВНА ЧАСТИНА

**Мета роботи**: Перевірити генератор випадкових чисел на відповідність закону розподілу.

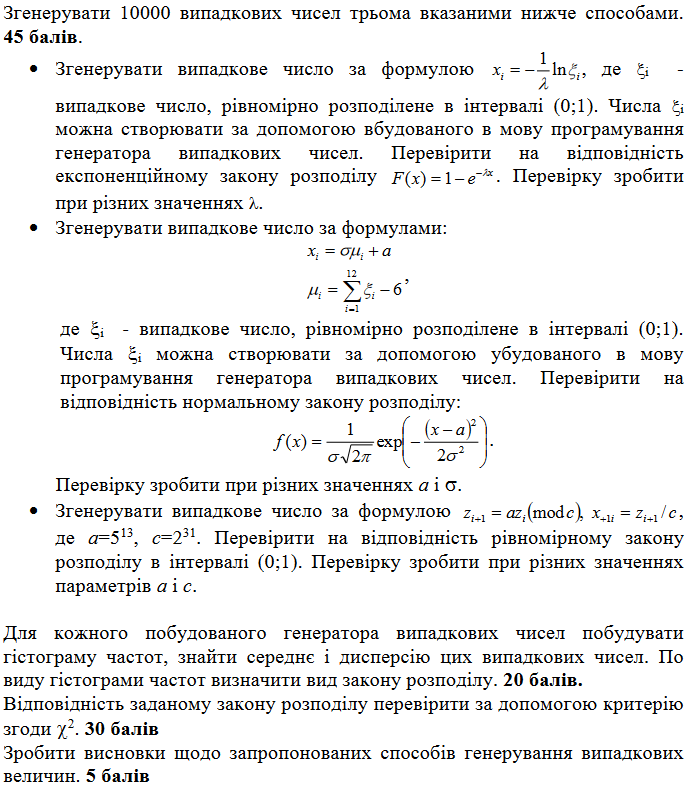


Рисунок 1.1 – Завдання лабораторного практикуму

# 

Рисунок 1.2 – Гістограма експоненційного розподілу при значенні λ = 10

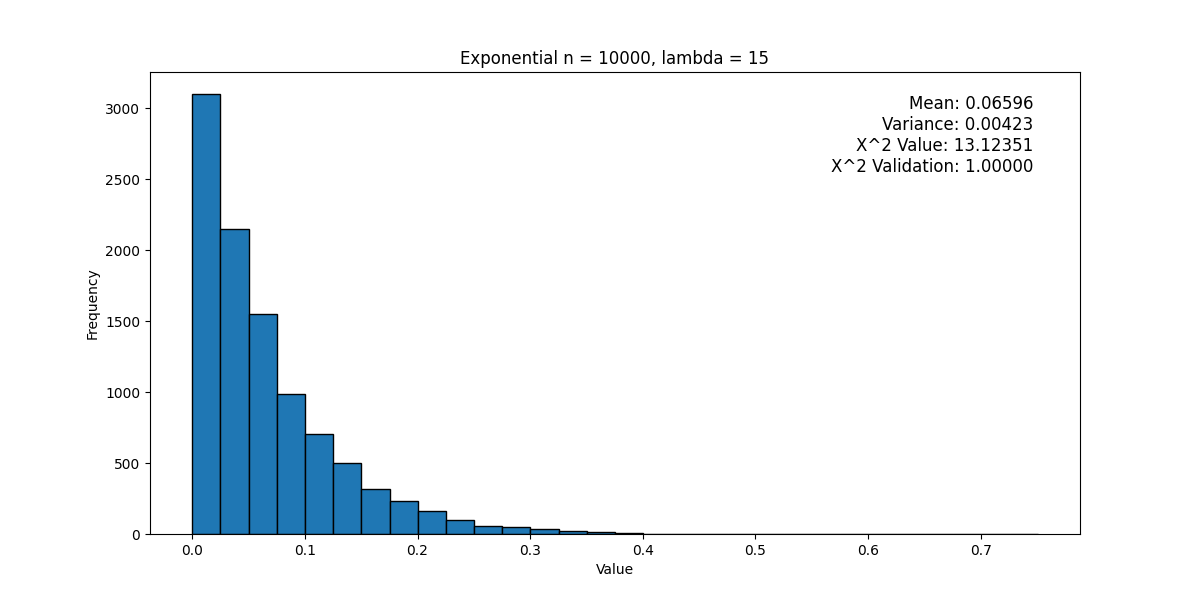


Рисунок 1.3 – Гістограма експоненційного розподілу при значенні λ = 15

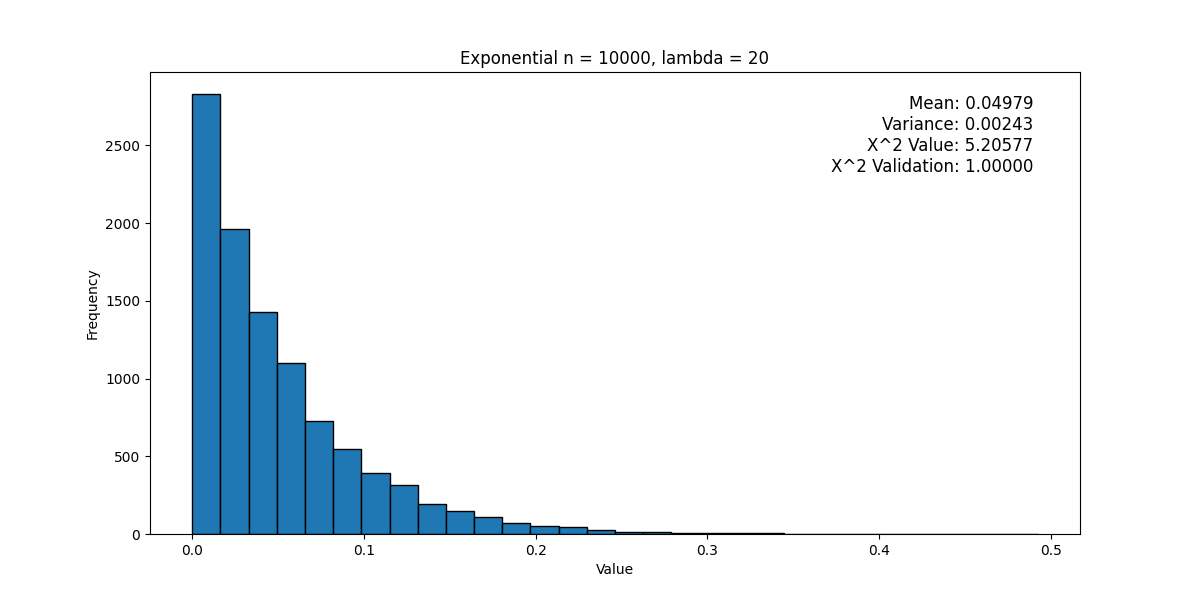


Рисунок 1.4 – Гістограма експоненційного розподілу при значенні λ = 20

З вигляду гістограм, можна зробити висновок, що розподіл значень дійсно нагадує експоненційний розподіл, оскільки наявна яскраво виражена асиметрія у бік кількості елементів ближче до 0 та швидке спадання частоти появи елементів. Також обчислення критично значення критерію X^2 підтверджує висунуту гіпотезу.

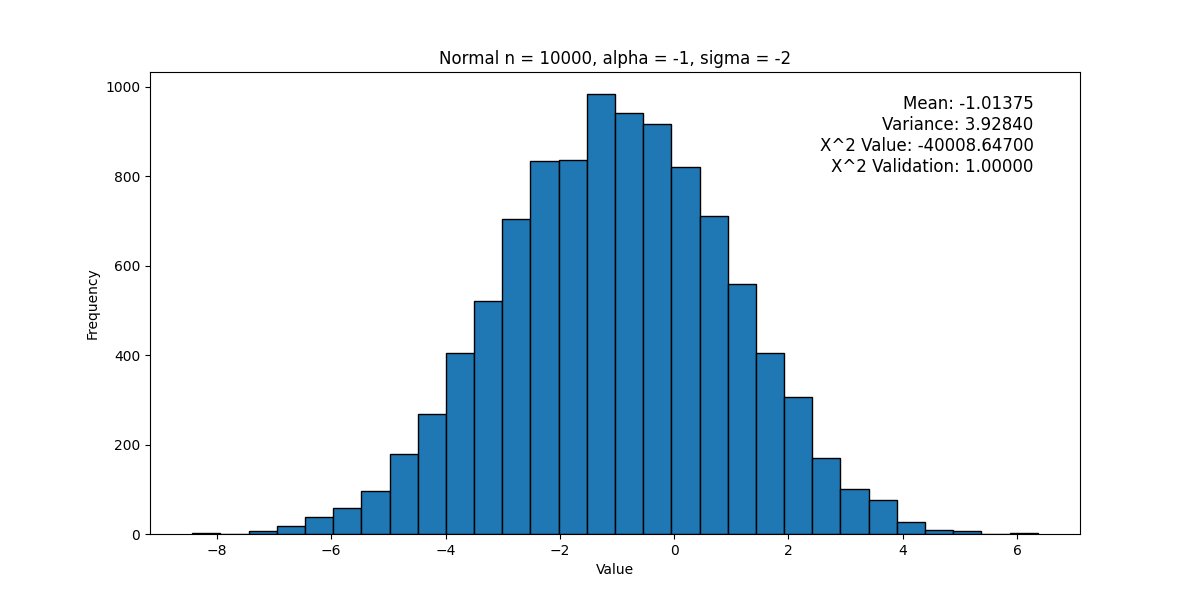


Рисунок 1.5 – Нормальний розподіл при значеннях α = -1, σ = -2

# 

Рисунок 1.6 – Нормальний розподіл при значеннях α = 5, σ = 125

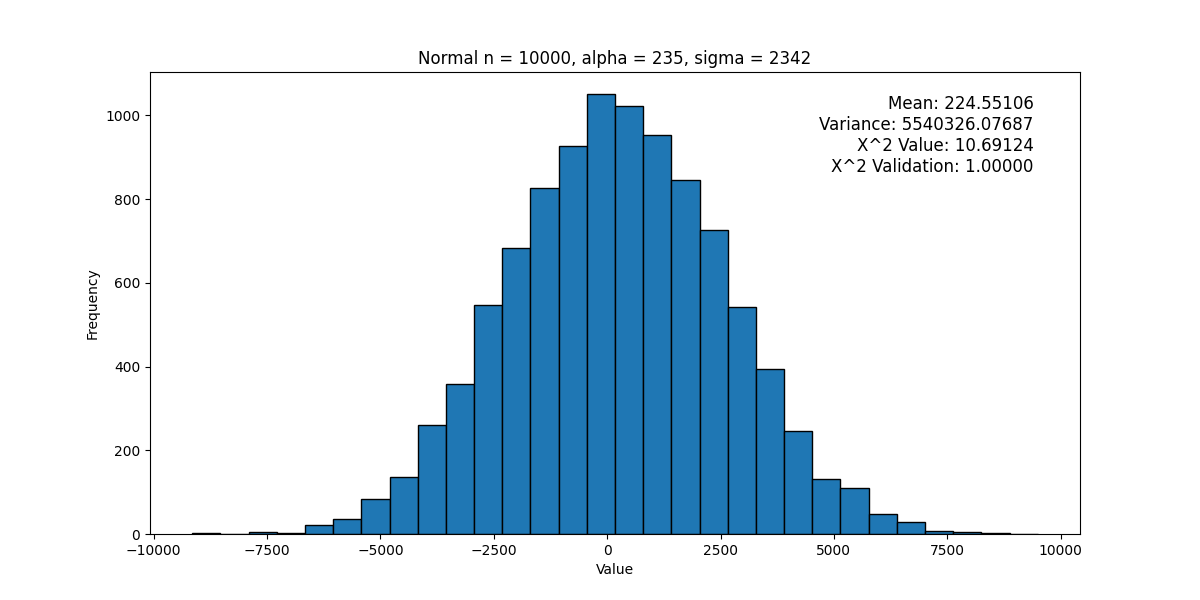


Рисунок 1.7 – Нормальний розподіл при значеннях α = 235, σ = 2342

Розглянувши гістограми частот, можна зробити висновок, що розподіл значень дійсно нагадує нормальний розподіл, оскільки наявна яскраво виражена симетричність частот появи елементів, що нагадує собою форму дзвону (гаусіанну форму). Також обчислення критично значення критерію X^2 підтверджує висунуту гіпотезу.

# 

Рисунок 1.8 – Рівномірний розподіл при значеннях a = 5^13, c = 2^31

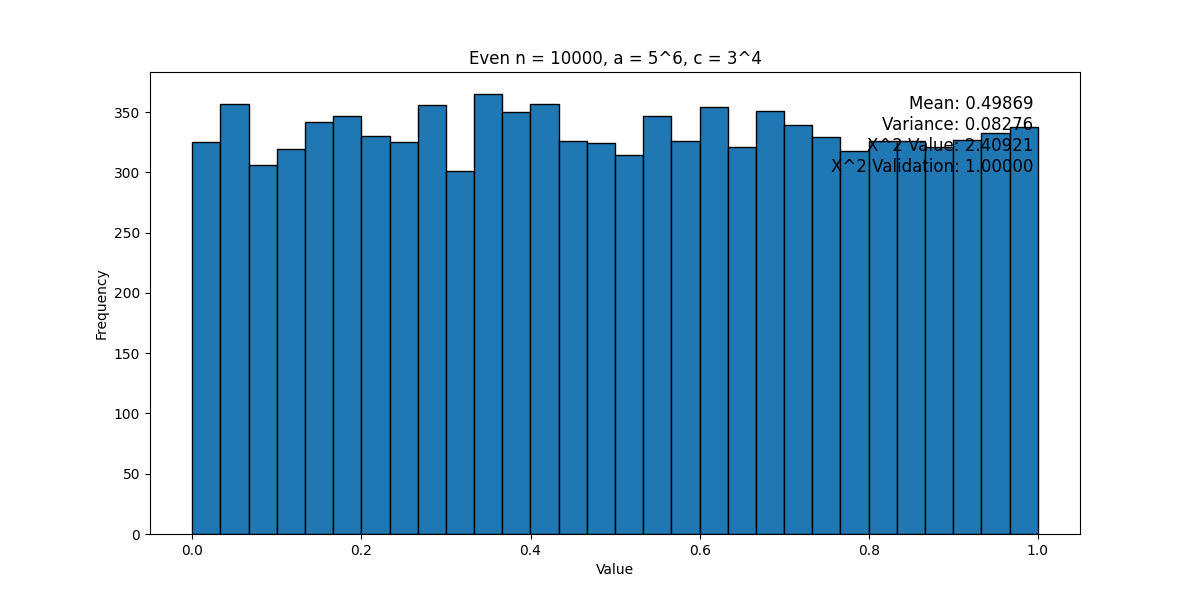


Рисунок 1.9 – Рівномірний розподіл при значеннях a = 5^6, c = 3^4

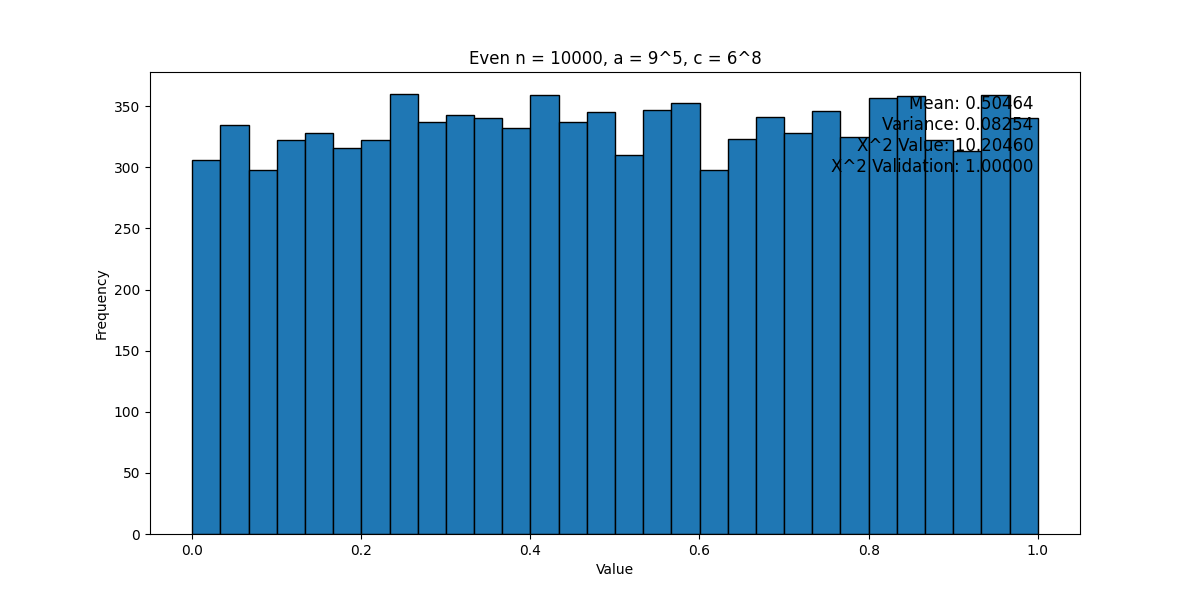


Рисунок 1.10 – Рівномірний розподіл при значеннях a = 9^5, c = 6^8

Завдяки гістограмам частот, можна зробити висновок, що розподіл значень дійсно нагадує рівномірний розподіл, оскільки легко можна помітити рівномірність розподілу частот появи елементів. Також обчислення критично значення критерію X^2 підтверджує висунуту гіпотез.

# ВИСНОВКИ

В ході виконання лабораторної роботи було досліджено основні поняття, пов’язані з генерацією випадкових чисел та їх відповідністю різним теоретичним законам розподілу. У процесі виконання практичних завдань були побудовані гістограми для різних типів розподілів: експоненційного, нормального та рівномірного, що дозволило візуально оцінити характер кожного розподілу. Також було проведено перевірку за критерієм χ², що підтвердило відповідність результатів генерації випадкових чисел відповідним теоретичним законам розподілу. Завдяки отриманим результатам можна зробити висновок, що застосовані алгоритми генерації випадкових чисел коректно відтворюють відповідні розподіли, а метод побудови гістограм дозволяє наочно оцінити характер розподілу значень.

# ДОДАТОК ПРОГРАМНИЙ КОД

import math

import random

from common import IGenerator

class EvenGenerator(IGenerator):

def \_\_init\_\_(self, a: float, c: float) -> None:

self.\_a = a

self.\_c = c

self.\_z = random.random()

def get\_parameters\_count(self) -> int:

return 2

def generate\_samples(self, numbers\_count: int) -> list[float]:

numbers: list[float] = []

for \_ in range(numbers\_count):

self.\_z = math.fmod(self.\_a \* self.\_z, self.\_c)

numbers.append(self.\_z / self.\_c)

return numbers

def calculate\_distribution(self, number: float) -> float:

if number < 0:

return 0

elif number > 1:

return 1

else:

return number

import math

import random

from common import IGenerator

class ExponentialGenerator(IGenerator):

def \_\_init\_\_(self, lambda\_value: float) -> None:

self.\_lambda\_value = lambda\_value

def get\_parameters\_count(self) -> int:

return 1

def generate\_samples(self, numbers\_count: int) -> list[float]:

numbers: list[float] = []

for \_ in range(numbers\_count):

numbers.append((-1 / self.\_lambda\_value) \* math.log(random.random()))

return numbers

def calculate\_distribution(self, number: float) -> float:

return 1 - math.exp(-self.\_lambda\_value \* number)

import math

import random

from common import IGenerator

class NormalGenerator(IGenerator):

def \_\_init\_\_(self, alpha: float, sigma: float) -> None:

self.\_alpha = alpha

self.\_sigma = sigma

def get\_parameters\_count(self) -> int:

return 2

def generate\_samples(self, numbers\_count: int) -> list[float]:

numbers: list[float] = []

for \_ in range(numbers\_count):

u = sum([random.random() for \_ in range(1, 13)]) - 6

numbers.append((self.\_sigma \* u + self.\_alpha))

return numbers

def calculate\_distribution(self, number: float) -> float:

return (1 + math.erf((number - self.\_alpha) / (2 \*\* 0.5 \* self.\_sigma))) / 2.0

from scipy import stats

from common import IGenerator

class StatisticsUtility:

@staticmethod

def calculate\_average(numbers: list[float]) -> float:

return sum(numbers) / numbers.\_\_len\_\_()

@staticmethod

def calculate\_variance(numbers: list[float]) -> float:

average = StatisticsUtility.calculate\_average(numbers)

return sum(((number - average) \*\* 2) for number in numbers) / numbers.\_\_len\_\_()

@staticmethod

def evaluate\_distribution(generator: IGenerator, samples: list[float], intervals\_count: int, significance\_level: float = 0.05) -> tuple[float, bool]:

MIN\_NUMBER = min(samples)

MAX\_NUMBER = max(samples)

MIN\_EXPECTED\_FREQUENCY = 5

LAST\_INTERVAL\_INDEX = intervals\_count - 1

intervals\_sizes = [0 for \_ in range(intervals\_count + 1)]

interval\_width = (MAX\_NUMBER - MIN\_NUMBER) / intervals\_count

for number in samples:

index = int((number - MIN\_NUMBER) / interval\_width)

if number == MAX\_NUMBER:

index -= 1

intervals\_sizes[index] += 1

chi2 = 0

cumulative\_observed = 0

left\_interval\_index = 0

right\_interval\_index = 0

for i in range(intervals\_count):

cumulative\_observed += intervals\_sizes[i]

if cumulative\_observed < MIN\_EXPECTED\_FREQUENCY and i != LAST\_INTERVAL\_INDEX:

continue

right\_interval\_index = (i + 1)

left\_boundary = MIN\_NUMBER + interval\_width \* left\_interval\_index

right\_boundary = MIN\_NUMBER + interval\_width \* right\_interval\_index

expected\_count = samples.\_\_len\_\_() \* (generator.calculate\_distribution(right\_boundary) - generator.calculate\_distribution(left\_boundary))

chi2 += ((cumulative\_observed - expected\_count) \*\* 2) / expected\_count

cumulative\_observed = 0

left\_interval\_index = i + 1

degrees\_of\_freedom = intervals\_count - 1 - generator.get\_parameters\_count()

chi\_critical\_value = stats.chi2.ppf(1.0 - significance\_level, degrees\_of\_freedom)

return (chi2, (chi2 < chi\_critical\_value))

import matplotlib.pyplot as plt

class VisualizerUtility:

\_output\_path: str = "/out"

@classmethod

def setup(cls, output\_path: str) -> None:

cls.\_output\_path = output\_path

@staticmethod

def save\_plot\_histogram(numbers: list[float], title: str = "", data: dict[str, float] = {}) -> None:

plt.figure(figsize = (12, 6))

plt.hist(numbers, bins = 30, edgecolor = 'black')

plt.title(title)

plt.xlabel('Value')

plt.ylabel('Frequency')

annotation\_text = "\n".join([f"{key}: {value:.5f}" for key, value in data.items()])

plt.annotate(annotation\_text, xy = (0.95, 0.95), xycoords = 'axes fraction',

fontsize = 12, verticalalignment = 'top', horizontalalignment = 'right')

plt.savefig(f"{VisualizerUtility.\_output\_path}{title}.png")

plt.close()

from utilities import StatisticsUtility, VisualizerUtility

from generators import ExponentialGenerator, NormalGenerator, EvenGenerator

def run\_even\_generators(numbers\_count: int, intervals\_count: int) -> None:

generator = EvenGenerator(5 \*\* 13, 2 \*\* 31)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Even n = {numbers\_count}, a = 5^13, c = 2^31", stats\_results)

generator = EvenGenerator(9 \*\* 5, 6 \*\* 8)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Even n = {numbers\_count}, a = 9^5, c = 6^8", stats\_results)

generator = EvenGenerator(5 \*\* 6, 3 \*\* 4)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Even n = {numbers\_count}, a = 5^6, c = 3^4", stats\_results)

def run\_normal\_generators(numbers\_count: int, intervals\_count: int) -> None:

generator = NormalGenerator(-1, -2)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Normal n = {numbers\_count}, alpha = -1, sigma = -2", stats\_results)

generator = NormalGenerator(5, 125)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Normal n = {numbers\_count}, alpha = 5, sigma = 125", stats\_results)

generator = NormalGenerator(235, 2342)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Normal n = {numbers\_count}, alpha = 235, sigma = 2342", stats\_results)

def run\_exponential\_generators(numbers\_count: int, intervals\_count: int) -> None:

generator = ExponentialGenerator(10)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Exponential n = {numbers\_count}, lambda = 10", stats\_results)

generator = ExponentialGenerator(15)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Exponential n = {numbers\_count}, lambda = 15", stats\_results)

generator = ExponentialGenerator(20)

samples = generator.generate\_samples(numbers\_count)

stats\_results = {

"Mean": StatisticsUtility.calculate\_average(samples),

"Variance": StatisticsUtility.calculate\_variance(samples),

"X^2 Value": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[0],

"X^2 Validation": StatisticsUtility.evaluate\_distribution(generator, samples, intervals\_count)[1]

}

VisualizerUtility.save\_plot\_histogram(samples, f"Exponential n = {numbers\_count}, lambda = 20", stats\_results)

if \_\_name\_\_ == "\_\_main\_\_":

VisualizerUtility.setup("../labs/1/docs/")

intervals\_count = 10

numbers\_count = 10\_000

run\_even\_generators(numbers\_count, intervals\_count)

run\_normal\_generators(numbers\_count, intervals\_count)

run\_exponential\_generators(numbers\_count, intervals\_count)