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.

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

- Згенерувати випадкове число за формулою $x_i = -\frac{1}{\lambda}\ln\xi_i$, де ξ_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 ξ_i - випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξ_i можна створювати за допомогою убудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність нормальному закону розподілу:

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

Перевірку зробити при різних значеннях a і σ.

- Згенерувати випадкове число за формулою $z_{i+1} = az_i \pmod c$, $x_{+1i} = z_{i+1}/c$, 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.

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.**
Відповідність заданому закону розподілу перевірити за допомогою критерію згоди χ^2. **30 балів**
Зробити висновки щодо запропонованих способів генерування випадкових величин. **5 балів**

Рисунок 1.1 – Завдання лабораторного практикуму

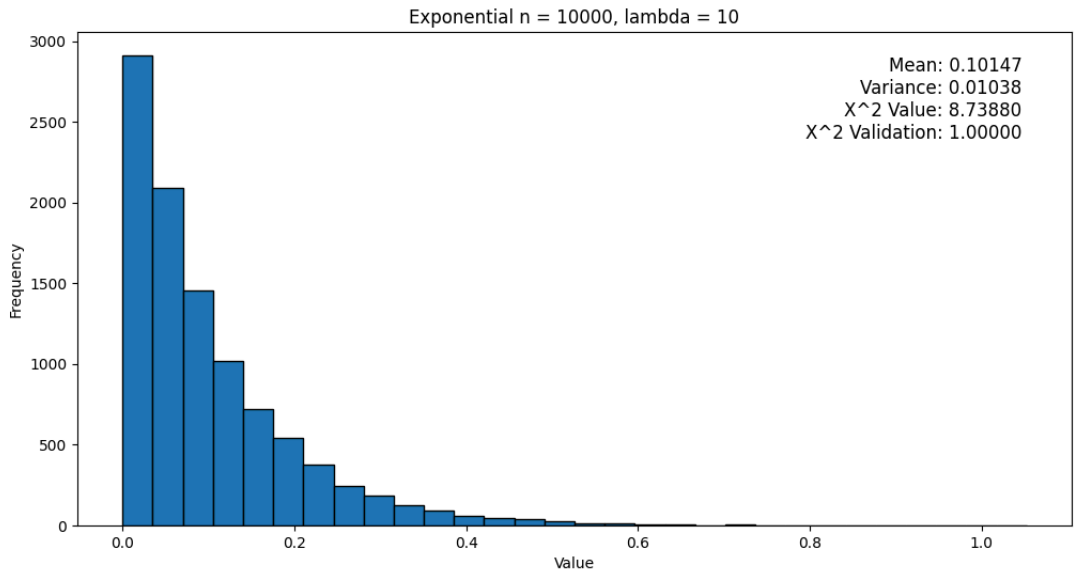

Рисунок 1.2 – Гістограма експоненційного розподілу при значенні λ = 10

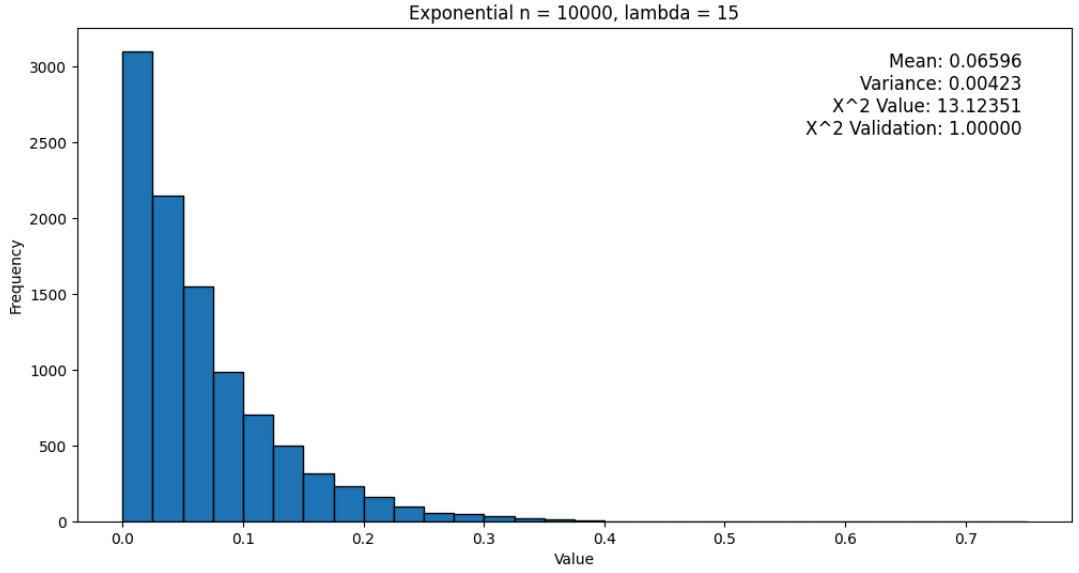

Рисунок 1.3 – Гістограма експоненційного розподілу при значенні λ = 15

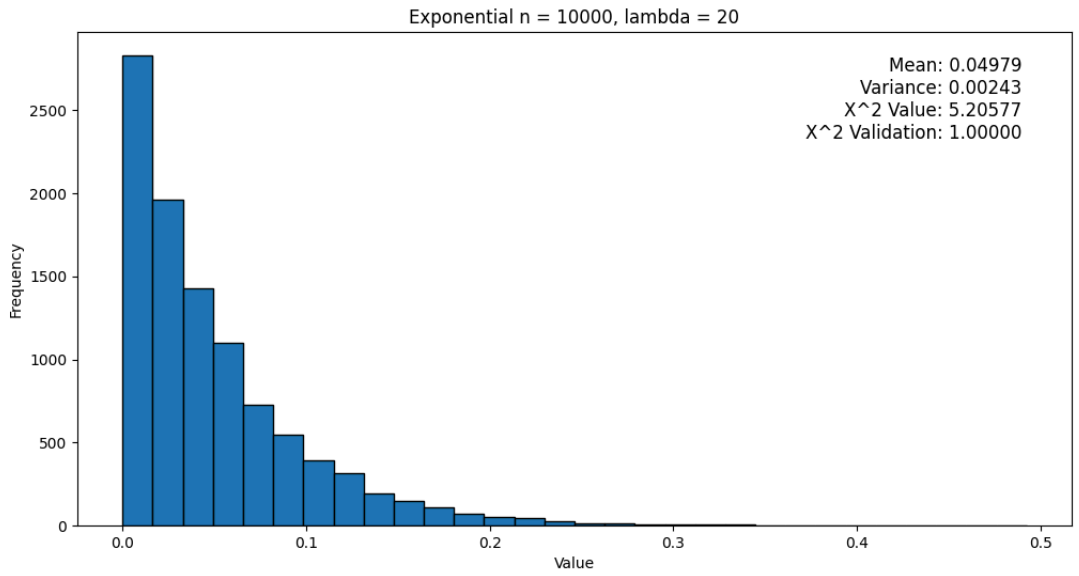

Рисунок 1.4 – Гістограма експоненційного розподілу при значенні λ = 20

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.

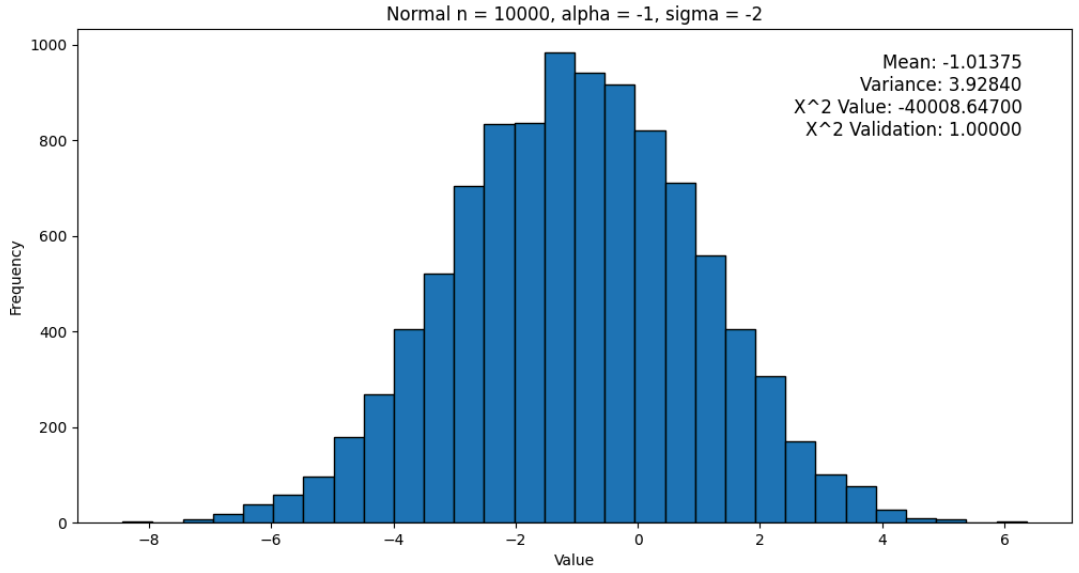

Рисунок 1.5 – Нормальний розподіл при значеннях α = -1, σ = -2

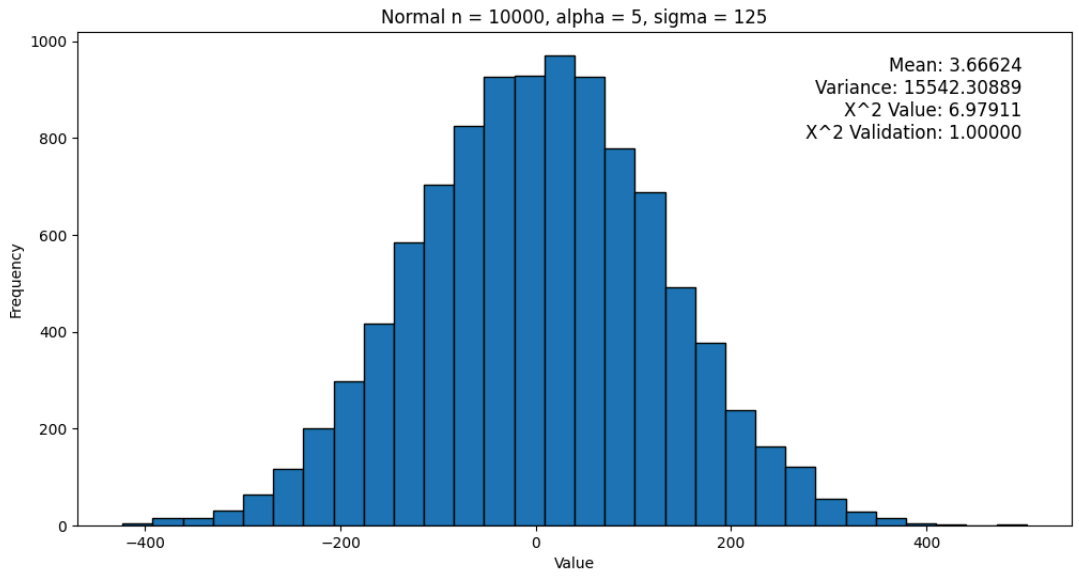

Рисунок 1.6 – Нормальний розподіл при значеннях α = 5, σ = 125

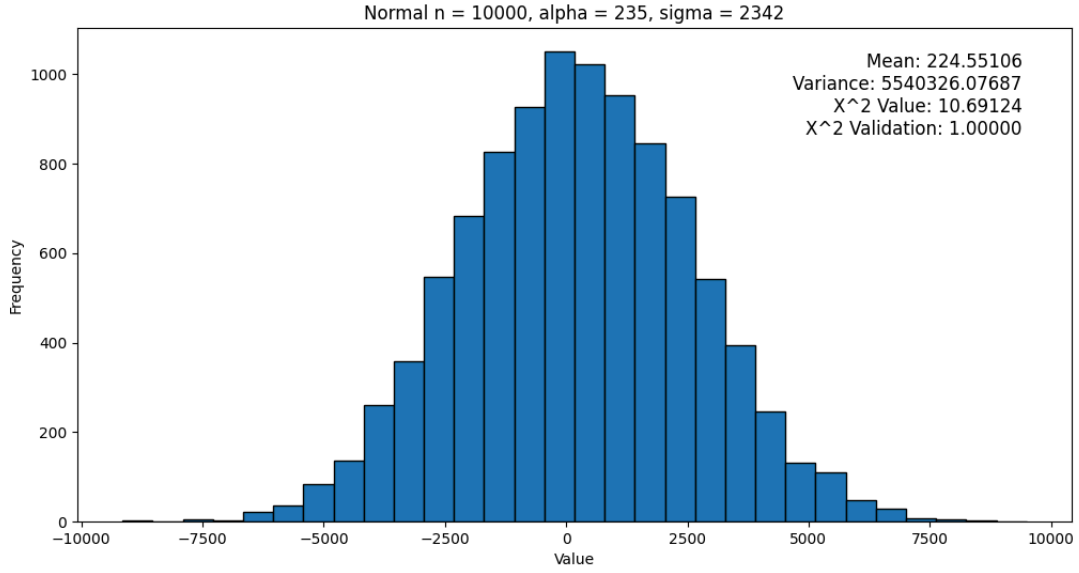

Рисунок 1.7 – Нормальний розподіл при значеннях α = 235, σ = 2342

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.

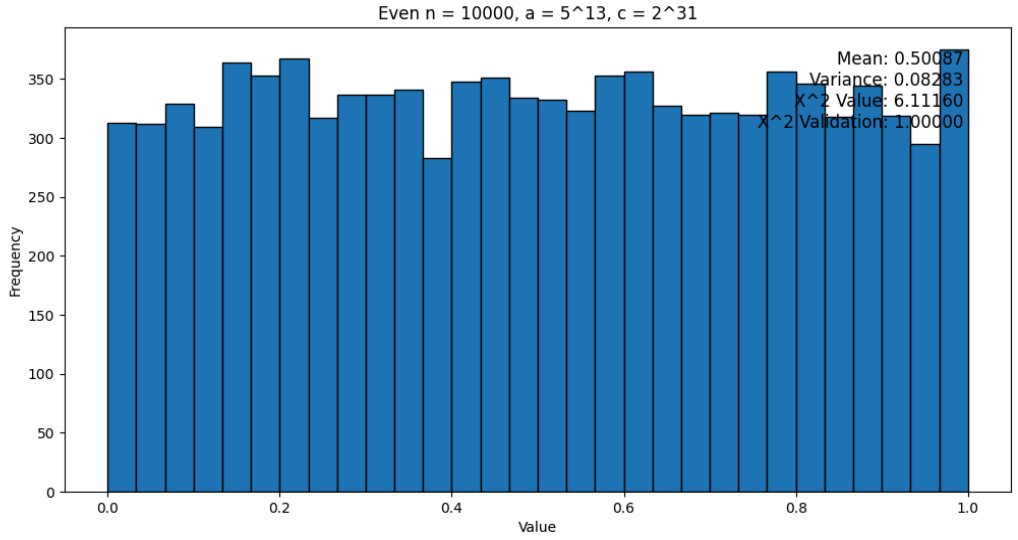

Рисунок 1.8 – Рівномірний розподіл при значеннях a = 5^13, c = 2^31

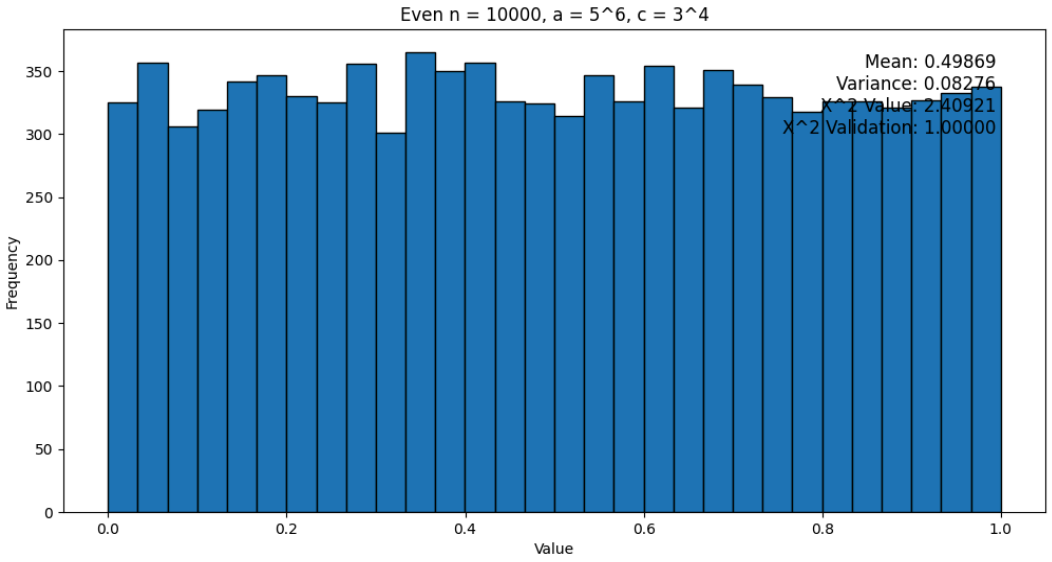

Рисунок 1.9 – Рівномірний розподіл при значеннях a = 5^6, c = 3^4

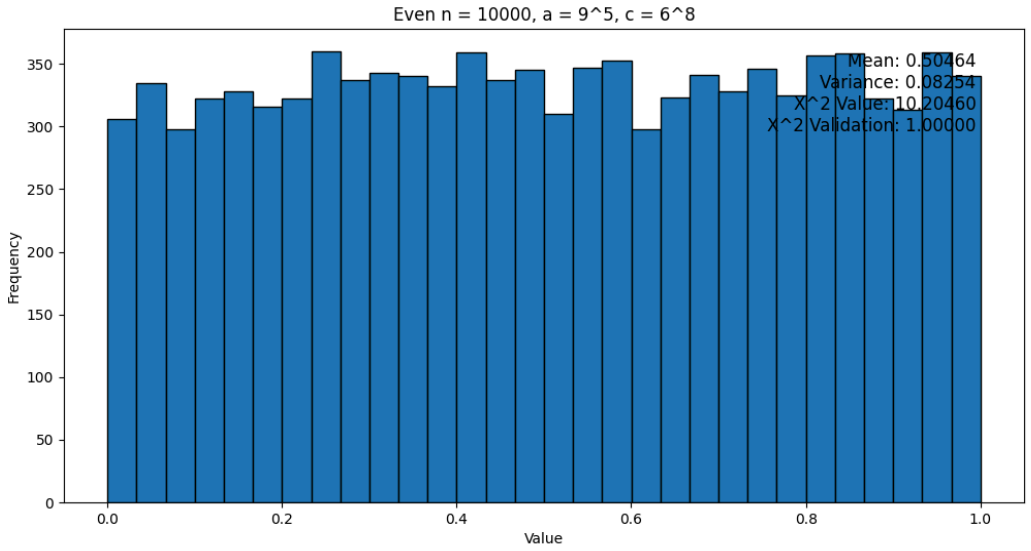

Рисунок 1.10 – Рівномірний розподіл при значеннях a = 9^5, c = 6^8

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 χ^2, 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)
```