Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»

КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

Лабораторная работа №6

«Исследование качества генераторов случайных чисел»

ДИСЦИПЛИНА: «Моделирование»

Выполнил: студент гр. ИУК4-72Б _____ (Сафронов Н.С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Никитенко У.В.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2023

Цель работы: изучить и практически освоить оценки качества генераторов случайных чисел (ГСЧ) в различных системах программирования по заданным теоретическим показателям, с помощью критериев согласия и с помощью нормированной автокорреляционной функции на предмет независимости случайных чисел.

Постановка задачи

Вариант 14

Выполняемые задания: 2, 4.3(4-6), 5.2.

Задание 2

1. Написать программу на языке программирования формирования простых трехзначных чисел с целью их использования в качестве начальных чисел в методе Фибоначчи. Рассчитать относительные погрешности по математическому ожиданию, дисперсии, стандартному отклонению.

2. Построить гистограммы для сформированных выборок (Zx и Zy) с разбивкой графического окна.

Задание 4.3

1. Полагая в формуле, написать в MATLAB(PYTHON) программу формирования случайных чисел, приняв следующие числа для расчета модуля в зависимости от номера варианта:

№3	№1: $N = 7.11*10^6$; №2: $N = 7.22*10^6$; №3: $N = 8.33*10^6$; №4: $N = 8.44*10^6$; №5: $N = 9.55*10^6$; №6: $N = 9.66*10^6$; №7: $N = 10.77*10^6$; №8: $N = 10.88*10^6$; №9: $N = 10.99*10^6$;

2. В качестве первого назначаемого случайного числа (в зависимости от номера варианта) принять следующие значения:

№3	№1: $m(31)$, №2: $m(32)$, №3: $m(33)$, №4: $m(34)$, №: $m(35)$, №6: $m(36)$, №7: $m(37)$, №8: $m(38)$, №9: $m(38)$, где m – массив простых чисел, сформированный с помощью выражения `m = primes(N);`

3. Вычислить период формируемой случайной последовательности;

4. Произвести статистический анализ созданного ГСЧ по линейному конгруэнтному методу;

5. Построить гистограммы полученных распределений случайных чисел.

6. Построить функции плотности и распределения для сформированных выборок случайных чисел. Совместить диаграммы с теоретическими функциями.

Задание 5.2.

По критерию Колмогорова – Смирнова протестировать выборки случайных чисел объема 100, 500, 1000, сформированных по линейному конгруэнтному методу.

Ход выполнения работы

Задание 2

```
Среднее выборочное: 524.94
Выборочная дисперсия: 69046.21
Выборочное стандартное отклонение: 262.77

Относительная погрешность по математическому ожиданию: 4.56%
Относительная погрешность по дисперсии: 2.29%
Относительная погрешность по стандартному отклонению: 1.14%
```

Рисунок 1 – Результаты расчёта относительных погрешностей по математическому ожиданию, дисперсии и стандартному отклонению

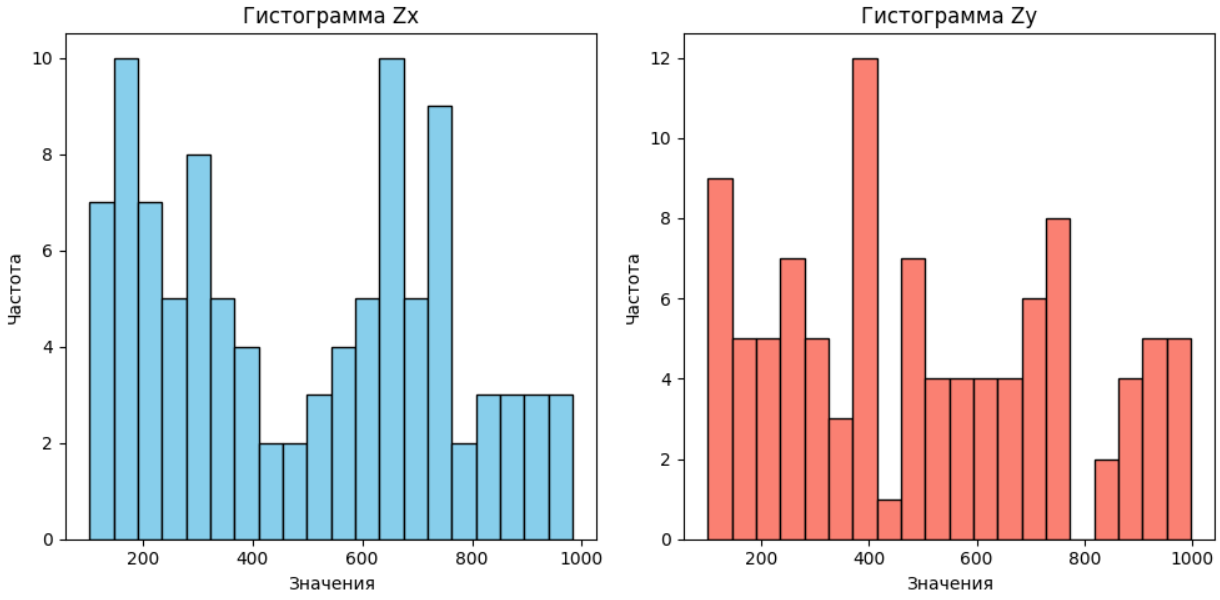

Рисунок 2 - Гистограммы для сформированных выборок Zx и Zy

Задание 4.3

```
При N=8440000 и R0=m(34)=149
Период формируемой случайной последовательности равен: 4219994
Среднее выборочное: 0.50
Выборочная дисперсия: 0.08
Выборочное стандартное отклонение: 0.29
Относительная погрешность по математическому ожиданию: 0.01%
Относительная погрешность по дисперсии: 2.10%
Относительная погрешность по стандартному отклонению: 1.06%
```

Рисунок 3 - Результат вычисления периода и погрешностей ГСЧ, созданного по линейному конгруэнтному методу при $N = 8.44 \cdot 10^6, R0 = m(34)$

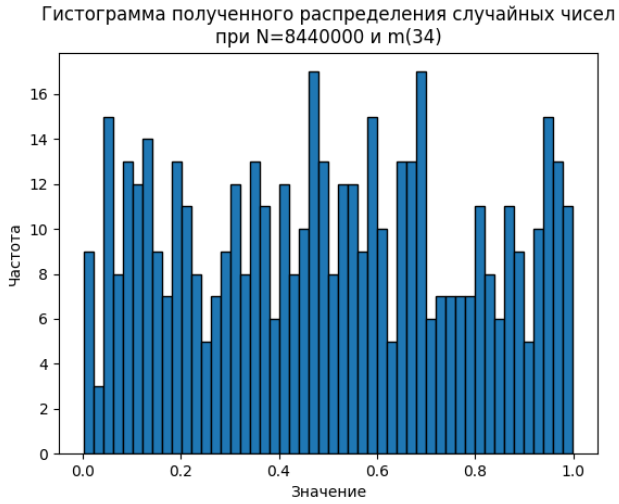

Рисунок 4 - Гистограмма распределения при $N = 8.44 \cdot 10^6, R0 = m(34)$

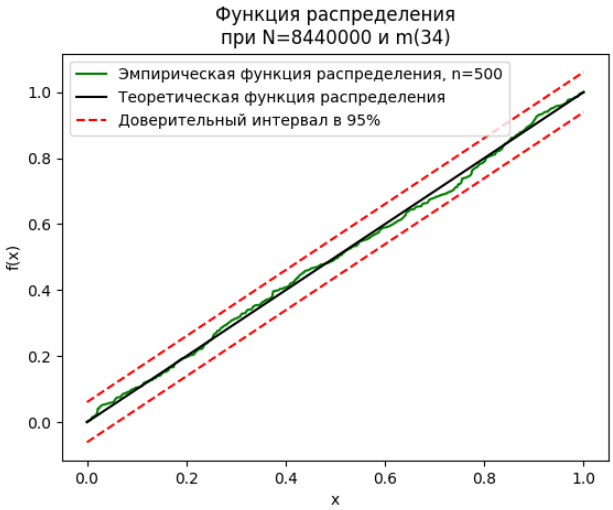

Рисунок 5 - Функция распределения при $N = 8.44 \cdot 10^6, R0 = m(34)$

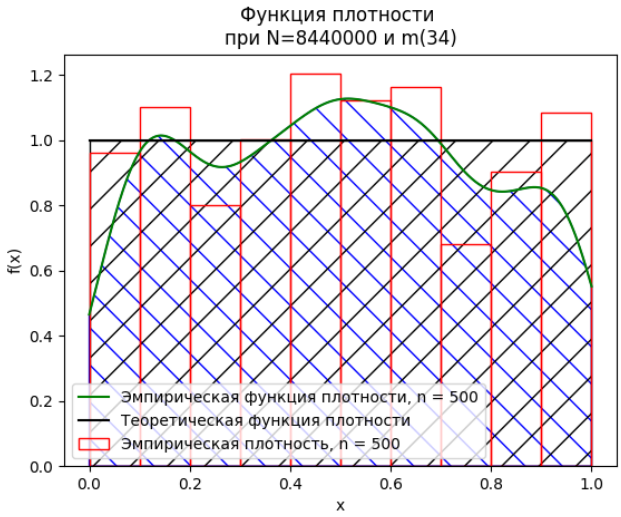

Рисунок 6 - Функция распределения при $N = 8.44 \cdot 10^6, R0 = m(34)$

```
При N=9550000 и R0=m(35)=151
Период формируемой случайной последовательности равен: 9549982
Среднее выборочное: 0.51
Выборочная дисперсия: 0.08
Выборочное стандартное отклонение: 0.29
Относительная погрешность по математическому ожиданию: 1.02%
Относительная погрешность по дисперсии: 1.62%
Относительная погрешность по стандартному отклонению: 0.81%
```

Рисунок 7 - Результат вычисления периода и погрешностей ГСЧ, созданного по линейному конгруэнтному методу при $N = 9.55 \cdot 10^6, R0 = m(35)$

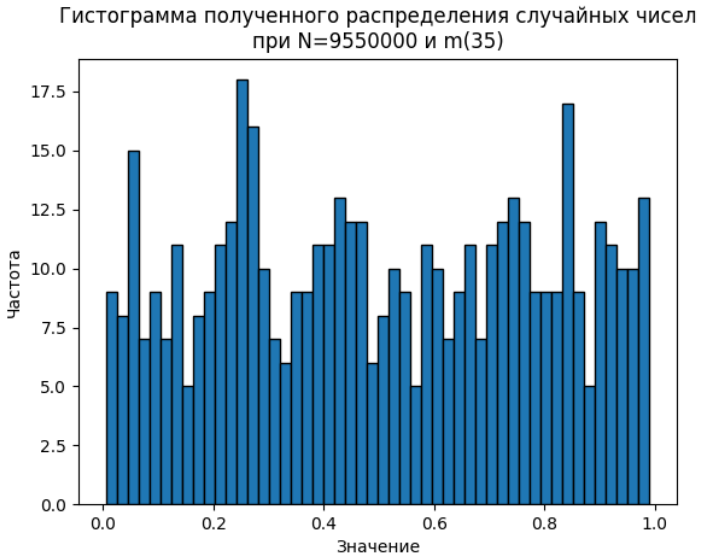

Рисунок 8 - Гистограмма распределения при $N = 9.55 \cdot 10^6, R0 = m(35)$

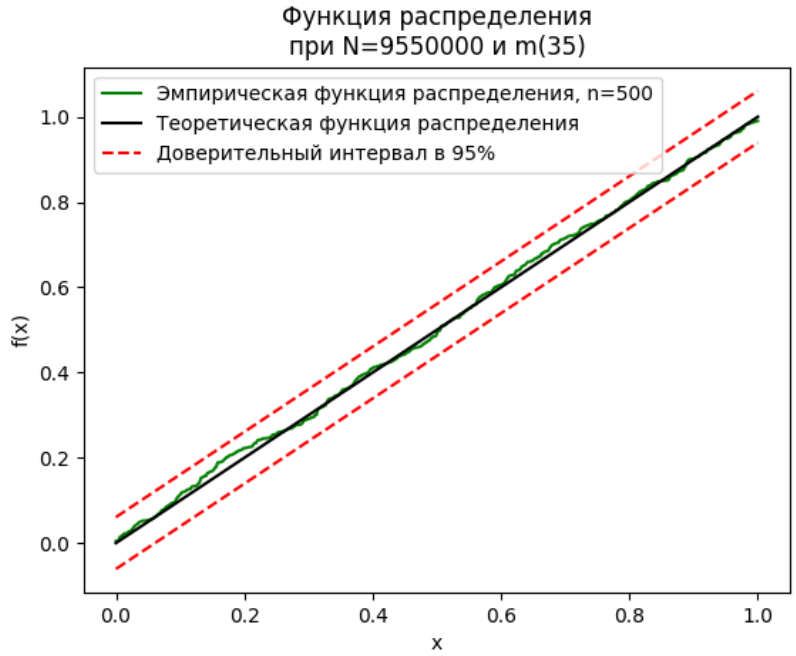

Рисунок 9 - Функция распределения при $N = 9.55 \cdot 10^6, R0 = m(35)$

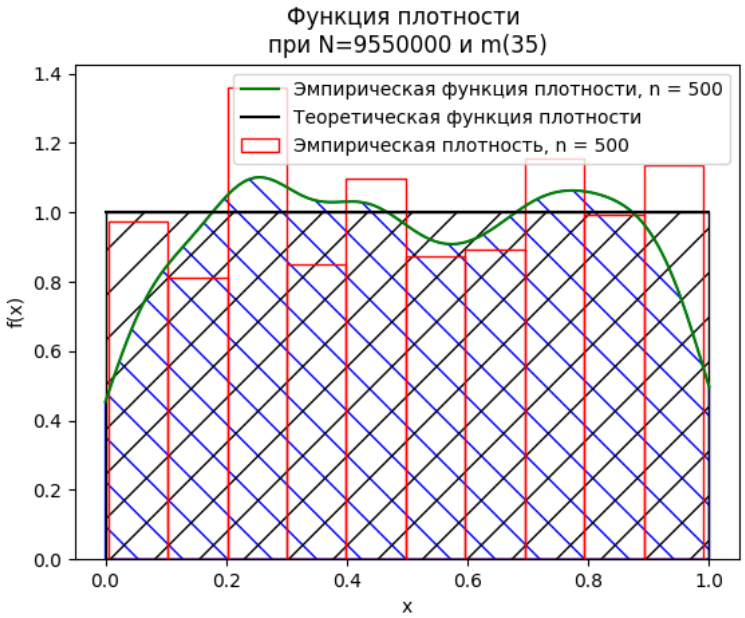

Рисунок 10 - Функция распределения при $N = 9.55 \cdot 10^6, R0 = m(35)$

```
При N=9660000 и R0=m(36)=157
Период формируемой случайной последовательности равен: 9659988
Среднее выборочное: 0.51
Выборочная дисперсия: 0.09
Выборочное стандартное отклонение: 0.29
Относительная погрешность по математическому ожиданию: 1.68%
Относительная погрешность по дисперсии: 4.40%
Относительная погрешность по стандартному отклонению: 2.18%
```

Рисунок 11 - Результат вычисления периода и погрешностей ГСЧ, созданного по линейному конгруэнтному методу при $N = 9.66 \cdot 10^6, R0 = m(36)$

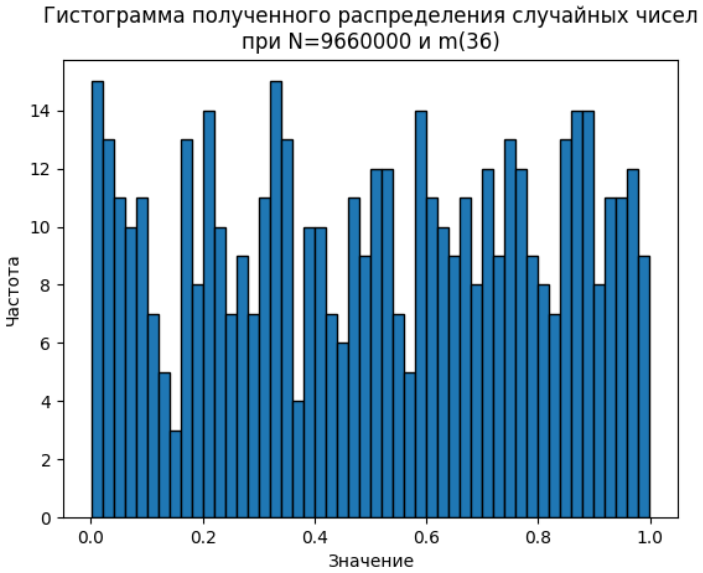

Рисунок 12 - Гистограмма распределения при $N = 9.66 \cdot 10^6, R0 = m(36)$

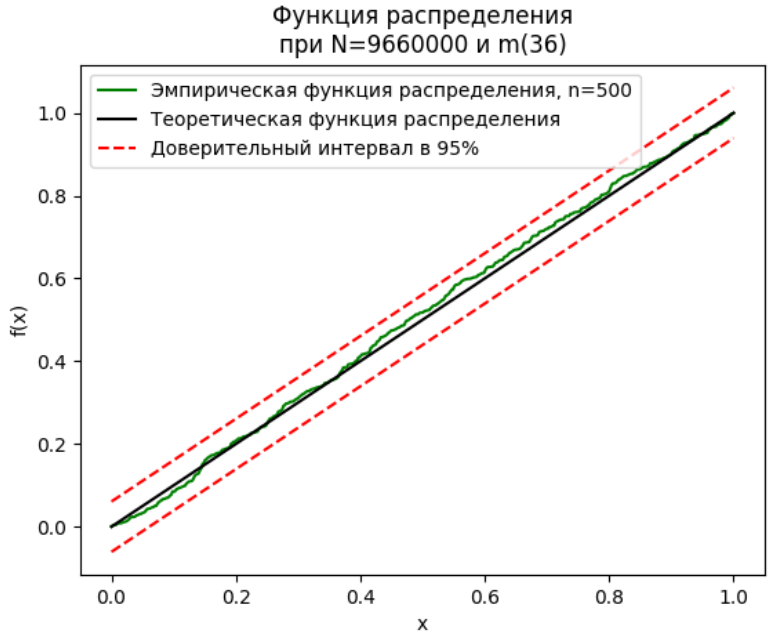

Рисунок 13 - Функция распределения при $N = 9.66 \cdot 10^6, R0 = m(36)$

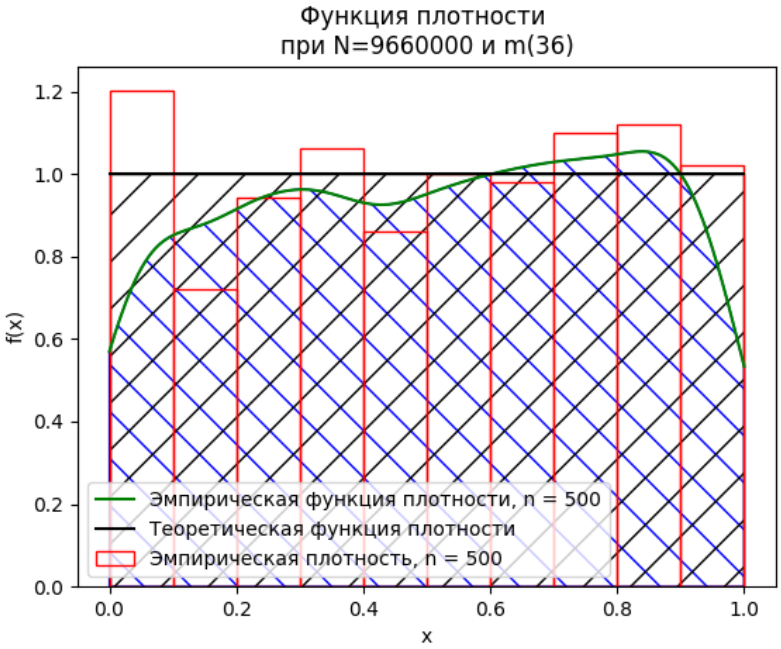

Рисунок 14 - Функция распределения при $N = 9.66 \cdot 10^6, R0 = m(36)$

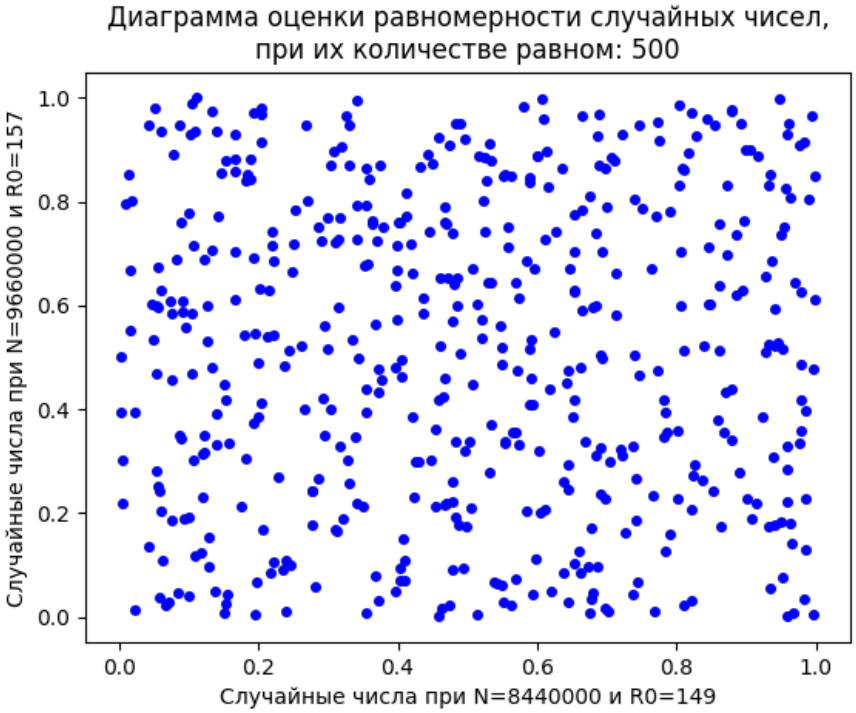

Рисунок 15 - Диаграмма оценки равномерности случайных чисел

Из графика выше (рис.15) можно сделать вывод, что разработанные ГСЧ работают корректно, так как точки равномерно распределены по диаграмме.

Задание 5.2

```
При размере выборки равном: 100
При экспоненциальном распределении и уровне значимости 0.05, нулевая гипотеза подтверждается: p = 1.5550790371345692e-07
При равномерном распределении и уровне значимости 0.05, нулевая гипотеза не подтверждается: p = 1.0
При нормальном распределении и уровне значимости 0.05, нулевая гипотеза подтверждается: p = 1.0245697148897385e-13

При размере выборки равном: 500
При экспоненциальном распределении и уровне значимости 0.05, нулевая гипотеза подтверждается: p = 1.4363212332187476e-27
При равномерном распределении и уровне значимости 0.05, нулевая гипотеза не подтверждается: p = 1.0
При нормальном распределении и уровне значимости 0.05, нулевая гипотеза подтверждается: p = 1.4765110719589765e-62

При размере выборки равном: 1000
При экспоненциальном распределении и уровне значимости 0.05, нулевая гипотеза подтверждается: p = 1.2295379067920271e-57
При равномерном распределении и уровне значимости 0.05, нулевая гипотеза не подтверждается: p = 1.0
При нормальном распределении и уровне значимости 0.05, нулевая гипотеза подтверждается: p = 2.9587730366111342e-123
```

Рисунок 16 - Проверка выборок по критерию Колмогорова – Смирнова

Из результата вычислений выше можно сделать вывод, что с увеличением выборки идёт увеличение точности проверки по критерию Колмогорова–Смирнова. Кроме того, т.к. нулевая гипотеза, гласящая о том, что 2 выборки берутся не из одного распределения вероятности, не подтверждается только при равномерном распределении, то можно сделать вывод что ГСЧ, созданный по линейному конгруэнтному методу, выдаёт равномерное распределение чисел.

Вывод: в ходе выполнения лабораторной работы были сформированы практические навыки оценки качества генераторов случайных чисел (ГСЧ) в различных системах программирования по заданным теоретическим показателям, с помощью критериев согласия и с помощью нормированной автокорреляционной функции на предмет независимости случайных чисел.

ПРИЛОЖЕНИЯ

Листинг программы

Задание 2

```
import math
import typing

import numpy as np
import matplotlib.pyplot as plt


def is_prime(n: typing.Union[int, float]) -> bool:
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True


def generate_prime_numbers() -> list[int]:
    primes = [num for num in range(100, 1000) if is_prime(num)]
    return primes


def calculate_errors(
        data: np.array,
        expected_mean: float,
        expected_var: float,
        expected_std_dev: float
) -> tuple:
    mean = np.mean(data)
    variance = np.var(data)
    std_dev = np.std(data)

    relative_error_mean = abs(mean - expected_mean) / expected_mean
    relative_error_var = abs(variance - expected_var) / expected_var
    relative_error_std_dev = abs(std_dev - expected_std_dev) /
expected_std_dev

    return relative_error_mean, relative_error_var,
relative_error_std_dev


if __name__ == '__main__':
    prime_numbers = generate_prime_numbers()

    print(f"Среднее выборочное: {np.mean(prime_numbers):.2f}")
    print(f"Выборочная дисперсия: {np.var(prime_numbers):.2f}")
    print(f"Выборочное стандартное отклонение:
{np.std(prime_numbers):.2f}")
```

```
    print()

    expected_mean = 550
    expected_var = 67500
    expected_std_dev = math.sqrt(expected_var)

    relative_errors = calculate_errors(
        prime_numbers,
        expected_mean,
        expected_var,
        expected_std_dev
    )

    print(
        f"Относительная погрешность по математическому ожиданию: "
        f"{relative_errors[0] * 100:.2f}%"
    )
    print(
        f"Относительная погрешность по дисперсии: "
        f"{relative_errors[1] * 100:.2f}%"
    )
    print(
        f"Относительная погрешность по стандартному отклонению: "
        f"{relative_errors[2] * 100:.2f}%"
    )
    Zx = np.random.choice(prime_numbers, size=100)
    Zy = np.random.choice(prime_numbers, size=100)

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.hist(Zx, bins=20, color='skyblue', edgecolor='black')
    plt.title('Гистограмма Zx')
    plt.xlabel('Значения')
    plt.ylabel('Частота')

    plt.subplot(1, 2, 2)
    plt.hist(Zy, bins=20, color='salmon', edgecolor='black')
    plt.title('Гистограмма Zy')
    plt.xlabel('Значения')
    plt.ylabel('Частота')

    plt.tight_layout()
    plt.show()
```

Задание 4.3

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

```
class LinearCongruentialGenerator:
    @classmethod
    def generate_primes(cls, n: int) -> list[int]:
        sieve = [True] * n
        for i in range(3, int(n ** 0.5) + 1, 2):
            if sieve[i]:
                sieve[i * i::2 * i] = [False] * ((n - i * i - 1) // (2
* i) + 1)
        return [2] + [i for i in range(3, n, 2) if sieve[i]]

    def __find_M(self, N):
        m = self.generate_primes(N)
        M = m[-1]
        return M

    def __find_A(self, M):
        a = int(M * (1 / 2 - math.sqrt(3) / 6))
        if a % 2 == 0:
            a += 1
        if a % 8 == 5:
            if math.gcd(a, M) == 1:
                return a
        i = 1
        while True:
            a1 = a + 2 * i
            if a1 % 8 == 5:
                if math.gcd(a1, M) == 1:
                    if a1 < (M - math.sqrt(M)):
                        return a1
            a2 = a - 2 * i
            if a2 % 8 == 5:
                if math.gcd(a2, M) == 1:
                    if a2 > (M // 100):
                        return a2
            i += 1

    def __init__(self, R0, N, c=0):
        self.__c = int(c)
        self.__Rk = int(R0)
        M = self.__find_M(N)
        self.__M = int(M)
        a = self.__find_A(M)
        self.__a = a

    def generate_int_number(self):
        number = (self.__a * self.__Rk + self.__c) % self.__M
        self.__Rk = number
        return number

    def generate_float_number(self):
        number = (self.__a * self.__Rk + self.__c) % self.__M
        self.__Rk = number
        number /= self.__M
        return number
```

```
    def generate_int_number_array(self, count):
        array = []
        for i in range(count):
            number = (self.__a * self.__Rk + self.__c) % self.__M
            self.__Rk = number
            array.append(number)
        return array

    def generate_float_number_array(self, count):
        array = []
        for i in range(count):
            number = (self.__a * self.__Rk + self.__c) % self.__M
            self.__Rk = number
            number /= self.__M
            array.append(number)
        return array


def print_period_sequence(N, order):
    m = LinearCongruentialGenerator.generate_primes(N)
    R0 = m[order]
    rnd = LinearCongruentialGenerator(R0, N)
    base = rnd.generate_int_number()
    i = 0
    while True:
        i += 1
        if rnd.generate_int_number() == base:
            break
    print(
        f"При N={N} и R0=m({order})={R0}\nПериод формируемой случайной
последовательности равен: {i}")


def print_distribution_parameters(N: int, order: int):
    m = LinearCongruentialGenerator.generate_primes(N)
    R0 = m[order]
    rnd = LinearCongruentialGenerator(R0, N)
    x = rnd.generate_float_number_array(500)
    mf = np.mean(x)
    print("Среднее выборочное: %f" % mf)
    sf2 = np.var(x)
    print("Выборочная дисперсия: %f" % sf2)
    sf = np.std(x)
    print("Выборочное стандартное отклонение: %f" % sf)
    m = 0.5
    Dm = abs((mf - m) / m) * 100
    print("Относительная  погрешность  по  математическому  ожиданию:
%f%%" % Dm)
    d = 1 / 12
    Dd = abs((sf2 - d) / d) * 100
    print("Относительная погрешность по дисперсии: %f%%" % Dd)
    sd = np.sqrt(d)
    Ds = abs((sf - sd) / sd) * 100
    print("Относительная погрешность по стандартному отклонению: %f%%"
% Ds)
```

```
def generate_histogram(N: int, order: int, size: int):
    m = LinearCongruentialGenerator.generate_primes(N)
    R0 = m[order]
    rnd = LinearCongruentialGenerator(R0, N)
    x = rnd.generate_float_number_array(size)
    plt.hist(x, bins=int(size / 10), edgecolor='black', linewidth=1)
    plt.title(f"Гистограмма полученного распределения случайных
чисел\n"
              f"при {N=} и m({order})")
    plt.xlabel("Значение")
    plt.ylabel("Частота")
    plt.show()


def generate_cdf(N, order, size):
    m = LinearCongruentialGenerator.generate_primes(N)
    R0 = m[order]
    rnd = LinearCongruentialGenerator(R0, N)
    a = 0
    b = 1
    n = size
    x = np.linspace(a, b, n)
    y = sorted(rnd.generate_float_number_array(n))
    plt.plot(
        x, y, "-g",
        label=f"Эмпирическая функция распределения, n={len(x)}"
    )
    y = (x - a) / (b - a)
    plt.plot(x, y, "-k", label="Теоретическая функция распределения")
    y = x - 1.36 / len(x) ** (0.5)
    plt.plot(x, y, "--r", label="Доверительный интервал в 95%")
    y = x + 1.36 / len(x) ** (0.5)
    plt.plot(x, y, "--r")
    plt.title("Функция распределения\n"
              f"при {N=} и m({order})")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.legend()
    plt.show()


def generate_pdf(N: int, order: int, size: int):
    m = LinearCongruentialGenerator.generate_primes(N)
    R0 = m[order]
    rnd = LinearCongruentialGenerator(R0, N)
    a = 0
    b = 1
    n = size
    x = np.linspace(a, b, n)
    y = rnd.generate_float_number_array(n)
    plt.hist(
        y, density=True, fc="none", ec="red",
        label="Эмпирическая плотность, n = %i" % len(x)
    )
    density = gaussian_kde(y)
    density.covariance_factor = lambda: .25
    density._compute_covariance()
```

```
    plt.plot(
        x, density(x), "-g",
        label="Эмпирическая функция плотности, n = %i" % len(x)
    )
    plt.fill_between(
        x, 0, density(x), color="none", hatch='\\',
        edgecolor="b"
    )
    y = x * 0 + 1 / (b - a)
    plt.plot(x, y, "-k", label="Теоретическая функция плотности")
    plt.fill_between(x, 0, y, color="none", hatch="/", edgecolor="k")
    plt.title("Функция плотности \n"
              f"при {N=} и m({order})")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.legend()
    plt.show()


def plot_uniformity_diagram(
        first_N: float,
        first_order: int,
        second_N: float,
        second_order: int,
        count: int
):
    first_random = LinearCongruentialGenerator(first_order, first_N)
    second_random      =      LinearCongruentialGenerator(second_order,
second_N)
    x = first_random.generate_float_number_array(count)
    y = second_random.generate_float_number_array(count)

    plt.plot(x, y, 'bo', markersize=4)
    plt.title(
        "Диаграмма оценки равномерности случайных чисел,"
        f"\nпри их количестве равном: {count}"
    )
    plt.xlabel(f"Случайные числа при N={first_N} и R0={first_order}")
    plt.ylabel(f"Случайные      числа      при      N={second_N}      и
R0={second_order}")
    plt.show()


if __name__ == '__main__':
    Ns = [int(8.44 * 10 ** 6), int(9.55 * 10 ** 6), int(9.66 * 10 **
6)]
    orders = [34, 35, 36]
    print()

    print_period_sequence(Ns[0], orders[0])
    print_distribution_parameters(Ns[0], orders[0])
    generate_histogram(Ns[0], orders[0], 500)
    generate_cdf(Ns[0], orders[0], 500)
    generate_pdf(Ns[0], orders[0], 500)
    print()

    print_period_sequence(Ns[1], orders[1])
```

```
    print_distribution_parameters(Ns[1], orders[1])
    generate_histogram(Ns[1], orders[1], 500)
    generate_cdf(Ns[1], orders[1], 500)
    generate_pdf(Ns[1], orders[1], 500)
    print()

    print_period_sequence(Ns[2], orders[2])
    print_distribution_parameters(Ns[2], orders[2])
    generate_histogram(Ns[2], orders[2], 500)
    generate_cdf(Ns[2], orders[2], 500)
    generate_pdf(Ns[2], orders[2], 500)
    print()

    plot_uniformity_diagram(
        Ns[0],
        LinearCongruentialGenerator.generate_primes(250)[orders[0]],
        Ns[2],
        LinearCongruentialGenerator.generate_primes(250)[orders[2]],
        500
    )
    print()
```

Задание 5.2

```
import math
import scipy.stats as sps


class LinearCongruentialGenerator:
    @classmethod
    def generate_primes(cls, n: int) -> list[int]:
        sieve = [True] * n
        for i in range(3, int(n ** 0.5) + 1, 2):
            if sieve[i]:
                sieve[i * i::2 * i] = [False] * ((n - i * i - 1) // (2
* i) + 1)
        return [2] + [i for i in range(3, n, 2) if sieve[i]]

    def __find_M(self, N):
        m = self.generate_primes(N)
        M = m[-1]
        return M

    def __find_A(self, M):
        a = int(M * (1 / 2 - math.sqrt(3) / 6))
        if a % 2 == 0:
            a += 1
        if a % 8 == 5:
            if math.gcd(a, M) == 1:
                return a
        i = 1
        while True:
            a1 = a + 2 * i
            if a1 % 8 == 5:
                if math.gcd(a1, M) == 1:
                    if a1 < (M - math.sqrt(M)):
```

```
                    return a1
            a2 = a - 2 * i
            if a2 % 8 == 5:
                if math.gcd(a2, M) == 1:
                    if a2 > (M // 100):
                        return a2
            i += 1

    def __init__(self, R0, N, c=0):
        self.__c = int(c)
        self.__Rk = int(R0)
        M = self.__find_M(N)
        self.__M = int(M)
        a = self.__find_A(M)
        self.__a = a

    def generate_int_number(self):
        number = (self.__a * self.__Rk + self.__c) % self.__M
        self.__Rk = number
        return number

    def generate_float_number(self):
        number = (self.__a * self.__Rk + self.__c) % self.__M
        self.__Rk = number
        number /= self.__M
        return number

    def generate_int_number_array(self, count):
        array = []
        for i in range(count):
            number = (self.__a * self.__Rk + self.__c) % self.__M
            self.__Rk = number
            array.append(number)
        return array

    def generate_float_number_array(self, count):
        array = []
        for i in range(count):
            number = (self.__a * self.__Rk + self.__c) % self.__M
            self.__Rk = number
            number /= self.__M
            array.append(number)
        return array


if __name__ == '__main__':
    counts = [100, 500, 1000]
    for count in counts:
        print(f"При размере выборки равном: {count}")
        rnd = LinearCongruentialGenerator(95, int(8.44 * 10 ** 6))
        x = rnd.generate_float_number_array(count)
        F0 = sps.expon.cdf(x)
        H0 = sps.kstest(x, F0, N=count)[1]
        if H0 < 0.95:
            print(
                f"При экспоненциальном распределении и уровне
значимости 0.05, "
```

```
                f"нулевая гипотеза подтверждается: p = {H0}")
        else:
            print(
                f"При экспоненциальном распределении и уровне
значимости 0.05, "
                f"нулевая гипотеза не подтверждается: p = {H0}")
        F1 = sps.uniform.cdf(x)
        H1 = sps.kstest(x, F1, N=count)[1]
        if H1 < 0.95:
            print(f"При равномерном распределении и уровне значимости
0.05, "
                  f"нулевая гипотеза подтверждается: p = {H1}")
        else:
            print(f"При равномерном распределении и уровне значимости
0.05, "
                  f"нулевая гипотеза не подтверждается: p = {H1}")
        F2 = sps.norm.cdf(x)
        H2 = sps.kstest(x, F2, N=count)[1]
        if H2 < 0.95:
            print(f"При нормальном распределении и уровне значимости
0.05, "
                  f"нулевая гипотеза подтверждается: p = {H2}")
        else:
            print(f"При нормальном распределении и уровне значимости
0.05, "
                  f"нулевая гипотеза не подтверждается: p = {H2}")
        print()
```