|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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.

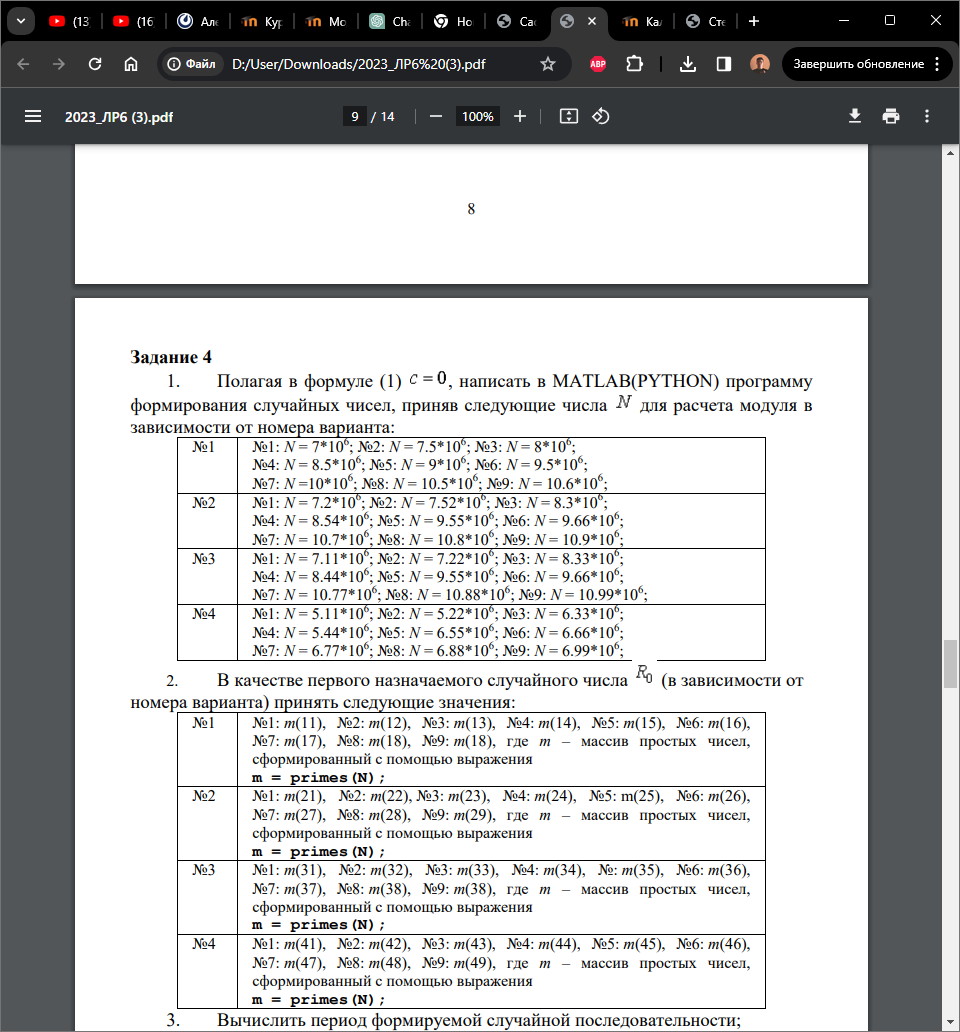
**Задание 2**

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.

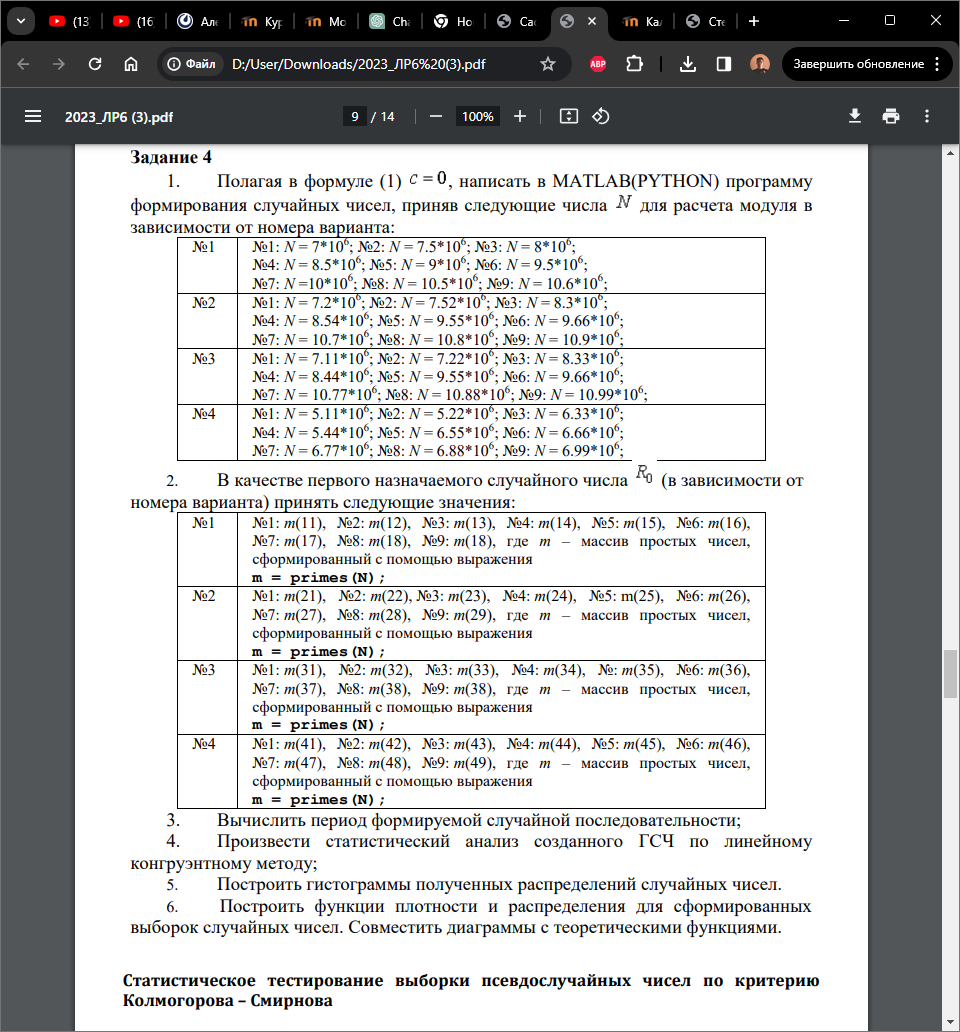
2. Построить гистограммы для сформированных выборок (Zx и Zy) с разбивкой графического окна.

**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:



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



3. Вычислить период формируемой случайной последовательности;

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

5. Построить гистограммы полученных распределений случайных чисел.

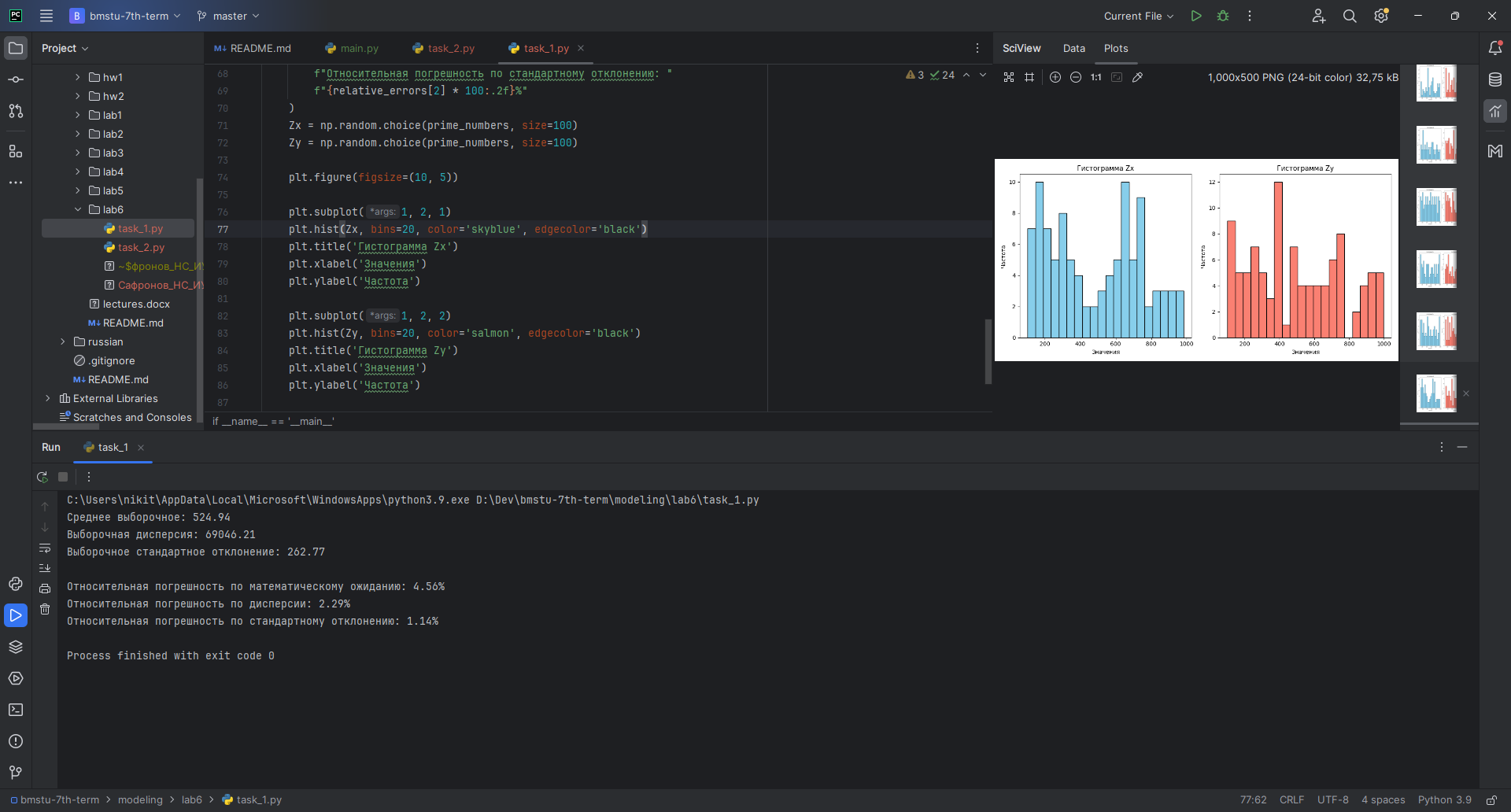
6. Построить функции плотности и распределения для сформированных выборок случайных чисел. Совместить диаграммы с теоретическими функциями.

**Задание 5.2**.

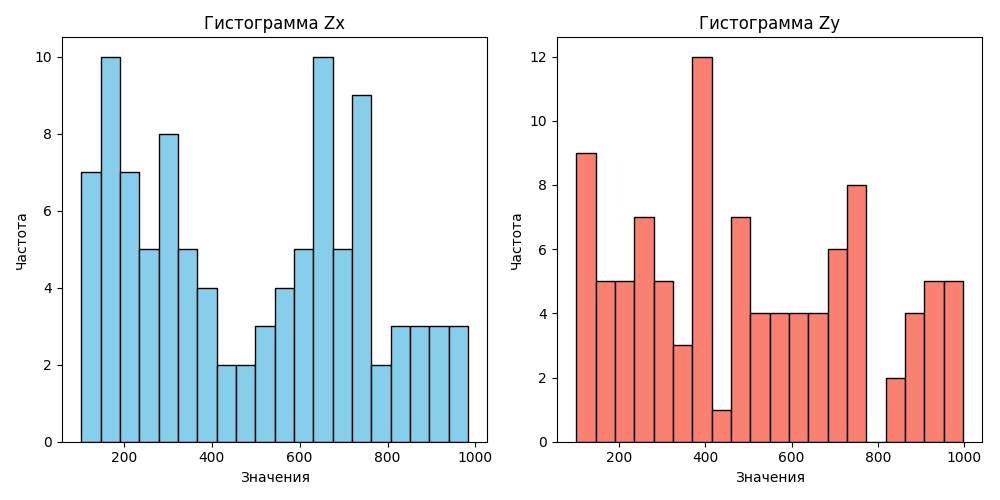
По критерию Колмогорова – Смирнова протестировать выборки случайных чисел объема 100, 500, 1000, сформированных по линейному конгруэнтному методу.

**Ход выполнения работы**

**Задание 2**

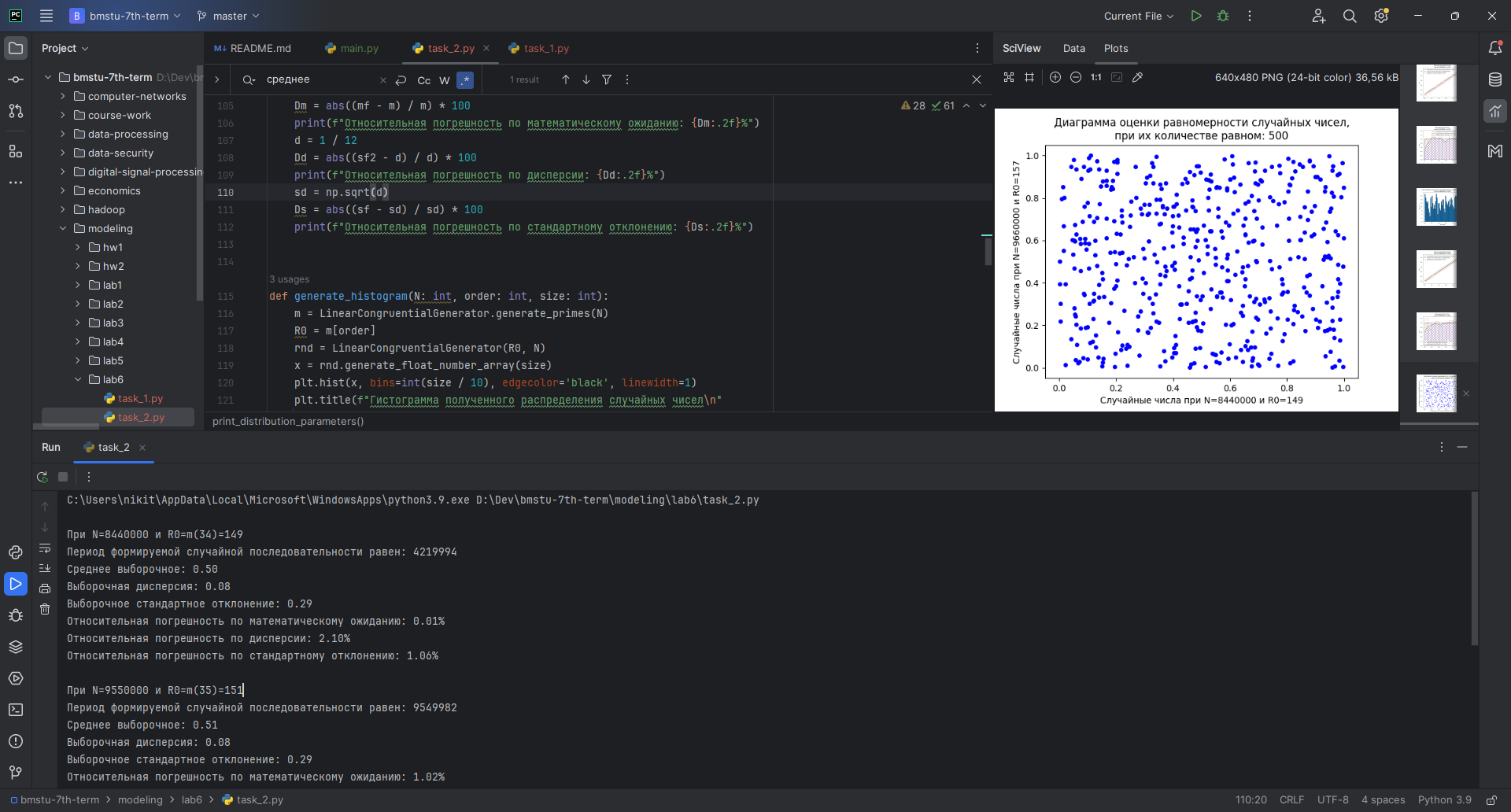


**Рисунок 1 –** Результаты расчёта относительных погрешностей по математическому ожиданию, дисперсии и стандартному отклонению

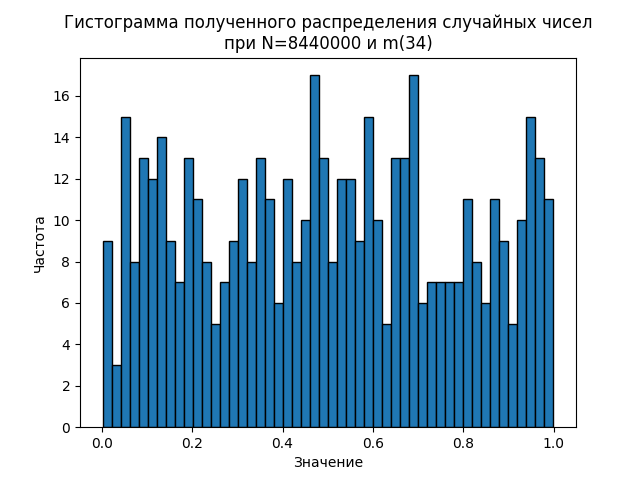


**Рисунок 2 -** Гистограммы для сформированных выборок Zx и Zy

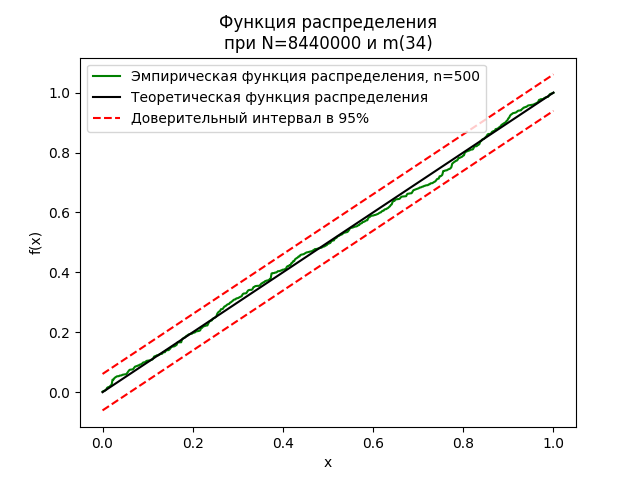
**Задание 4.3**



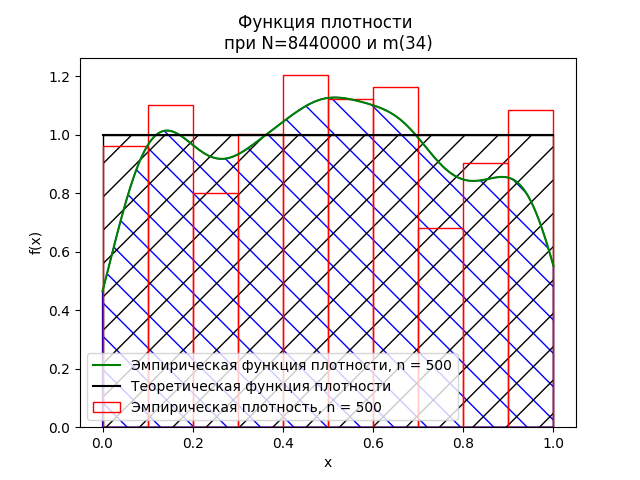
**Рисунок 3 -** Результат вычисления периода и погрешностей ГСЧ, созданного по линейному конгруэнтному методу при



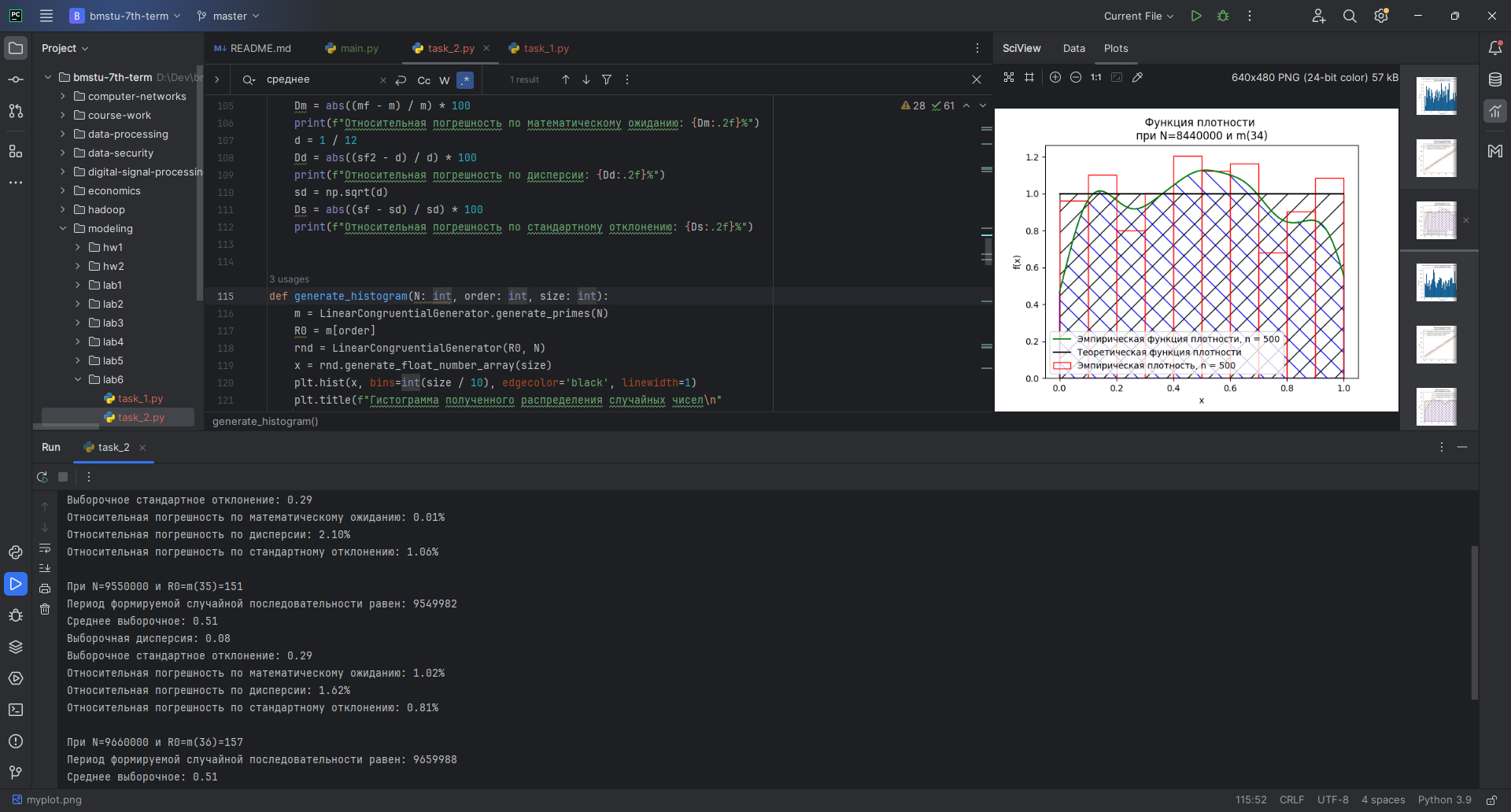
**Рисунок 4 -** Гистограмма распределения при



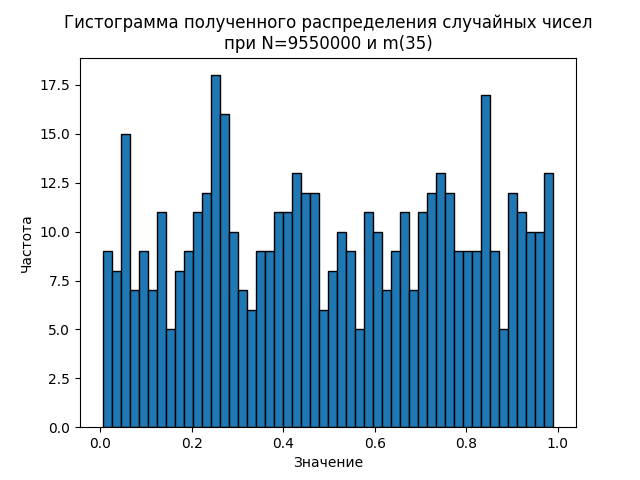
**Рисунок 5 -** Функция распределения при



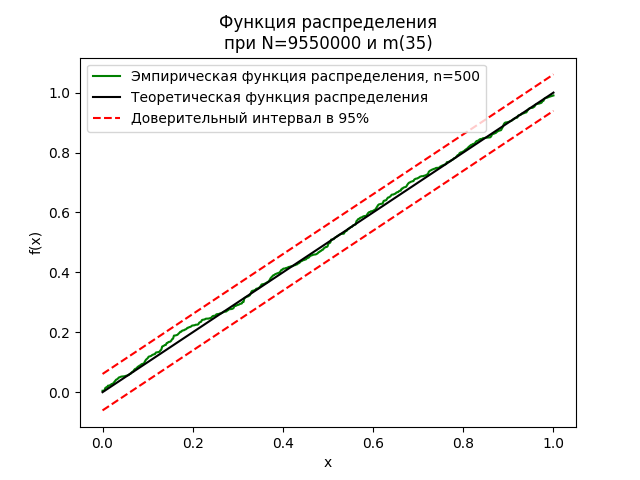
**Рисунок 6 -** Функция распределения при



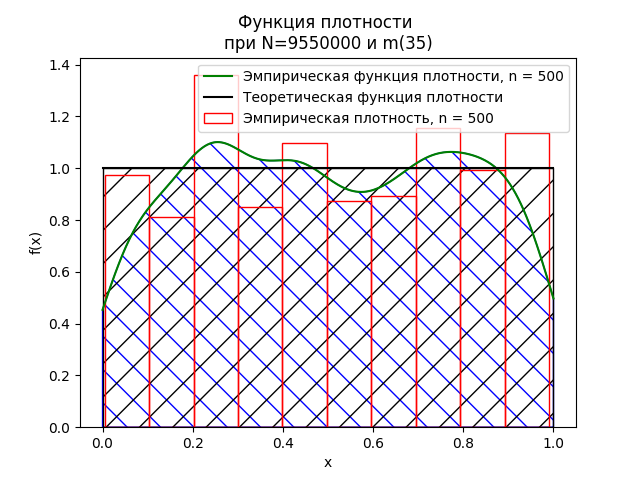
**Рисунок 7 -** Результат вычисления периода и погрешностей ГСЧ, созданного по линейному конгруэнтному методу при



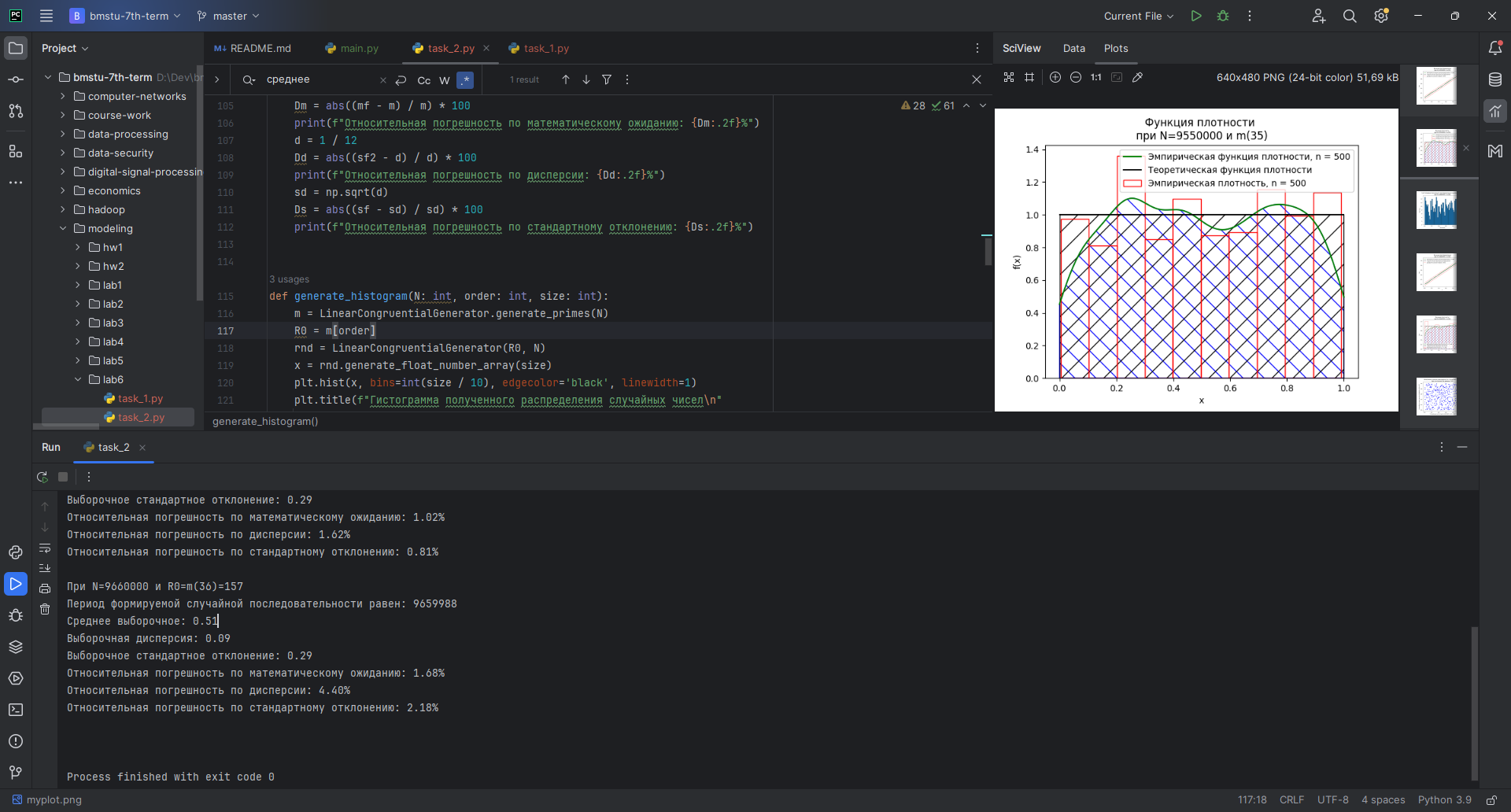
**Рисунок 8 -** Гистограмма распределения при



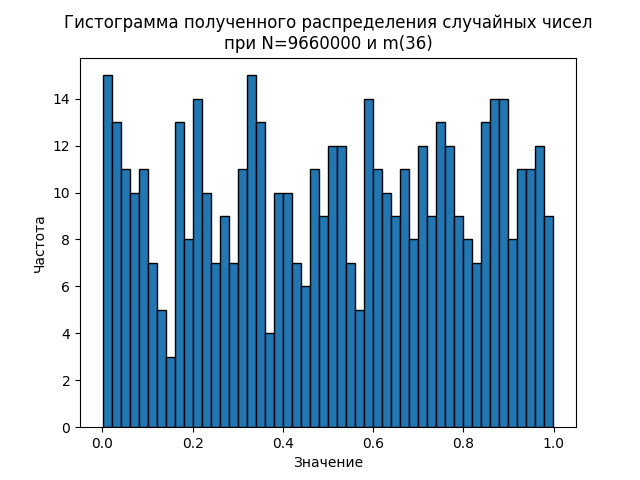
**Рисунок 9 -** Функция распределения при



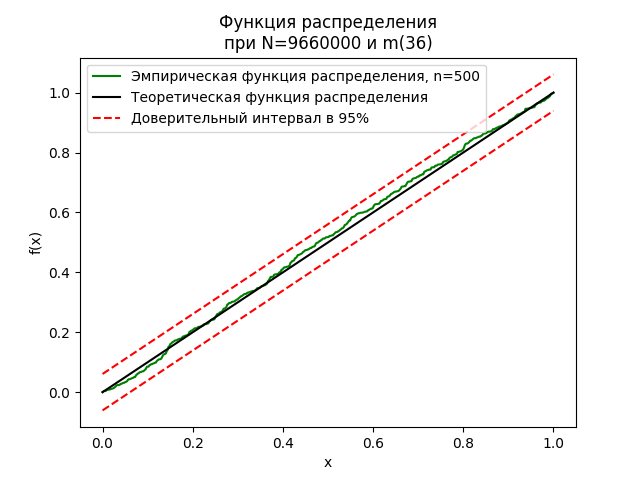
**Рисунок 10 -** Функция распределения при



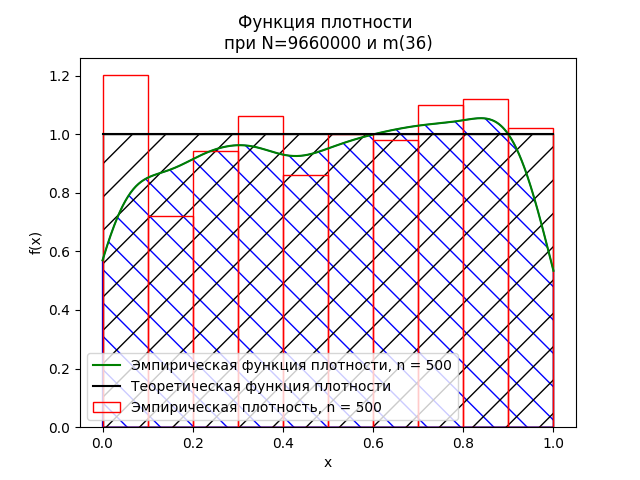
**Рисунок 11 -** Результат вычисления периода и погрешностей ГСЧ, созданного по линейному конгруэнтному методу при



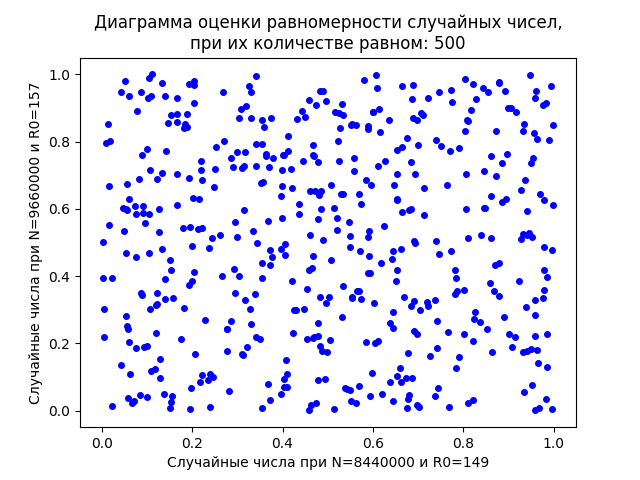
**Рисунок 12 -** Гистограмма распределения при



**Рисунок 13 -** Функция распределения при



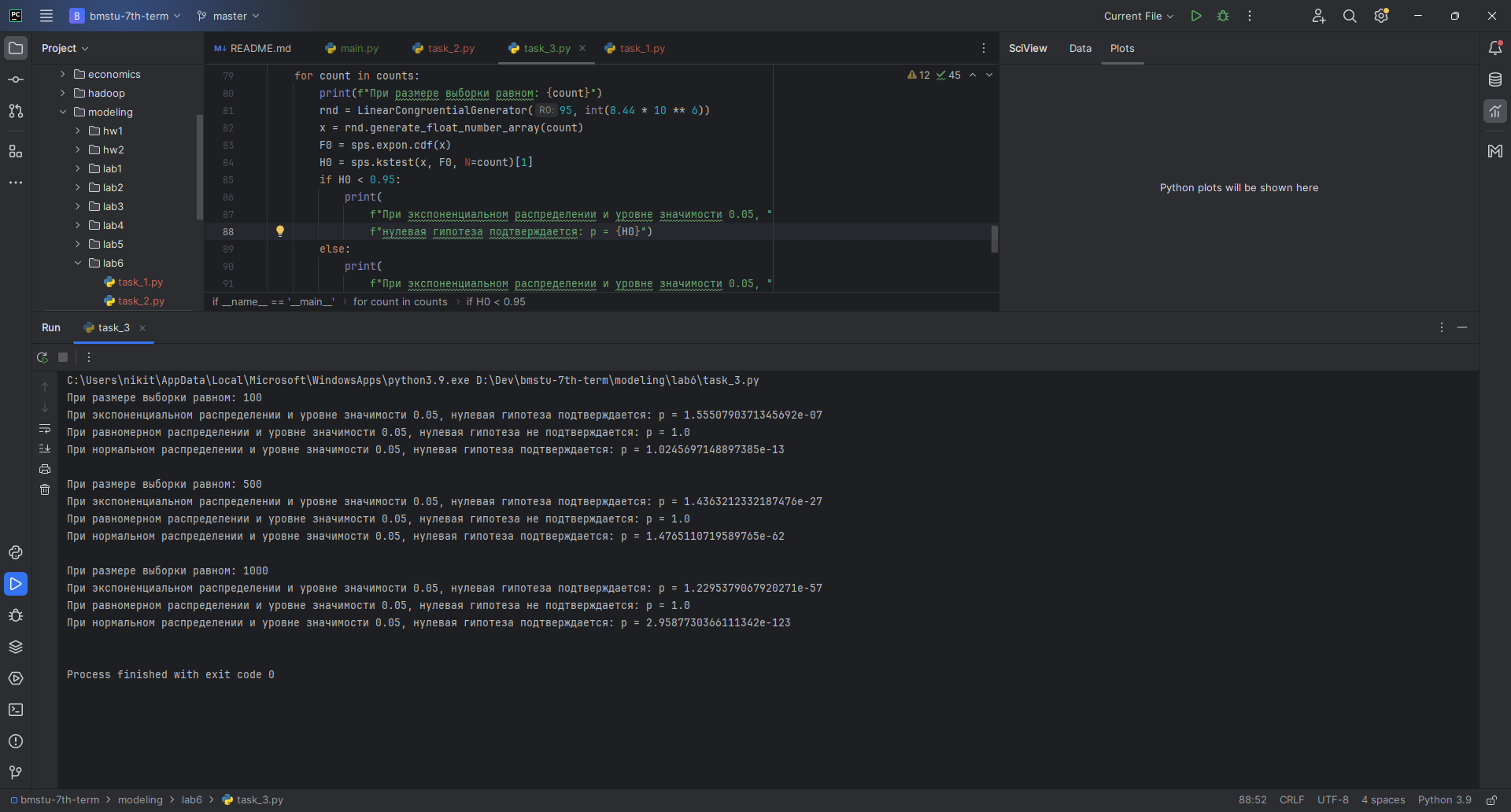
**Рисунок 14 -** Функция распределения при



**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**



**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()