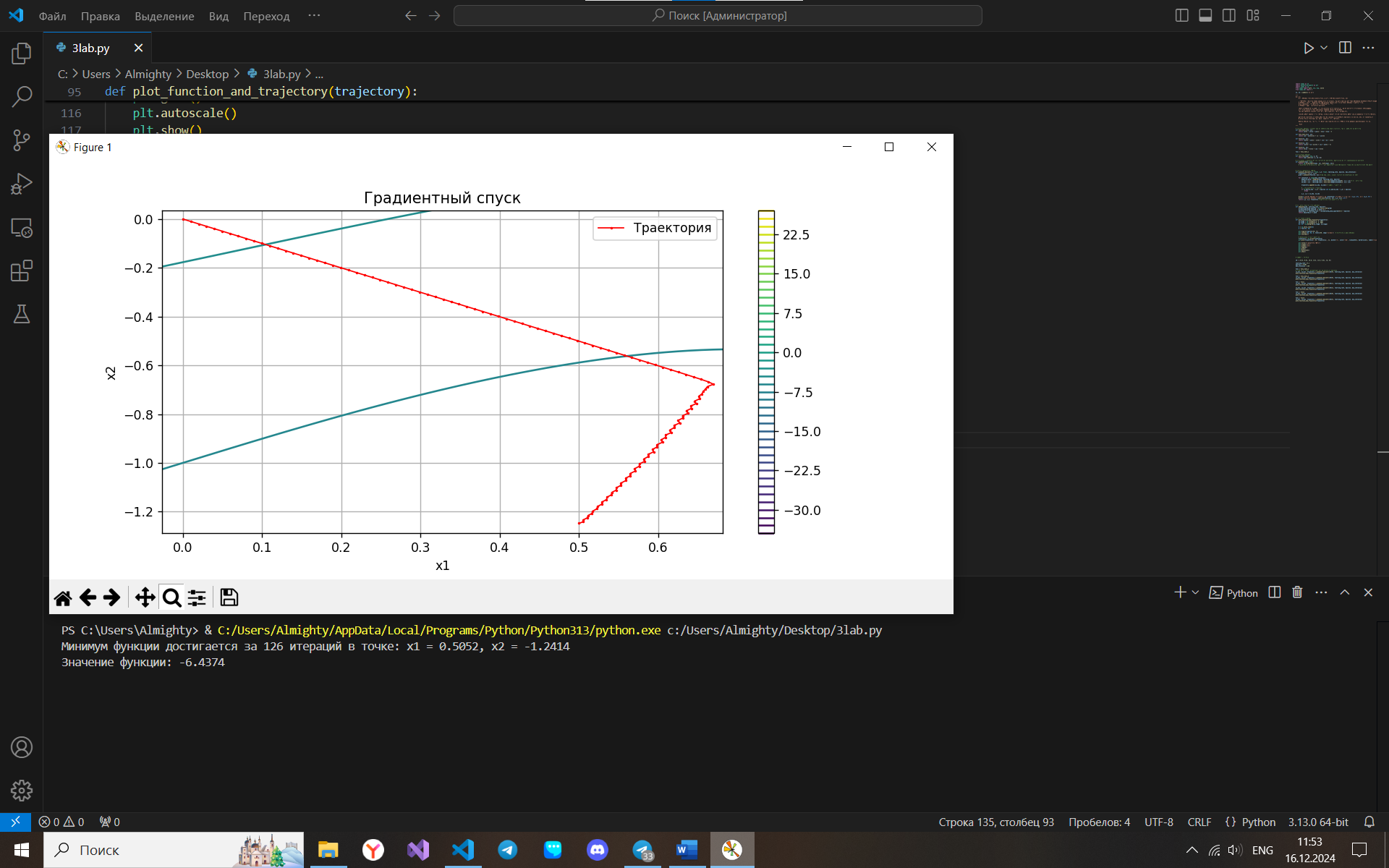
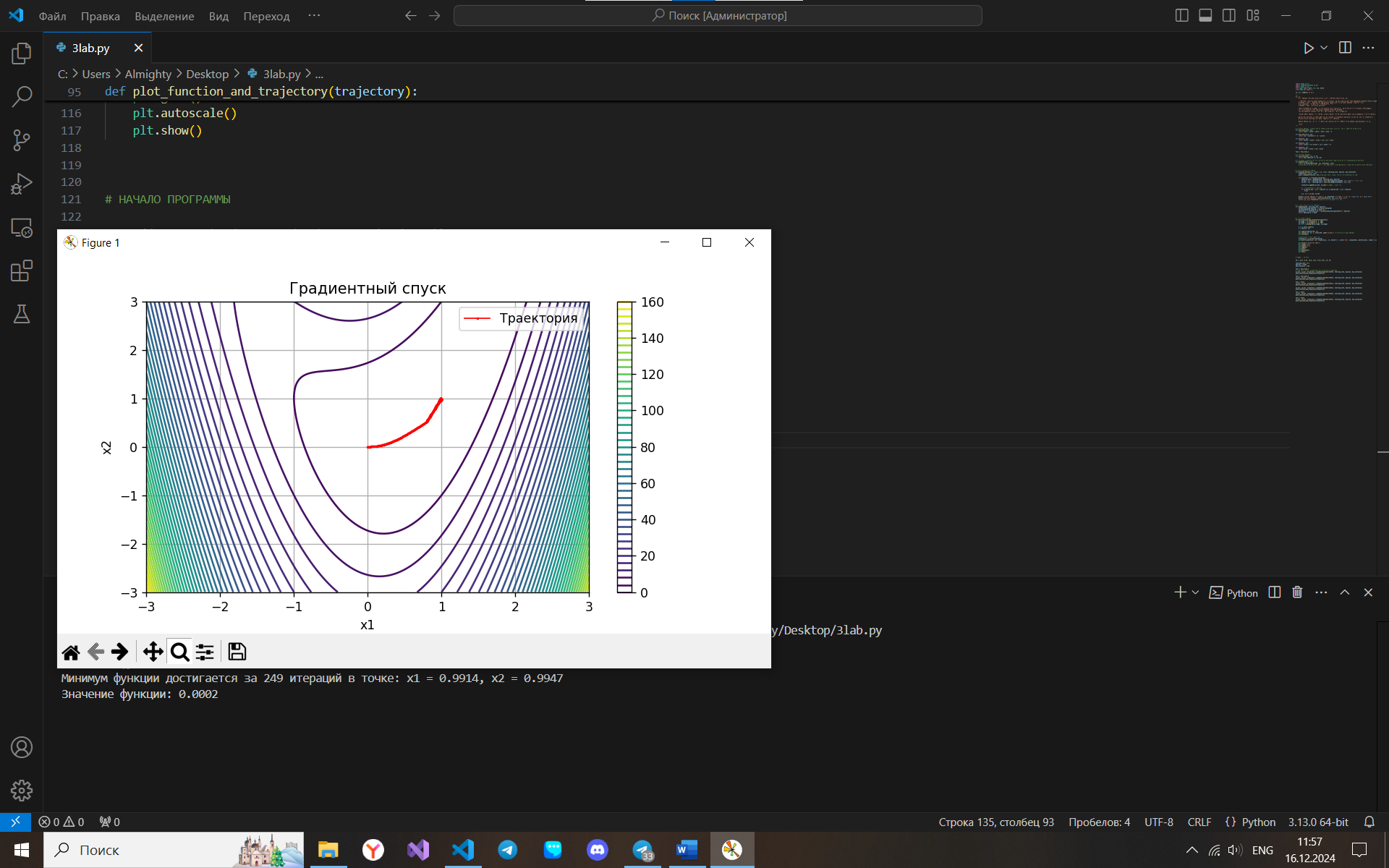
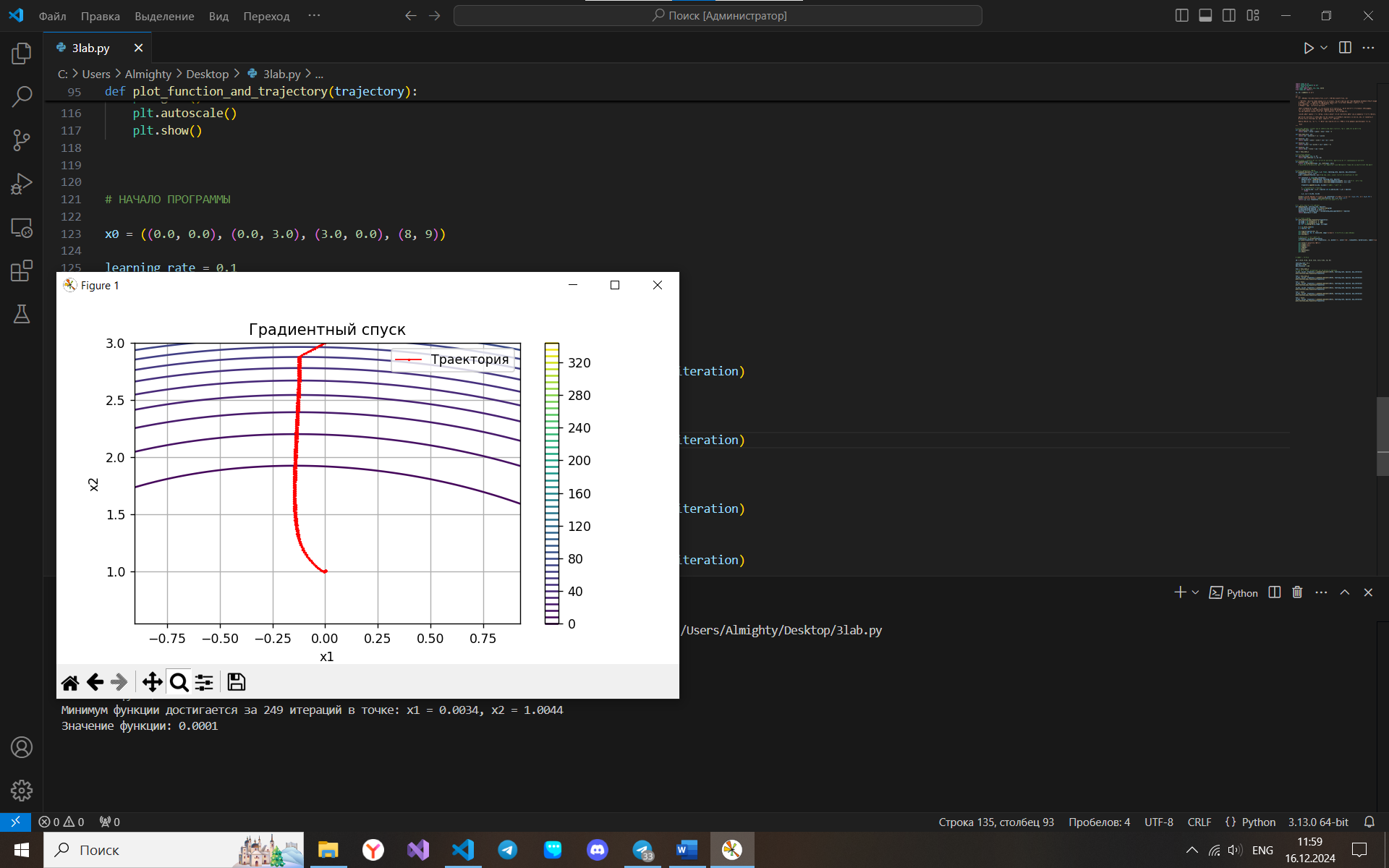
**Исследование численных методов оптимизации**

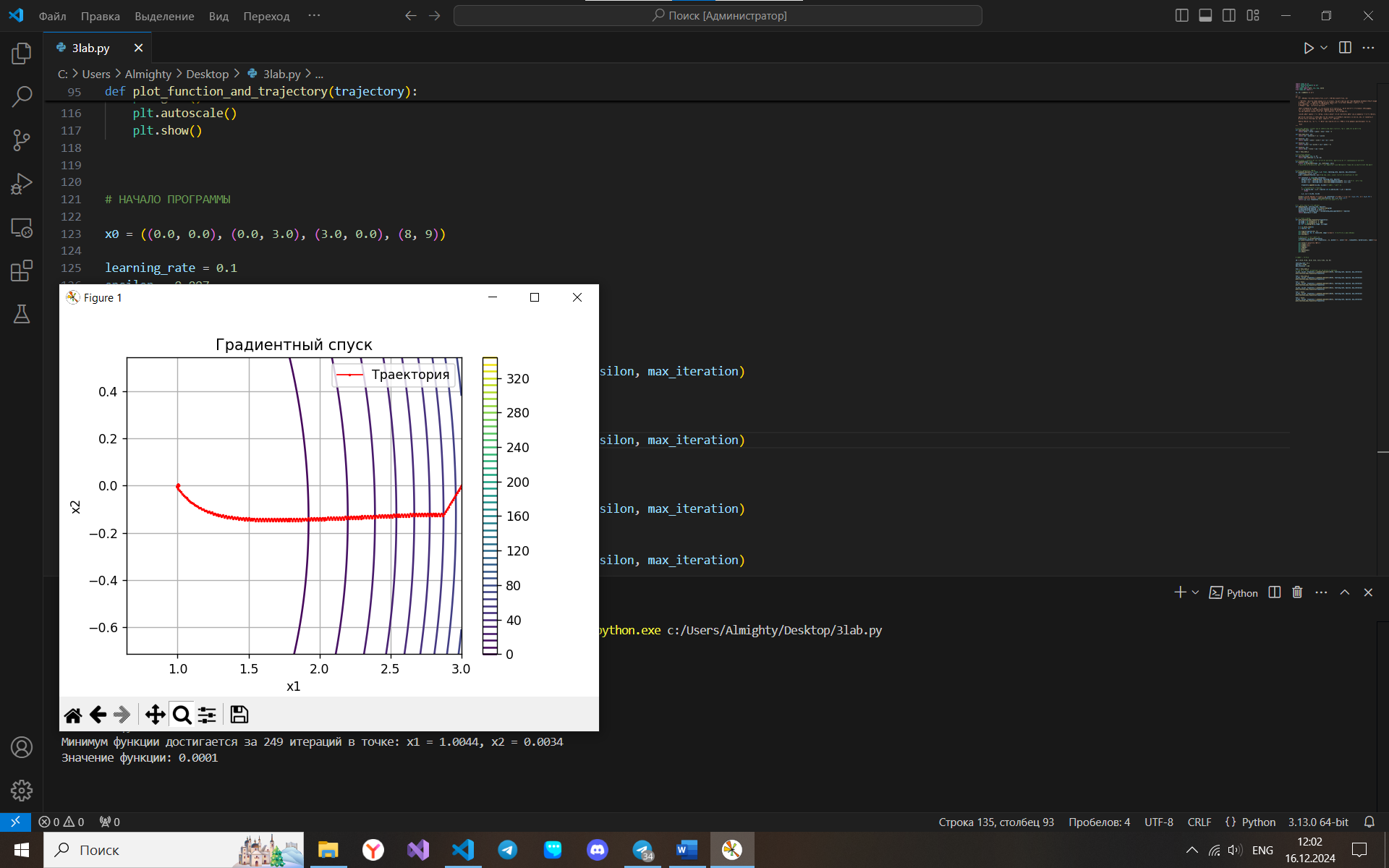
Реализовать программно метод градиентного спуска. Реализация должна предполагать останов при достижении заданного количества шагов, останов по величине ɛ, регулировать величину шага, задание начальной точки.

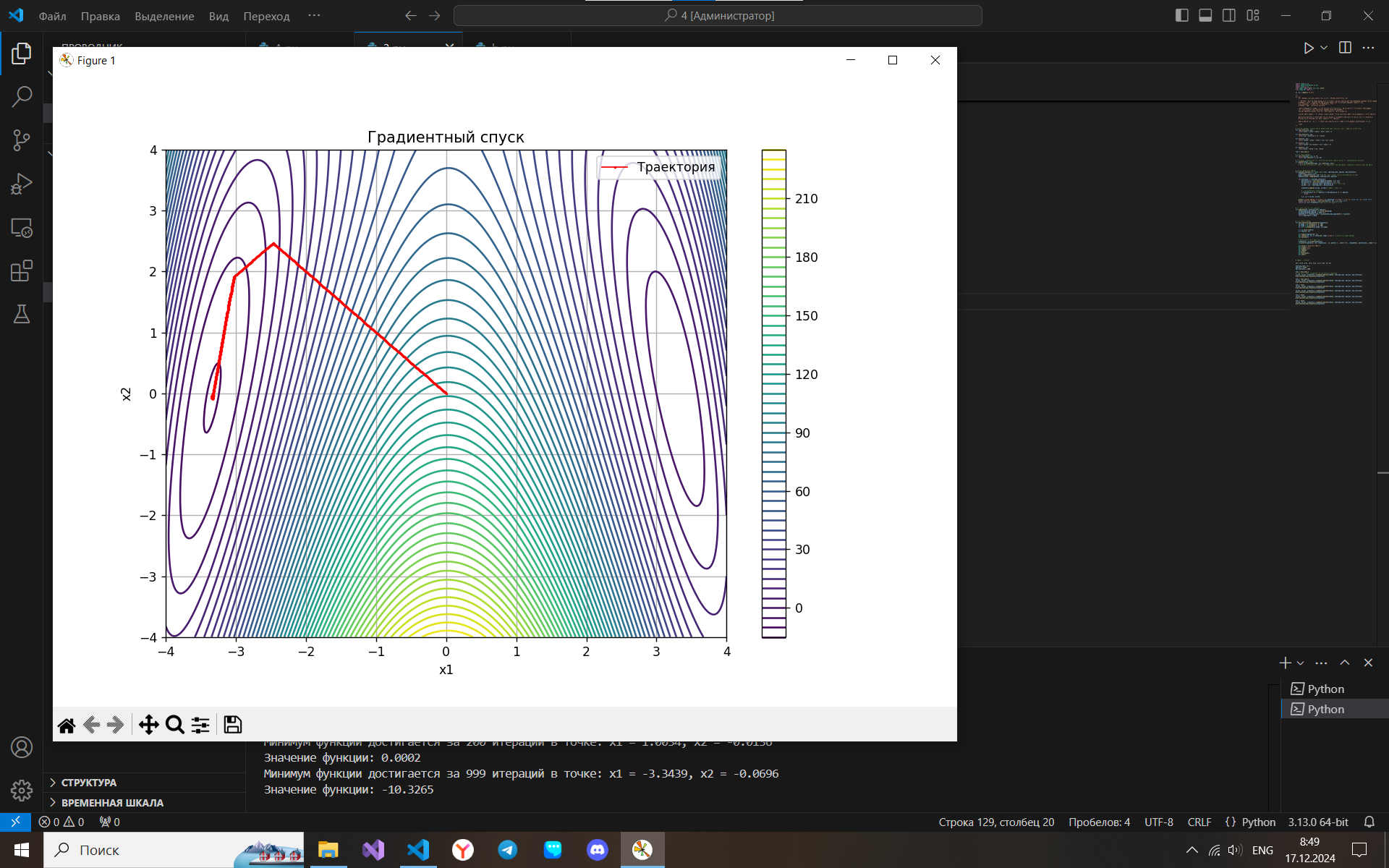
Провести исследование влияния параметров метода на результат (количество итераций, величина параметров, точность, задание начальной точки).











import numpy as np

import matplotlib.pyplot as plt

from sympy import diff, exp

from sympy.abc import \*

x1, x2 = symbols('x1 x2')

def func\_test\_1(x1, x2):

    return x1\*\*3 - x1\*x2 + x2\*\*2 - 2\*x1 + 3\*x2 - 4

def func\_test\_2(x1, x2):

    return (x2 - x1\*\*2)\*\*2 + (1 - x1)\*\*2

def func1(x1, x2):

    return (x2\*\*2 + x1\*\*2 - 1)\*\*2 + (x1 + x2 - 1)\*\*2

def func2(x1, x2):

    return (x1\*\*2 + x2 -11)\*\*2 + (x1 + x2\*\*2 - 7)

def func3(x1, x2):

    return 4\*(x1 - 5)\*\*2 + (x2 - 6)\*\*2

func = func\_test\_1

# вычисление функции

def calc\_func(expr: exp, a, b):

    return expr.subs({x1: a, x2: b})

# вычисление градиента (ФОРМУЛ частных производных), фактически просто объединение производных

def gradient(expr: exp):

    return np.array([diff(expr, x1), diff(expr, x2)])

# метод градиентного спуска

def gradient\_descent(x\_1: float, x\_2: float, learning\_rate, epsilon, max\_iteration):

    trajectory = [(x\_1, x\_2)]

    grad = gradient(func(x1, x2

    adaptive\_grad = adagrad(grad, learning\_rate, epsilon)

    for iteration in range(max\_iteration):

        derivative\_x1 = calc\_func(adaptive\_grad[0], x\_1, x\_2)

        derivative\_x2 = calc\_func(adaptive\_grad[1], x\_1, x\_2)

        x\_1 -= learning\_rate \* derivative\_x1 # корректировка

        x\_2 -= learning\_rate \* derivative\_x2

        trajectory.append((x\_1, x\_2)) # запись траектории

        if derivative\_x1 \*\* 2 < epsilon and derivative\_x2 \*\* 2 < epsilon:

            break

    print(f'Минимум функции достигается за {iteration} итераций в точке: x1 = {x\_1:.4f}, x2 = {x\_2:.4f}')

    print(f'Значение функции: {calc\_func(func(x1, x2), x\_1, x\_2):.4f}')

    return x\_1, x\_2, trajectory # фактически возвращается кортеж

# корректировка скорости обучения

def adagrad(grad, learning\_rate, epsilon):

    accumulated\_grad\_squared = np.zeros\_like(grad)

    accumulated\_grad\_squared += grad \*\* 2

    adjusted\_lr = learning\_rate / (accumulated\_grad\_squared\*\*0.5 + epsilon)

    return adjusted\_lr \* grad

# Построение графика

def plot\_function\_and\_trajectory(trajectory):

    x1\_range = np.linspace(-4, 4, 400)

    x2\_range = np.linspace(-4, 4, 400)

    X1, X2 = np.meshgrid(x1\_range, x2\_range)

    Z = np.zeros\_like(X1)

    Z = func(X1, X2)

    plt.figure(figsize=(10, 7))

    plt.contour(X1, X2, Z, levels=50, cmap='viridis')  # Контурный график функции

    plt.colorbar()

    # Извлечение точек траектории

    trajectory = np.array(trajectory)

    plt.plot(trajectory[:, 0], trajectory[:, 1], marker='o', color='red', linewidth=1, markersize=1, label='Траектория')

    plt.title('Градиентный спуск')

    plt.xlabel('x1')

    plt.ylabel('x2')

    plt.legend()

    plt.grid()

    plt.autoscale()

    plt.show()

# НАЧАЛО ПРОГРАММЫ

x0 = ((0.0, 0.0), (0.0, 3.0), (3.0, 0.0), (8, 9))

learning\_rate = 0.1

epsilon = 0.007

max\_iteration = 1000

func = func\_test\_1

# \* перед кортежем раскладывает его на несколько значений

x1\_min, x2\_min, trajectory = gradient\_descent(\*x0[0], learning\_rate, epsilon, max\_iteration)

plot\_function\_and\_trajectory(trajectory)

func = func\_test\_2

x1\_min, x2\_min, trajectory = gradient\_descent(\*x0[0], learning\_rate, epsilon, max\_iteration)

plot\_function\_and\_trajectory(trajectory)

func = func1

x1\_min, x2\_min, trajectory = gradient\_descent(\*x0[1], learning\_rate, epsilon, max\_iteration)

plot\_function\_and\_trajectory(trajectory)

x1\_min, x2\_min, trajectory = gradient\_descent(\*x0[2], learning\_rate, epsilon, max\_iteration)

plot\_function\_and\_trajectory(trajectory)

func = func2

x1\_min, x2\_min, trajectory = gradient\_descent(\*x0[0], learning\_rate, epsilon, max\_iteration)

plot\_function\_and\_trajectory(trajectory)

func = func3

x1\_min, x2\_min, trajectory = gradient\_descent(\*x0[3], learning\_rate, epsilon, max\_iteration)

plot\_function\_and\_trajectory(trajectory)