Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Дисциплина: Методы оптимизации**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

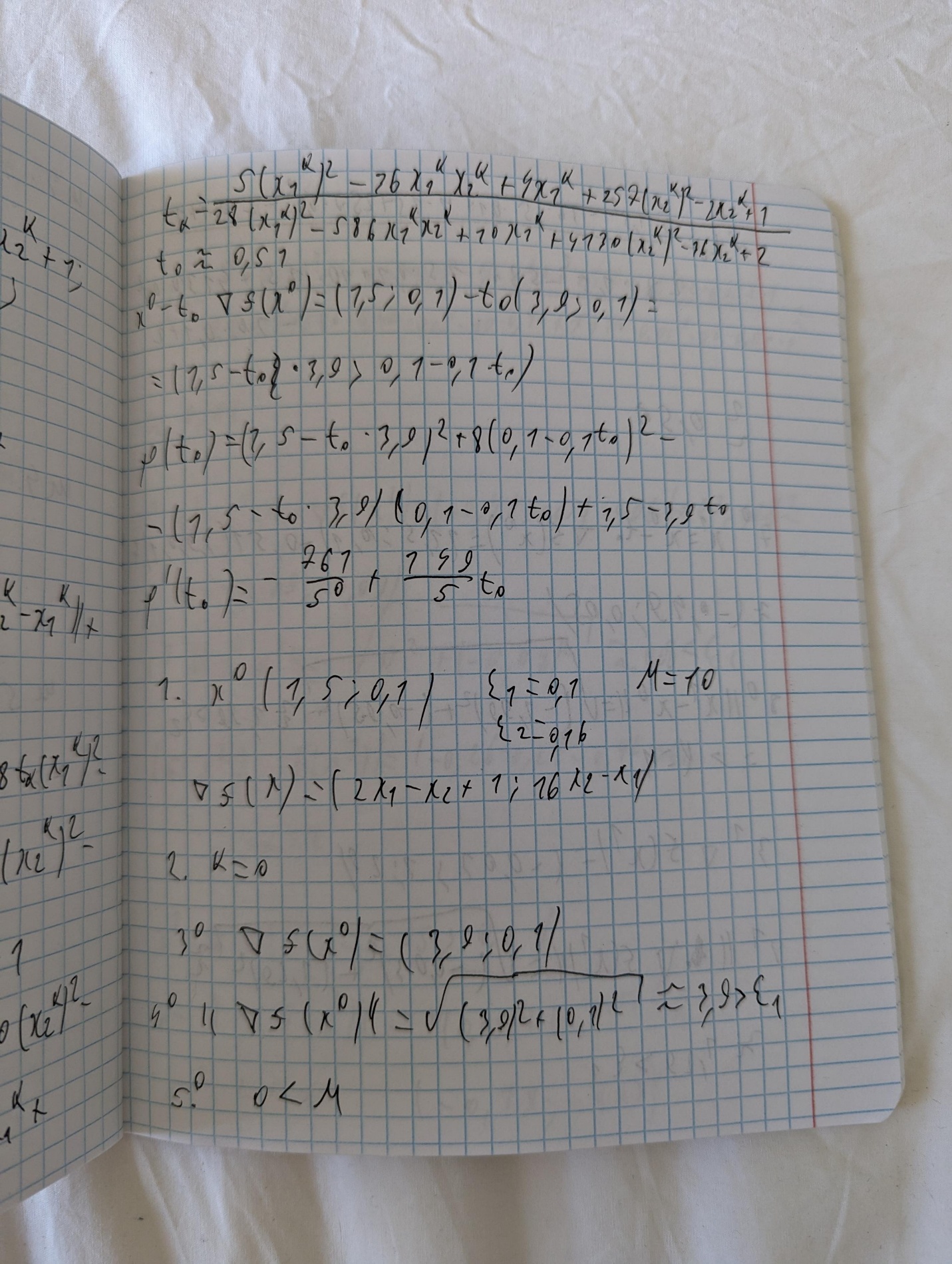
Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Черная

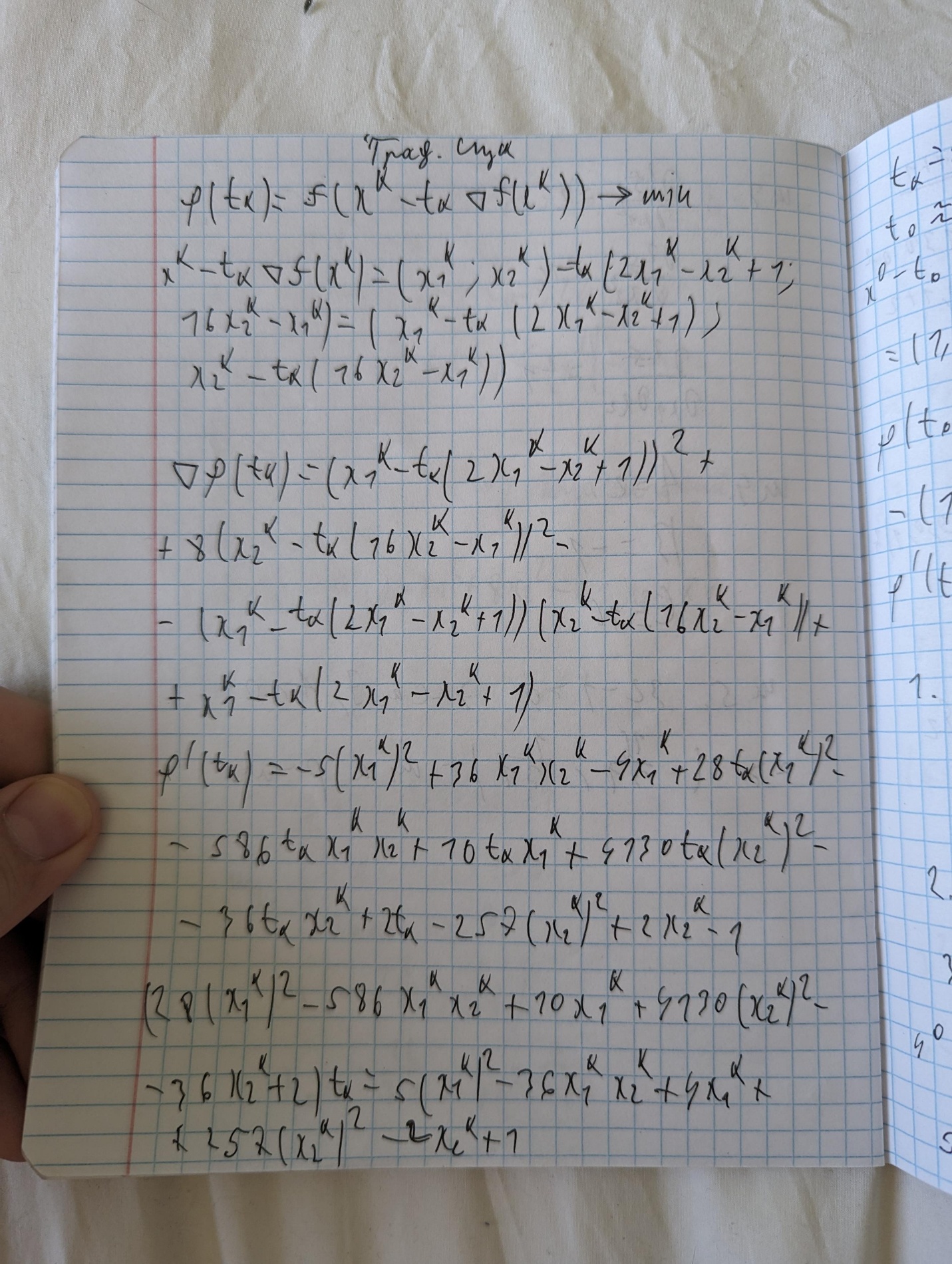
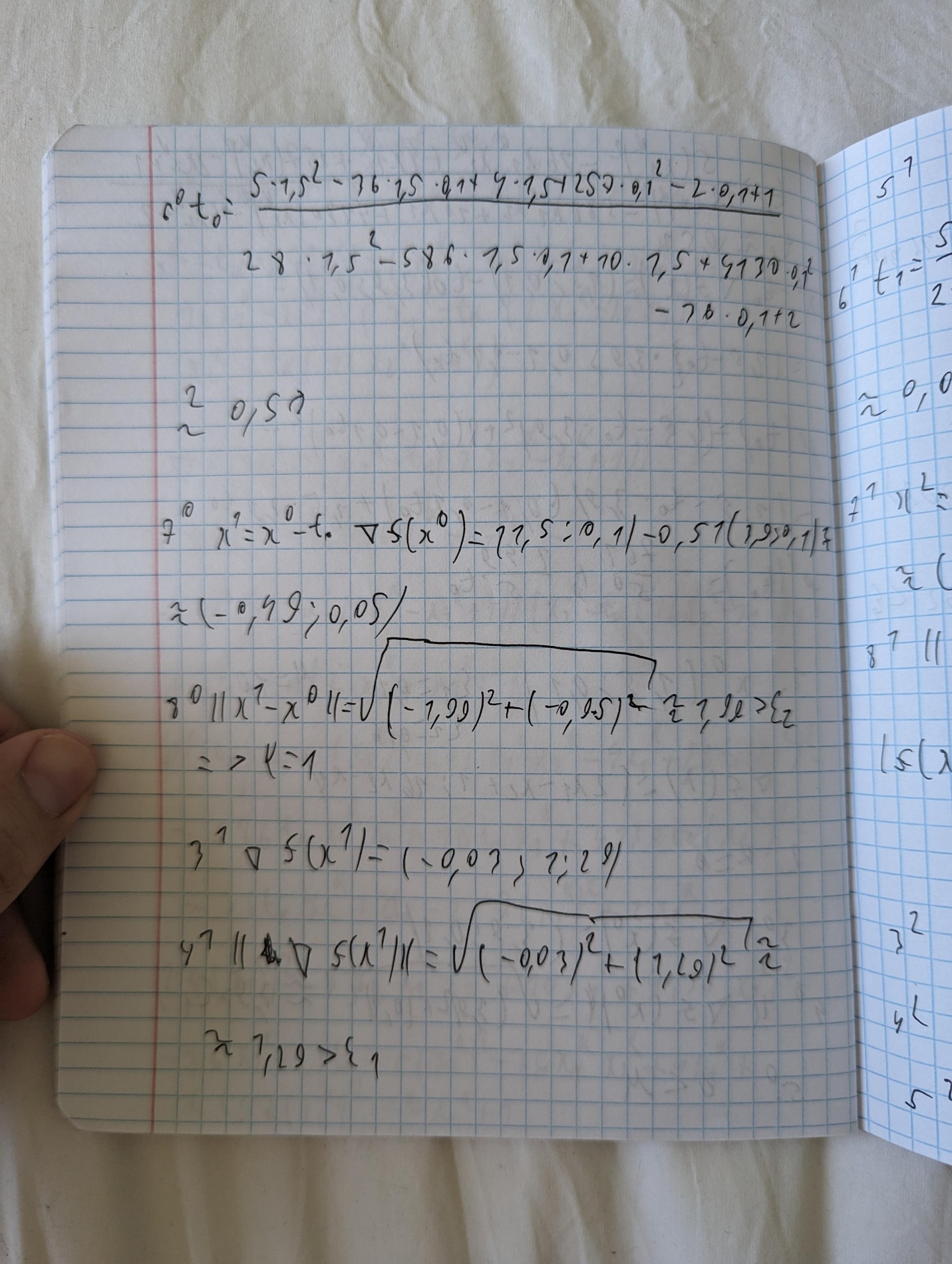
**Цель работы**: Целью данной лабораторной работы является изучение и применение методов градиентного спуска, Ньютона, Ньютона-Рафсона, Флетчера-Ривса.

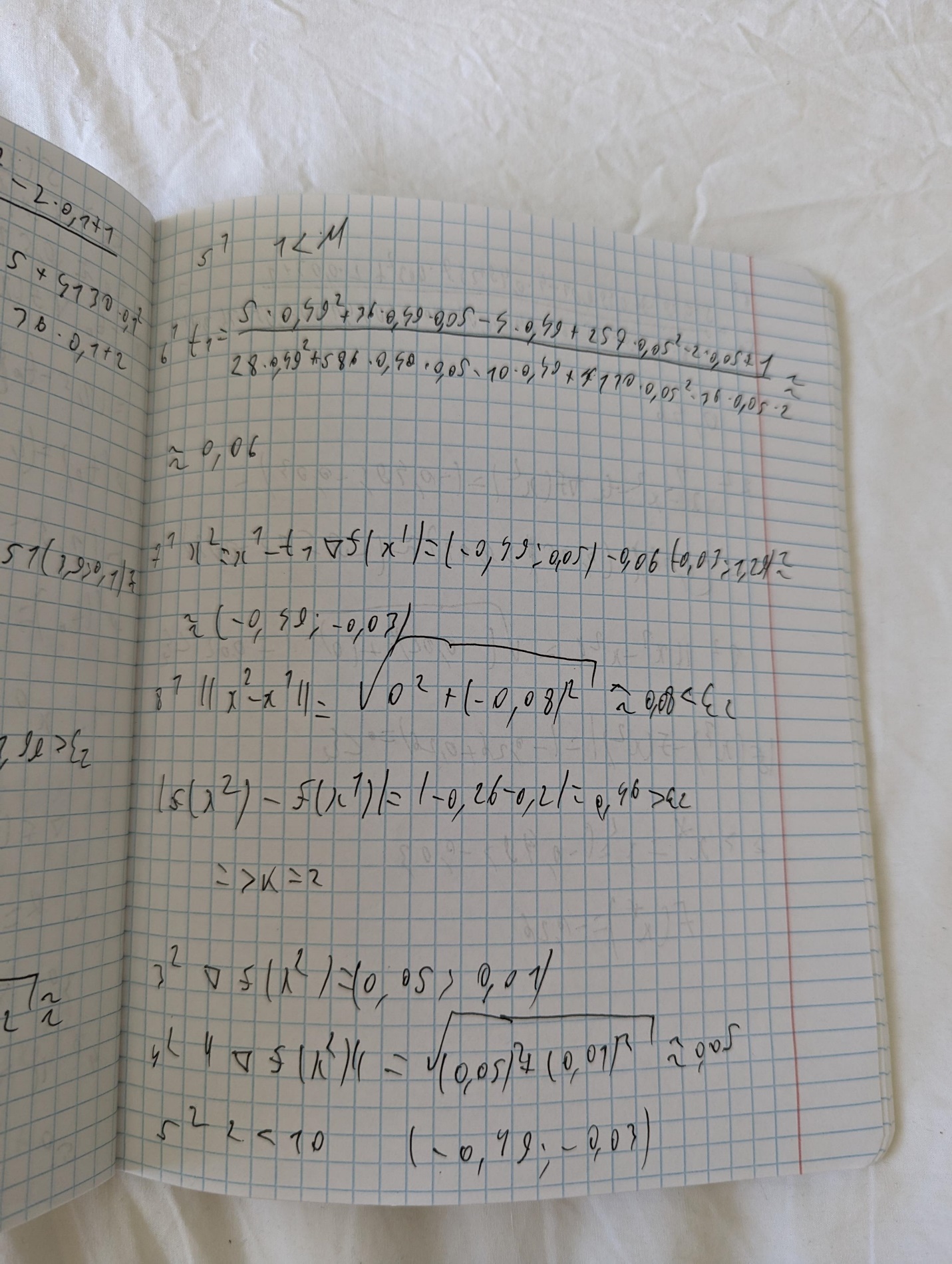
Функция:

=(1.5;0.1); =0.1; =0.16; M=10

**Задача 1.** Изучить метод градиентного спуска.

Это метод, при котором шаг для перемещения в пространстве параметров выбирается таким образом, чтобы достичь максимально быстрого уменьшения значения функции. На каждом шаге алгоритма вычисляется градиент функции в текущей точке, и производится перемещение в направлении, противоположном градиенту.



**Код**

import numpy as np

def f(x):

return x[0]\*\*2 + 8 \* x[1]\*\*2 - x[0] \* x[1] + x[0]

def t(x):

return ( 5\*(x[0]\*\*2) -36\*x[0]\*x[1]+4\*x[0]+257\*(x[1]\*\*2)-2\*x[1]+1 )/\

( 28\*(x[0]\*\*2) -586\*x[0]\*x[1] +10\*x[0]+4130\*(x[1]\*\*2) -36\*x[1]+2 )

def gradient\_descent(f, x0, E1, E2, M):

# Шаг 1 и 2: Инициализация

x = x0

k = 0

# Градиент функции

def grad\_f(x):

return np.array([2\*x[0] - x[1] + 1, 16\*x[1] - x[0]])

isTriggered=False

while True:

# Шаг 3: Вычислить градиент

grad = grad\_f(x)

# Шаг 4: Проверка критерия окончания

if np.linalg.norm(grad) < E1:

return x, k+1

# Шаг 5: Проверка неравенства

if k >= M:

return x, k+1

# Шаг 6: Вычислить шаг

alpha = t(x) # Можно использовать более сложные методы выбора шага

# Шаг 7: Обновить x

x\_new = x - alpha \* grad

# Шаг 8: Проверка условий сходимости

if np.linalg.norm(x\_new - x) < E2 and abs(f(x\_new) - f(x)) < E2:

return x\_new, k + 1

if(np.linalg.norm(x\_new-x)<E2 and abs(f(x\_new-f(x)))<E2):

if(isTriggered):

return x, k

else:

isTriggered=False

# Переход к следующей итерации

x = x\_new

k += 1

# Пример использования

x0 = np.array([1.5, 0.1])

E1 = 0.1

E2 = 0.16

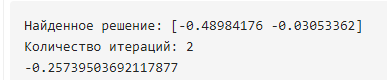
M = 10

result, iterations = gradient\_descent(f, x0, E1, E2, M)

print("Найденное решение:", result)

print("Количество итераций:", iterations)

print(f(result))



**Задача 2.** Изучить метод Ньютона.

Метод Ньютона предназначен для нахождения критических точек функции путем использования информации о первых и вторых производных. Основная идея состоит в том, чтобы делать шаги пропорционально обратной матрице Гессе функции, умноженной на градиент. Это приводит к более быстрой сходимости по сравнению с методом простого градиентного спуска.

**Код:**

import numpy as np

def f(x):

return x[0]\*\*2 + 8 \* x[1]\*\*2 - x[0] \* x[1] + x[0]

def t(x):

return ( 5\*(x[0]\*\*2) -36\*x[0]\*x[1]+4\*x[0]+257\*(x[1]\*\*2)-2\*x[1]+1 )/\

( 28\*(x[0]\*\*2) -586\*x[0]\*x[1] +10\*x[0]+4130\*(x[1]\*\*2) -36\*x[1]+2 )

def grad\_f(x):

return np.array([2\*x[0] - x[1] + 1, 16\*x[1]-x[0]])

def hessian\_f(x):

return np.array([[2, -1], [-1, 16]])

def newton\_method(x0, E1, E2, M):

x = x0

k = 0

isTriggered=False

while k < M:

grad = grad\_f(x)

H = hessian\_f(x)

H\_inv = np.linalg.inv(H)

if np.linalg.norm(grad) <= E1:

return x, k+1

x\_new = 0

if np.all(np.linalg.eigvals(H) > 0):

d = -np.dot(H\_inv, grad)

x\_new = x + d

else:

d = -grad

x\_new = x + t(x)\*d

if(np.linalg.norm(x\_new-x)<E2 and abs(f(x\_new-f(x)))<E2):

if(isTriggered):

return x, k

else:

isTriggered=False

if np.linalg.norm(x\_new - x) <= E2:

return x\_new, k+1

x = x\_new

k += 1

return x, k

x0 = np.array([1.5,0.1])

E1 = 0.1

E2 = 0.16

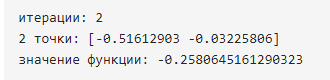
M = 10

x\_opt, iterations = newton\_method(x0, E1, E2, M)

print("итерации:", iterations)

print("2 точки:", x\_opt)

print("значение функции:",f(x\_opt))



**Задача 3.** Изучить метод Ньютона-Рафсона.

Этот метод является разновидностью метода Ньютона, адаптирован для решения систем нелинейных уравнений. Он использует аппроксимацию функции до второго порядка для нахождения её корней (то есть точек, где функция равна нулю), применяя при этом итерационные шаги на основе матрицы Гессиана. В контексте оптимизации метод Ньютона-Рафсона может использоваться для более сложных функций, где требуется более точное нахождение точек минимума.

**Код:**

import numpy as np

def f(x):

return x[0]\*\*2 + 8 \* x[1]\*\*2 - x[0] \* x[1] + x[0]

def grad\_f(x):

return np.array([2\*x[0] - x[1] + 1, 16\*x[1]-x[0]])

def hessian\_f(x):

return np.array([[2, -1], [-1, 16]])

def t(x, d):

return -(((16 \* d[1] - d[0]) \* x[1] + (2 \* d[0] - d[1]) \* x[0] + d[0]) / \

(16 \* (d[1] \*\* 2) - 2 \* d[0] \* d[1] + 2 \* (d[0] \*\* 2)))

def newton\_rafson\_method(x0, E1, E2, M):

x = x0

k = 0

isTriggered=False

while k < M:

grad = grad\_f(x)

H = hessian\_f(x)

H\_inv = np.linalg.inv(H)

if np.linalg.norm(grad) <= E1:

return x, k+1

if np.all(np.linalg.eigvals(H\_inv) > 0):

d = -np.dot(H\_inv, grad)

else:

d = -grad

x\_new = x + t(x, d) \* d

if np.linalg.norm(x\_new - x) <= E2:

return x\_new, k+1

if(np.linalg.norm(x\_new-x)<E2 and abs(f(x\_new-f(x)))<E2):

if(isTriggered):

return x, k+1

else:

isTriggered=False

x = x\_new

k += 1

return x, k

x0 = np.array([1.5,0.1])

E1 = 0.1

E2 = 0.16

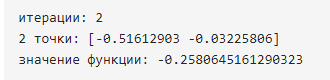
M = 10

x\_opt, iterations = newton\_rafson\_method(x0, E1, E2, M)

print("итерации:", iterations)

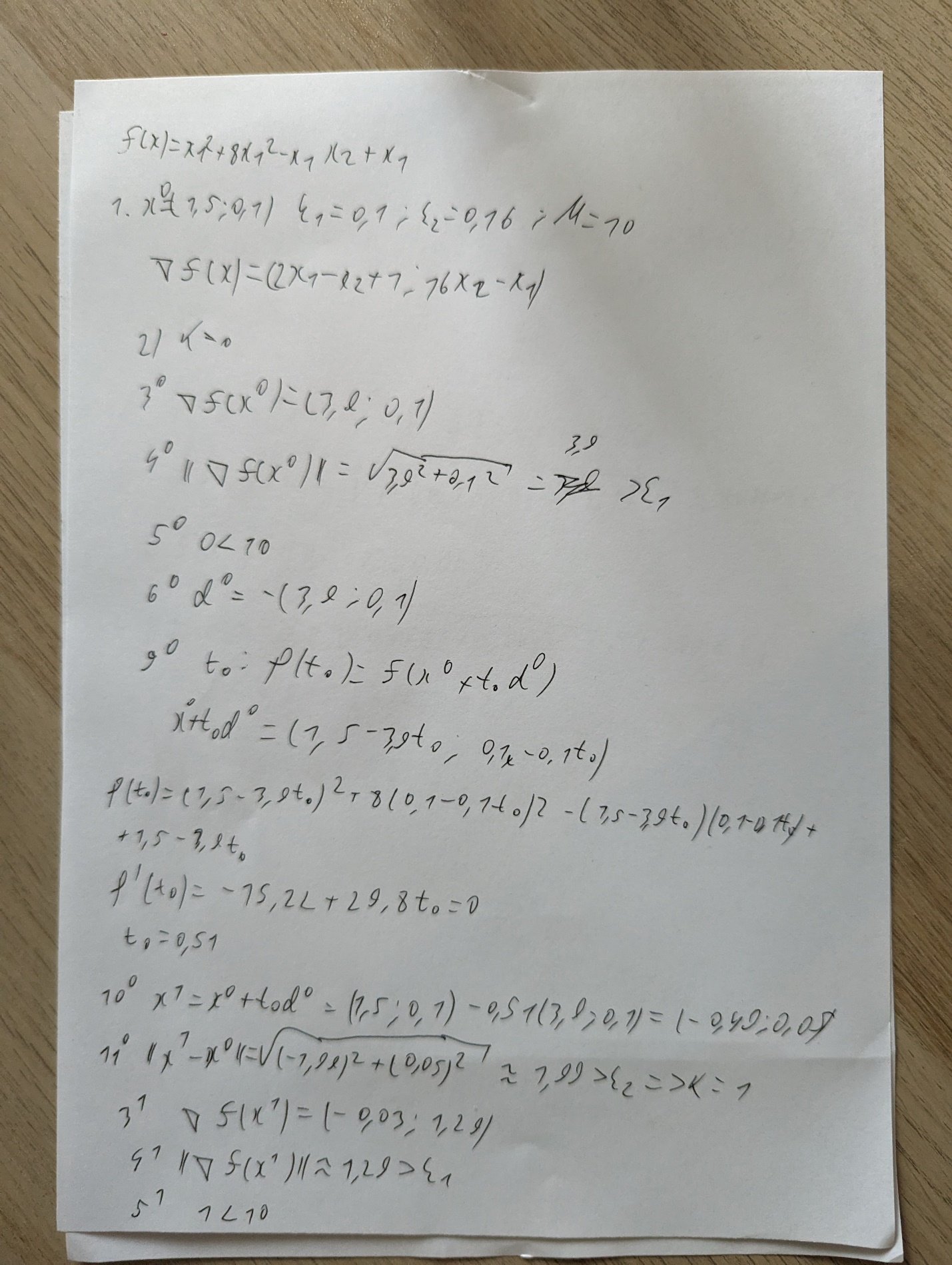
print("2 точки:", x\_opt)

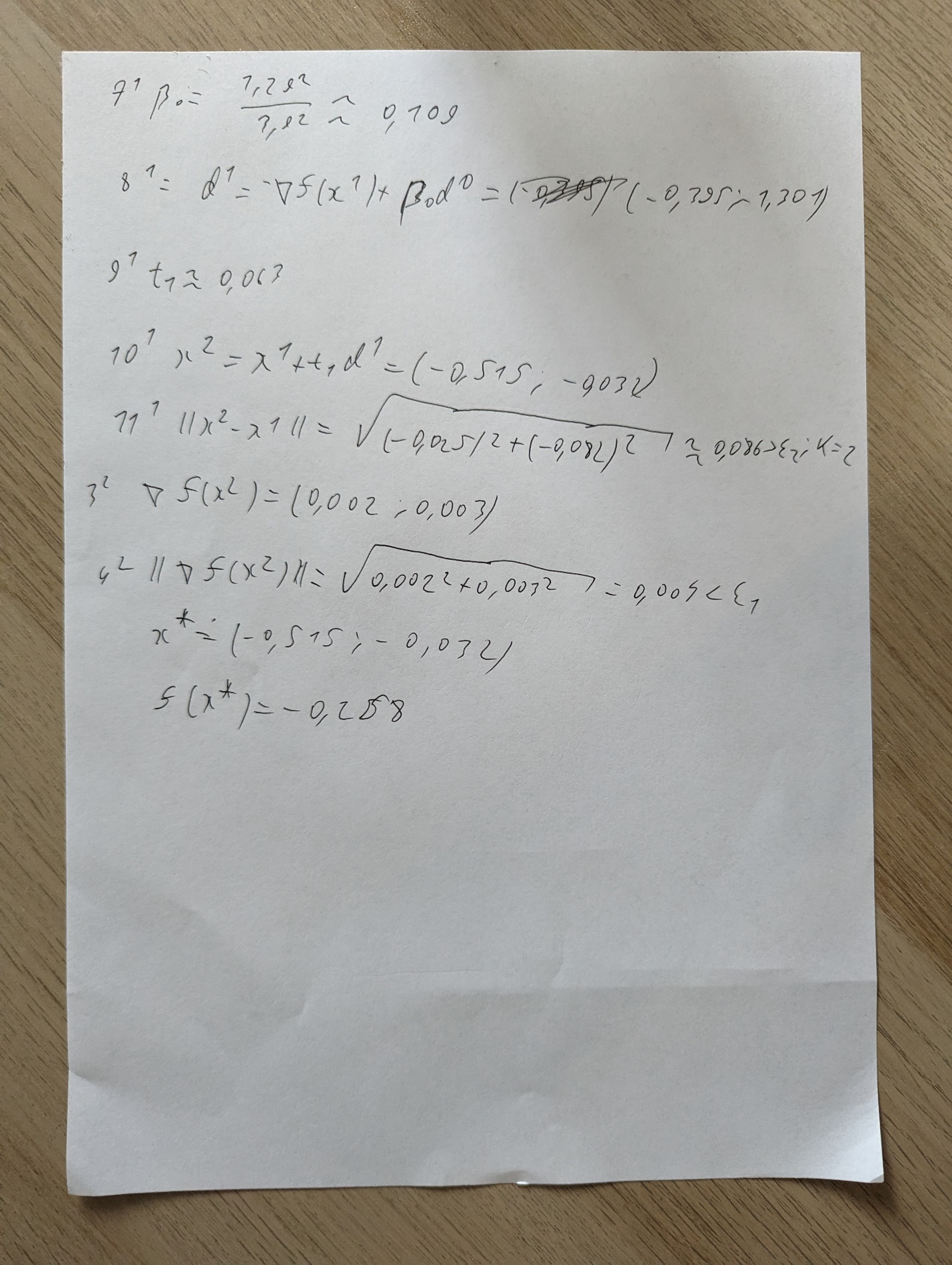
print("значение функции:",f(x\_opt))



**Задача 4.** Изучить метод Флетчера-Ривса.

Это метод оптимизации, который модифицирует классический метод наискорейшего спуска с помощью механизма “перезагрузки” направления поиска. Он использует градиенты функции для определения направления спуска, но вводит корректировку предыдущего направления спуска, что позволяет учесть информацию о кривизне пространства решений. Это делает метод Флетчера-Ривса более эффективным в случаях, когда пространство решений имеет сложную структуру.





**Код:**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

def f(x):

return x[0]\*\*2 + 8 \* x[1]\*\*2 - x[0] \* x[1] + x[0]

def grad\_f(x):

return np.array([2\*x[0] - x[1] + 1, 16\*x[1]-x[0]])

def hessian\_f(x):

return np.array([[2, -1], [-1, 16]])

def t(x, d):

return -(((16 \* d[1] - d[0]) \* x[1] + (2 \* d[0] - d[1]) \* x[0] + d[0]) / \

(16 \* (d[1] \*\* 2) - 2 \* d[0] \* d[1] + 2 \* (d[0] \*\* 2)))

def fletcher\_reeves(x0, E1, E2, M):

x = x0

k = 0

grad = grad\_f(x)

d = -grad

path = [x.copy()]

isTriggered=False

while k < M:

if np.linalg.norm(grad\_f(x)) < E1:

return x, k+1, path

alpha = t(x, d)

x\_new = x + alpha \* d

path.append(x\_new.copy())

if np.linalg.norm(x\_new - x) < E2 and abs(f(x\_new) - f(x)) < E2:

return x\_new, k+1, path

grad\_new = grad\_f(x\_new)

beta = np.linalg.norm(grad\_new) \*\* 2 / np.linalg.norm(grad) \*\* 2

d = -grad\_new + beta \* d

if(np.linalg.norm(x\_new-x)<E2 and abs(f(x\_new-f(x)))<E2):

if(isTriggered):

return x, k+1

else:

isTriggered=False

x = x\_new

grad = grad\_new

k += 1

return x, k, path

x0 = np.array([1.5,0.1])

E1 = 0.1

E2 = 0.16

M = 10

x\_opt, iterations, path = fletcher\_reeves(x0, E1, E2, M)

print("итерации:", iterations)

print("2 точки:", x\_opt)

print("значение функции:",f(x\_opt))

x\_range = np.linspace(-5, 5, 100)

y\_range = np.linspace(-5, 5, 100)

x\_mesh, y\_mesh = np.meshgrid(x\_range, y\_range)

z\_mesh = f([x\_mesh, y\_mesh])

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.plot\_surface(x\_mesh, y\_mesh, z\_mesh, alpha=0.5, cmap='viridis')

ax.view\_init(elev=25, azim=22)

path = np.array(path)

ax.plot(path[:, 0], path[:, 1], f(path.T), marker='o', color='r', label='Путь')

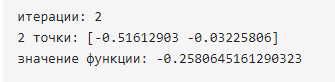
ax.set\_xlabel('X axis')

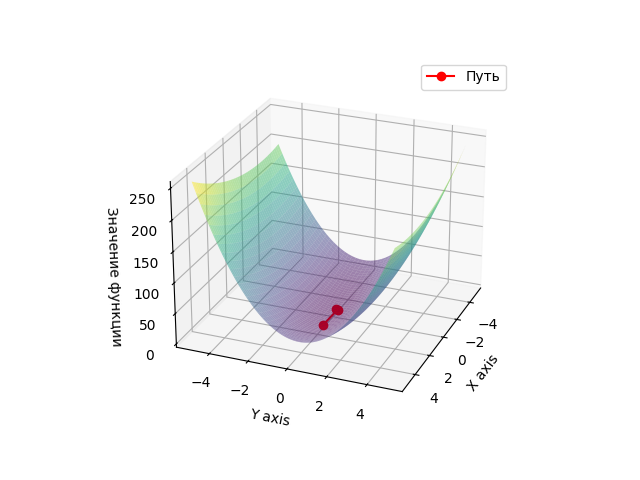
ax.set\_ylabel('Y axis')

ax.set\_zlabel('Значение функции')

ax.legend()

plt.show()



****

**Вывод:**

Метод градиентного спуска – этот метод эффективен для поиска локального минимума функции, но может быть медленным, если функция сложна или размерность пространства велика.

Метод Ньютона – обеспечивает более быструю сходимость по сравнению с методом градиентного спуска, благодаря использованию информации о вторых производных (гессиане). Однако, требует вычисления обратной матрицы гессиана, что может быть вычислительно затратным для больших размерностей.

Метод Ньютона-Рафсона – вариант метода Ньютона, адаптированный для решения задач оптимизации с учетом более сложных условий на шаг алгоритма. Также обеспечивает быструю сходимость, но требует аккуратного выбора параметров для избежания проблем с неопределенностью гессиана.

Метод Флетчера-Ривса – метод оптимизации, который использует концепции первого и второго порядка для определения направления поиска. Этот метод может быть более эффективным на практике, так как он адаптируется к форме функции и потенциально может сходиться быстрее в ситуациях, где другие методы застревают или слишком медленно прогрессируют.