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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Е. Головкин

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

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

**Цель работы**: получение знаний и практических навыков в области численной многомерной оптимизации методов для нахождения функции двух переменных.

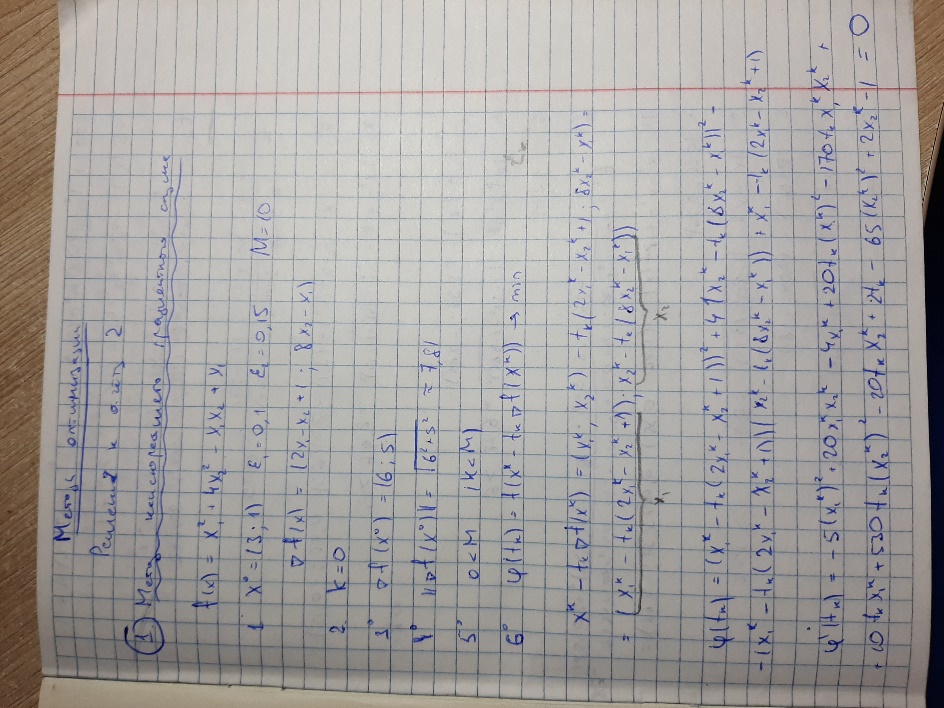
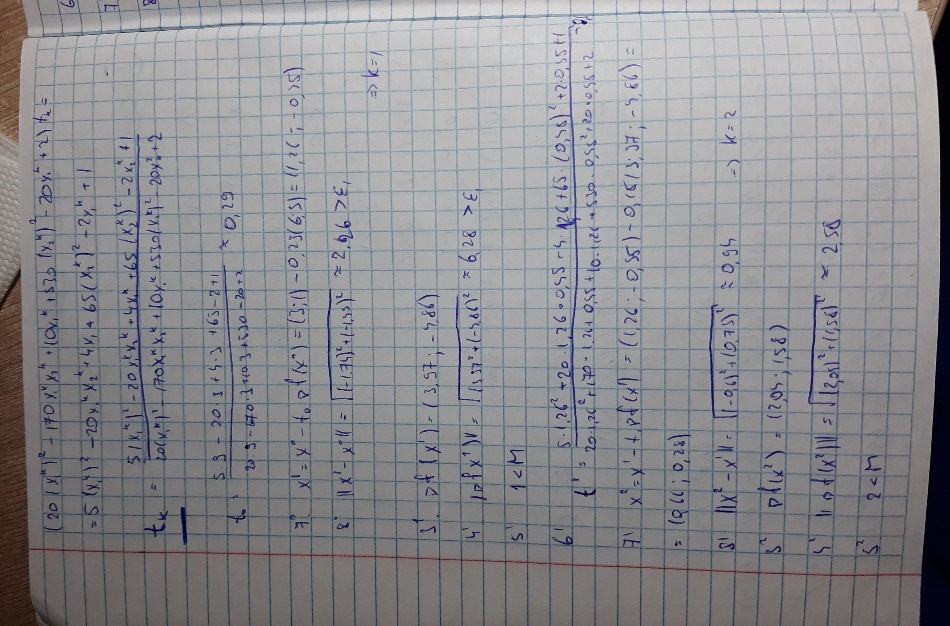
**Задание**: требуется написать программы для поиска минимума функции f(x) = методами: наискорейшего градиентного спуска, Ньютона, Ньютона-Рафсона, Флетчера-Ривса.

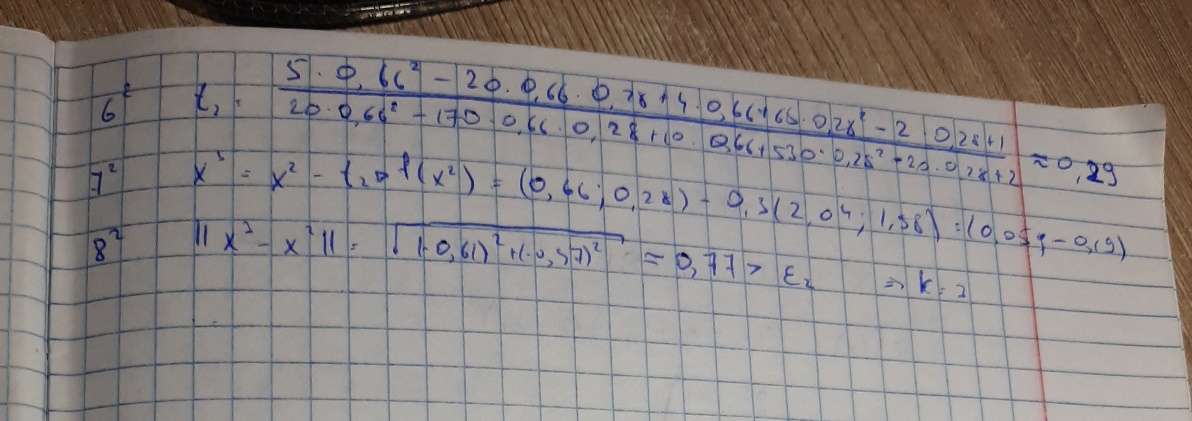
**Решение:**

**Метод наискорейшего градиентного спуска**

*Метод наискорейшего градиентного спуска* — это итеративный алгоритм оптимизации, используемый для нахождения минимума функции. Он начинает с начальной точки и делает шаги пропорциональные отрицательному градиенту функции в текущей точке. Размер шага определяется скоростью обучения. Алгоритм повторяет эти шаги до тех пор, пока не достигнет минимума или не выполнится условие остановки.

**Традиционная реализация**



**Программная реализация**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

def f(x):

return x[0] \*\* 2 + 4 \* x[1] \*\* 2 - x[0] \* x[1] + x[0]

def t(x):

return (5 \* (x[0] \*\* 2) - 20 \* x[0] \* x[1] + 4 \* x[0] + 65 \* (x[1] \*\* 2) - 2 \* x[1] + 1) / \

(20 \* (x[0] \*\* 2) - 170 \* x[0] \* x[1] + 10 \* x[0] + 530 \* (x[1] \*\* 2) - 20 \* x[1] + 2)

def gradient\_descent(f, x0, E1, E2, M): #1

x = x0

k = 0 #2

path = [] # To store points for plotting

def grad\_f(x): #3

return np.array([2 \* x[0] - x[1] + 1, 8 \* x[1] - x[0]])

flag = False

while True:

grad = grad\_f(x)

if np.linalg.norm(grad) < E1: #4

return x, k, path

if k >= M: #5

return x, k, path

alpha = t(x) #6

x\_new = x - alpha \* grad #7

path.append((x[0], x[1], f(x)))

if np.linalg.norm(x\_new - x) < E2 and abs(f(x\_new) - f(x)) < E2: #12

if (flag==False):

flag = True

else:

return x\_new, k + 1, path

else:

flag=False

x = x\_new

k += 1

x0 = np.array([3, 1])

E1 = 0.1

E2 = 0.15

M = 10

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

print("итерации:", iterations)

print("Точка минимума:", x\_opt)

print("Значение функции:",f(x\_opt))

# Plotting

fig = plt.figure()

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

path = np.array(path)

# Plot path

ax.plot(path[:, 0], path[:, 1], path[:, 2], 'bo-')

# Highlight start and end points

ax.scatter(x0[0], x0[1], f(x0), color='green', s=100, label='Start Point')

ax.scatter(result[0], result[1], f(result), color='red', s=100, label='End Point')

ax.set\_xlabel('X axis')

ax.set\_ylabel('Y axis')

ax.set\_zlabel('f(X, Y)')

ax.legend()

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')

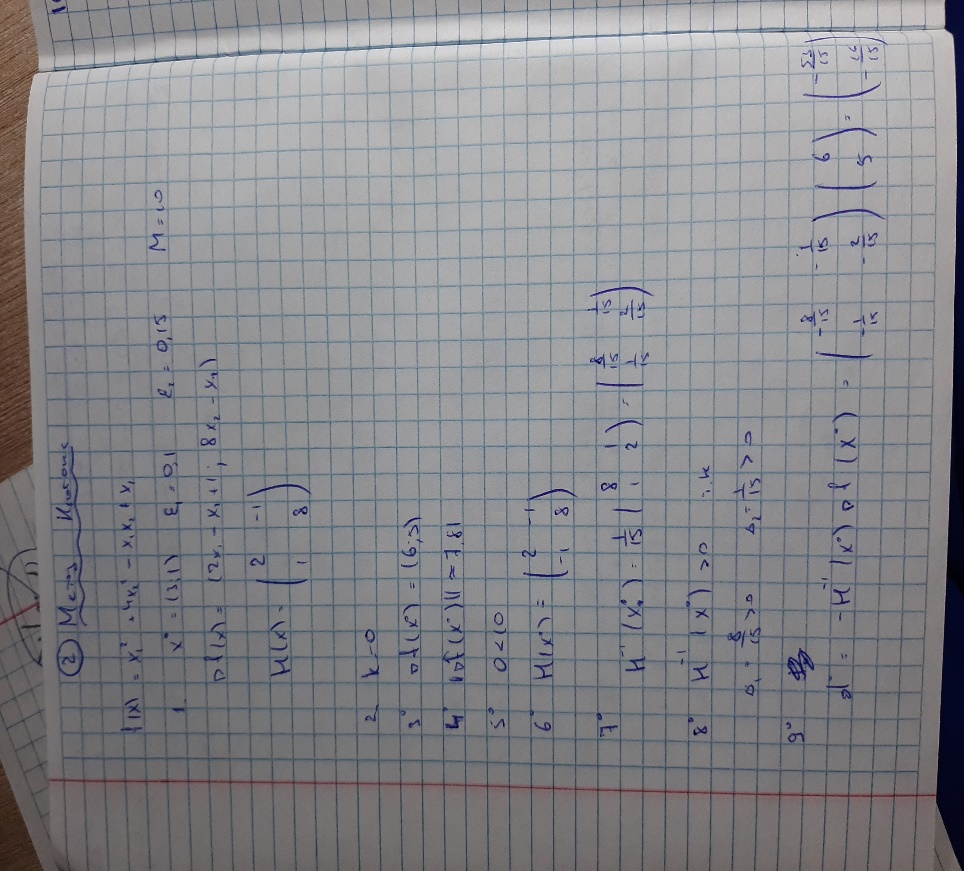
ax.legend()

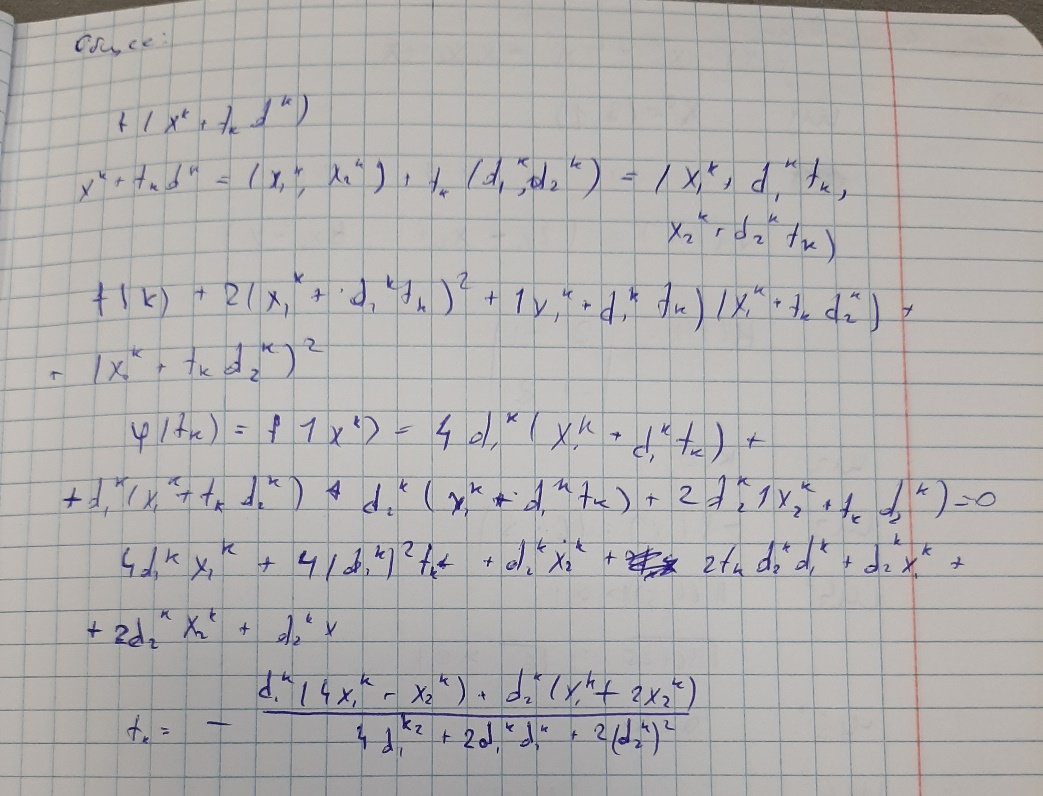
plt.show()

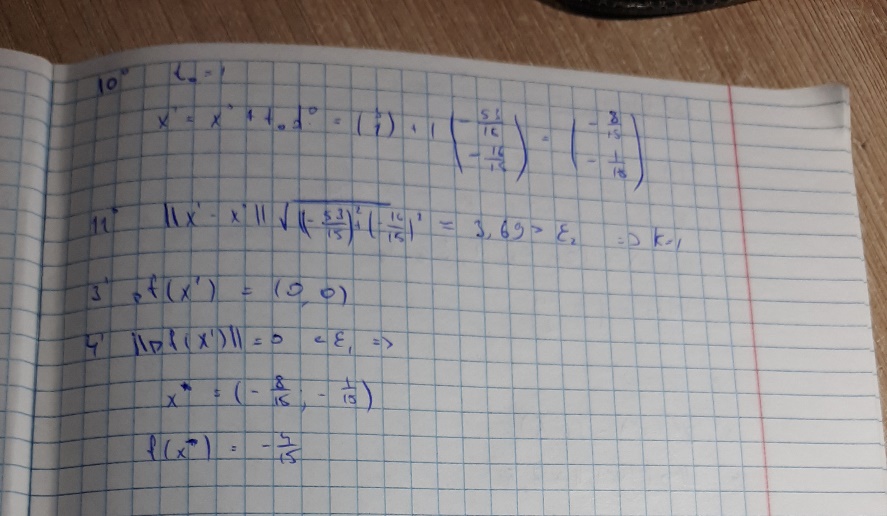
**Метод Ньютона**

*Метод Ньютона* — это метод, который использует градиенты и гессианы (матрицы вторых производных) функции для быстрого нахождения точек, где градиент равен нулю, что указывает на экстремумы. Алгоритм итеративный, на каждом шаге метод строит квадратичную аппроксимацию функции вокруг текущей точки и вычисляет направление к следующей точке как обратное гессиану умноженное на градиент.

**Традиционная реализация**







**Программная реализация**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

def f(x):

return x[0] \*\* 2 + 4 \* x[1] \*\* 2 - x[0] \* x[1] + x[0]

def grad\_f(x):

return np.array([2 \* x[0] - x[1] + 1, 8 \* x[1] - x[0]])

def hessian\_f(x):

return np.array([[2, -1], [-1, 8]])

def t(x):

return (5 \* (x[0] \*\* 2) - 20 \* x[0] \* x[1] + 4 \* x[0] + 65 \* (x[1] \*\* 2) - 2 \* x[1] + 1) / \

(20 \* (x[0] \*\* 2) - 170 \* x[0] \* x[1] + 10 \* x[0] + 530 \* (x[1] \*\* 2) - 20 \* x[1] + 2)

def newton\_method(x0, E1, E2, M): #1

x = x0

k = 0 #2

path = []

flag=False

while k < M: #5

grad = grad\_f(x) #3

H = hessian\_f(x) #6

H\_inv = np.linalg.inv(H) #7

path.append((x[0], x[1], f(x)))

if np.linalg.norm(grad) <= E1: #4

return x, k, path

x\_new = 0

if np.all(np.linalg.eigvals(H) > 0): #8

d = -np.dot(H\_inv, grad) #9

x\_new = x + d

else:

d = -grad #10

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

if np.linalg.norm(x\_new - x) < E2 and abs(f(x\_new) - f(x)) < E2: #12

if (flag==False):

flag = True

else:

return x\_new, k + 1, path

else:

flag=False

x = x\_new

k += 1

return x, k, path

x0 = np.array([3, 1])

E1 = 0.1

E2 = 0.15

M = 10

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

print("итерации:", iterations)

print("Точка минимума:", x\_opt)

print("Значение функции:",f(x\_opt))

# Plotting

fig = plt.figure()

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

path = np.array(path)

ax.plot(path[:,0], path[:,1], path[:,2], 'bo-') # plot trajectory

ax.scatter(x0[0], x0[1], f(x0), color='green', s=100, label='Start Point') # start point

ax.scatter(x\_opt[0], x\_opt[1], f(x\_opt), color='red', s=100, label='End Point') # end point

ax.set\_xlabel('X axis')

ax.set\_ylabel('Y axis')

ax.set\_zlabel('f(X, Y)')

ax.legend()

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')

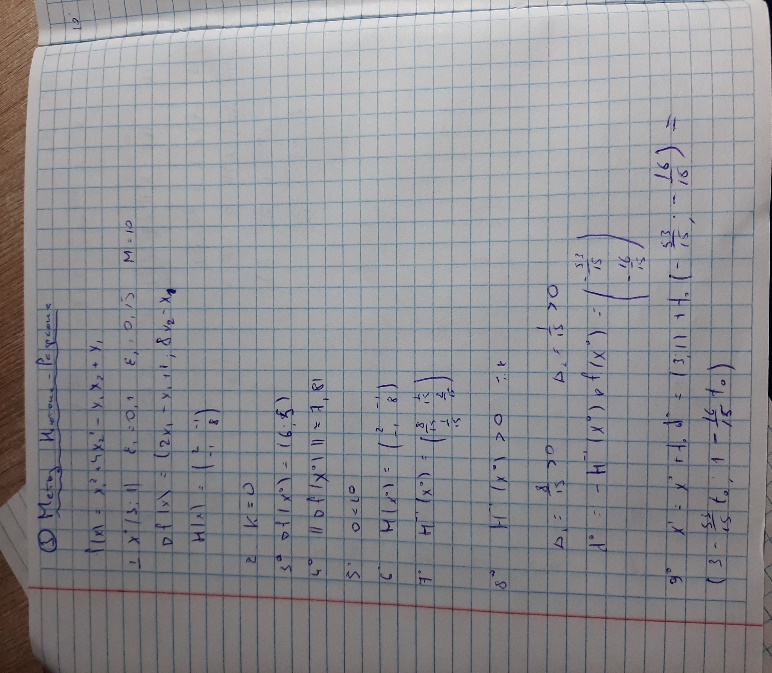
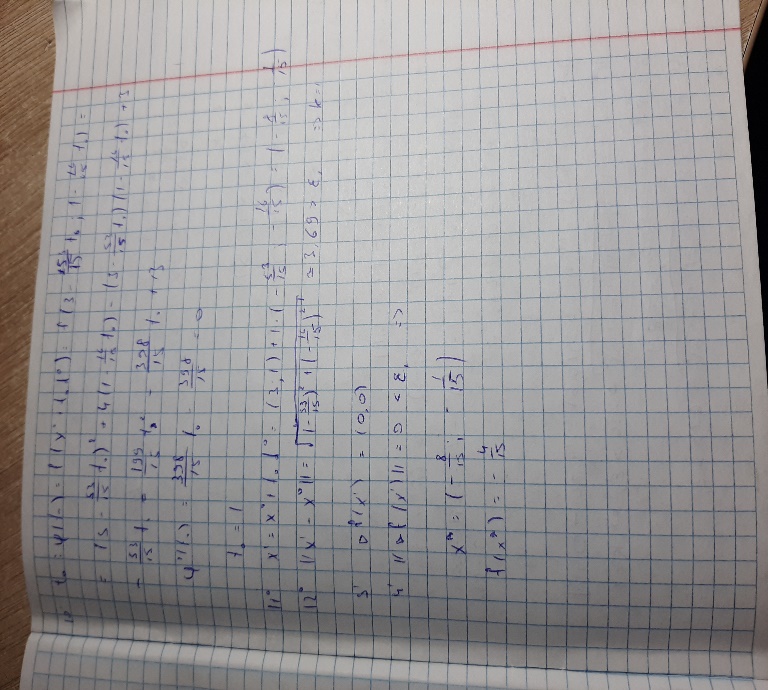
ax.legend()

plt.show()

**Метод Ньютона-Равсона**

*Метод Ньютона-Равсона* является улучшением метода Ньютона нахождения экстремума. Основное отличие заключается в том, что на очередной итерации каким-либо из методов одномерной оптимизации выбирается оптимальный шаг.

**Традиционная реализация**

**Программная реализация**

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 newton\_rafson\_method(x0, E1, E2, M): #1

x = x0

k = 0 #2

path = []

flag=False

while k < M: #5

grad = grad\_f(x) #3

H = hessian\_f(x) #6

H\_inv = np.linalg.inv(H) #7

path.append((x[0], x[1], f(x)))

if np.linalg.norm(grad) <= E1: #4

return x, k, path

if np.all(np.linalg.eigvals(H\_inv) > 0): #8

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

else:

d = -grad

x\_new = x + t(x, d) \* d #9,11

if np.linalg.norm(x\_new - x) < E2 and abs(f(x\_new) - f(x)) < E2: #12

if (flag==False):

flag = True

else:

return x\_new, k + 1, path

else:

flag=False

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

print("итерации:", iterations)

print("Точка минимума:", x\_opt)

print("Значение функции:",f(x\_opt))

# Plotting

fig = plt.figure()

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

path = np.array(path)

ax.plot(path[:,0], path[:,1], path[:,2], 'bo-') # plot trajectory

ax.scatter(x0[0], x0[1], f(x0), color='green', s=100, label='Start Point') # start point

ax.scatter(x\_opt[0], x\_opt[1], f(x\_opt), color='red', s=100, label='End Point') # end point

ax.set\_xlabel('X axis')

ax.set\_ylabel('Y axis')

ax.set\_zlabel('f(X, Y)')

ax.legend()

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')

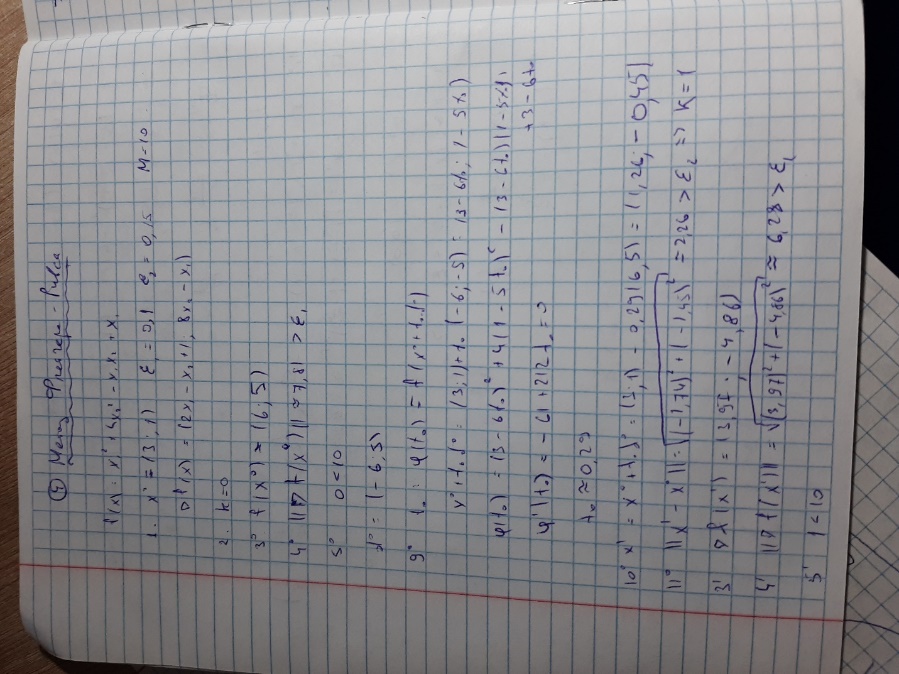
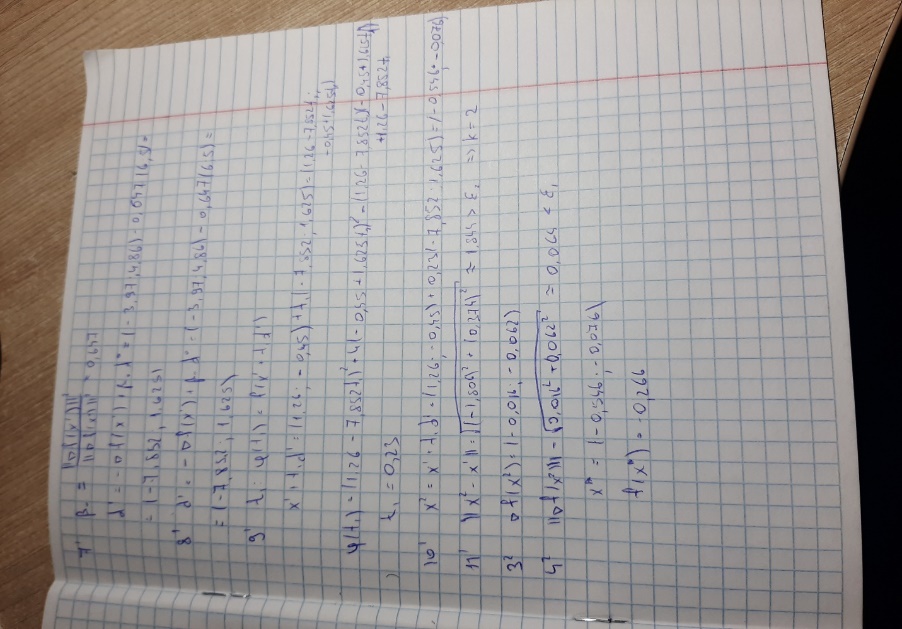
ax.legend()

plt.show()

**Метод Флетчера-Ривса**

*Метод Флетчера-Ривса* — это итерационный алгоритм первого порядка для оптимизации функций, основанный на методе наискорейшего спуска. Он использует градиенты функции для нахождения минимума, применяя формулу для обновления направления поиска, которая учитывает предыдущее направление и текущий градиент.

**Традиционная реализация**

**Программная реализация**

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): #1

x = x0

k = 0 #2

grad = grad\_f(x) #3

path = [x.copy()]

flag=False

while k < M: #5

if np.linalg.norm(grad\_f(x)) < E1: #4

return x, k, path

d = -grad #6

alpha = t(x, d)

x\_new = x + alpha \* d #10

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, path

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

beta = np.linalg.norm(grad\_new) \*\* 2 / np.linalg.norm(grad) \*\* 2 #7

d = -grad\_new + beta \* d #8

if np.linalg.norm(x\_new - x) < E2 and abs(f(x\_new) - f(x)) < E2: #12

if (flag==False):

flag = True

else:

return x\_new, k + 1, path

else:

flag=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("Точка минимума:", 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()

**Анализ результатов и вывод:**

С помощью классического метода был получен минимум (-8/15; -1/15) равный (-0,533;-0,067). Сравним это значение с другими полученными результатами.

Метод наискорейшего градиентного спуска – (-0,396;-0.0253)

Метод Ньютона – (0.533;-0.067)

Метод Ньютона-Равсона – (-0,516;0.323)

Метод-Флетчера-Ривса – (-0.490;-0.031)

Таким образом, лучшим методом для решения данной задачи оказался метод Ньютона, с помощью которого был получен точный результат.