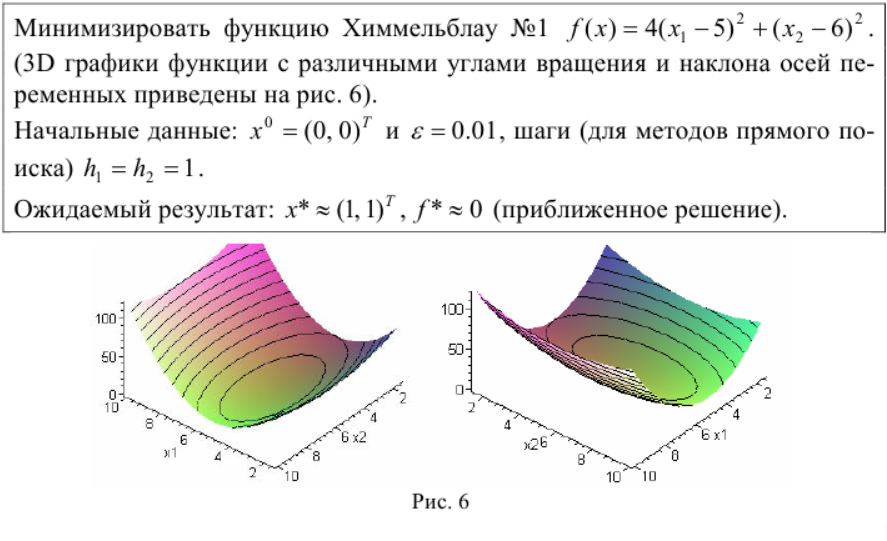
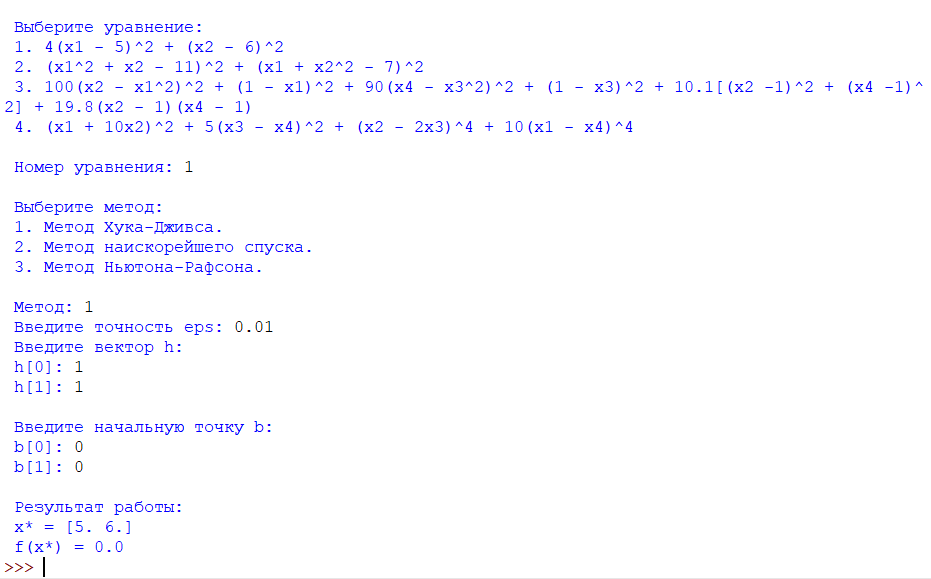
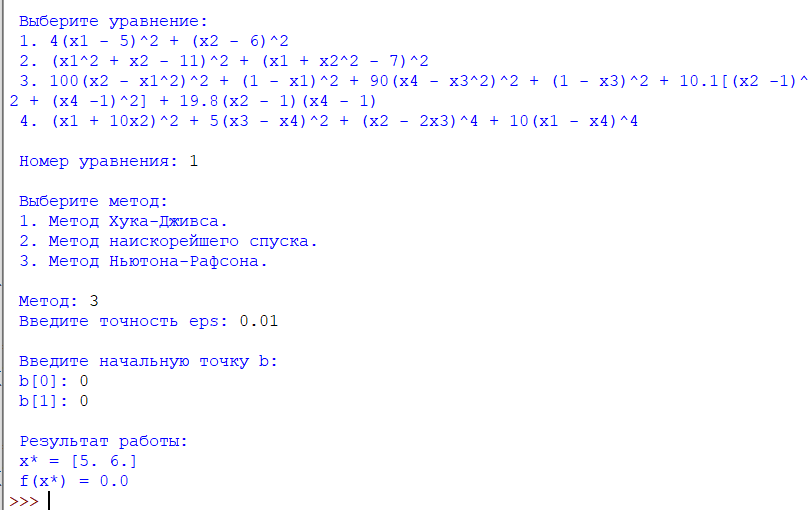
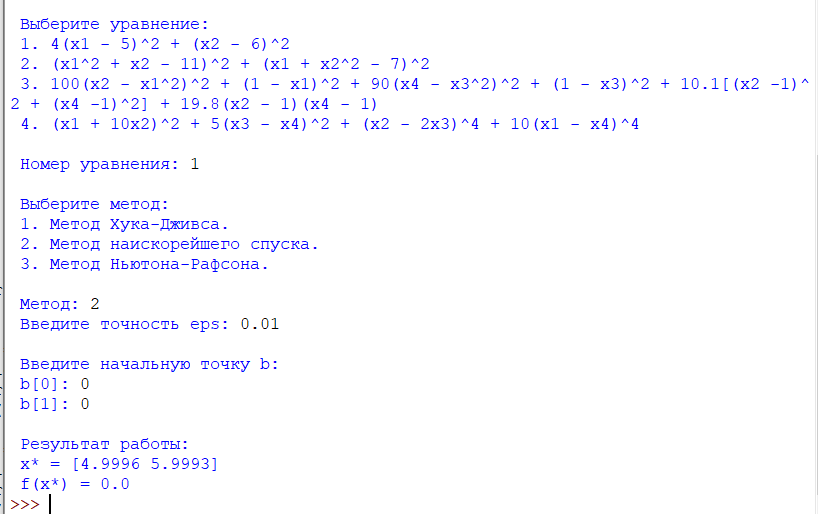
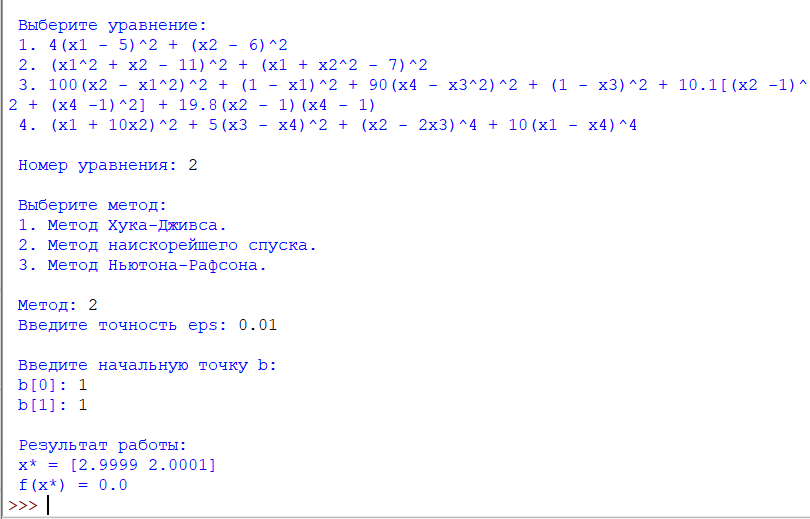
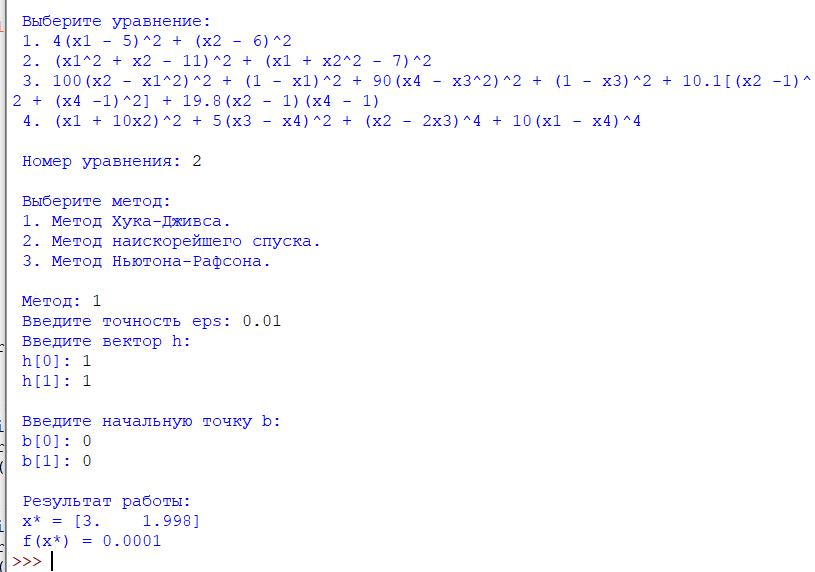
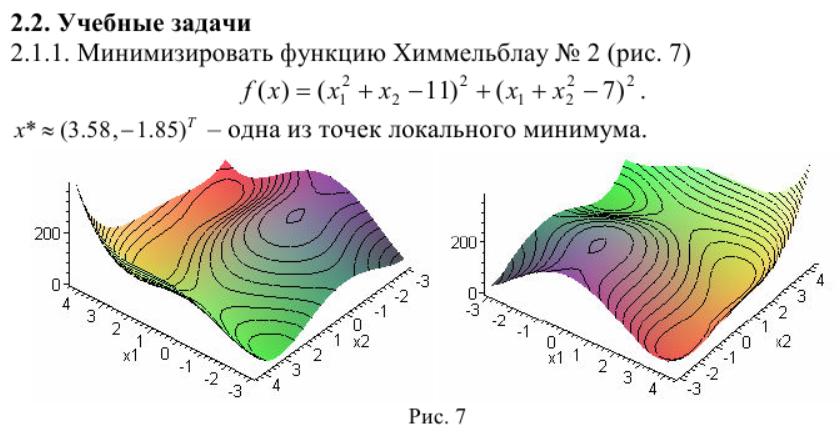
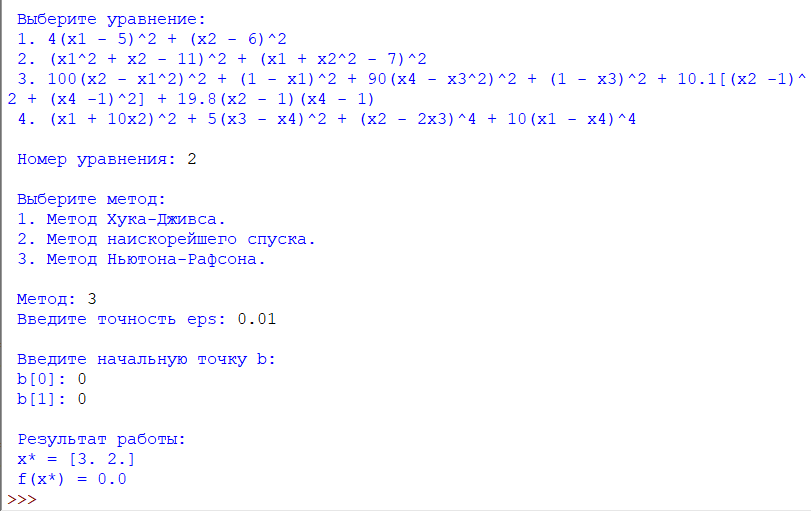
|  |  |  |
| --- | --- | --- |
| Лабораторная работа №2  Методы многомерной оптимизации | Ф.И.О. | Бокова Ольга Дмитриевна |
| Группа | ИВТ-363 |
| Преподаватель | Асанова Наталия Васильевна |
| Дата сдачи | 26.10.2022 |

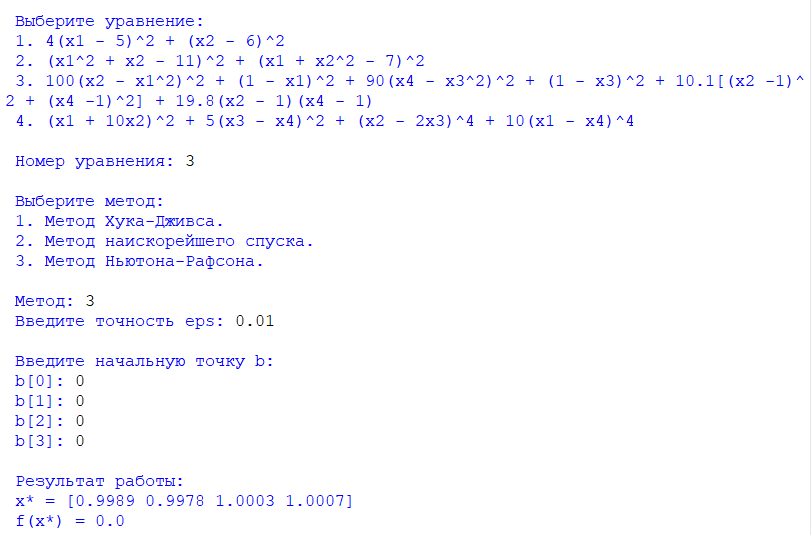
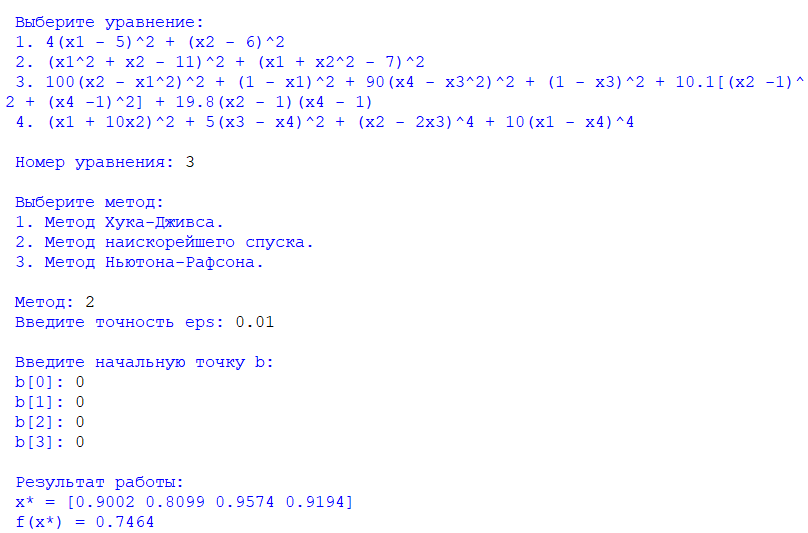
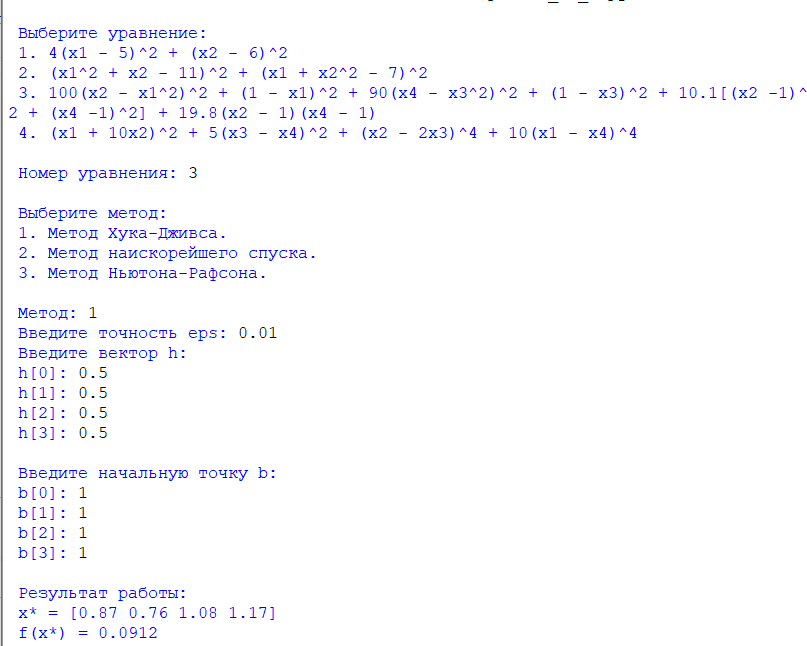
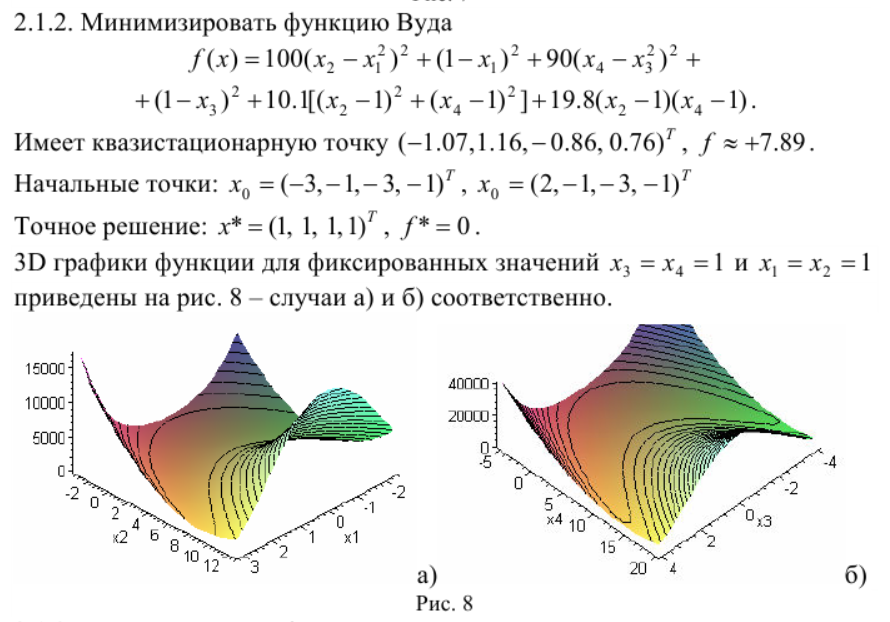


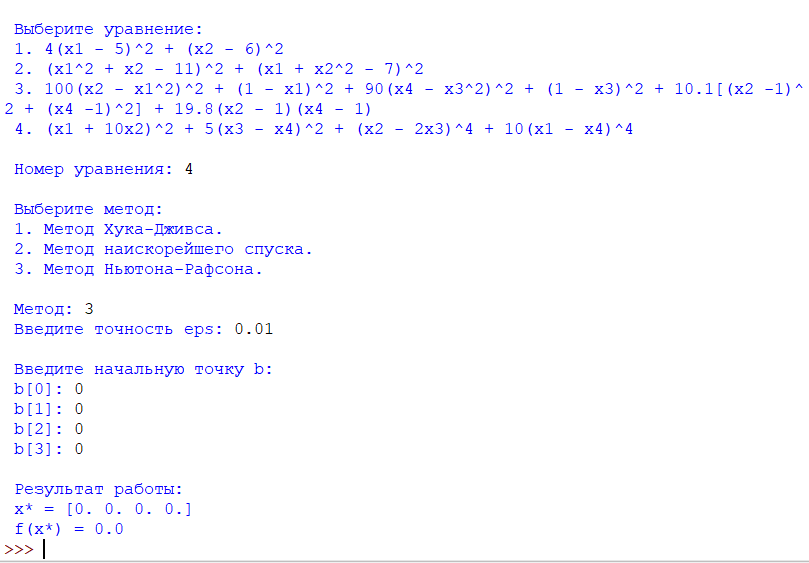
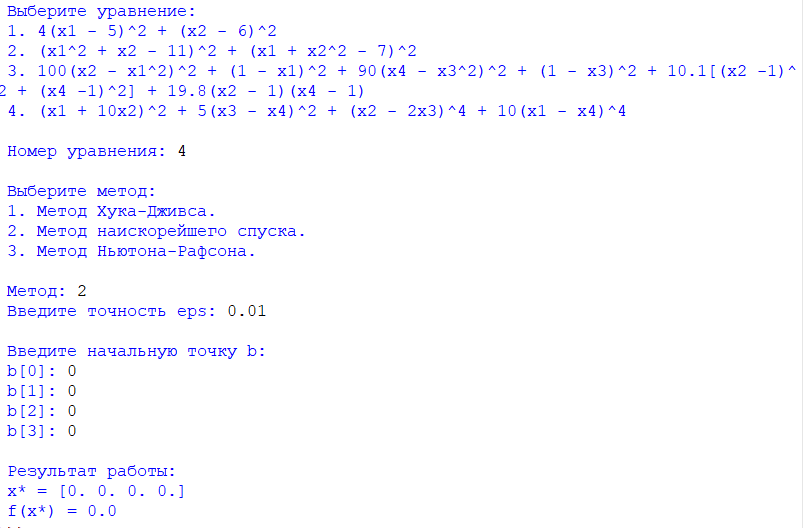
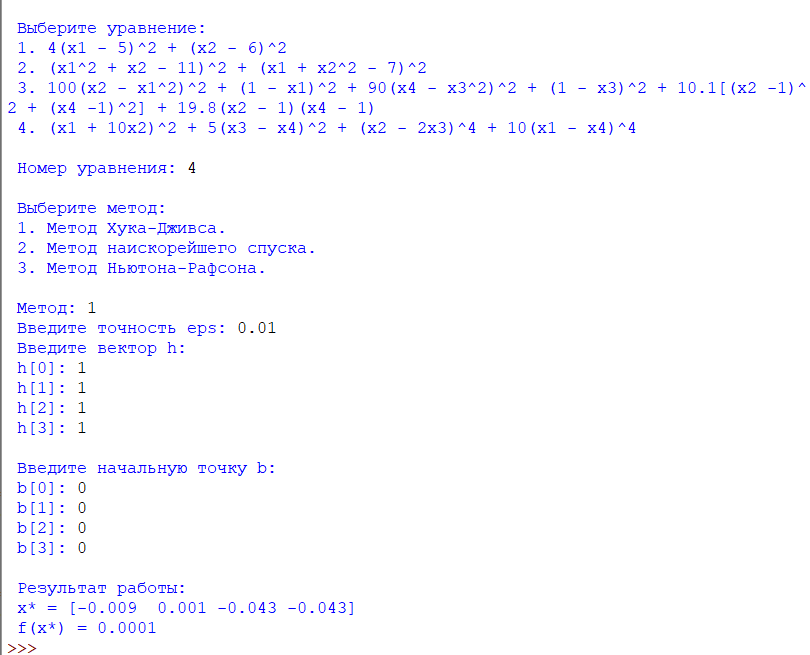
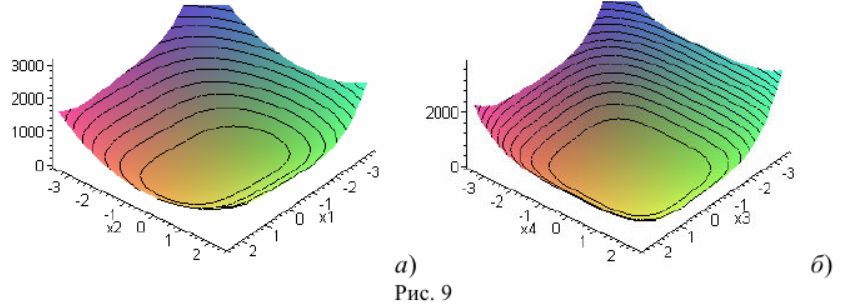
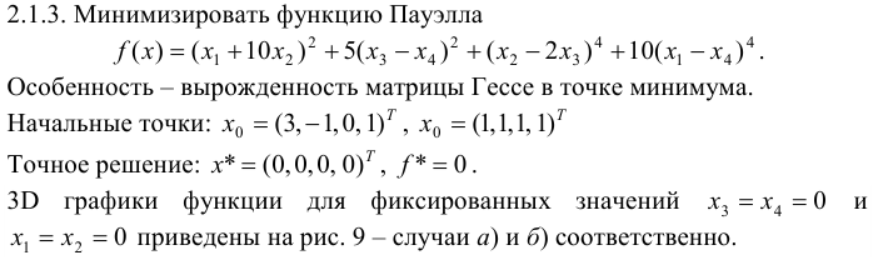












Код программы:

import numpy as np

import sympy.calculus.util

from sympy import \*

from sympy.calculus.util import minimum

from typing import Callable, List

from scipy import optimize

from scipy.optimize import minimize

import numdifftools as nd

Path = []

helper = 0.000000001

#Уравнения

def function\_ch(urav):

if urav == 1:

return lambda x: 4 \* (x[0] - 5) \*\* 2 + (x[1] - 6) \*\* 2

if urav == 2:

return lambda x: (x[0] \*\* 2 + x[1] - 11) \*\* 2 + (x[0] + x[1] \*\* 2 - 7) \*\* 2

if urav == 3:

return lambda x: 100 \* (x[1] - x[0] \*\* 2) \*\* 2 + (1 - x[0]) \*\* 2 + 90 \* (x[3] - x[2] \*\* 2) \*\* 2 + (1 - x[2]) \*\* 2 + 10.1 \* ((x[1] - 1) \*\* 2 + (x[3] - 1) \*\* 2) + 19.8 \* (x[1] - 1) \* (x[3] - 1)

if urav == 4:

return lambda x: (x[0] + 10 \* x[1]) \*\* 2 + 5 \* (x[2] - x[3]) \*\* 2 + (x[1] - 2 \* x[2]) \*\* 4 + 10 \* (x[0] - x[3]) \*\* 4

#Производная начальной функции

def f\_dif (b):

try:

if (urav == 1):

f\_xi = 4 \* (b[0] - 5)\*\*2 + (b[1] - 6)\*\*2

return f\_xi

if (urav == 2):

f\_xi = (b[0]\*\*2 + b[1] - 11)\*\*2 + (b[0] + b[1]\*\*2 - 7)\*\*2

return f\_xi

if (urav == 3):

f\_xi = 100 \* (b[1] - b[0]\*\*2)\*\*2 + (1 - b[0])\*\*2 + 90 \* (b[3] - b[2]\*\*2)\*\*2 + (1 - b[2])\*\*2 + 10.1 \* ((b[1] - 1)\*\*2 + (b[3] - 1)\*\*2) + 19.8 \* (b[1] - 1) \* (b[3] - 1)

return f\_xi

if (urav == 4):

f\_xi = (b[0] + 10 \* b[1])\*\*2 + 5 \* (b[2] - b[3])\*\*2 + (b[1] - 2 \* b[2])\*\*4 + 10 \* (b[0] - b[3])\*\*4

return f\_xi

except NameError:

print("\n\n Поменяйте x0 и h, они не подходят.")

exit()

#Метод наискорейшего спуска

def optimal\_gradient\_method(f: Callable[[List[float]], float], b: List[float], eps: float):

x = np.array(b)

def grad(f, xcur, eps) -> np.array:

return optimize.approx\_fprime(xcur, f, eps\*\*2)

gr = grad(f, x, eps)

a = 0.

iterat = 1

while any([abs(gr[i]) > eps for i in range(len(gr))]):

gr = grad(f, x, eps)

a = optimize.minimize\_scalar(lambda koef: f(\*[x+koef\*gr])).x

x += a\*gr

iterat += 1

if iterat > 20:

break

return x

#Метод Ньютона

def newton\_raphson(b: List[float], eps: float, f: Callable[..., float]):

xcur = np.array(b)

Path.append(xcur)

hess\_f = nd.Hessian(f)

n = len(b)

grad = optimize.approx\_fprime(xcur, f, eps \*\* 4) # step2

y = 0

while any([pow(abs(grad[i]), 1.5) > eps for i in range(n)]): # step3

y = y + 1

h = np.linalg.inv(hess\_f(xcur)) # step 4 & 5

pk = (-1 \* h).dot(grad) # step 6

a = optimize.minimize\_scalar(lambda a: f(xcur + pk \* a), bounds=(0,)).x # step7

xcur = xcur + a \* pk # step8

Path.append(xcur)

grad = optimize.approx\_fprime(xcur, f, eps \* eps) # step2

if y > 100:

break

return xcur # step10

urav = int((input("\n Выберите уравнение: \n 1. 4(x1 - 5)^2 + (x2 - 6)^2 \n 2. (x1^2 + x2 - 11)^2 + (x1 + x2^2 - 7)^2 \n \

3. 100(x2 - x1^2)^2 + (1 - x1)^2 + 90(x4 - x3^2)^2 + (1 - x3)^2 + 10.1[(x2 -1)^2 + (x4 -1)^2] + 19.8(x2 - 1)(x4 - 1) \n \

4. (x1 + 10x2)^2 + 5(x3 - x4)^2 + (x2 - 2x3)^4 + 10(x1 - x4)^4 \n\n Номер уравнения: ")))

list\_urav = list(range(1, 5))

iter = 0

if urav not in list\_urav:

while True:

urav = int((input(" Вы ввели неверный номер уравнения, пожалуйста, введите номер от 1 до 4: ")))

iter += 1

if (iter % 4 == 0):

print("\n Пожалуйста, выберите номер уравнения из следующего списка: \n 1. 4(x1 - 5)^2 + (x2 - 6)^2 \n 2. (x1^2 + x2 - 11)^2 + (x1 + x2^2 - 7)^2 \n \

3. 100(x2 - x1^2)^2 + (1 - x1)^2 + 90(x4 - x3^2)^2 + (1 - x3)^2 + 10.1[(x2 -1)^2 + (x4 -1)^2] + 19.8(x2 - 1)(x4 - 1) \n \

4. (x1 + 10x2)^2 + 5(x3 - x4)^2 + (x2 - 2x3)^4 + 10(x1 - x4)^4 \n\n Уравнение: ")

elif (urav in list\_urav):

break

method = int(input("\n Выберите метод: \n 1. Метод Хука-Дживса.\n 2. Метод наискорейшего спуска.\n 3. Метод Ньютона-Рафсона.\n\n Метод: "))

j = 0

if method not in [1, 2, 3]:

while True:

method = int(input(" Вы ввели неверный номер метода, пожалуйста, введите номер от 1 до 3: "))

j += 1

if (j % 4 == 0):

print("\n 1. Метод Хука-Дживса.\n 2. Метод наискорейшего спуска.\n 3. Метод Ньютона-Рафсона.\n\n Метод: ")

elif (method in [1 , 2]):

break

eps = float((input(" Введите точность eps: ")))

while eps <= 0:

eps = float((input(" Вы ввели eps, которой не соответствует условию: eps > 0 \n \

Пожалуйста введите подходящее значение eps: ")))

#пустые размерные матрицы для уравнений

if urav in [1, 2]:

h = np.zeros(2, float)

b = np.zeros(2, float)

f\_xi\_diff = np.zeros(2, float)

x\_st = np.zeros(2, float)

elif urav in [3, 4]:

h = np.zeros(4, float)

b = np.zeros(4, float)

f\_xi\_diff = np.zeros(4, float)

x\_st = np.zeros(4, float)

if method == 1:

z = 0.1

#z = float((input(" Введите z: ")))

print(" Введите вектор h: ")

for i in range(len(h)):

h[i] = float(input(f" h[{i}]: "))

print("\n Введите начальную точку b: ")

for i in range(len(b)):

b[i] = float(input(f" b[{i}]: "))

#Метод Хука-Дживса

#2.1.2.2

# Исследующий поиск

def utilSearch(b):

try:

key\_1st = 1 # step2

for i in range(0, len(b)):

if key\_1st == 1:

fb = f\_dif(b) # step1

b[i] = b[i] + h[i] \* 1

f = f\_dif(b) # step3

if f + helper< fb: # step4

fb = f

else:

b[i] = b[i] - 2 \* h[i] \* 1

f = f\_dif(b) # step5

if f + + helper< fb: # step6

fb = f

key\_1st = 0

return b, fb

except NameError:

print("Ошибка в исследующем поиске!")

#2.1.2.1

if method == 1:

k = 0

key\_3rd = 1

while True:

if key\_3rd == 1:

k += 1

xk = b # step1

# step2

b2, fb2 = utilSearch(xk) # 2.1.2.2

xk = b + 2 \* (b2 - b) # step3

x, fx = utilSearch(xk) # step4

b = b2 # step5

if fx + helper< f\_dif(b): # step6

b2 = x # step3

key\_3rd = 0

elif fx - helper> f\_dif(b): # step7

key\_3rd = 1

else: # step8

if urav in [1, 2]:

if pow((pow(h[0], 2) + pow(h[1], 2)), 0.5) <= eps + helper:

x\_st = b # step10

break

else: # step9

h = z \* h

key\_3rd = 1

elif urav in [3, 4]:

if pow((pow(h[0], 2) + pow(h[1], 2) + pow(h[2], 2) + pow(h[3], 2)), 0.5) <= eps + helper:

x\_st = b

break

else:

h = z \* h

key\_3rd = 1

answer = x\_st

print ("\n Результат работы:")

if method == 2:

function2 = function\_ch(urav)

answer = optimal\_gradient\_method(function2, b, eps)

print("\n Результат работы:")

if method == 3:

function3 = function\_ch(urav)

answer = newton\_raphson(b, eps, function3)

print("\n Результат работы:")

print (f" x\* = {np.round(answer, 4)} \n f(x\*) = {np.round(f\_dif(answer), 4)}")