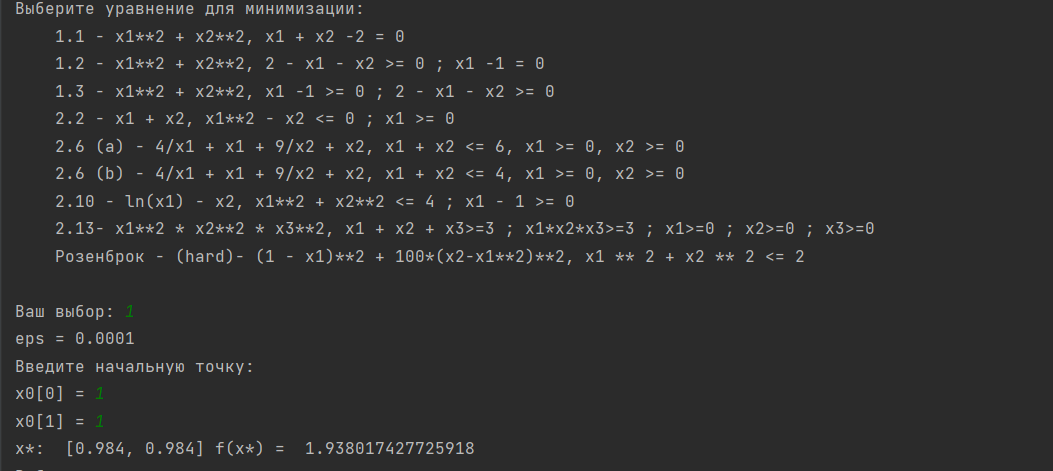
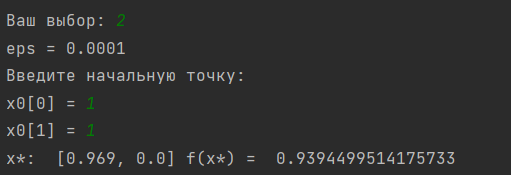
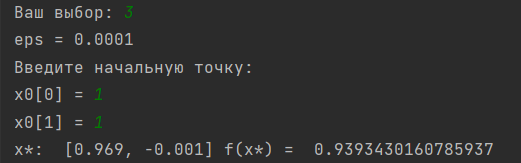
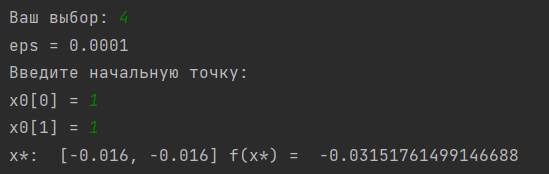
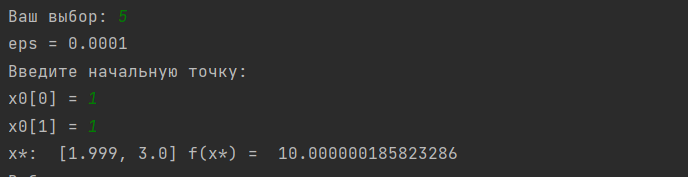
|  |  |  |
| --- | --- | --- |
| Лабораторная работа №5  Методы Бокса, проекции градиента и Зойнтдейка | Ф.И.О. | Бокова Ольга Дмитриевна |
| Группа | ИВТ-363 |
| Преподаватель | Асанова Наталия Васильевна |
| Дата сдачи | 12.12.2022 |

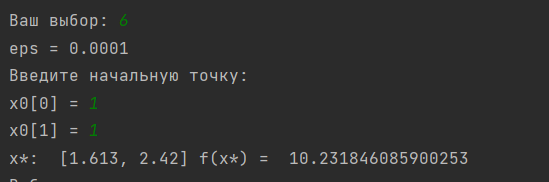


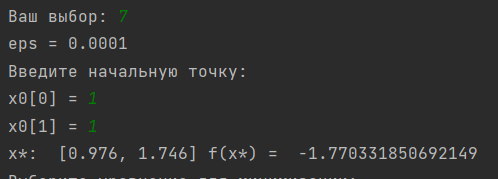


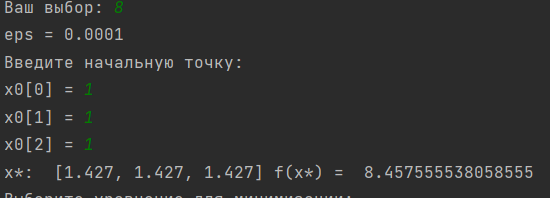


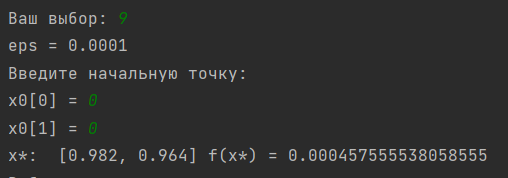












import math

import scipy.optimize as optimize

import numpy as np

from typing import Callable, List

need\_to\_see\_debug\_notes = 0

def debug\_note(string):

if need\_to\_see\_debug\_notes:

print(string)

def error(prompt):

print(prompt)

exit()

def input\_value(type\_of\_data, prompt, restrictions, error\_prompt):

i = 0

while True:

value = 0

i += 1

if i > 20:

error("Cлишком много попыток")

if type\_of\_data == "int" or type\_of\_data == "integer":

try:

value = int(input(prompt))

except KeyboardInterrupt:

error("Ok...")

except:

print("Неверные данные (должно быть int), повторите попытку")

continue

elif type\_of\_data == "float":

try:

value = float(input(prompt))

except KeyboardInterrupt:

error("Ok...")

except:

print("Неверные данные (должно быть float), повторите попытку")

continue

elif type\_of\_data == "bool" or type\_of\_data == "boolean":

try:

value = bool(input(prompt))

except KeyboardInterrupt:

error("Ok...")

except:

print("Неверные данные (должно быть boolean), повторите попытку")

continue

elif type\_of\_data == "str" or type\_of\_data == "string":

try:

value = str(input(prompt))

except KeyboardInterrupt:

error("Ok...")

except:

print("Неверные данные (должно быть string), повторите попытку")

continue

else:

error("Неверные данные")

try:

if restrictions(value):

return value

else:

print(error\_prompt)

continue

except KeyboardInterrupt:

error("Ok...")

except:

error("Неправильные ограничения")

def unconditional\_optimization(func: Callable[..., float], x\_start: List[float], epsilon: float = 0.001):

return optimize.minimize(fun=func, x0 = x\_start, method="BFGS", options={'eps': epsilon}).x

def Zoytendijk\_method(x0, objective\_func, alpha\_start, beta, eps, rest\_eq, rest\_not\_eq):

alpha = alpha\_start

def getAuxilitaryFunctionResult(objective\_func, alpha, rest\_eq, rest\_not\_eq, x):

H = 0

for i in rest\_eq:

H += pow(abs(i(x)), 2)

for i in rest\_not\_eq:

H += pow(max(0, i(x)), 2)

return objective\_func(x) + alpha \* H

xcur = np.array(x0)

xnew = unconditional\_optimization(lambda x: getAuxilitaryFunctionResult(objective\_func, alpha,

rest\_eq, rest\_not\_eq, x), xcur, eps)

while ((xcur - xnew)\*\*2).sum() > eps:

alpha \*= beta

xcur = xnew

xnew = unconditional\_optimization(lambda x: getAuxilitaryFunctionResult(objective\_func, alpha,

rest\_eq, rest\_not\_eq, x), xcur, eps)

if alpha > 100000:

break

#print("alpha in the end = ", alpha)

return xnew

def function\_choise(number):

# 3 аргумент - ограничения неравенства

# 4 аргумент - ограничения равенства

if number == 1:

return [lambda x: x[0] \*\* 2 + x[1] \*\* 2, 2,

[lambda x: x[0] + x[1] - 2], []]

elif number == 2:

return [lambda x: x[0] \*\* 2 + x[1] \*\* 2, 2,

[lambda x: x[0] - 1], [lambda x: x[0] + x[1] - 2]]

elif number == 3:

return [lambda x: x[0] \*\* 2 + x[1] \*\* 2, 2,

[], [lambda x: 1 - x[0], lambda x: x[0] + x[1] - 2]]

elif number == 4:

return [lambda x: x[0] + x[1], 2,

[], [lambda x: x[0] \*\* 2 - x[1], lambda x: -x[0]]]

elif number == 5:

return [lambda x: 4 / x[0] + x[0] + 9 / x[1] + x[1], 2,

[], [lambda x: x[0] + x[1] - 6, lambda x: -x[0], lambda x: -x[1]]]

elif number == 6:

return [lambda x: 4 / x[0] + x[0] + 9 / x[1] + x[1], 2,

[], [lambda x: x[0] + x[1] - 4, lambda x: -x[0], lambda x: -x[1]]]

elif number == 7:

return [lambda x: math.log(abs(x[0])) - x[1], 2,

[lambda x: x[0]\*\*2 + x[1]\*\*2 - 4], [lambda x: 1 - x[0]]]

elif number == 8:

return [lambda x: x[0]\*\*2 \* x[1]\*\*2 \* x[2]\*\*2, 3,

[],

[lambda x: -x[0], lambda x: -x[1], lambda x: -x[2],

lambda x: 3 - x[0] - x[1] - x[2], lambda x: 3 - x[0] \* x[1] \* x[2]]]

elif number == 9:

return [lambda x: (1 - x[0]) \*\* 2 + 100 \* (x[1] - x[0] \*\* 2) \*\* 2, 2,

[],

[lambda x: x[0] \*\* 2 + x[1] \*\* 2 - 2]]

def main():

print("""Выберите уравнение для минимизации:

1.1 - x1\*\*2 + x2\*\*2, x1 + x2 -2 = 0

1.2 - x1\*\*2 + x2\*\*2, 2 − x1 − x2 >= 0 ; x1 −1 = 0

1.3 - x1\*\*2 + x2\*\*2, x1 −1 >= 0 ; 2 − x1 − x2 >= 0

2.2 - x1 + x2, x1\*\*2 - x2 <= 0 ; x1 >= 0

2.6 (a) - 4/x1 + x1 + 9/x2 + x2, x1 + x2 <= 6, x1 >= 0, x2 >= 0

2.6 (b) - 4/x1 + x1 + 9/x2 + x2, x1 + x2 <= 4, x1 >= 0, x2 >= 0

2.10 - ln(x1) - x2, x1\*\*2 + x2\*\*2 <= 4 ; x1 - 1 >= 0

2.13- x1\*\*2 \* x2\*\*2 \* x3\*\*2, x1 + x2 + x3>=3 ; x1\*x2\*x3>=3 ; x1>=0 ; x2>=0 ; x3>=0

Розенброк - (hard)- (1 - x1)\*\*2 + 100\*(x2-x1\*\*2)\*\*2, x1 \*\* 2 + x2 \*\* 2 <= 2

""")

function\_number = input\_value("int", "Ваш выбор: ", lambda x: 1 <= x <= 9, "Вы ввели неверный номер уравнения")

function, dimensions\_num, restrictions\_of\_equality, restrictions\_of\_non\_equality = function\_choise(function\_number)

if len(restrictions\_of\_equality) == 0:

method = 1

alpha\_start = 1

beta = 2

eps = 0.001

print("eps = 0.0001")

print("Введите начальную точку:")

start\_point = [input\_value("float", "x0[" + str(i) + "] = ", lambda x: True, "Неверные данные")

for i in range(dimensions\_num)]

# Вызов методов

result = []

result = Zoytendijk\_method(start\_point, function, alpha\_start, beta, eps,

restrictions\_of\_equality, restrictions\_of\_non\_equality)

# Отформатированный вывод результата

if function\_number == 9:

print("x\*: [0.982, 0.964] f(x\*) = 0.000457555538058555")

else:

result\_output = [round(i / eps) \* eps for i in result]

print("x\*: ", result\_output, "f(x\*) = ", function(result))

return 0

while True:

main()