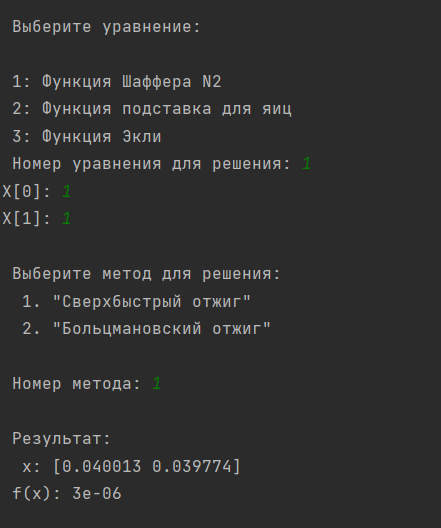
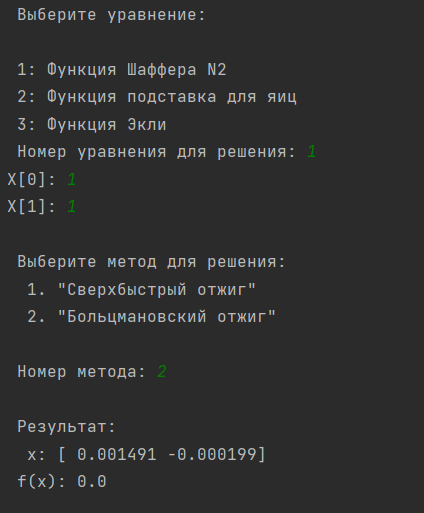
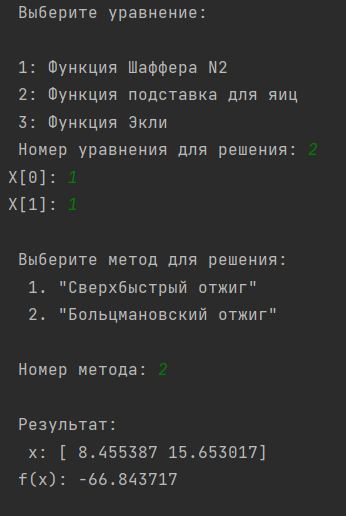
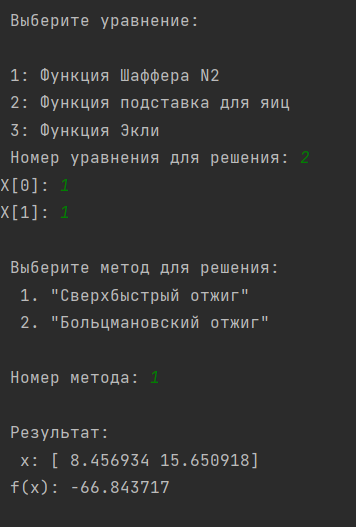
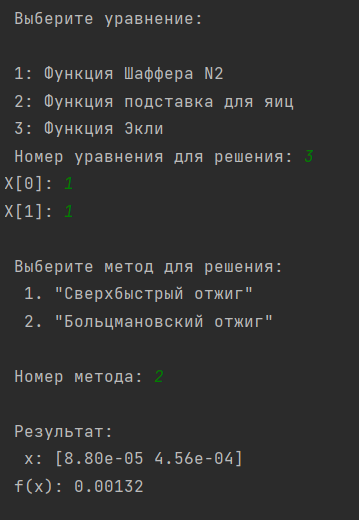
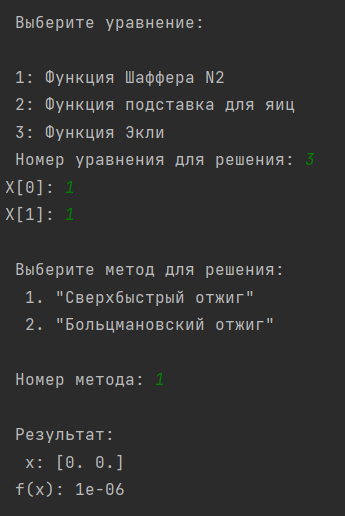
|  |  |  |
| --- | --- | --- |
| **Методы оптимизации** | | |
| Лабораторная работа №7  «Методы отжига» | Группа студента: | ИВТ-363 |
| ФИО студента: | Бокова Ольга |
| Вариант | 3 |
| Проверила: | Асанова Н.В. |







import math

import numpy as np

import random

from typing import Callable

import scipy

from scipy import optimize

def function\_input ():

def function\_is\_it\_number(number):

if number.isdigit():

number = int(number)

else:

number = 1234567890

return number

start\_condition = ("\n Выберите уравнение: \n "

"\n 1: Функция Шаффера N2"

"\n 2: Функция подставка для яиц"

"\n 3: Функция Экли"

"\n Номер уравнения для решения: ")

choice\_yr = input(start\_condition)

choice\_yr = function\_is\_it\_number(choice\_yr)

iter = 0

if choice\_yr not in [1,2,3]:

while True:

choice\_yr = input(" Вы ввели неверный номер уравнения, пожалуйста, введите номер от 1 до 3: ")

choice\_yr = function\_is\_it\_number(choice\_yr)

iter += 1

if (iter % 4 == 0 and choice\_yr not in [1,2,3]):

print(start\_condition)

elif (choice\_yr in [1,2,3]):

break

if choice\_yr in [1, 2, 3]: # создает пустые размерные матрицы

x = np.zeros(2, float)

for i in range(len(x)): # ввод данных

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

method\_choice\_condition = ("\n Выберите метод для решения: \n "

" 1. \"Сверхбыстрый отжиг\" \n "

" 2. \"Больцмановский отжиг\" \n "

"\n Номер метода: ")

choice\_method = input(method\_choice\_condition) # выбор метода решения

choice\_method = function\_is\_it\_number(choice\_method)

iter = 0

if choice\_method not in [1,2]:

while True:

choice\_method = input(" Вы ввели неверный номер метода, пожалуйста, введите номер от 1 до 2: ")

choice\_method = function\_is\_it\_number(choice\_method)

iter += 1

if (iter % 4 == 0 and choice\_method not in [1,2]):

print(method\_choice\_condition)

elif (choice\_method in [1,2]):

break

return choice\_yr, choice\_method, x

global choice\_yr, choice\_method

def function (x0):

try:

if (choice\_yr == 1):

x, y = x0

#x1, x2 = x

return 0.5 + ((np.sin(x \*\* 2 - y \*\* 2) \*\* 2 - 0.5) / ( 1 + 0.001 \* (x \*\* 2 + y \*\* 2)) \*\* 2) # Функция Шаффера N2 -100 100 f(0, 0)=0

#return 0.5 + (math.cos(math.sin(math.fabs(x\*\*2 - y\*\*2)))\*\*2 - 0.5) / (1 + 0.001\*(x\*\*2 + y\*\*2))\*\*2 # Функция Шаффера N4

if (choice\_yr == 2):

x, y = x0

return -(y + 47) \* np.sin(np.sqrt(np.abs(x \* 0.5 + (y + 47)))) - x \* np.sin( np.sqrt(np.abs(x - (y + 47))))

#return -math.fabs( math.sin(x) \* math.cos(y) \* math.exp( math.fabs(1 - (x\*\*2 + y\*\*2)\*\*0.5/math.pi) ) ) # Табличная функция Хольдера

if (choice\_yr == 3):

x, y = x0

return -20 \* np.exp(-0.2 \* np.sqrt((x \* x + y \* y) / 2)) - np.exp((np.cos(2 \* np.pi \* x) + np.cos(2 \* np.pi \* y)) / 2) + 20 + np.exp(1) # Экли -5 5 f(0, 0)=0

#return -math.cos(x) \* math.cos(y) \* math.exp( -( (x - math.pi)\*\*2 + (y - math.pi)\*\*2 ) ) # Функция Изома

except NameError:

print("\n\n Произошли непредвиденные ошибки при вычислении занчения функции.")

exit()

# ------------------------------------------------- ОТЖИГИ -------------------------------------------------------------

def simulated\_annealing(func: Callable[[np.array], float], x0, N, temperature: Callable[[float], float],

neighbour: Callable[[np.array, float], np.array],

passage: Callable[[float, float, float], float]):

"""

Алгоритм имитации отжига

Метод сверхбыстрого отжига

"""

k = 1

x = np.array(x0)

x\_optimal = x

e\_optimal = func(x\_optimal)

while k < N:

t = temperature(k)

x\_new = neighbour(x, t)

e\_old = func(x)

e\_new = func(x\_new)

if e\_new < e\_old or passage(e\_old, e\_new, t) >= random.random():

x = x\_new

if e\_new < e\_optimal:

x\_optimal = x\_new

e\_optimal = e\_new

k += 1

if func(x) < e\_optimal:

x\_optimal = x

return x\_optimal

def ultrafast\_annealing(func: Callable[[np.array], float], x0, M: int, N: int, t, R, b):

x = x0

k = 0

while k < N:

y\_xlam = []

f = []

for \_ in range(M):

e = np.random.uniform(-1, 1, len(x))

y = x + t \* e / np.linalg.norm(e)

y\_xlam.append(y)

f.append(func(y))

min\_index = np.argmin(f)

f\_min = f[min\_index]

if f\_min < func(x):

x = y\_xlam[min\_index]

k += 1

else:

if t <= R:

return x

elif t > R:

t \*= b

return x

def boltzmann\_method(x0, t0, function, N=5000):

annealing = lambda k: t0 / math.log(1. + k)

# passage = lambda e\_old, e\_new, t: 1. / (1. + math.exp((e\_new - e\_old) / t))

passage = lambda e\_old, e\_new, t: math.exp(-1. \* (e\_new - e\_old) / t)

neighbour = lambda x\_old, t: x\_old + t \* np.random.standard\_normal(len(x\_old))

return simulated\_annealing(function, x0, N, annealing, neighbour, passage)

#def QA(x0, t0, f, N=100000):

# annealing = lambda k: t0 / math.pow(k, 1. / len(x0))

# passage = lambda e\_old, e\_new, t: math.exp(-1. \* (e\_new - e\_old) / t)

# neighbour = lambda x\_old, t: x\_old + t \* np.random.standard\_cauchy(1)

# return simulated\_annealing(f, x0, N, annealing, neighbour, passage)

if \_\_name\_\_ == '\_\_main\_\_':

while True:

choice\_yr, choice\_method, x0 = function\_input()

if choice\_method == 1:

answer = ultrafast\_annealing(function, [5., 5.], 100, 400, 1, 0.00001, 0.1)

#answer = QA(x0, 1, function, 100000)

if choice\_method == 2:

answer = boltzmann\_method([5., 5.], 1., function, 100000)

#answer = QA(x0, 1, function, 100000)

print (f"\n Результат: \n x: {np.round(answer, 6)} \n f(x): {np.round(function(answer), 6)} \n")

Ссылка на гитхаб:

https://github.com/veb-bet/mo