ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент, к.т.н., доцент		А.А.Сенцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

ГЛОБАЛЬНАЯ ОПТИМИЗАЦИЯ

по курсу: МЕТОДЫ ОПТИМИЗАЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4242М		М.В.Климов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

1. Цель работы и вариант

Ознакомиться с известными численными методами глобальной безусловной оптимизации, получить опыт решения глобальной оптимизационной задачи, приобрести навыки использования математических программных средств для решения задач на глобальный экстремум.

Вариант 7.

7. **Функция**:

$$f(x,y) = -\frac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}.$$

Интервал поиска по каждой переменной [–8; 8], минимум при $x = y = 0$.

37. **Функция**:

$$f(x,y) = 100\left(y - x^3\right)^2 + (1-x)^2.$$

Интервал поиска по каждой переменной [–5; 5], минимум при $x = 1, y = 1$.

Методом Basin-hopping

2. Вычисления:

Basin-hopping - это один из методов, основанный на идее многократных попыток локальной оптимизации из различных случайных начальных точек. Алгоритм глобальной оптимизации «basin-hopping» для поиска минимума функции f(x) можно описать следующим образом.

1. Любым детерминированным методом найти x* – точку минимума функции f(x) из случайной начальной точки x0.

2. Если f(x*) наименьшего значения найденного ранее, обнулить счетчик неудачных попыток (t = 0), запомнить полученное значение как наименьшее, перейти к шагу 1. Иначе t = t + 1.

3. Если t > tmax, завершить алгоритм. Иначе перейти к шагу 1.

Алгоритм завершается, когда tmax раз не удается получить локальный минимум, меньший найденного ранее.

Методом для поиска локального минимума в пределах начальных точек был выбран перебор ближайших значений в с шагом e и переход к наименьшему из них.

Программа получает на вход максимально допустимое количество неудачных попыток нахождения минимума от случайно выбранной начальной точки. Далее идёт подсчёт минимумов от случайно выбранных точек, и, если найденный минимум меньше запомненного (по-умолчанию, меньше первого подсчитанного), программа берёт его за новый эталон и обнуляет счётчик неудачных попыток. Иначе, счётчик инкрементируется. При достижении на счётчике числа, данного на вход, подсчёт заканчивается и выводятся все найденные минимумы при каждой из начальных точек. Тот же алгоритм используется и для второй функции.

Для защиты от ввода некорректных начальных данных максимум неудачных попыток была взята переменная типа float, берущаяся под модулем. При попытке пользователем ввести вместо этой переменной отрицательного или дробного числа, подсчёт всё равно будет вестись до ближайшего целого числа, которое больше или равно, чем модуль введённого числа.

Код программы приведён в приложении 1, а результат выполнения представлен на рисунках 1-3.

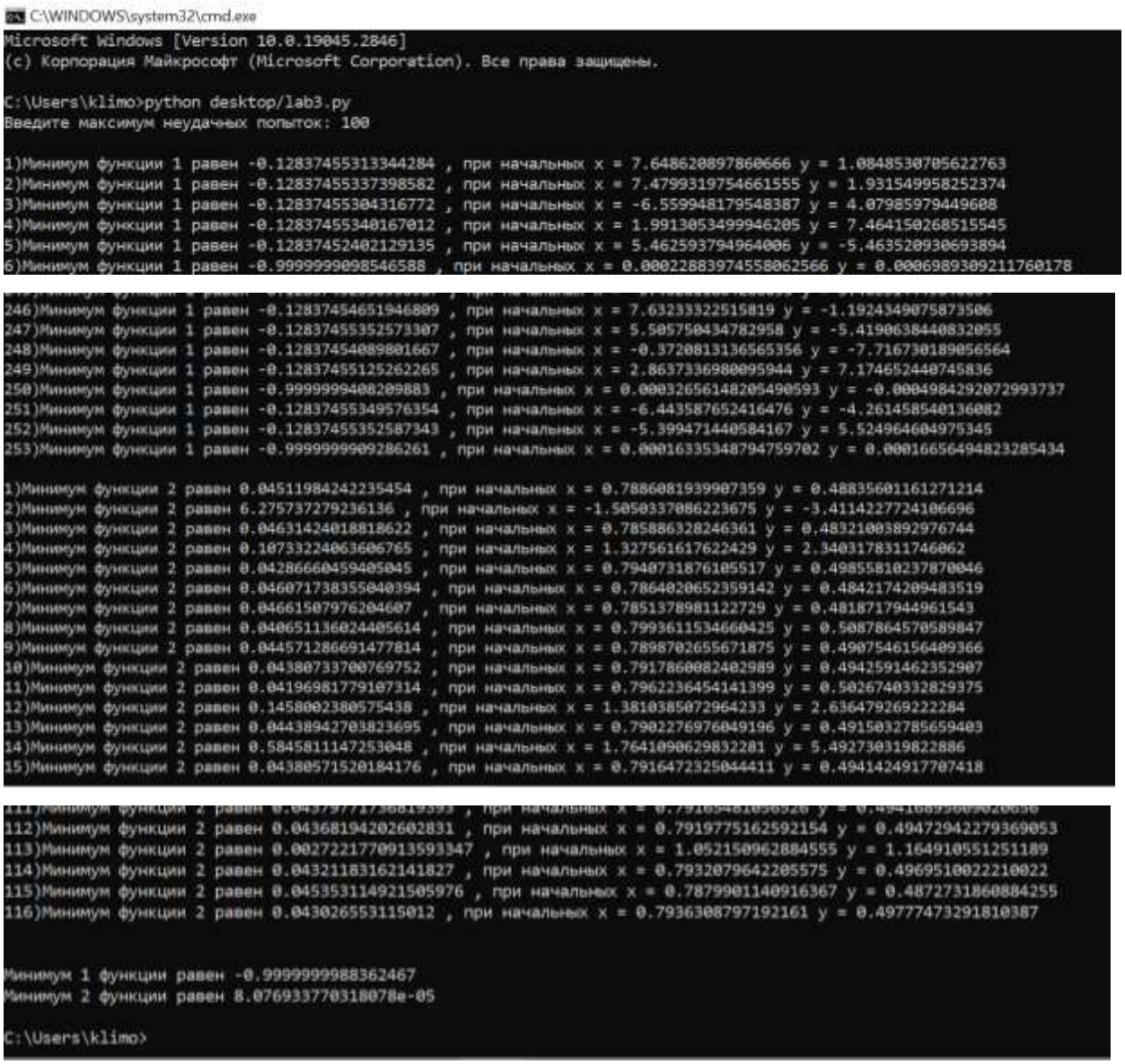

Рисунок 1-3. Результат выполнения программы в консоли.

3. Выводы.

В процессе выполнения лабораторной работы была решена поставленная задача по нахождению минимума двух функций методом Basin-hopping. При тестировании результаты отличались даже при начальных данных, однако одни были в пределах 0.001 от предполагаемого результата, вследствие чего, можно сказать, что функции решались верно.

ПРИЛОЖЕНИЕ 1

Содержимое программы

```
import numpy as np
import matplotlib.pyplot as plt
import math;
import random;

def fun7(x):
    return  (-1*(math.sin(((x[1,0]  **  2)  +  (x[0,0]  **  2))  **
(1/2)))/(((x[1,0] ** 2) + (x[0,0] ** 2)) ** (1/2)))

def fun37(x):
    return ((100*((x[1,0] - (x[0,0] ** 3)) ** (2)))+((1 - x[0,0]) **
(2)))

def dfp1(fun,  x0, e):
    result = []
    n = 0
    x1 =  np.mat([[1.0], [1.0]])
    x2 =  np.mat([[1.0], [1.0]])
    x3 =  np.mat([[1.0], [1.0]])
    x4 =  np.mat([[1.0], [1.0]])
    x =  np.mat([[1.0], [1.0]])
    x1[0,0] = x0[0,0] + e
    x1[1,0] = x0[1,0] + e
    x2[0,0] = x0[0,0] - e
    x2[1,0] = x0[1,0] - e
    x3[0,0] = x0[0,0] + e
    x3[1,0] = x0[1,0] - e
    x4[0,0] = x0[0,0] - e
    x4[1,0] = x0[1,0] + e
    x[0,0] = x1[0,0]
    x[1,0] = x1[1,0]
    while ((x[0,0] != x0[0,0]) and (x[1,0] != x0[1,0])):
```

```
        x[0,0] = x0[0,0]
        x[1,0] = x0[1,0]
        if (fun(x1) < fun(x0)):
            x0[0,0] = x1[0,0]
            x0[1,0] = x1[1,0]
        else:
            if (fun(x2) < fun(x0)):
                x0[0,0] = x2[0,0]
                x0[1,0] = x2[1,0]
            else:
                if (fun(x3) < fun(x0)):
                    x0[0,0] = x3[0,0]
                    x0[1,0] = x3[1,0]
                else:
                    if (fun(x4) < fun(x0)):
                        x0[0,0] = x4[0,0]
                        x0[1,0] = x4[1,0]
        x1[0,0] = x0[0,0] + e
        x1[1,0] = x0[1,0] + e
        x2[0,0] = x0[0,0] - e
        x2[1,0] = x0[1,0] - e
        x3[0,0] = x0[0,0] + e
        x3[1,0] = x0[1,0] - e
        x4[0,0] = x0[0,0] - e
        x4[1,0] = x0[1,0] + e
        result.append(fun(x0))
        n=n+1

    return result


tmax = abs(float(input('Введите максимум неудачных попыток: ')))
e = 0.001
n = 0
```

```
resultfun = []
resultx = []
resulty = []
resultfun1 = []
resultx1 = []
resulty1 = []

x0 =  np.mat([[16*random.random() - 8], [16*random.random() - 8]])
result = dfp1(fun7, x0, e)
n = len(result)
min0 = result[n-1]
resultfun.append(min0)
resultx.append(x0[0,0])
resulty.append(x0[1,0])
t = 0
while (t<tmax):
    x =  np.mat([[16*random.random() - 8], [16*random.random() - 8]])
    result1 = dfp1(fun7, x, e)
    n = len(result1)
    min1 = result1[n-1]
    resultfun.append(min1)
    resultx.append(x[0,0])
    resulty.append(x[1,0])
    if (min1<min0):
        result = result1
        min0 = min1
        x0 = x
        t = 0
    else:
        t = t+1
print(f'')
n = len(resultfun)
i = 0
while(i<n):
```

```
    print(f'{i+1})Минимум функции 1 равен {resultfun[i]} , при начальных
x = {resultx[i]} y = {resulty[i]}')
    i = i+1

mini0 = min0

x0 =  np.mat([[10*random.random() - 5], [10*random.random() - 5]])
result = dfp1(fun37, x0, e)
n = len(result)
min0 = result[n-1]
resultfun1.append(min0)
resultx1.append(x0[0,0])
resulty1.append(x0[1,0])
t = 0
while (t<tmax):
    x =  np.mat([[10*random.random() - 5], [10*random.random() - 5]])
    result1 = dfp1(fun37, x, e)
    n = len(result1)
    min1 = result1[n-1]
    resultfun1.append(min1)
    resultx1.append(x[0,0])
    resulty1.append(x[1,0])
    if (min1<min0):
        result = result1
        min0 = min1
        x0 = x
        t = 0
    else:
        t = t+1
print(f'')
n = len(resultfun1)
i = 0
while(i<n):
```

```
    print(f'{i+1})Минимум функции 2 равен {resultfun1[i]} , при
начальных x = {resultx1[i]} y = {resulty1[i]}')
    i = i+1
print(f'')
print(f'')
print(f'Минимум 1 функции равен {mini0}')
print(f'Минимум 2 функции равен {min0}')
```

ПРИЛОЖЕНИЕ 2

Вопросы:

1. Метод штрафных функций. Описание и пример.

Метод штрафных функций так же, как и аналитические методы условной оптимизации, основан на переходе от задачи с ограничением к задаче без ограничений:

$$F(\mathbf{x}) = f(\mathbf{x}) + rP(\mathbf{x}) \to \min$$

Здесь r – некоторое число, называемое величиной штрафа.

P(x) – функция штрафа (penalty function), удовлетворяющая условиям:

$$P(\mathbf{x}) = \begin{cases} 0, \text{если } \mathbf{q}(\mathbf{x}) = 0, g_i(\mathbf{x}) \ge 0, i = \overline{1,s}; \\ > 0, \text{если } \mathbf{q}(\mathbf{x}) \ne 0 \text{ или } \exists g_i(\mathbf{x}) < 0, i = \overline{1,s}. \end{cases}$$

Функция штрафа равна нулю, если все ограничения выполняются и больше нуля, если нарушено хотя бы одно из ограничений.

Наиболее простые способы формирования такой функции:

$$P(\mathbf{x}) = \sum_{j=1}^{m} q_j^2(\mathbf{x}) - \sum_{i=1}^{s} \min(0, g_i(\mathbf{x})),$$

$$P(\mathbf{x}) = \sum_{j=1}^{m} |q_j(\mathbf{x})| - \sum_{i=1}^{s} \min(0, g_i(\mathbf{x})).$$

Алгоритм поиска экстремума методом штрафных функций можно сформулировать в виде трех шагов.

Шаг 1. Выбрать произвольное начальное значение r. Обычно выбирают r = 1.

Шаг 2. Найти решение x* задачи каким-либо численным методом.

Шаг 3. Если P(x *) > ε, увеличить r и перейти к шагу 2. Иначе: x* – искомая точка

минимума.

В качестве начальной точки при численном решении задачи безусловной оптимизации на шаге 2 рекомендуется выбирать точку x*, полученную на предыдущей итерации метода штрафных функций.

Пример:

Экстремум квадратичной формы при линейном ограничении. Пусть нужно найти точку минимума функции $f(x) = x_1^2 + x_2^2$ при ограничении $q(x) = 2 – x_1 – x_2 = 0$.

Составим функцию:

$F(x) = x_1^2 + x_2^2 + r(x_1 + x_2 – 2)^2$ -> min

Возьмем для начала r = 1. Минимум можно искать любым известным численным

методом. Получим ответ:

$x_1 = x_2 = 0{,}66$.

Посчитаем значение штрафа:

$P = (2 \cdot 0{,}66 – 2)^2 = 0{,}46$.

Таким образом, мы нашли ответ с точностью $\varepsilon = 0{,}46$. Увеличим r до r = 2:

$x_1 = x_2 = 0{,}80$

Точность решения составляет $P = (2 \cdot 0{,}8 – 2)^2 = 0{,}16$.

2. Градиент и его применение для решения задач оптимизации, пример

Градиентом называют вектор, составленный из частных производных функции в какой-либо точке.

Точки, в которых градиент равен 0 называют стационарными.

Также понятие градиента входит в необходимое условие экстремума функции: *Если x0 является точкой экстремума функции, определённой в окрестности точки x0, то градиент функции в точке x0 либо не существует, либо равен нулю.*

Следовательно, точки, в которых градиент равен 0 или не существует, называют критическими точками.

Также, одним из наиболее ярких методов применения градиента, можно назвать метод градиентного спуска, использующийся для многомерной локальной оптимизации. Суть данного метода заключается в том, чтобы взять начальную точку для функции, вычислить в ней градиент и изменить

рассматриваемую координату на градиент в данной точке, взятый с каким-то множителем, получив таким образом следующую точку, более близкую к экстремуму:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \nabla f(\mathbf{x}_i)$$

Пример:

Пусть дана функция $f(x) = x_1^2 + x_2^2$. Требуется найти минимум с заданной точностью.

Частные производные данной функции равны $[2x_1, 2x_2]$

Возьмём начальную точку [1,1] и множитель приращения равный 0.75

x0 = [1,1]

x1 = x0 – 0.75*[2,2] = [-0.5,-0.5]

x2 = x1 – 0.75*[-1,-1] = [0.25,0.25]

x3 = x2 – 0.75*[0.5,0.5] = [-0.125,-0.125]

…

Таким образом можно заметить, что с каждым циклом подсчёт приближается к точке [0,0], которая и является искомым экстремумом. И несмотря, на то, что данного значения таким образом достигнуть не удастся, но всё же удастся приблизиться к нему на расстояние равное заданной точности.

3. Оценить вычислительную сложность реализации алгоритма, программа которой приведена в отчете.

Простите, но я не совсем понимаю, как это оценивать и по каким критериям. Не могли бы вы уточнить, как это примерно делается и что именно нужно оценивать?

Первое и единственное, что приходит в голову это отметить, что алгоритм локальной оптимизации вычисляет функцию 3, 5, 7 или 9 раз за каждый цикл (в последней итерации цикла функция всегда вычисляется 9 раз). Данное количество вычислений можно изменить на 5 вычислений за каждый цикл, если перед сравнением «запоминать» все результаты подсчёта функции.