МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Липецкий государственный технический университет
Факультет автоматизации и информатики

Домашняя работа №4
по математическому программированию

Студент Станиславчук С. М.
Группа АС-21-1

Руководитель Качановский Ю. П.

Липецк 2023 г.

Содержание

1. Задание

Вариант: 14

Метод Нелдера-Мида для функции:

f(x)=((x_1-5)^2)/2 + ((x_2-3)^2)/3+4,

x_1=(-2, +7)^T,

x_2=(-2, +7)^T

При lamda = 2, alpha = 1, beta = 0.5, gamma = 2

Минимальное число отражений: 4

2. Теория

Алгоритм заключается в формировании симплекса (simplex) и последующего его деформирования в направлении минимума, посредством трех операций:

1) Отражение (reflection);
2) Растяжение (expansion);
3) Сжатие (contract);

Инициализация симплекса: начинаем с исходной точки (вектора переменных) и создаем симплекс, который представляет собой набор точек в пространстве переменных.

Оценка значений функции: для каждой точки симплекса вычисляются значения целевой функции.

Сортировка точек: Точки симплекса сортируются по значениям функции, так что лучшая точка (с наименьшим значением) становится первой, а худшая точка - последней.

Центроид: вычисляется центроид симплекса, который представляет собой среднее значение координат всех точек симплекса, за исключением худшей.

Отражение: относительно центроида проводится отражение худшей точки. Это создает новую отраженную точку.

Оценка отраженной точки: оценивается значение функции в отраженной точке.

Выбор действия: В зависимости от результатов отражения могут выполняться следующие действия:

Если отраженная точка лучше (имеет меньшее значение функции), чем лучшая точка, то проводится растяжение.
Если отраженная точка лучше, чем вторая худшая точка, но не лучше лучшей, то отраженная точка заменяет худшую точку.
Если отраженная точка не дает улучшения, проводится сжатие.
Если ни отражение, ни растяжение, ни сжатие не улучшают ситуацию, выполняется редукция (уменьшение размера симплекса).
Проверка критериев останова: проводятся проверки на достижение максимального числа итераций, отсутствие улучшений и другие критерии останова. Если одно из условий выполняется, алгоритм завершает работу и возвращает лучшую точку.

Алгоритм продолжает эти шаги до тех пор, пока не выполняются критерии останова. В результате работы алгоритма получается набор точек, а лучшая из них считается приближенным оптимальным решением.

Основные параметры алгоритма, такие как коэффициенты отражения (alpha), растяжения (gamma), сжатия (beta) и уменьшения симплекса (_lambda), заданы по условию.

3. Решение на ЯП Python с комментариями

Описание метода Нелдера-Мида с приведенными шагами в программном коде:

3.1 Описание алгоритма на примере программы

1) Инициализация

Здесь мы объявляем такие переменные как: шаг, начальная точка, точки останова (конец, если нет улучшений, макс. итераций), размерность пространства параметров, первоначальное значение целевой функции в начальной точке, счетчик итераций без улучшений, хранение результатов в виде кортежа: [[точка, значение], ...], а также параметры alpha, beta, gamma, lamda.

А также цикл, который создает начальный симплекс вокруг начальной точки x_start. Для каждой координаты i в диапазоне от 0 до dim-1, создается новая точка x, которая отличается от x_start только в этой координате на величину step. Этот симплекс затем используется как исходный.

```
def nelder_mead(f, x_start,
                step=0.1, no_improve_thr=10e-6,
                no_improv_break=10, max_iter=0,
                alpha=1., gamma=2., beta=0.5, _lambda=2):


dim = len(x_start)            # Размерность пространства параметров
prev_best = f(x_start)        # Первоначальное значение целевой функции
not_improved = 0              # Счетчик итераций без улучшений
res = [[x_start, prev_best]]  # Хранение результатов


    # Цикл, генерирующий исходный симплекс
    for i in range(dim):
        x = copy.copy(x_start)
        x[i] = x[i] + step
        score = f(x)
        res.append([x, score])


iters = 0      # Объявление переменной для подсчета числа итераций
```

2) Главный цикл, в котором и происходят все вычисления.

Бесконечный цикл while True:

```
while True:
```

2.0) Сортировка

```
res.sort(key=lambda x: x[1])  # Лучшая точка (минимум) будет в начале списка.
best = res[0][1]              # Текущее лучшее значение целевой функции
```

2.1) Проверка на точку останова

```
if max_iter and iters >= max_iter:
    return res[0]
iters += 1              # Увеличиваем счетчик итераций.
```

2.2) Вывод в консоль результатов

```
print (f'Лучшее значение среди всех минимумов на итерации [{iters}]:', best)
print("Simplex:")
for point in res:
    print(point[0], point[1])
```

2.3) Проверка на улучшение

```
if best < prev_best - no_improve_thr:
    # произошло улучшение
    not_improved = 0
    prev_best = best
else:
    # улучшение не произошло
    not_improved += 1

if not_improved >= no_improv_break:
    # возвращаем текущий лучший результат
    return res[0]
```

2.4) Центроид

В этом блоке кода вычисляется центроид, который представляет собой среднее значение координат точек симплекса, за исключением худшей точки. Центроид используется для вычисления остальных точек

$$x_0 = \frac{1}{N-1} \sum_{i=1}^{N-1} x_i$$

```
x0 = [0.] * dim          # Инициализация координат центроида
 # Вычисление суммы координат точек, за исключением худшей точки
 for tup in res[:-1]:
     for i, c in enumerate(tup[0]):
         x0[i] += c / (len(res)-1)
```

2.5) Отражение

Отражение выполняется относительно центроида симплекса в направлении худшей точки. Затем проверяется, улучшилось ли значение функции в отраженной точке по сравнению с худшей точкой и второй лучшей точкой.

$$x_r = x_0 + \alpha \cdot (x_0 - x_w)$$

```
xr = x0 + alpha*(x0 - res[-1][0])     # Вычисление отраженной точки
 rscore = f(xr)                        # Оценка значения целевой функции в отраженной точке
 if res[0][1] <= rscore < res[-2][1]: # Проверка условия отражения
      refl_number += 1                # Подсчет числа отражений(по условию должно быть > 4)
     # Замена худшей точки отраженной точкой
     del res[-1]
     res.append([xr, rscore])
     continue
```

2.6) Растяжение

Если значение функции в отраженной точке меньше, чем значение функции в лучшей точке симплекса, то выполняется экспансия. Растяжение происходит в направлении центроида симплекса.

$$x_e = x_0 + \gamma \cdot (x_0 - x_w)$$

```
     if rscore < res[0][1]:                  # Проверка условия экспансии
     # Вычисление экспансии
     xe = x0 + gamma*(x0 - res[-1][0])
     escore = f(xe)
     if escore < rscore:                  # Проверка условия улучшения
         # Замена худшей точки экспансией
         del res[-1]
         res.append([xe, escore])
         continue
     else:
```

```
        # В случае неулучшения замена худшей точки отраженной точкой
        del res[-1]
        res.append([xr, rscore])
        continue
```

2.7) Сжатие

Если значение функции в отраженной точке больше или равно значению функции в худшей точке симплекса, то выполняется сжатие. Сжатие происходит в направлении центроида симплекса.

$$x_c = x_0 + \beta \cdot (x_0 - x_w)$$

```
xc = x0 + beta*(x0 - res[-1][0])        # Вычисление сжатия
cscore = f(xc)
if cscore < res[-1][1]:                 # Проверка условия сжатия
        # Замена худшей точки сжатием
        del res[-1]
        res.append([xc, cscore])
        continue
```

2.8) Редукция

В случае, если ни один из предыдущих шагов не привел к улучшению значения целевой функции, происходит уменьшение симплекса. Каждая точка симплекса сжимается в направлении лучшей точки симплекса. Уменьшение симплекса направлено на уменьшение размера симплекса в случае, если предыдущие шаги не привели к улучшению значения целевой функции. Каждая точка симплекса сжимается в направлении лучшей точки, и процесс повторяется.

$$x_{\text{red}} = x_1 + \lambda \cdot (x_{\text{tup}} - x_1)$$

```
x1 = res[0][0]          # Запоминание координат лучшей точки
nres = []               # Инициализация нового списка для хранения новых точек симплекса
# Применение уменьшения симплекса к каждой точке
for tup in res:
    redx = x1 + _lambda*(tup[0] - x1)
    score = f(redx)
    nres.append([redx, score])
# Обновление списка точек симплекса
res = nres
```

Конец алгоритма.

3.2 Полный код программы:

```
import copy
import numpy as np


def nelder_mead(f, x_start,
                step=1, no_improve_thr=10e-6,
                no_improv_break=10, max_iter=0,
                alpha=1., gamma=2., beta=0.5, _lambda=2):

    # init
    dim = len(x_start)
    prev_best = f(x_start)
    not_improved = 0
    res = [[x_start, prev_best]]

    # Используйте x_1 и x_2 для инициализации симплекса
    for point in range(dim):
        x1 = copy.copy(x_start)
        x1[point] = x1[point] + step
        score_x1 = f(x1)
        res.append([x1, score_x1])

    # simplex iter
    iters = 0
    refl_number = 0
    while True:
        # =====Сортировка=====
        res.sort(key=lambda x: x[1])  # Лучшая точка (минимум) будет в начале списка.
        best = res[0][1]            # Текущее лучшее значение целевой функции

        # ======Проверка на максимальное кол-во итераций======
        if max_iter and iters >= max_iter:
            return res[0]
        iters += 1              # Увеличиваем счетчик итераций.

        # ======Вывод в консоль результатов=====
```

```
        print(f'\n==============================[{iters}]==============================')
        print(f'Лучшее значение среди всех минимумов на итерации [{iters}]: {best:.6f}')
        print("Simplex:")
        for point in res:
            print(f'{[f"{coord:.6f}" for coord in point[0]]}  {point[1]:.6f}')
        print(f"Reflection number: {refl_number}")
        print(f"------------------------------Точки------------------------------")
        if best < prev_best - no_improve_thr:
            # произошло улучшение
            not_improved = 0
            prev_best = best
        else:
            # улучшение не произошло
            not_improved += 1

        if not_improved >= no_improv_break:
            # возвращаем текущий лучший результат
            return res[0]

        # =====Центроид=====
        x0 = [0.] * dim         # Инициализация координат центроида
        # Вычисление суммы координат точек, за исключением худшей точки
        for tup in res[:-1]:
            for i, c in enumerate(tup[0]):
                x0[i] += c / (len(res)-1)

        print(f"Центроид: {[f'{coord:.6f}' for coord in x0]}")

        # =====Отражение=====
        xr = x0 + alpha*(x0 - res[-1][0])       # Вычисление отраженной точки
        rscore = f(xr)                          # Оценка значения целевой функции в отраженной
точке
        if res[0][1] <= rscore < res[-2][1]:    # Проверка условия отражения
            refl_number += 1                    # Подсчет числа отражений
            # Замена худшей точки отраженной точкой
            del res[-1]
            res.append([xr, rscore])
            print(f"Отражение: {[f'{coord:.6f}' for coord in xr]}")
            continue

        # =====Растяжение=====
```

```
        if rscore < res[0][1]:                  # Проверка условия экспансии
            # Вычисление экспансии
            xe = x0 + gamma*(x0 - res[-1][0])
            escore = f(xe)
            if escore < rscore:                 # Проверка условия улучшения
                # Замена худшей точки экспансией
                del res[-1]
                res.append([xe, escore])
                print(f"Растяжение: {[f'{coord:.6f}' for coord in xe]}")
                continue
            else:
                # В случае неулучшения замена худшей точки отраженной точкой
                del res[-1]
                res.append([xr, rscore])
                print(f"Отражение: {[f'{coord:.6f}' for coord in xr]}")
                continue

        # =====Сжатие=====
        xc = x0 + beta*(x0 - res[-1][0])        # Вычисление сжатия
        cscore = f(xc)
        if cscore < res[-1][1]:                 # Проверка условия сжатия
            # Замена худшей точки сжатием
            del res[-1]
            res.append([xc, cscore])
            print(f"Сжатие: {[f'{coord:.6f}' for coord in xc]}")
            continue

        # =====Редукция=====
        x1 = res[0][0]                          # Запоминание координат лучшей точки
        nres = []                               # Инициализация нового списка для хранения новых
точек симплекса
        # Применение уменьшения симплекса к каждой точке
        for tup in res:
            redx = x1 + _lambda*(tup[0] - x1)
            score = f(redx)
            nres.append([redx, score])
        # Обновление списка точек симплекса
        res = nres


if __name__ == "__main__":
```

```
    import numpy as np

    def f(x):

        return ((x[0] - 5) ** 2) / 2 + ((x[1] - 3) ** 2) / 3 + 4


    print(f'\n\nИтоговый (лучший) результат:', '{Симлекс, Точка минимума}', nelder_mead(f,
np.array([-2, 7, -2, 7])))
```

3.3 Результат программы

```
==============================[1]==============================

Лучшее значение среди всех минимумов на итерации [1]: 27.333333

Simplex:

['-1.000000', '7.000000', '-2.000000', '7.000000']  27.333333

['-2.000000', '7.000000', '-2.000000', '7.000000']  33.833333

['-2.000000', '7.000000', '-1.000000', '7.000000']  33.833333

['-2.000000', '7.000000', '-2.000000', '8.000000']  33.833333

['-2.000000', '8.000000', '-2.000000', '7.000000']  36.833333

Reflection number: 0

------------------------------Точки------------------------------

Центроид: ['-1.750000', '7.000000', '-1.750000', '7.250000']

Отражение: ['-1.500000', '6.000000', '-1.500000', '7.500000']


==============================[2]==============================

Лучшее значение среди всех минимумов на итерации [2]: 27.333333

Simplex:

['-1.000000', '7.000000', '-2.000000', '7.000000']  27.333333

['-1.500000', '6.000000', '-1.500000', '7.500000']  28.125000

['-2.000000', '7.000000', '-2.000000', '7.000000']  33.833333

['-2.000000', '7.000000', '-1.000000', '7.000000']  33.833333

['-2.000000', '7.000000', '-2.000000', '8.000000']  33.833333

Reflection number: 1

------------------------------Точки------------------------------

Центроид: ['-1.625000', '6.750000', '-1.625000', '7.125000']

Отражение: ['-1.250000', '6.500000', '-1.250000', '6.250000']


==============================[3]==============================

Лучшее значение среди всех минимумов на итерации [3]: 27.333333

Simplex:

['-1.000000', '7.000000', '-2.000000', '7.000000']  27.333333

['-1.250000', '6.500000', '-1.250000', '6.250000']  27.614583

['-1.500000', '6.000000', '-1.500000', '7.500000']  28.125000

['-2.000000', '7.000000', '-2.000000', '7.000000']  33.833333

['-2.000000', '7.000000', '-1.000000', '7.000000']  33.833333
```

```
Reflection number: 2
------------------------------Точки------------------------------
Центроид: ['-1.437500', '6.625000', '-1.687500', '6.937500']
Растяжение: ['-0.312500', '5.875000', '-3.062500', '6.812500']

==============================[4]==============================
Лучшее значение среди всех минимумов на итерации [4]: 20.866536
Simplex:
['-0.312500', '5.875000', '-3.062500', '6.812500']  20.866536
['-1.000000', '7.000000', '-2.000000', '7.000000']  27.333333
['-1.250000', '6.500000', '-1.250000', '6.250000']  27.614583
['-1.500000', '6.000000', '-1.500000', '7.500000']  28.125000
['-2.000000', '7.000000', '-2.000000', '7.000000']  33.833333
Reflection number: 2
------------------------------Точки------------------------------
Центроид: ['-1.015625', '6.343750', '-1.953125', '6.890625']
Растяжение: ['0.953125', '5.031250', '-1.859375', '6.671875']

==============================[5]==============================
Лучшее значение среди всех минимумов на итерации [5]: 13.563924
Simplex:
['0.953125', '5.031250', '-1.859375', '6.671875']  13.563924
['-0.312500', '5.875000', '-3.062500', '6.812500']  20.866536
['-1.000000', '7.000000', '-2.000000', '7.000000']  27.333333
['-1.250000', '6.500000', '-1.250000', '6.250000']  27.614583
['-1.500000', '6.000000', '-1.500000', '7.500000']  28.125000
Reflection number: 2
------------------------------Точки------------------------------
Центроид: ['-0.402344', '6.101562', '-2.042969', '6.683594']
Отражение: ['0.695312', '6.203125', '-2.585938', '5.867188']

==============================[6]==============================
Лучшее значение среди всех минимумов на итерации [6]: 13.563924
Simplex:
['0.953125', '5.031250', '-1.859375', '6.671875']  13.563924
['0.695312', '6.203125', '-2.585938', '5.867188']  16.685170
['-0.312500', '5.875000', '-3.062500', '6.812500']  20.866536
['-1.000000', '7.000000', '-2.000000', '7.000000']  27.333333
['-1.250000', '6.500000', '-1.250000', '6.250000']  27.614583
Reflection number: 3
------------------------------Точки------------------------------
```

Центроид: ['0.083984', '6.027344', '-2.376953', '6.587891']

Растяжение: ['2.751953', '5.082031', '-4.630859', '7.263672']

==========[7]==========

Лучшее значение среди всех минимумов на итерации [7]: 7.971809

Simplex:

['2.751953', '5.082031', '-4.630859', '7.263672'] 7.971809

['0.953125', '5.031250', '-1.859375', '6.671875'] 13.563924

['0.695312', '6.203125', '-2.585938', '5.867188'] 16.685170

['-0.312500', '5.875000', '-3.062500', '6.812500'] 20.866536

['-1.000000', '7.000000', '-2.000000', '7.000000'] 27.333333

Reflection number: 3

----------Точки----------

Центроид: ['1.021973', '5.547852', '-3.034668', '6.653809']

Растяжение: ['5.065918', '2.643555', '-5.104004', '5.961426']

==========[8]==========

Лучшее значение среди всех минимумов на итерации [8]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['2.751953', '5.082031', '-4.630859', '7.263672'] 7.971809

['0.953125', '5.031250', '-1.859375', '6.671875'] 13.563924

['0.695312', '6.203125', '-2.585938', '5.867188'] 16.685170

['-0.312500', '5.875000', '-3.062500', '6.812500'] 20.866536

Reflection number: 3

----------Точки----------

Центроид: ['2.366577', '4.739990', '-3.545044', '6.441040']

Отражение: ['5.045654', '3.604980', '-4.027588', '6.069580']

==========[9]==========

Лучшее значение среди всех минимумов на итерации [9]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.045654', '3.604980', '-4.027588', '6.069580'] 4.123043

['2.751953', '5.082031', '-4.630859', '7.263672'] 7.971809

['0.953125', '5.031250', '-1.859375', '6.671875'] 13.563924

['0.695312', '6.203125', '-2.585938', '5.867188'] 16.685170

Reflection number: 4

----------Точки----------

Центроид: ['3.454163', '4.090454', '-3.905457', '6.491638']

Отражение: ['6.213013', '1.977783', '-5.224976', '7.116089']

==========[10]==========

Лучшее значение среди всех минимумов на итерации [10]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.045654', '3.604980', '-4.027588', '6.069580'] 4.123043

['6.213013', '1.977783', '-5.224976', '7.116089'] 5.084009

['2.751953', '5.082031', '-4.630859', '7.263672'] 7.971809

['0.953125', '5.031250', '-1.859375', '6.671875'] 13.563924

Reflection number: 5

----------Точки----------

Центроид: ['4.769135', '3.327087', '-4.746857', '6.602692']

Сжатие: ['6.677139', '2.475006', '-6.190598', '6.568100']

==========[11]==========

Лучшее значение среди всех минимумов на итерации [11]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.045654', '3.604980', '-4.027588', '6.069580'] 4.123043

['6.213013', '1.977783', '-5.224976', '7.116089'] 5.084009

['6.677139', '2.475006', '-6.190598', '6.568100'] 5.498271

['2.751953', '5.082031', '-4.630859', '7.263672'] 7.971809

Reflection number: 5

----------Точки----------

Центроид: ['5.750431', '2.675331', '-5.136791', '6.428799']

Сжатие: ['7.249670', '1.471981', '-5.389757', '6.011362']

==========[12]==========

Лучшее значение среди всех минимумов на итерации [12]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.045654', '3.604980', '-4.027588', '6.069580'] 4.123043

['6.213013', '1.977783', '-5.224976', '7.116089'] 5.084009

['6.677139', '2.475006', '-6.190598', '6.568100'] 5.498271

['7.249670', '1.471981', '-5.389757', '6.011362'] 7.308788

Reflection number: 5

----------Точки----------

Центроид: ['5.750431', '2.675331', '-5.136791', '6.428799']

Отражение: ['4.251192', '3.878681', '-4.883825', '6.846235']

==========[13]==========

Лучшее значение среди всех минимумов на итерации [13]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.045654', '3.604980', '-4.027588', '6.069580'] 4.123043

['4.251192', '3.878681', '-4.883825', '6.846235'] 4.537717

['6.213013', '1.977783', '-5.224976', '7.116089'] 5.084009

['6.677139', '2.475006', '-6.190598', '6.568100'] 5.498271

Reflection number: 6

------------------------------Точки------------------------------

Центроид: ['5.143944', '3.026250', '-4.810098', '6.498333']

Отражение: ['3.610749', '3.577494', '-3.429599', '6.428565']

==============================[14]==============================

Лучшее значение среди всех минимумов на итерации [14]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.045654', '3.604980', '-4.027588', '6.069580'] 4.123043

['4.251192', '3.878681', '-4.883825', '6.846235'] 4.537717

['3.610749', '3.577494', '-3.429599', '6.428565'] 5.076175

['6.213013', '1.977783', '-5.224976', '7.116089'] 5.084009

Reflection number: 7

------------------------------Точки------------------------------

Центроид: ['4.493378', '3.426178', '-4.361254', '6.326452']

==============================[15]==============================

Лучшее значение среди всех минимумов на итерации [15]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

['5.025391', '4.566406', '-2.951172', '6.177734'] 4.818199

['3.436466', '5.113808', '-4.663647', '7.731045'] 6.711713

['7.360107', '1.312012', '-5.345947', '8.270752'] 7.734822

['2.155581', '4.511433', '-1.755194', '6.895704'] 8.806837

Reflection number: 7

------------------------------Точки------------------------------

Центроид: ['5.221971', '3.408945', '-4.516192', '7.035239']

Сжатие: ['6.755166', '2.857701', '-5.896692', '7.105007']

==============================[16]==============================

Лучшее значение среди всех минимумов на итерации [16]: 4.044524

Simplex:

['5.065918', '2.643555', '-5.104004', '5.961426'] 4.044524

```
['5.025391', '4.566406', '-2.951172', '6.177734']  4.818199
['6.755166', '2.857701', '-5.896692', '7.105007']  5.547053
['3.436466', '5.113808', '-4.663647', '7.731045']  6.711713
['7.360107', '1.312012', '-5.345947', '8.270752']  7.734822
Reflection number: 7
------------------------------Точки------------------------------
Центроид: ['5.070735', '3.795367', '-4.653879', '6.743803']
Сжатие: ['3.926049', '5.037045', '-4.307844', '5.980328']

==============================[17]==============================
Лучшее значение среди всех минимумов на итерации [17]: 4.044524
Simplex:
['5.065918', '2.643555', '-5.104004', '5.961426']  4.044524
['5.025391', '4.566406', '-2.951172', '6.177734']  4.818199
['6.755166', '2.857701', '-5.896692', '7.105007']  5.547053
['3.926049', '5.037045', '-4.307844', '5.980328']  5.959870
['3.436466', '5.113808', '-4.663647', '7.731045']  6.711713
Reflection number: 7
------------------------------Точки------------------------------
Центроид: ['5.193131', '3.776177', '-4.564928', '6.306124']
Сжатие: ['6.071463', '3.107362', '-4.515569', '5.593663']

==============================[18]==============================
Лучшее значение среди всех минимумов на итерации [18]: 4.044524
Simplex:
['5.065918', '2.643555', '-5.104004', '5.961426']  4.044524
['6.071463', '3.107362', '-4.515569', '5.593663']  4.577859
['5.025391', '4.566406', '-2.951172', '6.177734']  4.818199
['6.755166', '2.857701', '-5.896692', '7.105007']  5.547053
['3.926049', '5.037045', '-4.307844', '5.980328']  5.959870
Reflection number: 7
------------------------------Точки------------------------------


Итоговый (лучший) результат: {Симлекс, Точка минимума} [array([ 5.06591797,  2.64355469, -
5.10400391,  5.96142578]), 4.04452367623647]


Process finished with exit code 0
```

* Можем заметить, что одно из условий задачи (минимальное число отражений: 4) выполнилось, т.к. на последней [18] итерации метода общее число отражений равно 7.

4. Ответ.

Минимум функции, который нашелся методом Нелдера-Мида для функции: f(x)=((x_1-5)^2)/2 + ((x_2-3)^2)/3+4 :

= 4.0445 ≈ 4.