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. Решение на ЯП Python с комментариями

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

2.1 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.

2.2 Полный код программы:

```
import copy

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):

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

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

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


        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)

        # =====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])
            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])
                continue
            else:
                # В случае неулучшения замена худшей точки отраженной точкой
```

```
                del res[-1]
                res.append([xr, rscore])
                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])
            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'Итоговый (лучший) результат:', '{Симлекс, Точка минимума}', nelder_mead(f,
np.array([0., 0., 0.])))
```

2.3 Результат программы

```
Лучшее значение среди всех минимумов на итерации [1]: 19.005000000000003
Simplex:
[0.1 0.  0. ] 19.005000000000003
[0.  0.1 0. ] 19.303333333333335
[0. 0. 0.] 19.5
[0.  0.  0.1] 19.5
Reflection number: 0
Лучшее значение среди всех минимумов на итерации [2]: 19.005000000000003
Simplex:
[0.1 0.  0. ] 19.005000000000003
[ 0.06666667  0.06666667 -0.1       ] 19.037037037037038
[0.  0.1 0. ] 19.303333333333335
[0. 0. 0.] 19.5
Reflection number: 1
Лучшее значение среди всех минимумов на итерации [3]: 18.35648148148148
Simplex:
[ 0.16666667  0.16666667 -0.1       ] 18.35648148148148
[0.1 0.  0. ] 19.005000000000003
[ 0.06666667  0.06666667 -0.1       ] 19.037037037037038
[0.  0.1 0. ] 19.303333333333335
Reflection number: 1
Лучшее значение среди всех минимумов на итерации [4]: 17.822592592592596
Simplex:
[ 0.33333333  0.03333333 -0.2       ] 17.822592592592596
[ 0.16666667  0.16666667 -0.1       ] 18.35648148148148
[0.1 0.  0. ] 19.005000000000003
[ 0.06666667  0.06666667 -0.1       ] 19.037037037037038
Reflection number: 1
Лучшее значение среди всех минимумов на итерации [5]: 17.1437037037037
Simplex:
[ 0.46666667  0.06666667 -0.1       ] 17.1437037037037
[ 0.33333333  0.03333333 -0.2       ] 17.822592592592596
[ 0.16666667  0.16666667 -0.1       ] 18.35648148148148
[0.1 0.  0. ] 19.005000000000003
Reflection number: 1
Лучшее значение среди всех минимумов на итерации [6]: 15.450925925925926
```

Рисунок 1 – Результат итераций 1-6

```
Лучшее значение среди всех минимумов на итерации [35]: 4.049650820200891
Simplex:
[ 5.24113533  3.24846145 -2.07499365] 4.049650820200891
[ 4.70597413  2.2549792  -1.13367034] 4.228244269902831
[ 4.56378262  2.35191451 -1.50833206] 4.235147734169684
[ 4.25950256  3.29433296 -0.93152128] 4.303045525463744
Reflection number: 9
Лучшее значение среди всех минимумов на итерации [36]: 4.049650820200891
Simplex:
[ 5.24113533  3.24846145 -2.07499365] 4.049650820200891
[ 5.12569476  2.2805111  -1.89273739] 4.18045434521297
[ 4.70597413  2.2549792  -1.13367034] 4.228244269902831
[ 4.56378262  2.35191451 -1.50833206] 4.235147734169684
Reflection number: 9
Лучшее значение среди всех минимумов на итерации [37]: 4.049650820200891
Simplex:
[ 5.24113533  3.24846145 -2.07499365] 4.049650820200891
[ 5.48475353  2.83738665 -1.89260219] 4.1263073586783205
[ 5.12569476  2.2805111  -1.89273739] 4.18045434521297
[ 4.70597413  2.2549792  -1.13367034] 4.228244269902831
Reflection number: 10
Лучшее значение среди всех минимумов на итерации [38]: 4.049650820200891
Simplex:
[ 5.24113533  3.24846145 -2.07499365] 4.049650820200891
[ 5.48475353  2.83738665 -1.89260219] 4.1263073586783205
[ 5.57280474  3.05569    -2.36333144] 4.165086428426862
[ 5.12569476  2.2805111  -1.89273739] 4.18045434521297
Reflection number: 10
Лучшее значение среди всех минимумов на итерации [39]: 4.049650820200891
Simplex:
[ 5.24113533  3.24846145 -2.07499365] 4.049650820200891
[ 5.72837173  2.42631186 -1.71021073] 4.374968715942203
[ 5.90447416  2.86291856 -2.65166924] 4.415300523691379
[ 5.0102542   1.31256075 -1.71048113] 4.949202976538823
Reflection number: 10
Итоговый (лучший) результат: {Симлекс, Точка минимума} [array([ 5.24113533,  3.24846145, -2.07499365]), 4.049650820200891]

Process finished with exit code 0
```

Рисунок 2 – результат итераций 34-39

* Можем заметить, что одно из условий задачи (минимальное число отражений: 4) выполнилось, т.к. на последней [39] итерации метода общее число отражений равно 10.

3. Ответ.

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

= 4.0497 ≈ 4.