Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

Отчёт расчетной работе № 3

Дисциплина: Системный анализ и принятие решений.

Выполнил студент гр. 5130901/10101 _____ Д.Л. Симоновский
(подпись)

Руководитель _____ А.Г. Сиднев
(подпись)

“18” октябрь 2023 г.

Санкт-Петербург
2023

Оглавление

1. Условие

Дана задача нелинейного программирования в следующей форме:

$$max\{C_{11}{x_1}^2 + C_{22}{x_2}^2 + C_{12}x_1x_2 + C_1x_1 + C_2x_2\}$$

При следующих значениях:

Вариант	C_{11}	C_{22}	C_{12}	C_1	C_2
17	-7	-7	2	34	50

Итоговая задача:

$$max\{-7{x_1}^2 - 7{x_2}^2 + 2x_1x_2 + 34x_1 + 50x_2\}$$

Градиент целевой функции:

$$\nabla f(x_1, x_2) = \begin{bmatrix} \frac{\delta f(x_1, x_2)}{\delta x_1} \\ \frac{\delta f(x_1, x_2)}{\delta x_2} \end{bmatrix} = \begin{bmatrix} -14x_1 + 2x_2 + 34 \\ 2x_1 - 14x_2 + 50 \end{bmatrix}$$

Матрица Гессе:

$$\mathrm{H}(x_1, x_2) = \begin{bmatrix} \frac{\delta f(x_1, x_2)^2}{\delta {x_1}^2} & \frac{\delta f(x_1, x_2)^2}{\delta x_1 x_2} \\ \frac{\delta f(x_1, x_2)^2}{\delta x_1 x_2} & \frac{\delta f(x_1, x_2)^2}{\delta {x_2}^2} \end{bmatrix} = \begin{bmatrix} -14 & 2 \\ 2 & -14 \end{bmatrix}$$

2. Ход решения

Создадим функции, которые будем использовать в ходе решения данной расчетной работы.
Функция для получения коэффициентов C:

```
def get_C():
    return np.array([-7, -7, 2, 34, 50])
```

Функция для получения матрицы Гессе, которая необходима в ходе использования многих методов:

```
def get_H(C):
    return np.array([[C[0] * 2, C[2]], [C[2], C[1] * 2]])
```

Функция для получения градиента целевой функции:

```
def calculate_dfX(X, C):
    return np.array([C[0] * 2 * X[0] + C[2] * X[1] + C[3], C[1] * 2 * X[1] + C[2] * X[0] +
                     + C[4]])
```

Функция для получения значения целевой функции:

```
def calculate_fX(X, C):
    return C[0] * X[0] ** 2 + C[1] * X[1] ** 2 + C[2] * X[0] * X[1] + C[3] * X[0] + C[4] *
           + X[1]
```

Напишем функцию для отрисовки графика функции. Она будет получать значения x, y, чтоб нарисовать путь к итоговому решению:

```
def draw(x, y, C):
    x_1 = np.arange(min(x) - 1, max(x) + 1.01, 0.01)
    x_2 = np.arange(min(y) - 1, max(y) + 1.01, 0.01)
    x_1, x_2 = np.meshgrid(x_1, x_2)
    w = (C[0] * x_1 ** 2 + C[1] * x_2 ** 2 + C[2] * x_1 * x_2 + C[3] * x_1 + C[4] * x_2)
    plt.plot(x, y, '.-')
    contours = plt.contour(x_1, x_2, w, 15)
    contours.clabel()
```

Так же создадим ряд функций для вывода данных в виде таблицы:

```
table = None


def create_table(header):
    global table
    table = PrettyTable(header)


def add_row(row):
    table.add_row(row)


def print_table():
    print(table)
```

Естественно, это не самый правильный способ решения с точки зрения языка Python, однако цель этого расчетного задания – получить результат, поэтому автор решил оставить вывод в такой форме.
Так же стоит отметить, что во всех заданиях начальной точкой выбраны пары (1; 5), (5; 1), (-2; -3), а необходимая точность решения 0.01.
В дальнейшем код приводится не будет, его можно найти в приложении.

2.1. Метод наискорейшего подъема:

Траектория поиска решения:

$$X^{(i+1)} = X^{(i)} + t^{(i)}K^{(i)}$$

Где $t^{(i)}$- длина шага, $K^{(i)}$- вектор направления.

$$t^{(i)} = -\frac{\nabla^T f(X^{(i)})K^{(i)}}{{K^{(i)}}^T H(X^{(i)})K^{(i)}}$$

$$K^{(i)} = \nabla f(X)$$

Вызовем программу и получим следующий результат:

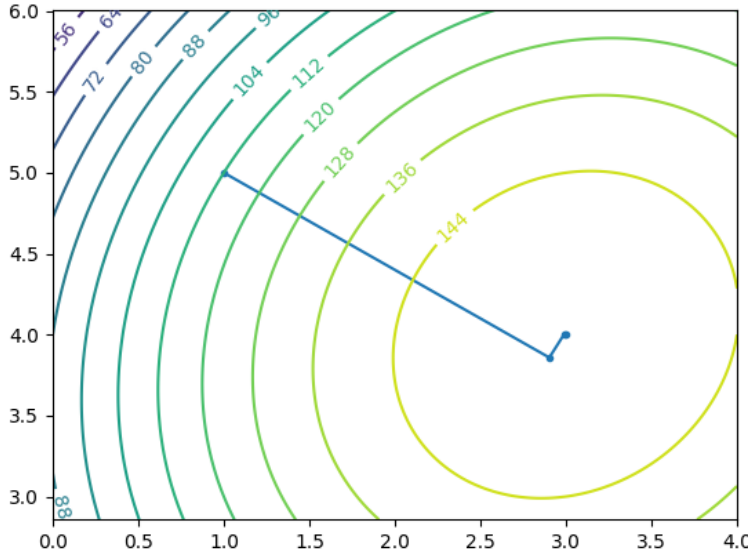

Рис. 1. Метод наискорейшего подъема. (1,5)

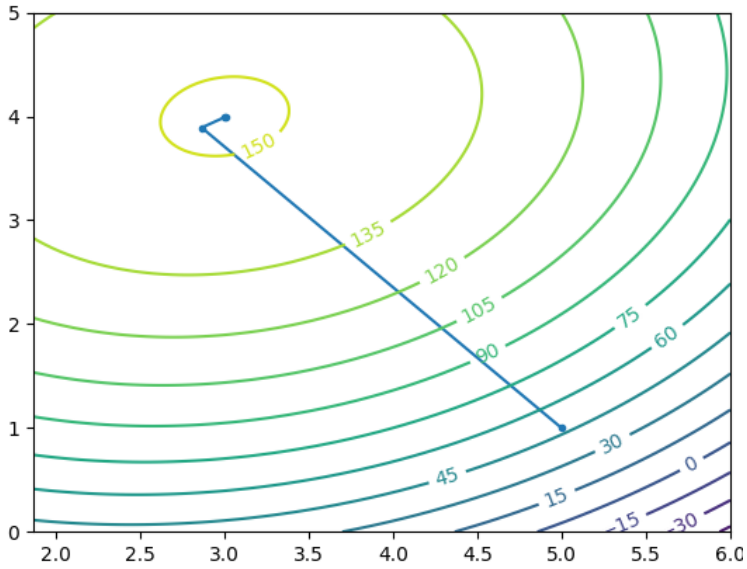

Рис. 2. Метод наискорейшего подъема. (5,1)

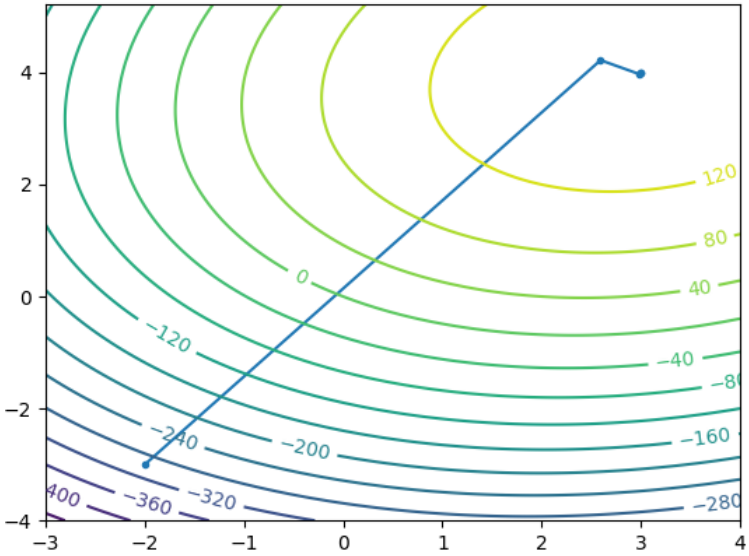

Рис. 3. Метод наискорейшего подъема. (-2, -3)

Видим, что графики действительно соответствуют ожидаемомым для метода наискорейшего подъема.

Промежуточные данные выглядят следующим образом:

Для (1; 5):

i	x1	x2	fX
1	1.000	5.000	112.000
2	2.903	3.858	150.821
3	2.991	4.005	150.999
4	3.000	3.999	151.000

Для (5; 1):

i	x1	x2	fX
1	5.000	1.000	48.000
2	2.863	3.891	150.816
3	3.004	3.995	151.000
4	3.000	4.000	151.000

Для (-2, -3):

i	x1	x2	fX
1	-2.000	-3.000	-297.000
2	2.595	4.220	149.332
3	2.981	3.974	150.994
4	2.998	4.001	151.000
5	3.000	4.000	151.000

По таблице видно, что алгоритм нашел ответ для всех начальных точек, получил результат (3; 4), а целевая функция приняла значение 151.

2.2. Метод Ньютона

Траектория поиска решения:

$$X^{(i+1)} = X^{(i)} + t^{(i)}K^{(i)}$$

Где $t^{(i)}$- длина шага, $K^{(i)}$- вектор направления.

$$t^{(i)} = t \equiv 1$$

$$K^{(i)} = -H^{-1}(X^{(i)})\nabla f(X)$$

Вызовем программу и получим следующий результат:

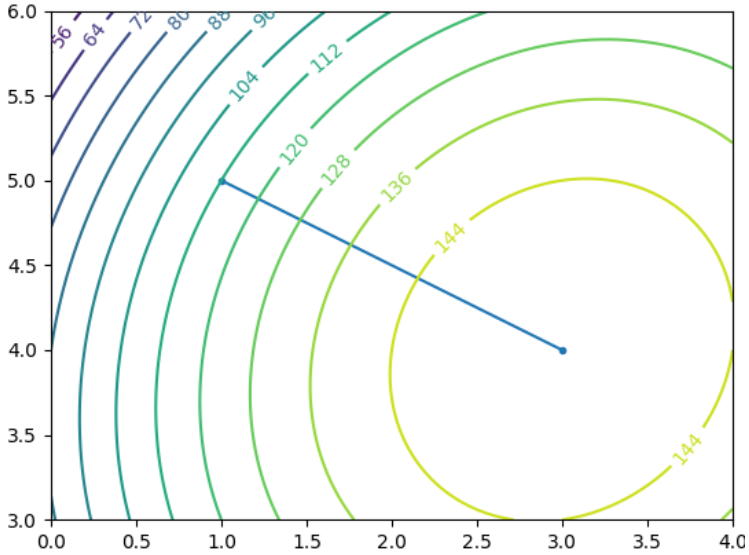

Рис. 4. Метод Ньютона. (1,5)

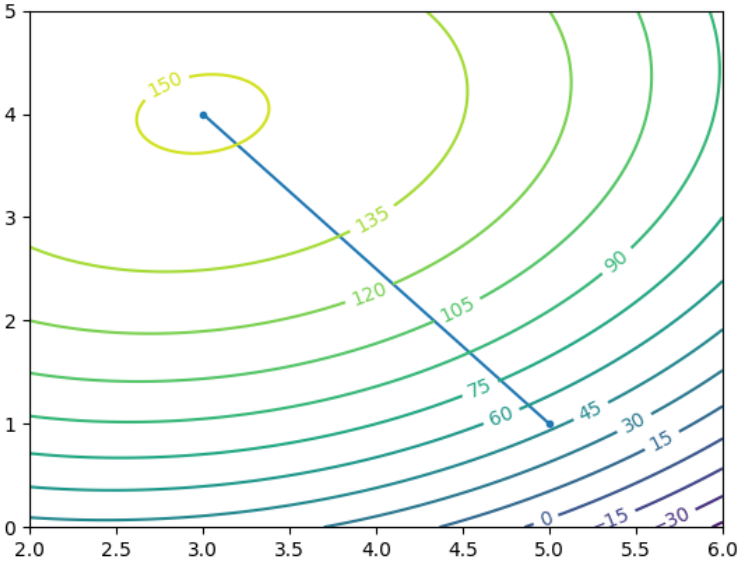

Рис. 5. Метод Ньютона. (5,1)

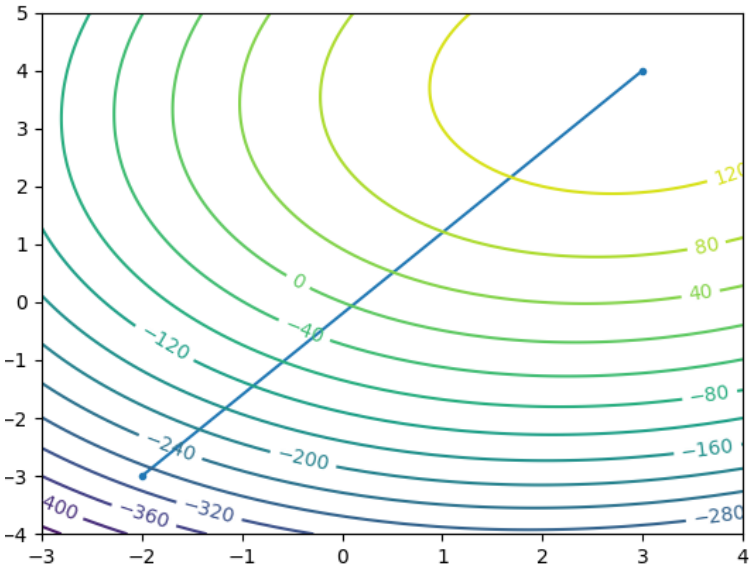

Рис. 6. Метод Ньютона. (-2, -3)

Промежуточные данные выглядят следующим образом:

Для (1; 5):

i	x1	x2	fX
1	1.000	5.000	112.000
2	3.000	4.000	151.000

Для (5; 1):

i	x1	x2	fX
1	5.000	1.000	48.000
2	3.000	4.000	151.000

Для (-2, -3):

i	x1	x2	fX
1	-2.000	-3.000	-297.000
2	3.000	4.000	151.000

По таблице видно, что алгоритм нашел ответ для всех начальных точек, получил результат (3; 4), а целевая функция приняла значение 151.

Стоит отметить, что метод Ньютона сделал это за минимальное число итераций, а именно одну.

2.3. Метод сопряженных градиентов

Траектория поиска решения:

$$X^{(i+1)} = X^{(i)} + t^{(i)}K^{(i)}$$

Где $t^{(i)}$- длина шага, $K^{(i)}$- вектор направления.

$$t^{(i)} == -\frac{\nabla^T f(X^{(i)})K^{(i)}}{{K^{(i)}}^T H(X^{(i)})K^{(i)}}$$

$$K^{(i)} = \begin{cases} \nabla f\left(X^{(i)}\right), & i = 0 \\ \nabla f\left(X^{(i)}\right) + \dfrac{\left\|\nabla f\left(X^{(i)}\right)\right\|^2}{\left\|\nabla f\left(X^{(i-1)}\right)\right\|^2} \nabla f\left(X^{(i-1)}\right), & i \neq 0 \end{cases}$$

Вызовем программу и получим следующий результат:

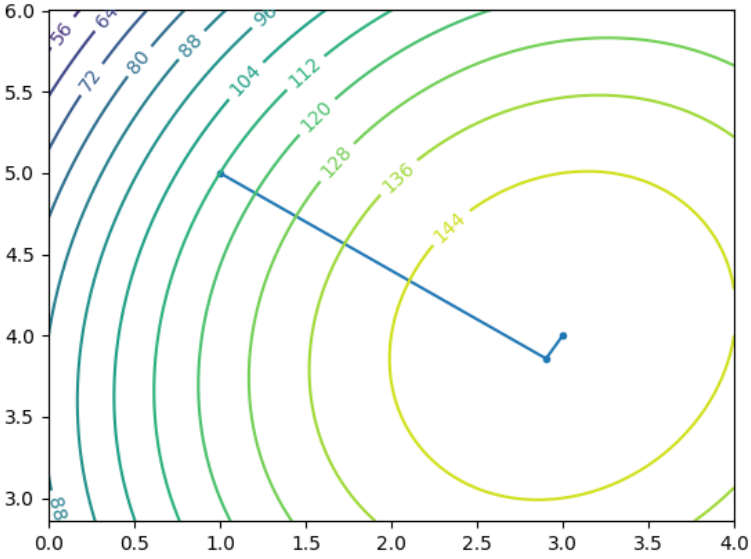

Рис. 7. Метод сопряженных градиентов. (1,5)

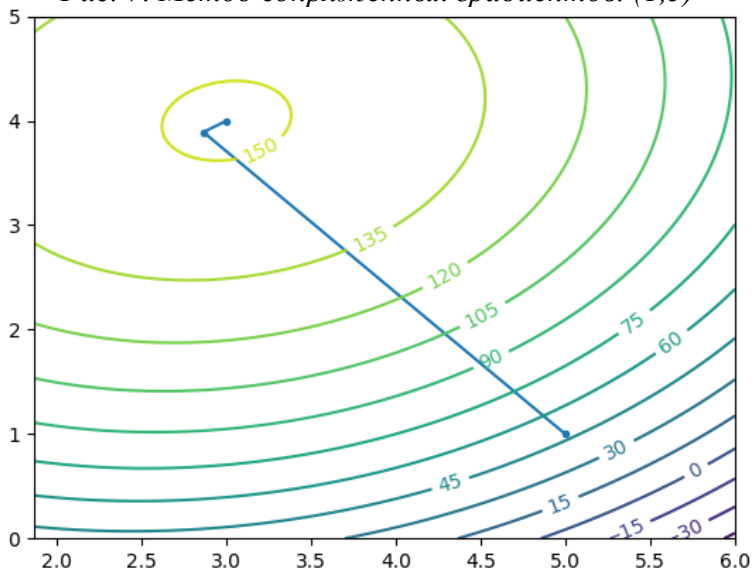

Рис. 8. Метод сопряженных градиентов. (5,1)

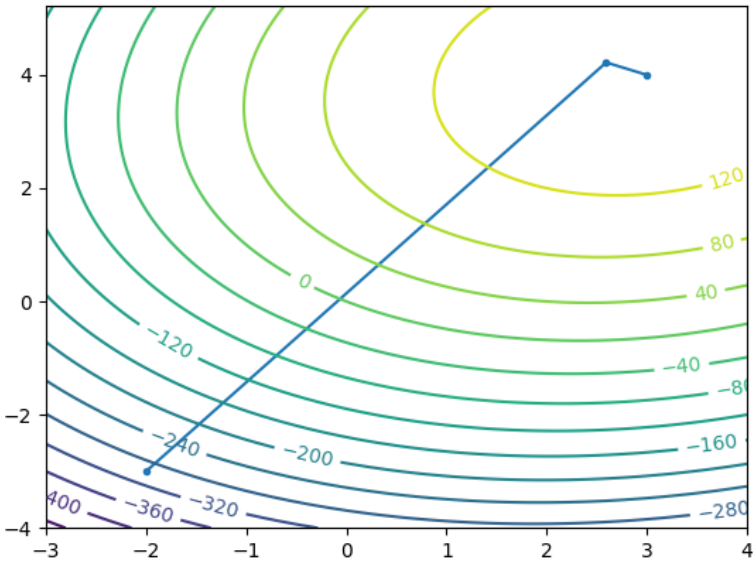

Рис. 9. Метод сопряженных градиентов. (-2, -3)

Промежуточные данные выглядят следующим образом:

Для (1; 5):

i	x1	x2	fX
1	1.000	5.000	112.000
2	2.903	3.858	150.821
3	3.000	4.000	151.000

Для (5; 1):

i	x1	x2	fX
1	5.000	1.000	48.000
2	2.863	3.891	150.816
3	3.000	4.000	151.000

Для (-2, -3):

i	x1	x2	fX
1	-2.000	-3.000	-297.000
2	2.595	4.220	149.332
3	3.000	4.000	151.000

По таблице видно, что алгоритм нашел ответ для всех начальных точек, получил результат (3; 4), а целевая функция приняла значение 151.

2.4. Метод релаксации

Траектория поиска решения:

$$X^{(i+1,j)} = X^{(i,j)} + t^{(i)}K^{(i,j)}$$

Где $t^{(i)}$- длина шага, $K^{(i)}$- вектор направления.

$$t^{(i)} == -\frac{\nabla^T f(X^{(i)})K^{(i)}}{{K^{(i)}}^T H(X^{(i)})K^{(i)}}$$

$$K^{(i,j)} = [{K_1}^{(i,j)} \dots {K_n}^{(i,j)}]$$

$$K^{(i,j)} = \begin{cases} \frac{\delta f(X^{(i)})}{\delta x_j}, k = j \\ 0, \quad k \neq j \end{cases}$$

Вызовем программу и получим следующий результат:

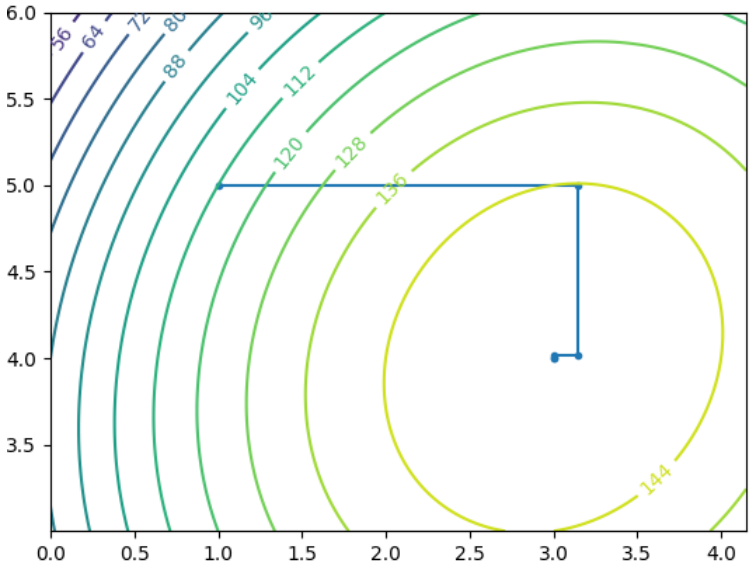

Рис. 10. Метод релаксации. (1,5)

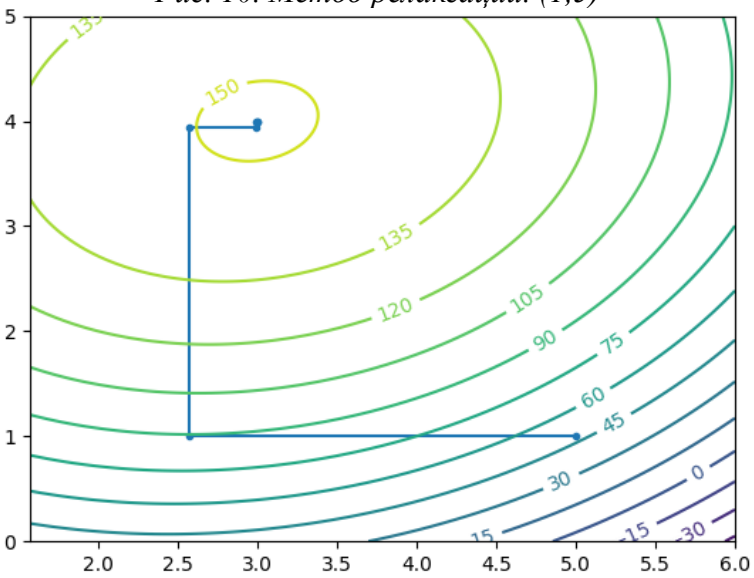

Рис. 11. Метод релаксации. (5,1)

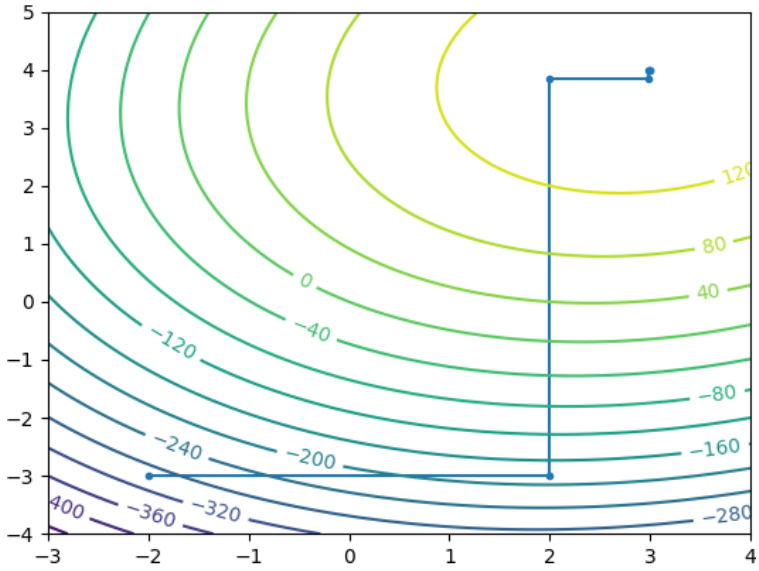

Рис. 12. Метод релаксации. (-2, -3)

Промежуточные данные выглядят следующим образом:

Для (1; 5):

i	x1	x2	fX
1	1.000	5.000	112.000
2	3.143	5.000	144.143
3	3.143	4.020	150.860
4	3.003	4.020	150.997
5	3.003	4.000	151.000
6	3.000	4.000	151.000

Для (5; 1):

i	x1	x2	fX
1	5.000	1.000	48.000
2	2.571	1.000	89.286
3	2.571	3.939	149.741
4	2.991	3.939	150.974
5	2.991	3.999	150.999
6	3.000	3.999	151.000
7	3.000	4.000	151.000

Для (-2, -3):

i	x1	x2	fX
1	-2.000	-3.000	-297.000
2	2.000	-3.000	-185.000
3	2.000	3.857	144.143
4	2.980	3.857	150.860
5	2.980	3.997	150.997
6	3.000	3.997	151.000
7	3.000	4.000	151.000

По таблице видно, что алгоритм нашел ответ для всех начальных точек, получил результат (3; 4), а целевая функция приняла значение 151.

Специфическая черта этого метода прослеживается как на графиках, так и на таблицах, он двигается вдоль осей координат. На графике это особенно видно, однако и на таблицах можно это заметить, что она из переменных изменяется в то время, как вторая – нет.

2.5. Метод метрики Бройдена

Траектория поиска решения:

$$X^{(i+1)} = X^{(i)} + t^{(i)}K^{(i)}$$

Где $t^{(i)}$- длина шага, $K^{(i)}$- вектор направления.

$$t^{(i)} == -\frac{\nabla^T f(X^{(i)})K^{(i)}}{{K^{(i)}}^T H(X^{(i)})K^{(i)}}$$

$$K^{(i,j)} = -\eta^{(i)}\nabla f(X^{(i)})$$

$$\eta^{(i)} = \begin{cases} -E, & i = 0 \\ \eta^{(i-1)} + \Delta\eta^{(i-1)}, & i \neq 0 \end{cases}$$

$$\Delta\eta^{(i-1)} = A^{(i-1)} - B^{(i-1)}$$

$$A^{(i-1)} = \frac{(\Delta X^{(i-1)})(\Delta X^{(i-1)})^T}{(\Delta X^{(i-1)})^T(\Delta g^{(i-1)})}$$

$$B^{(i-1)} = \frac{\eta^{(i-1)}(\Delta g^{(i-1)})(\Delta g^{(i-1)})^T(\eta^{(i-1)})^T}{(\Delta g^{(i-1)})^T(\eta^{(i-1)})^T(\Delta g^{(i-1)})}$$

Вызовем программу и получим следующий результат:

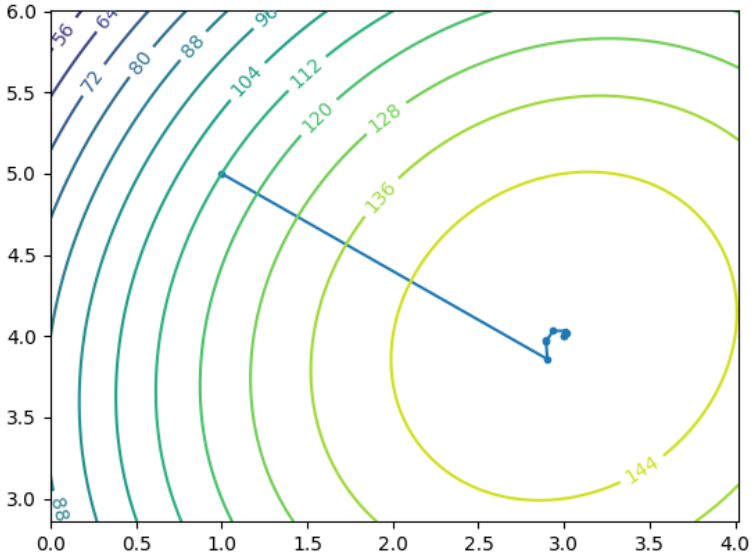

Рис. 13. Метод метрики Бройдена. (1,5)

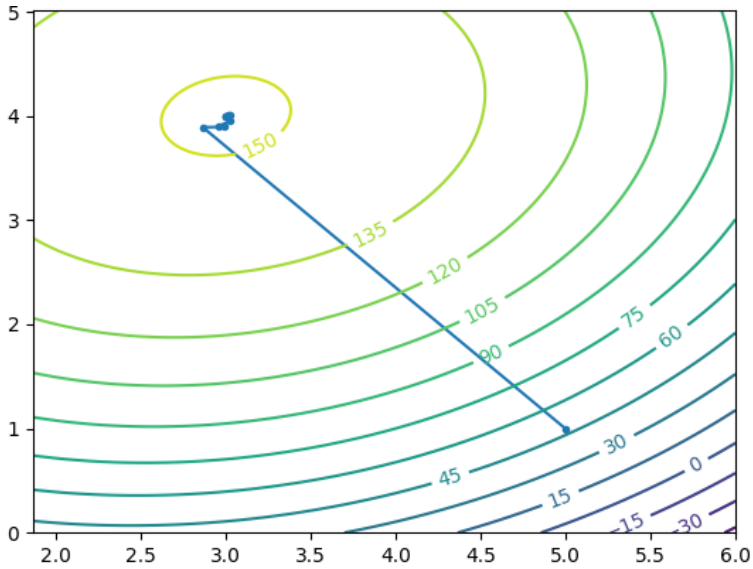

Рис. 14. Метод метрики Бройдена. (5,1)

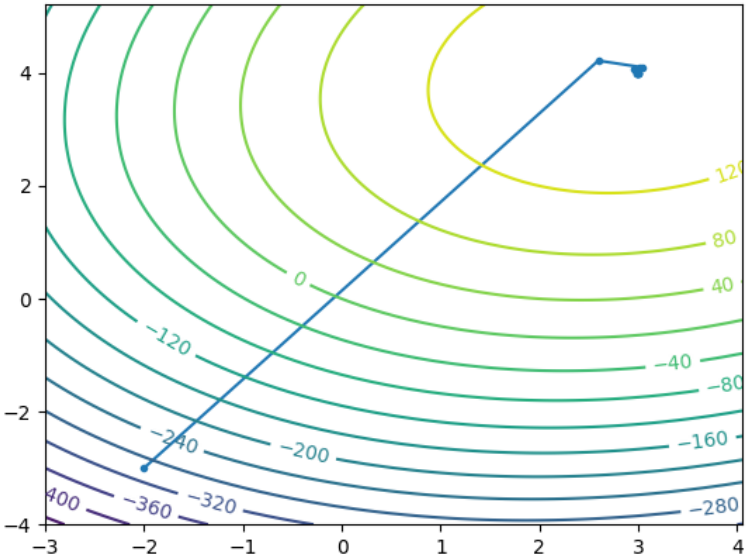

Рис. 15. Метод метрики Бройдена. (-2, -3)

Промежуточные данные выглядят следующим образом:

Для (1; 5):

i	x1	x2	fX
1	1.000	5.000	112.000
2	2.903	3.858	150.821
3	2.895	3.978	150.924
4	2.896	3.968	150.924
5	2.936	4.033	150.959
6	3.005	4.032	150.993
7	3.018	4.018	150.996
8	3.001	3.999	151.000
9	3.000	4.000	151.000

Для (5; 1):

i	x1	x2	fX
1	5.000	1.000	48.000
2	2.863	3.891	150.816
3	2.988	3.895	150.924
4	2.956	3.905	150.932
5	3.027	3.949	150.974
6	3.023	4.005	150.997
7	3.017	4.013	150.997
8	3.004	3.997	151.000
9	2.999	3.999	151.000
10	3.000	3.999	151.000
11	3.000	4.000	151.000

Для (-2, -3):

i	x1	x2	fX
1	-2.000	-3.000	-297.000
2	2.595	4.220	149.332
3	3.040	4.102	150.924
4	3.009	4.100	150.931
5	2.958	4.064	150.954
6	2.978	3.991	150.996
7	2.978	3.991	150.996
8	2.986	4.006	150.998
9	3.000	4.007	151.000
10	3.004	4.003	151.000
11	3.001	3.999	151.000
12	3.000	4.000	151.000

По таблице видно, что алгоритм нашел ответ для всех начальных точек, получил результат (3; 4), а целевая функция приняла значение 151.

3. Вывод:

В ходе расчетного задания были использованы некоторые методы для безусловной оптимизации заданной задачи нелинейного программирования.
По результатам работы видно, что наибыстрейшим методом является метод Ньютона, как и ожидалось, остальные методы тоже показали себя с хорошей стороны, успешно найдя точку максимума и её координаты во всех случаях.

4. Ссылки:

С кодом программы можно ознакомиться на github: https://github.com/DafterT/SADM