

Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

ОТЧЕТ

по расчетному заданию

«Нелинейное программирование. Безусловная оптимизация»

Системный анализ и принятие решений

Работу выполнил студент

группа 33501/4 Дьячков В.В.

Преподаватель

_____ Сабонис С.С.

Санкт-Петербург

10 ноября 2017 г.

Содержание

1	Техническое задание	3
2	Исходные данные	3
3	Аналитическое решение	3
4	Решение методом релаксации	4
4.1	Описание метода	4
4.2	Исходный код	5
4.3	Траектории поиска	5
5	Решение методом наискорейшего подъема	7
5.1	Описание метода	7
5.2	Исходный код	7
5.3	Траектории поиска	7
6	Решение методом Ньютона	9
6.1	Описание метода	9
6.2	Исходный код	9
6.3	Траектории поиска	9
7	Решение методом сопряженных градиентов	11
7.1	Описание метода	11
7.2	Исходный код	11
7.3	Траектории поиска	12
8	Решение методом Бройдена	13
8.1	Описание метода	13
8.2	Исходный код	13
8.3	Траектории поиска	14

Список иллюстраций

4.1	Траектории поиска методом релаксации	6
5.1	Траектории поиска методом наискорейшего подъема	8
6.1	Траектории поиска методом Ньютона	10
7.1	Траектории поиска методом сопряженных градиентов	12
8.1	Траектории поиска методом Бройдена	15

1. Техническое задание

1. Записать необходимые условия оптимальности для задачи и решить задачу аналитически;
2. Решить задачу методом релаксации;
3. Решить задачу методом наискорейшего подъема;
4. Решить задачу методом Ньютона;
5. Решить задачу методом сопряженных градиентов;
6. Решить задачу методом Бройдена.

2. Исходные данные

Вариант 32 Дана задача нелинейного программирования:

$$\max f(X) = \max (-31x_1^2 - 34x_2^2 + 4x_1x_2 + 286x_1 + 388x_2)$$

3. Аналитическое решение

Необходимое условие максимума первого порядка. Пусть $f(X)$ дифференцируема в точке $X^* \in R^n$. Тогда если X^* – локальный экстремум, то $f'(X^*) = 0$.

Достаточное условие максимума второго порядка. Пусть $f(X)$ дважды дифференцируема в точке $X^* \in R^n$. Тогда если $f'(X^*) = 0$, матрица $H(X^*)$ отрицательно определена (полуопределена), то X^* – строгий (нестрогий) локальный экстремум.

Вычислим частные производные 1 и 2 порядка:

$$\begin{aligned} \frac{\partial f}{\partial x_1}(X) &= -62x_1 + 4x_2 + 286 & \frac{\partial f}{\partial x_2}(X) &= -68x_2 + 4x_1 + 388 \\ \frac{\partial^2 f}{\partial x_1^2}(X) &= -62 & \frac{\partial^2 f}{\partial x_1 \partial x_2}(X) &= 4 & \frac{\partial^2 f}{\partial x_2^2}(X) &= -68 \end{aligned}$$

Следовательно матрица Гессе $H(X)$ равна:

$$H(X) = H = \begin{pmatrix} -62 & 4 \\ 4 & -68 \end{pmatrix}$$

Критерий отрицательной определенности квадратичной формы. Для отрицательной определенности квадратичной формы необходимо и достаточно, чтобы угловые миноры четного порядка ее матрицы были положительны, а нечетного порядка — отрицательны.

Найдем главные миноры матрицы H :

$$\Delta_1 = |-62| = -62 \qquad \Delta_2 = \begin{vmatrix} -62 & 4 \\ 4 & -68 \end{vmatrix} = 4200$$

По критерию отрицательной определенности квадратичной формы, матрица H отрицательно определена (выполнено достаточное условие максимума второго порядка).

Найдем точку X^* :

$$\begin{aligned} \begin{cases} -68x_2 + 4x_1 + 388 = 0 \\ -62x_1 + 4x_2 + 286 = 0 \end{cases} &\Rightarrow \begin{cases} x_1 = 17x_2 - 97 \\ -62 \cdot (17x_2 - 97) + 4x_2 + 286 = 0 \end{cases} \Rightarrow \\ \begin{cases} x_1 = 17x_2 - 97 \\ -1050x_2 + 6300 = 0 \end{cases} &\Rightarrow \begin{cases} x_1 = 17x_2 - 97 \\ x_2 = 6 \end{cases} \Rightarrow \begin{cases} x_1 = 5 \\ x_2 = 6 \end{cases} \end{aligned}$$

Следовательно максимум функции $f(X)$ достигается в точке $X^* = (5, 6)$: $f(X^*) = 1879$.

4. Решение методом релаксации

4.1. Описание метода

Траектория поиска решения:

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

где $t^{(i)}$ — длина шага, $K^{(i)}$ — вектор направления.

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

$$K^{(i,j)} = \begin{bmatrix} K_1^{(i,j)} & \dots & K_n^{(i,j)} \end{bmatrix}$$

$$K_k^{(i,j)} = \begin{cases} \frac{\partial f}{\partial x_j}(X^{(i,j)}), & k = j \\ 0, & k \neq j \end{cases}$$

4.2. Исходный код

```
1 import numpy as np
2 from sympy import *
3
4
5 def relaxation(y, f, symbols, x_init, max_step):
6     M = len(symbols)
7     derivs = [y.diff(sym) for sym in symbols]
8     H = np.array(hessian(y, symbols))
9
10    X = [np.array(x_init, dtype=float)]
11    step = 0
12    while step < max_step:
13        x = X[-1]
14        for j in range(M):
15            x = x.copy()
16            x_curr = {symbols[j]: X[-1][j] for j in range(M)}
17            grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
18            grad = grad.astype(np.float32)
19
20            K = np.zeros(M)
21            K[j] = derivs[j].subs(x_curr)
22            t = -grad.T.dot(K[np.newaxis].T) / K.dot(H).dot(K)
23
24            x[j] += t * K[j]
25            X.append(x)
26
27            if np.allclose(X[-2], X[-1]) or np.allclose(f(X[-2]), f(X[-1])):
28                break
29
30    step += 1
31
32    return X
```

Листинг 1: relaxation.py

4.3. Траектории поиска

Выберем 4 начальные точки разной степени удаленности от оптимальной ($x_1 = 5, x_2 = 6$):

- | | |
|-----------------------|-------------------------|
| 1. $x_1 = 0, x_2 = 0$ | 3. $x_1 = 6, x_2 = 5$ |
| 2. $x_1 = 1, x_2 = 8$ | 4. $x_1 = 10, x_2 = 10$ |

x_1	x_2	$f(X)$
0.000	0.000	0.000
4.613	0.000	659.645
4.613	5.977	1 874.372
4.999	5.977	1 878.982
4.999	6.000	1 879.000

Таблица 4.1: $X = (0, 0)$

x_1	x_2	$f(X)$
1.000	8.000	1 215.000
5.129	8.000	1 743.516
5.129	6.008	1 878.486
5.000	6.008	1 878.998
5.000	6.000	1 879.000

Таблица 4.2: $X = (1, 8)$

x_1	x_2	$f(X)$
6.000	5.000	1 810.000
4.935	5.000	1 845.129
4.935	5.996	1 878.871
5.000	5.996	1 879.000
5.000	6.000	1 879.000

Таблица 4.3: $X = (6, 5)$

x_1	x_2	$f(X)$
10.000	10.000	640.000
5.258	10.000	1 337.065
5.258	6.015	1 876.943
5.001	6.015	1 878.992
5.001	6.000	1 879.000

Таблица 4.4: $X = (10, 10)$

На рис. 4.1 изображены траектории поиска точки максимума $X = (5, 6)$ методом релаксации при разных начальных точках.

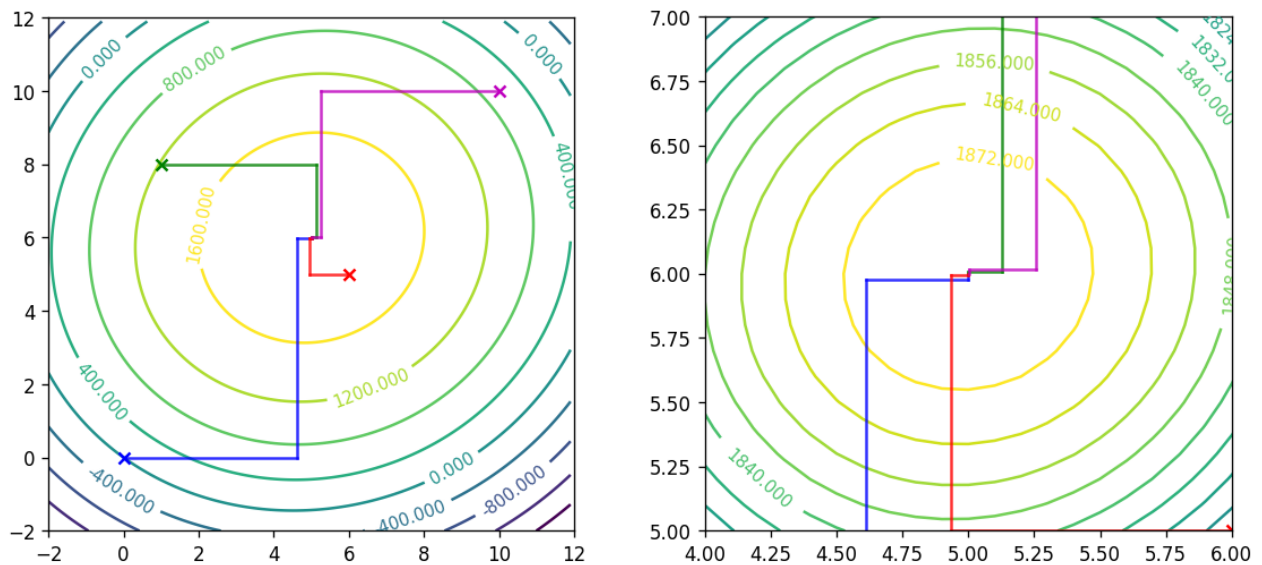


Рис. 4.1: Траектории поиска методом релаксации

5. Решение методом наискорейшего подъема

5.1. Описание метода

Траектория поиска решения:

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

где $t^{(i)}$ – длина шага, $K^{(i)}$ – вектор направления.

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

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

5.2. Исходный код

```
1 import numpy as np
2 from sympy import *
3
4
5 def steepest_ascent(y, f, symbols, x_init, max_step):
6     M = len(symbols)
7     derivs = [y.diff(sym) for sym in symbols]
8     H = np.array(hessian(y, symbols))
9
10    X = [np.array(x_init, dtype=np.float32)]
11    step = 0
12    while step < max_step:
13        x_curr = {symbols[j]: X[-1][j] for j in range(M)}
14
15        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
16        grad = grad.astype(np.float32)
17
18        t = -grad.dot(grad[np.newaxis].T) / grad.dot(H).dot(grad)
19        x_temp = np.array(X[-1] + t * grad, dtype=np.float32)
20
21        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp), rtol=1
22            e-6):
23            break
24
25        X.append(x_temp)
26        step += 1
27
28    return X
```

Листинг 2: steepest_ascent.py

5.3. Траектории поиска

Выберем 4 начальные точки разной степени удаленности от оптимальной ($x_1 = 5, x_2 = 6$):

1. $x_1 = 0, x_2 = 0$

2. $x_1 = 1, x_2 = 8$

3. $x_1 = 6, x_2 = 5$

4. $x_1 = 10, x_2 = 10$

x_1	x_2	$f(X)$
0.000	0.000	0.000
4.608	6.251	1 871.693
4.981	5.977	1 878.972
4.998	6.001	1 879.000

Таблица 5.1: $X = (0, 0)$

x_1	x_2	$f(X)$
1.000	8.000	1 215.000
4.817	5.734	1 875.745
4.980	6.010	1 878.984
4.999	5.999	1 879.000

Таблица 5.2: $X = (1, 8)$

x_1	x_2	$f(X)$
6.000	5.000	1 810.000
5.047	6.040	1 878.886
5.002	5.998	1 879.000

Таблица 5.3: $X = (6, 5)$

x_1	x_2	$f(X)$
10.000	10.000	640.000
5.148	5.841	1 877.369
5.007	6.005	1 878.998
5.000	6.000	1 879.000

Таблица 5.4: $X = (10, 10)$

На рис. 5.1 изображены траектории поиска точки максимума $X = (5, 6)$ методом наискорейшего подъема при разных начальных точках.

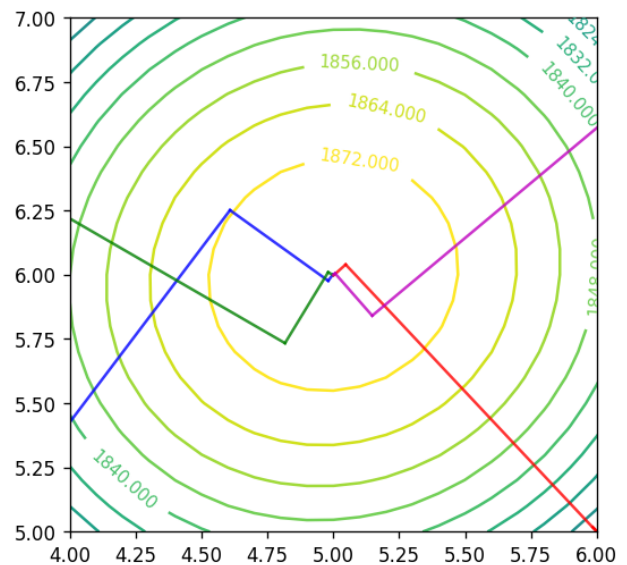
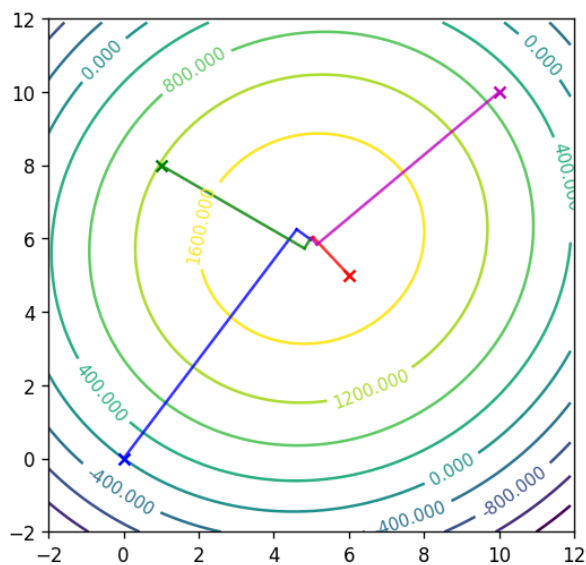


Рис. 5.1: Траектории поиска методом наискорейшего подъема

6. Решение методом Ньютона

6.1. Описание метода

Траектория поиска решения:

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

где $t^{(i)}$ – длина шага, $K^{(i)}$ – вектор направления.

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

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

Для квадратичных функций метод Ньютона сходится за 1 шаг.

6.2. Исходный код

```
1 import numpy as np
2 from sympy import *
3
4
5 def newton(y, f, symbols, x_init, max_step):
6     M = len(symbols)
7     derivs = [y.diff(sym) for sym in symbols]
8     Hinv = np.array(hessian(y, symbols).inv())
9
10    X = [np.array(x_init, dtype=np.float32)]
11    step = 0
12    while step < max_step:
13        x_curr = {symbols[j]: X[-1][j] for j in range(M)}
14
15        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
16        grad = grad.astype(np.float32)
17
18        x_temp = np.array(X[-1] - Hinv.dot(grad))
19        x_temp = x_temp.astype(np.float32)
20
21        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp)):
22            break
23
24        X.append(x_temp)
25        step += 1
26
27    return X
```

Листинг 3: newton.py

6.3. Траектории поиска

Выберем 4 начальные точки разной степени удаленности от оптимальной ($x_1 = 5, x_2 = 6$):

1. $x_1 = 0, x_2 = 0$

2. $x_1 = 1, x_2 = 8$

3. $x_1 = 6, x_2 = 5$

4. $x_1 = 10, x_2 = 10$

x_1	x_2	$f(X)$
0.000	0.000	0.000
5.000	6.000	1 879.000

Таблица 6.1: $X = (0, 0)$

x_1	x_2	$f(X)$
1.000	8.000	1 215.000
5.000	6.000	1 879.000

Таблица 6.2: $X = (1, 8)$

x_1	x_2	$f(X)$
6.000	5.000	1 810.000
5.000	6.000	1 879.000

Таблица 6.3: $X = (6, 5)$

x_1	x_2	$f(X)$
10.000	10.000	640.000
5.000	6.000	1 879.000

Таблица 6.4: $X = (10, 10)$

На рис. 6.1 изображены траектории поиска точки максимума $X = (5, 6)$ методом Ньютона при разных начальных точках.

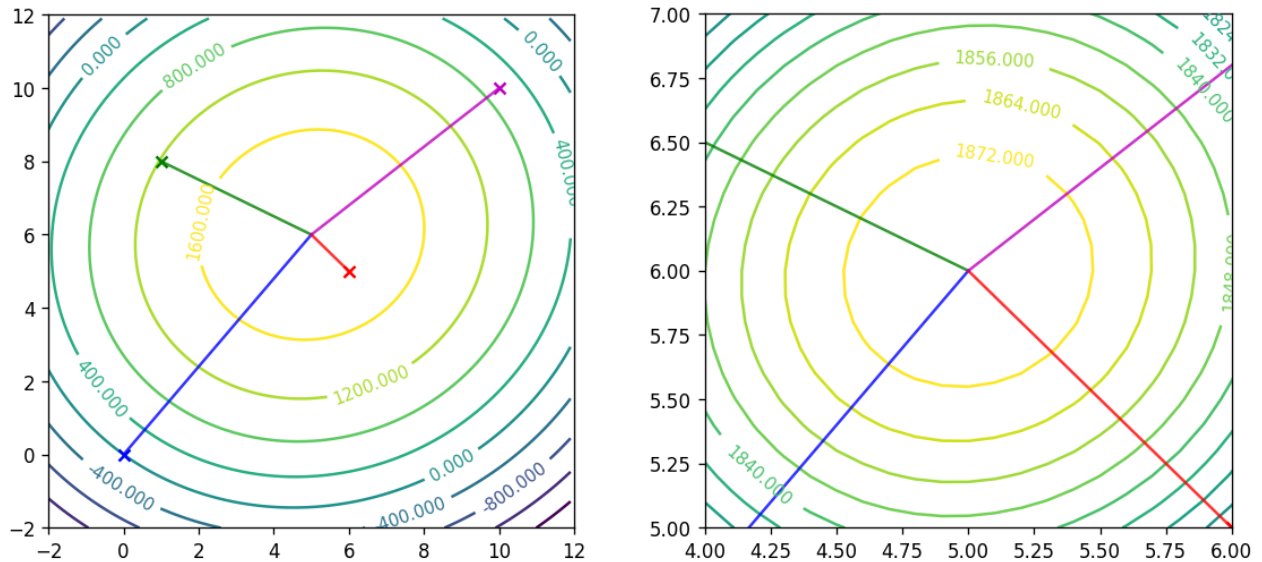


Рис. 6.1: Траектории поиска методом Ньютона

7. Решение методом сопряженных градиентов

7.1. Описание метода

Траектория поиска решения:

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

где $t^{(i)}$ – длина шага, $K^{(i)}$ – вектор направления.

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

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

Для квадратичных функций число шагов равно числу переменных.

7.2. Исходный код

```
1 import numpy as np
2 from numpy.linalg import norm
3 from sympy import *
4
5
6 def conjugate_gradients(y, f, symbols, x_init, max_step):
7     M = len(symbols)
8     derivs = [y.diff(sym) for sym in symbols]
9     H = np.array(hessian(y, symbols))
10    X = [np.array(x_init, dtype=np.float32)]
11    step = 0
12    while step < max_step:
13        x_curr = {symbols[j]: X[-1][j] for j in range(M)}
14        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
15        grad = grad.astype(np.float32)
16        K = grad
17        if step > 0:
18            x_prev = {symbols[j]: X[-2][j] for j in range(M)}
19            grad_prev = np.array([derivs[i].subs(x_prev) for i in range(M)])
20            grad_prev = grad_prev.astype(np.float32)
21            K += norm(grad) ** 2 / norm(grad_prev) ** 2 * grad_prev
22
23        t = -grad.dot(grad[np.newaxis].T) / grad.dot(H).dot(grad)
24        x_temp = np.array(X[-1] + t * K, dtype=np.float32)
25        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp)):
26            break
27        X.append(x_temp)
28        step += 1
29    return X
```

Листинг 4: conjugate_gradients.py

7.3. Траектории поиска

Выберем 4 начальные точки разной степени удаленности от оптимальной ($x_1 = 5, x_2 = 6$):

1. $x_1 = 0, x_2 = 0$
2. $x_1 = 1, x_2 = 8$
3. $x_1 = 6, x_2 = 5$
4. $x_1 = 10, x_2 = 10$

x_1	x_2	$f(X)$
0.000	0.000	0.000
4.608	6.251	1 871.693
5.002	5.999	1 879.000

Таблица 7.1: $X = (0, 0)$

x_1	x_2	$f(X)$
1.000	8.000	1 215.000
4.817	5.734	1 875.745
5.001	6.001	1 879.000

Таблица 7.2: $X = (1, 8)$

x_1	x_2	$f(X)$
6.000	5.000	1 810.000
5.047	6.040	1 878.886
5.000	6.000	1 879.000

Таблица 7.3: $X = (6, 5)$

x_1	x_2	$f(X)$
10.000	10.000	640.000
5.148	5.841	1 877.369
5.000	6.000	1 879.000

Таблица 7.4: $X = (10, 10)$

На рис. 7.1 изображены траектории поиска точки максимума $X = (5, 6)$ методом сопряженных градиентов при разных начальных точках.

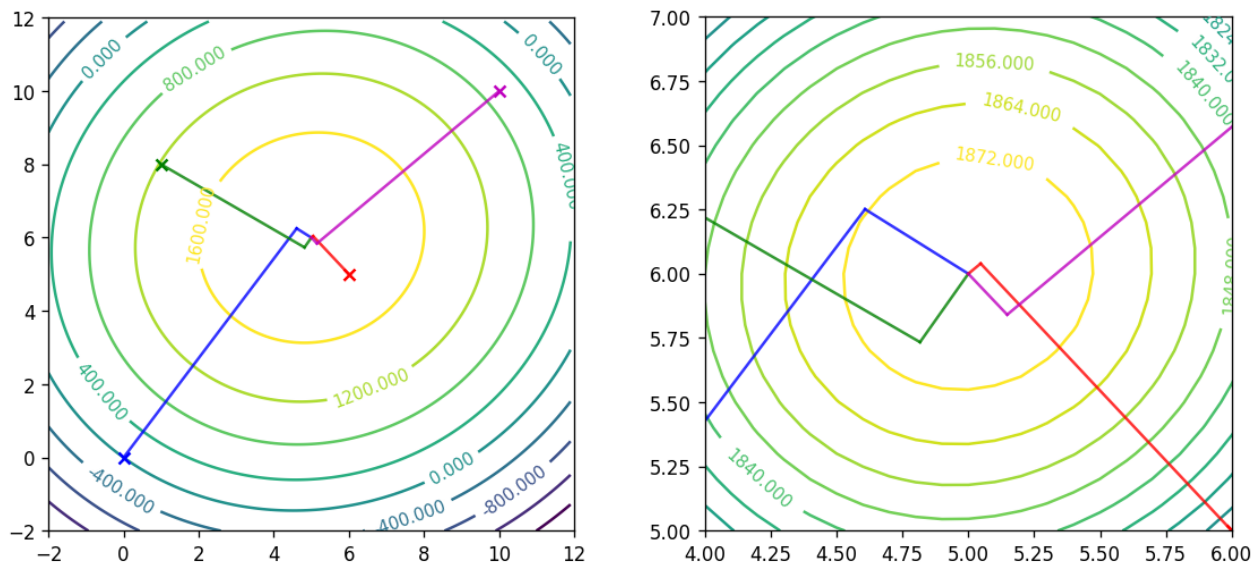


Рис. 7.1: Траектории поиска методом сопряженных градиентов

8. Решение методом Бройдена

8.1. Описание метода

Траектория поиска решения:

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

где $t^{(i)}$ – длина шага, $K^{(i)}$ – вектор направления.

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

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

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

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

$$z^{(i-1)} = X^{(i)} - X^{(i-1)} - \eta^{(i-1)} \Delta g^{(i-1)}$$

$$\Delta g^{(i-1)} = \nabla f(X^{(i)}) - \nabla f(X^{(i-1)})$$

8.2. Исходный код

```
1 import numpy as np
2 from sympy import *
3
4
5 def broyden(y, f, symbols, x_init, max_step):
6     M = len(symbols)
7     derivs = [y.diff(sym) for sym in symbols]
8     H = np.array(hessian(y, symbols))
9     X = [np.array(x_init, dtype=np.float32)]
10    G = []
11    N = []
12    Z = []
13    step = 0
14    while step < max_step:
15        x_curr = {symbols[j]: X[-1][j] for j in range(M)}
16
17        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
18        grad = grad.astype(np.float32)
19
20        if step == 0:
21            n = -np.eye(M)
22            N.append(n)
```

```

23     else :
24         x_prev = {symbols[j]: X[-2][j] for j in range(M)}
25         grad_prev = np.array([derivs[i].subs(x_prev) for i in range(M)])
26         grad_prev = grad_prev.astype(np.float32)
27
28         delta_g = grad - grad_prev
29         delta_X = X[-1] - X[-2]
30         z = delta_X - N[-1].dot(delta_g)
31         Z.append(z)
32
33         zm = z[np.newaxis] # 1x2
34         delta_nu = zm.T.dot(zm) / zm.dot(delta_g[np.newaxis].T)
35         n = N[-1] + delta_nu
36         N.append(n)
37
38         K = -n.dot(grad[np.newaxis].T).flatten()
39         t = -grad.dot(grad[np.newaxis].T) / grad.dot(H).dot(grad[np.newaxis].T)
40         x_temp = np.array(X[-1] + t * K, dtype=np.float32)
41
42         if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp)):
43             break
44
45         X.append(x_temp)
46         step += 1
47
48     return X

```

Листинг 5: broyden.py

8.3. Траектории поиска

Выберем 4 начальные точки разной степени удаленности от оптимальной ($x_1 = 5, x_2 = 6$):

1. $x_1 = 0, x_2 = 0$
2. $x_1 = 1, x_2 = 8$
3. $x_1 = 6, x_2 = 5$
4. $x_1 = 10, x_2 = 10$

x_1	x_2	$f(X)$
0.000	0.000	0.000
4.608	6.251	1 871.693
4.997	6.002	1 879.000

Таблица 8.1: $X = (0, 0)$

x_1	x_2	$f(X)$
1.000	8.000	1 215.000
4.817	5.734	1 875.745
4.998	5.997	1 879.000

Таблица 8.2: $X = (1, 8)$

x_1	x_2	$f(X)$
6.000	5.000	1 810.000
5.047	6.040	1 878.886
5.000	6.000	1 879.000

Таблица 8.3: $X = (6, 5)$

x_1	x_2	$f(X)$
10.000	10.000	640.000
5.148	5.841	1 877.369
5.000	6.000	1 879.000

Таблица 8.4: $X = (10, 10)$

На рис. 8.1 изображены траектории поиска точки максимума $X = (5, 6)$ методом Бройдена при разных начальных точках.

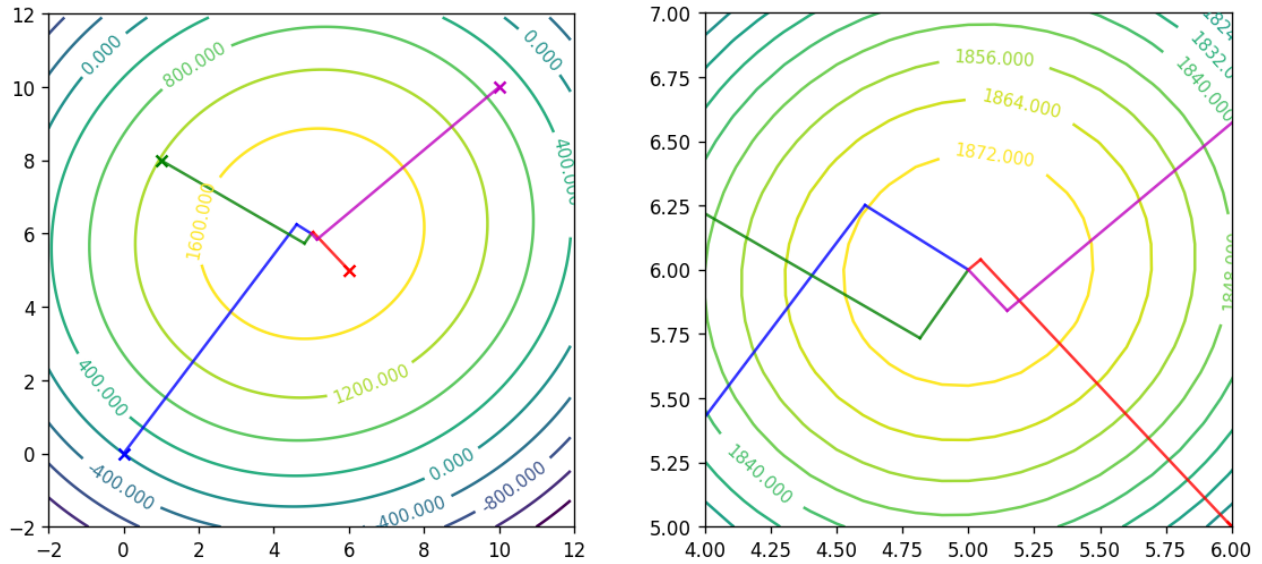


Рис. 8.1: Траектории поиска методом Бройдена