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
19 ноября 2017 г.

Содержание

Список иллюстраций

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:

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

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:

$$\frac{\partial f}{\partial x_1}(X) = -62x_1 + 4x_2 + 286 \qquad \frac{\partial f}{\partial x_2}(X) = -68x_2 + 4x_1 + 388$$

$$\frac{\partial^2 f}{\partial x_1^2}(X) = -62 \qquad \frac{\partial^2 f}{\partial x_1 \partial x_2}(X) = 4 \qquad \frac{\partial^2 f}{\partial x_2^2}(X) = -68$$

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:

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

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

Найдем точку X^*:

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

$$\begin{cases} x_1 = 17x_2 - 97 \\ -1050x_2 + 6300 = 0 \end{cases} \Rightarrow \begin{cases} x_1 = 17x_2 - 97 \\ x_2 = 6 \end{cases} \Rightarrow \begin{cases} x_1 = 5 \\ x_2 = 6 \end{cases}$$

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.

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

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

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

4.2. Исходный код

```
import numpy as np
from sympy import *


def relaxation(y, f, symbols, x_init, max_step):
    M = len(symbols)
    derivs = [y.diff(sym) for sym in symbols]
    H = np.array(hessian(y, symbols))

    X = [np.array(x_init, dtype=float)]
    step = 0
    while step < max_step:
        x = X[-1]
        for j in range(M):
            x = x.copy()
            x_curr = {symbols[j]: X[-1][j] for j in range(M)}
            grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
            grad = grad.astype(np.float32)

            K = np.zeros(M)
            K[j] = derivs[j].subs(x_curr)
            t = -grad.T.dot(K[np.newaxis].T) / K.dot(H).dot(K)

            x[j] += t * K[j]
            X.append(x)

        if np.allclose(X[-2], X[-1]) or np.allclose(f(X[-2]), f(X[-1])):
            break

        step += 1

    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. $x_1 = 0, x_2 = 0$
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
4.613	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.

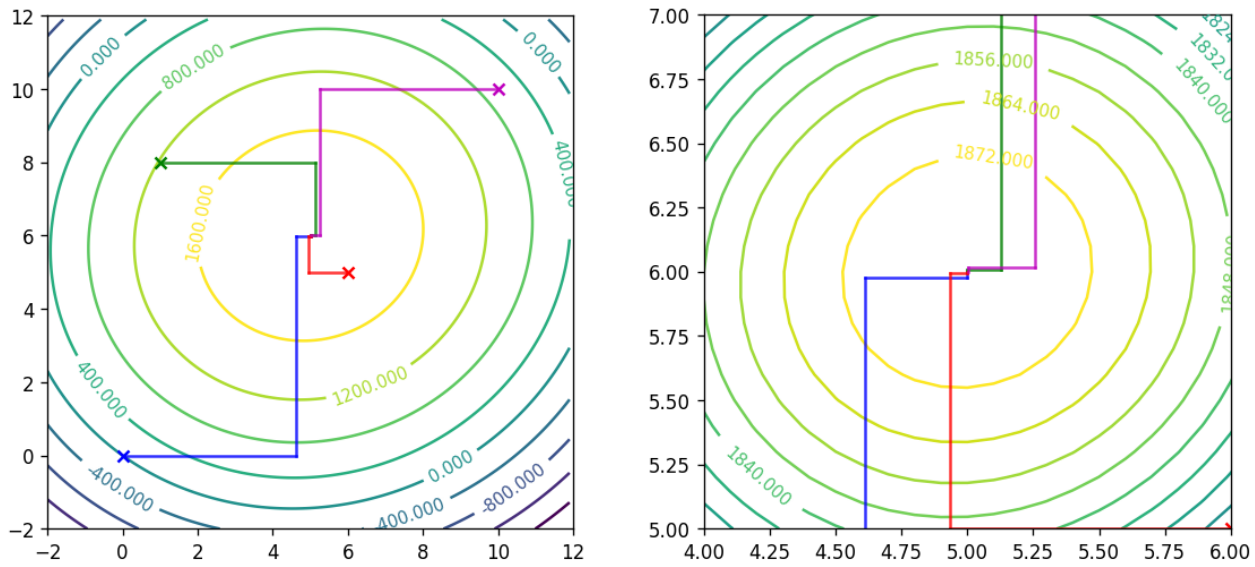

Рис. 4.1: Траектории поиска методом релаксации

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.

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

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

5.2. Исходный код

```
import numpy as np
from sympy import *


def steepest_ascent(y, f, symbols, x_init, max_step):
    M = len(symbols)
    derivs = [y.diff(sym) for sym in symbols]
    H = np.array(hessian(y, symbols))

    X = [np.array(x_init, dtype=np.float32)]
    step = 0
    while step < max_step:
        x_curr = {symbols[j]: X[-1][j] for j in range(M)}

        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
        grad = grad.astype(np.float32)

        t = -grad.dot(grad[np.newaxis].T) / grad.dot(H).dot(grad)
        x_temp = np.array(X[-1] + t * grad, dtype=np.float32)

        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp), rtol=1
    e-6):
            break

        X.append(x_temp)
        step += 1

    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.

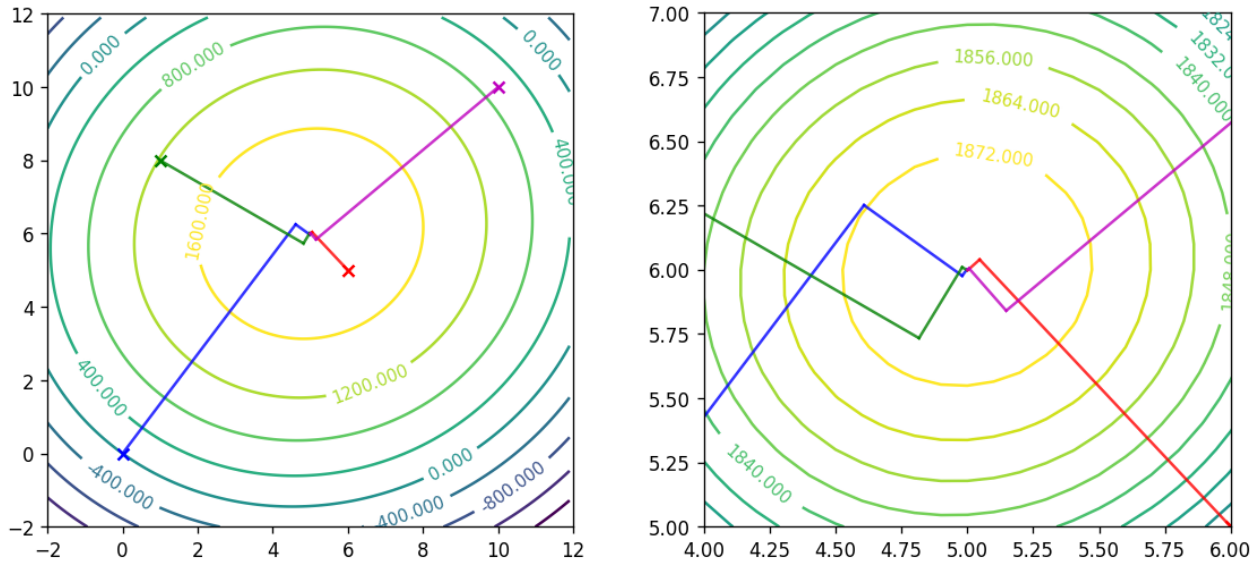

Рис. 5.1: Траектории поиска методом наискорейшего подъема

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

```python
import numpy as np
from sympy import *


def newton(y, f, symbols, x_init, max_step):
    M = len(symbols)
    derivs = [y.diff(sym) for sym in symbols]
    Hinv = np.array(hessian(y, symbols).inv())

    X = [np.array(x_init, dtype=np.float32)]
    step = 0
    while step < max_step:
        x_curr = {symbols[j]: X[-1][j] for j in range(M)}

        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
        grad = grad.astype(np.float32)

        x_temp = np.array(X[-1] - Hinv.dot(grad))
        x_temp = x_temp.astype(np.float32)

        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp)):
            break

        X.append(x_temp)
        step += 1

    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.

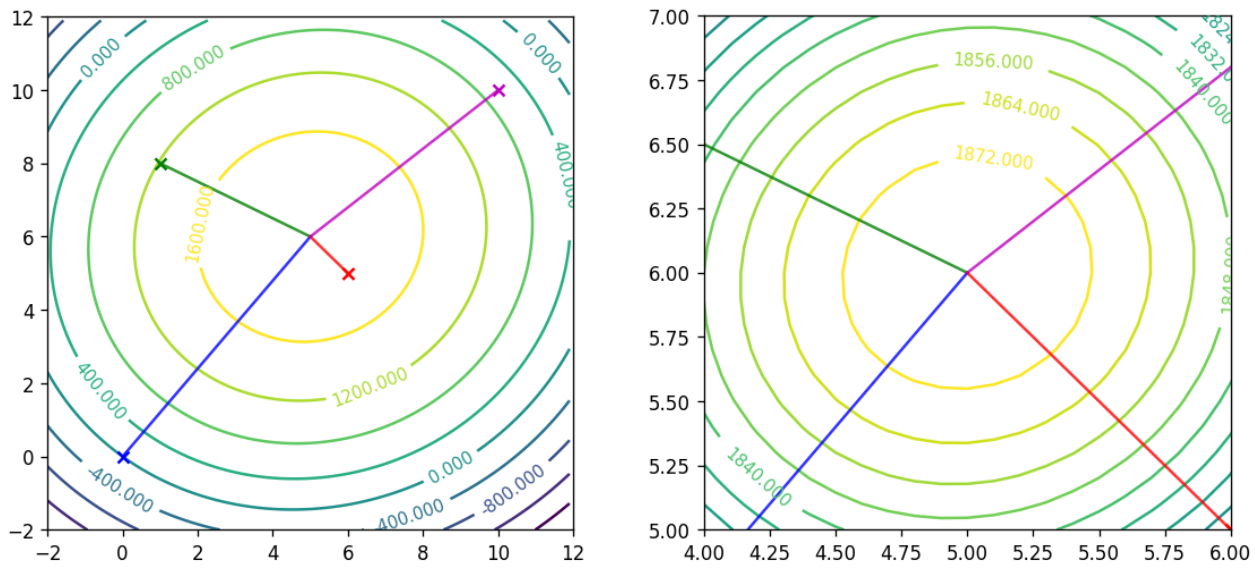

Рис. 6.1: Траектории поиска методом Ньютона

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.

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

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

Для квадратичных функций число шагов равно числу переменных.

7.2. Исходный код

```python
import numpy as np
from numpy.linalg import norm
from sympy import *


def conjugate_gradients(y, f, symbols, x_init, max_step):
    M = len(symbols)
    derivs = [y.diff(sym) for sym in symbols]
    H = np.array(hessian(y, symbols))
    X = [np.array(x_init, dtype=np.float32)]
    step = 0
    while step < max_step:
        x_curr = {symbols[j]: X[-1][j] for j in range(M)}
        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
        grad = grad.astype(np.float32)
        K = grad
        if step > 0:
            x_prev = {symbols[j]: X[-2][j] for j in range(M)}
            grad_prev = np.array([derivs[i].subs(x_prev) for i in range(M)])
            grad_prev = grad_prev.astype(np.float32)
            K += norm(grad) ** 2 / norm(grad_prev) ** 2 * grad_prev

        t = -grad.dot(grad[np.newaxis].T) / grad.dot(H).dot(grad)
        x_temp = np.array(X[-1] + t * K, dtype=np.float32)
        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp)):
            break
        X.append(x_temp)
        step += 1
    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.

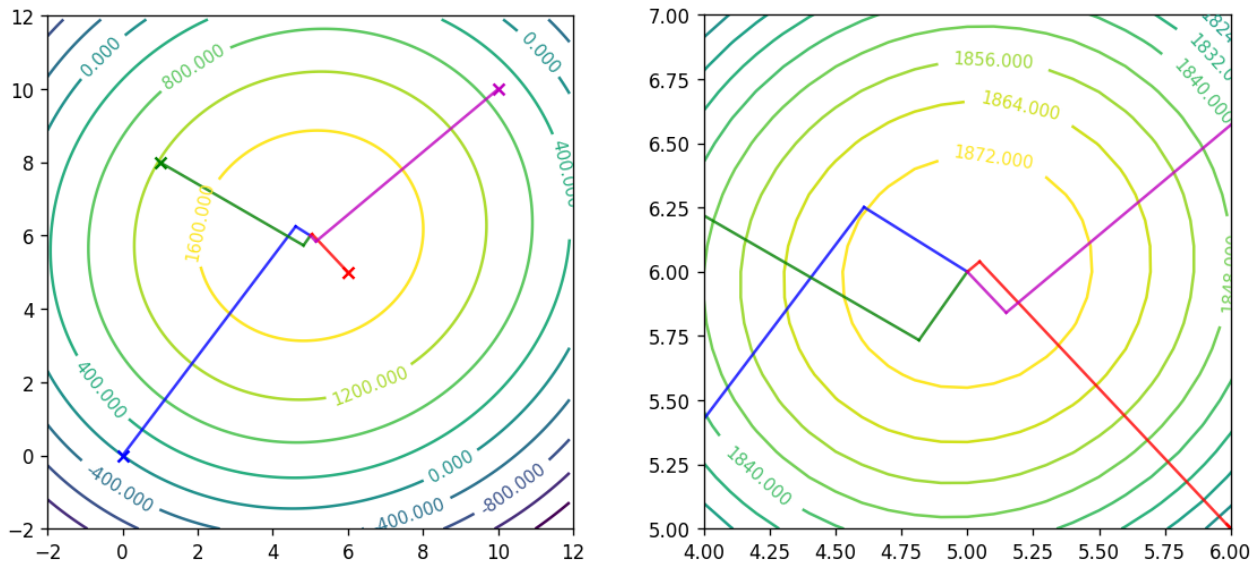

Рис. 7.1: Траектории поиска методом сопряженных градиентов

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.

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

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

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

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

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

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

8.2. Исходный код

```
import numpy as np
from sympy import *


def broyden(y, f, symbols, x_init, max_step):
    M = len(symbols)
    derivs = [y.diff(sym) for sym in symbols]
    H = np.array(hessian(y, symbols))
    X = [np.array(x_init, dtype=np.float32)]
    G = []
    N = []
    Z = []
    step = 0
    while step < max_step:
        x_curr = {symbols[j]: X[-1][j] for j in range(M)}

        grad = np.array([derivs[i].subs(x_curr) for i in range(M)])
        grad = grad.astype(np.float32)

        if step == 0:
            n = -np.eye(M)
            N.append(n)
```

```
        else:
            x_prev = {symbols[j]: X[-2][j] for j in range(M)}
            grad_prev = np.array([derivs[i].subs(x_prev) for i in range(M)])
            grad_prev = grad_prev.astype(np.float32)

            delta_g = grad - grad_prev
            delta_X = X[-1] - X[-2]
            z = delta_X - N[-1].dot(delta_g)
            Z.append(z)

            zm = z[np.newaxis]  # 1x2
            delta_nu = zm.T.dot(zm) / zm.dot(delta_g[np.newaxis].T)
            n = N[-1] + delta_nu
            N.append(n)

        K = -n.dot(grad[np.newaxis].T).flatten()
        t = -grad.dot(grad[np.newaxis].T) / grad.dot(H).dot(grad[np.newaxis].T)
        x_temp = np.array(X[-1] + t * K, dtype=np.float32)

        if np.allclose(X[-1], x_temp) or np.allclose(f(X[-1]), f(x_temp)):
            break

        X.append(x_temp)
        step += 1

    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.

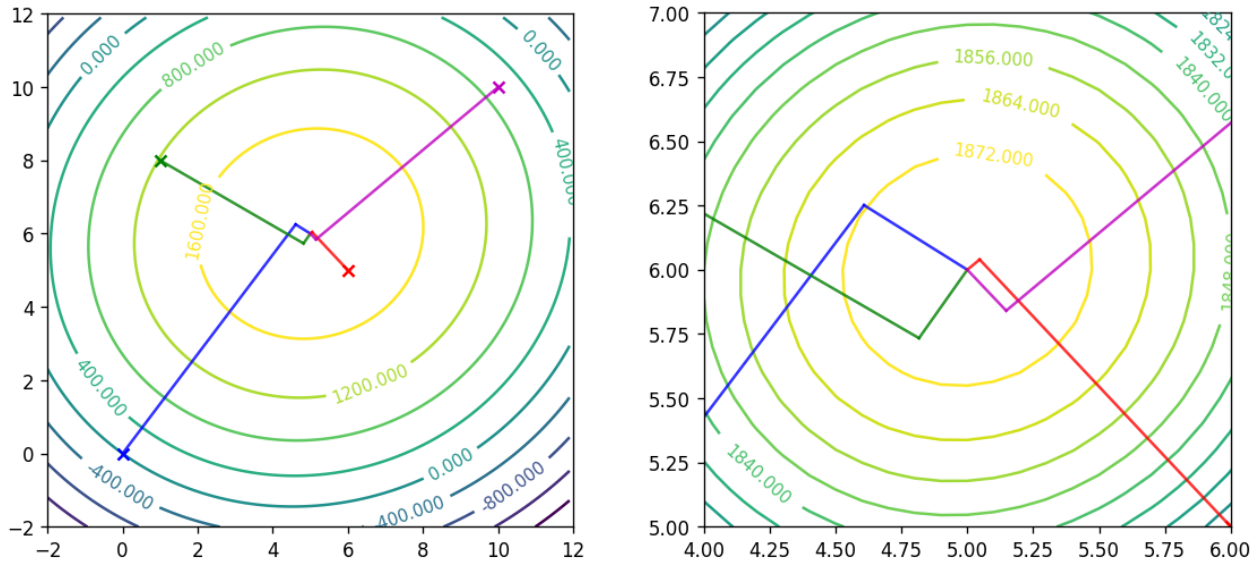

Рис. 8.1: Траектории поиска методом Бройдена