Отчёт по поиску экстремума функции методами оптимизации

14 апреля 2025 г.

1 Постановка задачи

Найти минимум функции

$$f(x_1, x_2) = 2x_1^2 + 4x_2^2 - 5x_1x_2 + 11x_1 + 8x_2 - 3$$

методами:

- покоординатного спуска,
- градиентного спуска с фиксированным шагом,
- наискорейшего спуска (с поиском оптимального шага).

Точность остановки алгоритмов: $\varepsilon = 10^{-4}$. Во всех случаях начальное приближение выбрано как

$$x^{(0)} = (0, 0).$$

2 Метод покоординатного спуска

2.1 Алгоритм

При покоординатном спуске функция минимизируется поочерёдно по каждому аргументу, то есть:

1. зафиксировав x_2 (и прочие координаты), минимизируем функцию по x_1;
2. затем, зафиксировав уже обновлённое x_1, минимизируем функцию по x_2.

На каждой подзадаче используется решение одномерной задачи оптимизации (приравнивание производной к нулю).

2.2 Ручные вычисления

Итерация 1:
Начальное приближение:

$$M^0 = (x_1^0, x_2^0) = (0, 0).$$

Шаг 1. Минимизация по x_1 при фиксированном $x_2 = 0$.
При $x_2 = 0$ функция принимает вид:

$$f(x_1, 0) = 2x_1^2 + 11x_1 - 3.$$

Найдём стационарную точку, приравняв производную по x_1 к нулю:

$$\frac{df}{dx_1} = 4x_1 + 11 = 0 \quad \Longrightarrow \quad x_1^* = -\frac{11}{4} = -2.75.$$

Получаем обновлённую точку:

$$M^1 = (-2.75,\ 0).$$

Шаг 2. Минимизация по x_2 при фиксированном $x_1 = -2.75$.
Подставляем $x_1 = -2.75$ в $f(x_1, x_2)$:

$$f(-2.75, x_2) = 2(2.75^2) + 4x_2^2 - 5(-2.75)x_2 + 11(-2.75) + 8x_2 - 3.$$

Вычислим части по шагам:

$$2(2.75^2) = 2 \cdot 7.5625 = 15.125;$$

$$-5(-2.75)x_2 = +13.75x_2, \quad \text{и } 11(-2.75) = -30.25;$$

$$8x_2 \quad \text{и } -3.$$

Таким образом, соберём:

$$f(-2.75, x_2) = 4x_2^2 + (13.75 + 8)x_2 + (15.125 - 30.25 - 3) = 4x_2^2 + 21.75x_2 - 18.125.$$

Приравнивая производную по x_2 к нулю:

$$\frac{df}{dx_2} = 8x_2 + 21.75 = 0 \quad \Longrightarrow \quad x_2^* = -\frac{21.75}{8} = -2.71875.$$

Получаем:

$$M^2 = (-2.75,\ -2.71875).$$

Итерация 2:
Начинаем с $M^2 = (-2.75,\ -2.71875)$.

Шаг 1. Минимизация по x_1 при фиксированном $x_2 = -2.71875$.
Запишем функцию:

$$f(x_1, -2.71875) = 2x_1^2 + 4(2.71875^2) - 5x_1(-2.71875) + 11x_1 + 8(-2.71875) - 3.$$

Заметим, что:

$$4(2.71875^2) = 29.5625, \quad -5x_1(-2.71875) = +13.59375\,x_1, \quad 11x_1 \text{ даёт суммарно } 24.59375\,x_1,$$

а константная часть:

$$29.5625 + 8(-2.71875) - 3 = 29.5625 - 21.75 - 3 = 4.8125.$$

Таким образом,

$$f(x_1, -2.71875) = 2x_1^2 + 24.59375\,x_1 + 4.8125.$$

Приравниваем производную:

$$\frac{df}{dx_1} = 4x_1 + 24.59375 = 0 \quad \Longrightarrow \quad x_1^* = -\frac{24.59375}{4} = -6.14844.$$

Получаем:

$$M^3 = (-6.14844,\ -2.71875).$$

Шаг 2. Минимизация по x_2 при фиксированном $x_1 = -6.14844$.
При $x_1 = -6.14844$ функция:

$$f(-6.14844, x_2) = 4x_2^2 + \alpha\, x_2 + \beta.$$

Вычисления дают:

$$\alpha \approx 38.74219, \quad \beta \approx 4.97290.$$

Приравниваем производную:

$$8x_2 + 38.74219 = 0 \quad \Longrightarrow \quad x_2^* = -\frac{38.74219}{8} = -4.84277.$$

Получаем:

$$M^4 = (-6.14844,\ -4.84277).$$

Итерация 3:
Начинаем с $M^4 = (-6.14844,\ -4.84277)$.

Шаг 1. Минимизация по x_1.
При вычислении аналогичным способом получаем уравнение:

$$4x_1 + 35.21387 = 0 \quad \Longrightarrow \quad x_1^* = -\frac{35.21387}{4} = -8.80347.$$

Обновлённая точка:

$$M^5 = (-8.80347,\ -4.84277).$$

Шаг 2. Минимизация по x_2.
При фиксированном $x_1 = -8.80347$ получаем:

$$8x_2 + 52.01735 = 0 \quad \Longrightarrow \quad x_2^* = -\frac{52.01735}{8} = -6.50217.$$

Таким образом,

$$M^6 = (-8.80347,\ -6.50217).$$

Вывод для покоординатного спуска:
После трёх полных циклов (каждый цикл — оптимизация по x_1 и x_2) получаем приближённое решение:

$$x^* \approx (-8.80347,\ -6.50217).$$

3 Метод градиентного спуска с фиксированным шагом

3.1 Алгоритм

Итерационное правило имеет вид:

$$x^{(k+1)} = x^{(k)} - \lambda \nabla f(x^{(k)}),$$

где фиксированный шаг выбран, как в данном примере, $\lambda = 0.1$. Градиент функции:

$$\nabla f(x_1, x_2) = \begin{pmatrix} 4x_1 - 5x_2 + 11 \\ 8x_2 - 5x_1 + 8 \end{pmatrix}.$$

3.2 Ручные вычисления

Итерация 1:
При $x^{(0)} = (0, 0)$:

$$\nabla f(0, 0) = \begin{pmatrix} 11 \\ 8 \end{pmatrix}.$$

Обновляем:

$$x^{(1)} = (0, 0) - 0.1\,(11, 8) = (-1.1, -0.8).$$

Вычисление функции даёт:

$$f(-1.1, -0.8) \approx -20.92.$$

Итерация 2:
При $x^{(1)} = (-1.1, -0.8)$:

$$\nabla f(-1.1, -0.8) = \begin{pmatrix} 4(-1.1) - 5(-0.8) + 11 \\ 8(-0.8) - 5(-1.1) + 8 \end{pmatrix} = \begin{pmatrix} 10.6 \\ 7.1 \end{pmatrix}.$$

Обновляем:

$$x^{(2)} = (-1.1, -0.8) - 0.1\,(10.6, 7.1) = (-2.16, -1.51),$$

а значение функции:

$$f(-2.16, -1.51) \approx -36.70.$$

Итерация 3:
При $x^{(2)} = (-2.16, -1.51)$:

$$\nabla f(-2.16, -1.51) \approx \begin{pmatrix} 9.91 \\ 6.72 \end{pmatrix}.$$

Обновляем:

$$x^{(3)} = (-2.16, -1.51) - 0.1\,(9.91, 6.72) \approx (-3.151, -2.182),$$

и функция равна:

$$f(-3.151, -2.182) \approx -50.59.$$

Вывод для градиентного спуска:
После трёх итераций получаем:

$$x^* \approx (-3.151, -2.182).$$

4 Метод наискорейшего спуска

4.1 Алгоритм

Метод наискорейшего спуска отличается тем, что на каждом шаге выбирается оптимальный шаг λ_k по направлению антиградиента:

$$x^{(k+1)} = x^{(k)} - \lambda_k \nabla f(x^{(k)}),$$

где λ_k определяется как решение одномерной задачи:

$$\min_{\lambda} f\Big(x^{(k)} - \lambda \nabla f(x^{(k)})\Big).$$

4.2 Ручные вычисления

Итерация 1:
При $x^{(0)} = (0, 0)$ имеем:

$$\nabla f(0,0) = \begin{pmatrix} 11 \\ 8 \end{pmatrix}.$$

Положим

$$\varphi(\lambda) = f(-11\lambda, -8\lambda).$$

Подставляем в функцию:

$$\begin{aligned} \varphi(\lambda) &= 2(11\lambda)^2 + 4(8\lambda)^2 - 5\,(11\lambda)(8\lambda) + 11(-11\lambda) + 8(-8\lambda) - 3 \\ &= 242\lambda^2 + 256\lambda^2 - 440\lambda^2 - 121\lambda - 64\lambda - 3 \\ &= 58\lambda^2 - 185\lambda - 3. \end{aligned}$$

Найдем минимум $\varphi(\lambda)$:

$$\varphi'(\lambda) = 116\lambda - 185 = 0 \quad \Longrightarrow \quad \lambda_0 = \frac{185}{116} \approx 1.59483.$$

Обновляем:

$$x^{(1)} = (0,0) - 1.59483\,(11, 8) \approx (-17.543, -12.7586).$$

Вычисляем:

$$f(x^{(1)}) \approx -149.04.$$

Итерация 2:
При $x^{(1)} = (-17.543, -12.7586)$:

$$\nabla f(x^{(1)}) = \begin{pmatrix} 4(-17.543) - 5(-12.7586) + 11 \\ 8(-12.7586) - 5(-17.543) + 8 \end{pmatrix} \approx \begin{pmatrix} 4.621 \\ -6.354 \end{pmatrix}.$$

Для оптимального шага решается одномерная задача для функции

$$\varphi(\lambda) = f\Big(x^{(1)} - \lambda \nabla f(x^{(1)})\Big),$$

что после свёртки приводит к квадратному многочлену:

$$\varphi(\lambda) = 351.208\lambda^2 - 62.427\lambda - 149.202.$$

Из условия $\varphi'(\lambda) = 0$ получаем:

$$702.416\lambda - 62.427 = 0 \quad \Longrightarrow \quad \lambda_1 \approx 0.0889.$$

Обновляем:

$$x^{(2)} = (-17.543, -12.7586) - 0.0889\,(4.621, -6.354) \approx (-17.953, -12.1946),$$

с $f(x^{(2)}) \approx -153.25$.

Итерация 3:
При $x^{(2)} = (-17.953, -12.1946)$ градиент:

$$\nabla f(x^{(2)}) \approx \begin{pmatrix} 0.161 \\ 0.208 \end{pmatrix}$$

почти равен нулю, поэтому дальнейшее обновление будет незначительным. Можно принять:

$$x^{(3)} \approx x^{(2)}.$$

Вывод для метода наискорейшего спуска:
Приближённое решение после двух значимых итераций:

$$x^* \approx (-17.953, -12.1946),$$

что близко к аналитически найденному минимуму (решением $\nabla f(x) = 0$):

$$x^* = \left(-\frac{128}{7}, -\frac{87}{7}\right) \approx (-18.2857, -12.4286).$$

5 Программная реализация на Python

Ниже приведён пример программы, реализующей все описанные методы.

```
import numpy as np
from scipy.optimize import minimize_scalar

def f(x):
    """ : f(x1, x2) = 2x1^2 + 4x2^2 - 5x1x2 + 11x1 + 8x2 - 3"""
    return 2 * x[0]**2 + 4 * x[1]**2 - 5 * x[0] * x[1] + 11 * x[0] + 8 * x[1] - 3

def grad_f(x):
    """  """
    return np.array([4 * x[0] - 5 * x[1] + 11, 8 * x[1] - 5 * x[0] + 8])

def coord_descent(x0, epsilon=1e-4, max_iter=1000):
    """
          f.

    Args:
        x0 (numpy.ndarray):  .
        epsilon (float): .
        max_iter (int):   .

    Returns:
        tuple:       .
    """

    x = x0.copy()
    f_values = [f(x)]
    x_history = [x.copy()] # Store the history of x

    for _ in range(max_iter):
        x_prev = x.copy()

        #   x1
        res_x1 = minimize_scalar(lambda x1: f([x1, x[1]]))
        x[0] = res_x1.x

        #   x2
        res_x2 = minimize_scalar(lambda x2: f([x[0], x2]))
        x[1] = res_x2.x

        x_history.append(x.copy())
        f_values.append(f(x))
```

```
        if np.abs(f(x) - f(x_prev)) < epsilon:
            break

    return x, f(x), x_history, f_values


def gradient_descent(x0, epsilon=1e-4, learning_rate=0.01, max_iter=1000):
    """
          f.

    Args:
        x0 (numpy.ndarray):  .
        epsilon (float): .
        learning_rate (float):   .
        max_iter (int):   .

    Returns:
        tuple:       .
    """

    x = x0.copy()
    f_values = [f(x)]
    x_history = [x.copy()] # Store the history of x

    for _ in range(max_iter):
        x_prev = x.copy()
        grad = grad_f(x)
        x = x - learning_rate * grad

        x_history.append(x.copy())
        f_values.append(f(x))

        if np.linalg.norm(x - x_prev) < epsilon or np.abs(f(x) - f(x_prev)) < epsilon:
            break

    return x, f(x), x_history, f_values

def steepest_descent(x0, epsilon=1e-4, max_iter=1000):
    """
          f.

    Args:
        x0 (numpy.ndarray):  .
        epsilon (float): .
        max_iter (int):   .

    Returns:
        tuple:       .
    """

    x = x0.copy()
    f_values = [f(x)]
    x_history = [x.copy()]

    for _ in range(max_iter):
        x_prev = x.copy()
        grad = grad_f(x)
        direction = -grad

        #
        res_alpha = minimize_scalar(lambda alpha: f(x + alpha * direction))
        alpha = res_alpha.x

        x = x + alpha * direction

        x_history.append(x.copy())
```

```
        f_values.append(f(x))


        if np.linalg.norm(x - x_prev) < epsilon or np.abs(f(x) - f(x_prev)) < epsilon:
            break

    return x, f(x), x_history, f_values

#
x0 = np.array([0.0, 0.0])

#
x_min_coord_descent, f_min_coord_descent, coord_descent_history, coord_descent_f_values =
    coord_descent(x0)
x_min_grad_descent, f_min_grad_descent, grad_descent_history, grad_descent_f_values =
    gradient_descent(x0, learning_rate=0.01) # You might need to tune learning_rate
x_min_steepest_descent, f_min_steepest_descent, steepest_descent_history,
    steepest_descent_f_values = steepest_descent(x0)

#
print("  :")
print(" :", x_min_coord_descent)
print("  :", f_min_coord_descent)

print("\n  :")
print(" :", x_min_grad_descent)
print("  :", f_min_grad_descent)

print("\n  :")
print(" :", x_min_steepest_descent)
print("  :", f_min_steepest_descent)
```