Практикум на ЭВМ Метод барьерных функций с методом Давидона-Флетчера-Пауэлла

Титушин Александр Дмитриевич

группа 411

29 сентября 2024 г.

Метод Барьерных Функций: Основная Идея

- Преобразует задачу с ограничениями:

$$\min f(x) \quad \text{при} \quad g_i(x) \leq 0, \quad i = 1, \ldots, m$$

в задачу без ограничений, добавляя барьерную функцию:

$$\phi(x, \mu) = f(x) + \mu \sum_{i=1}^{m} B_i(x)$$

где $B_i(x)$ - барьерная функция, например $-\log(-g_i(x))$.

- С каждой итерацией параметр μ уменьшается, что приводит к последовательному приближению к решению исходной задачи.

Метод Давидона-Флетчера-Пауэлла (DFP)

- Метод квази-Ньютона для минимизации функции.
- Основная идея: Итерирующее обновление приближения инверсии Гессиана с использованием градиента функции.

Метод Давидона-Флетчера-Пауэлла (DFP): Шаги алгоритма

1) **Инициализация:** Задать начальную точку x_0 и начальную аппроксимацию обратной Гессианской матрицы $H_0 = I$.

2) **Вычисление градиента:** В каждой итерации вычислить градиент функции ∇f_k в текущей точке x_k.

$$\nabla f_k = \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)\Bigg|_{x=x_k}$$

3) **Определение направления поиска:** Найти направление поиска p_k с использованием текущей аппроксимации обратной Гессианской матрицы H_k.

$$p_k = -H_k \nabla f_k$$

Метод Давидона-Флетчера-Пауэлла (DFP): Шаги алгоритма

4) **Линейный поиск:** Найти оптимальное значение шага α_k, чтобы минимизировать функцию вдоль направления p_k.

$$x_{k+1} = x_k + \alpha_k p_k$$

5) **Вычисление новых градиентов:** Вычислить новый градиент ∇f_{k+1} в точке x_{k+1}.

Метод Давидона-Флетчера-Пауэлла (DFP): Шаги алгоритма

6) **Обновление аппроксимации обратной Гессианской матрицы:** Используя формулы обновления DFP:

$$
\begin{aligned}
s_k &= x_{k+1} - x_k \\
y_k &= \nabla f_{k+1} - \nabla f_k \\
\rho_k &= \frac{1}{y_k^T s_k} \\
H_{k+1} &= \left(I - \rho_k s_k y_k^T\right) H_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T
\end{aligned}
$$

Проверка сходимости: Если $\|\nabla f_{k+1}\| < \epsilon$, остановиться. Иначе, перейти к следующей итерации.

Алгоритм в коде на Python

```
# Метод Давидона-Флетчера-Пауэлла (DFP) с барьерной функцией
def dfp_method(f, x0, mu, tol=1e-6, max_iter=100):
    n = len(x0)  # Размерность задачи
    H_k = np.eye(n)  # Инициализация матрицы H_k как единичной матрицы
    x_k = x0  # Начальная точка

    for _ in range(max_iter):
        # Вычисление градиента для комбинированной функции
        grad = gradient(lambda x: combined_func(x, mu), x_k)

        if np.linalg.norm(grad) < tol:
            break  # Если норма градиента меньше толерантности, то считаем, что решение достигнуто

        # Направление поиска
        p_k = -np.dot(H_k, grad)
        # Алгоритм линейного поиска для определения шагового размера
        alpha = line_search(lambda x: combined_func(x, mu), x_k, p_k)

        # Обновление новой точки решения
        x_k_next = x_k + alpha * p_k
        # Вычисление нового градиента
        grad_next = gradient(lambda x: combined_func(x, mu), x_k_next)

        # Обновление параметров для метода DFP
        s_k = x_k_next - x_k
        y_k = grad_next - grad
        rho = 1.0 / (np.dot(y_k, s_k))

        # Обновление матрицы H_k
        if rho > 0:
            H_k = (np.eye(n) - rho * np.outer(s_k, y_k)).dot(H_k).dot(np.eye(n) - rho * np.outer(y_k, s_k)) + rho * np.outer(s_k, s_k)

        x_k = x_k_next  # Переход к новой итерации

    return x_k
```

Алгоритм в коде на Python

```
import numpy as np
import matplotlib.pyplot as plt

# Линейный поиск для определения шагового размера
def line_search(f, x, p, alpha0=1, c1=1e-4, rho=0.9):
    alpha = alpha0
    # Условие Армихо для линейного поиска
    while f(x + alpha * p) > f(x) + c1 * alpha * np.dot(gradient(f, x), p):
        alpha *= rho
    return alpha

# Функция для вычисления градиента
def gradient(f, x, epsilon=1e-8):
    grad = np.zeros_like(x)
    for i in range(len(x)):
        x_h = x.copy()
        x_h[i] += epsilon
        grad[i] = (f(x_h) - f(x)) / epsilon
    return grad
```

Пример 1

$$f(x) = (x_0 - 3)^2 + (x_1 - 2)^2$$

Ограничения:

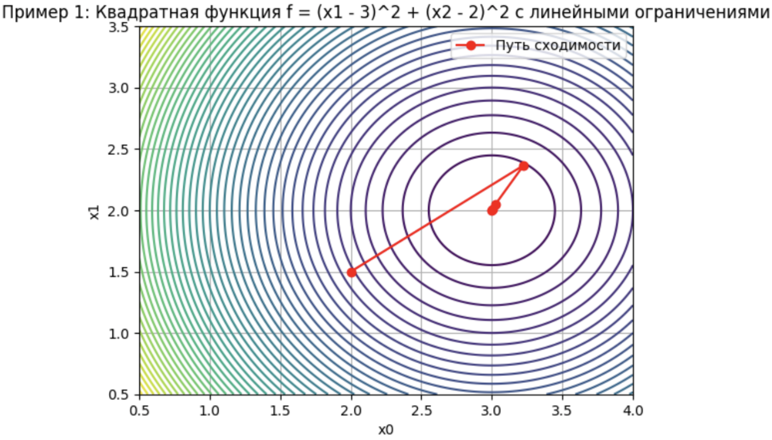
$$x_0 \geq 1, \quad x_1 \geq 1$$

```
# Целевая функция
def objective(x):
    return (x[0] - 3)**2 + (x[1] - 2)**2

# Барьерная функция для ограничений x[0] > 1 и x[1] > 1
def barrier(x):
    if x[0] <= 1 or x[1] <= 1:
        return float('inf')
    return -np.log(x[0] - 1) - np.log(x[1] - 1)

# Комбинированная функция, включающая целевую и барьерную функции
def combined_func(x, mu):
    return objective(x) + mu * barrier(x)
```

Пример 1

```
# Пример 1 ](f = (x1 - 3)^2 + (x2 - 2)^2)
x0 = np.array([2.0, 1.5])  # Начальная точка
mu_start = 1.0  # Начальное значение барьерного параметра
mu_end = 1e-3  # Конечное значение барьерного параметра
mu_reduction = 0.1  # Фактор уменьшения барьерного параметра

# Итерации метода барьерных функций
x_sol = x0
mu = mu_start
history = [x_sol.copy()]  # Сохраняем для графиков сходимости

while mu > mu_end:
    x_sol = dfp_method(lambda x: combined_func(x, mu), x_sol, mu)
    mu *= mu_reduction
    history.append(x_sol.copy())

print("Оптимальное решение:", x_sol)
print("Значение целевой функции в оптимуме:", objective(x_sol))

x_vals = np.linspace(0.5, 4, 400)
y_vals = np.linspace(0.5, 3.5, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = (X - 3)**2 + (Y - 2)**2

plt.contour(X, Y, Z, levels=50)
plt.plot([x[0] for x in history], [x[1] for x in history], 'ro-', label='Путь сходимости')
plt.xlim(0.5, 4)
plt.ylim(0.5, 3.5)
plt.xlabel('x0')
plt.ylabel('x1')
plt.legend()
plt.title('Пример 1: Квадратная функция f = (x1 - 3)^2 + (x2 - 2)^2 с линейными ограничениями')
plt.grid(True)
plt.show()
```

Пример 1



Пример 1: Квадратная функция f = (x1 - 3)^2 + (x2 - 2)^2 с линейными ограничениями
Путь сходимости
3.5
3.0
2.5
2.0
1.5
1.0
0.5
x1
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x0

Пример 2

$$f(x) = 2x_0^2 + x_0 x_1 + x_1^2$$

Ограничения:

$$x_0 \geq -1, \quad x_1 \geq -1$$

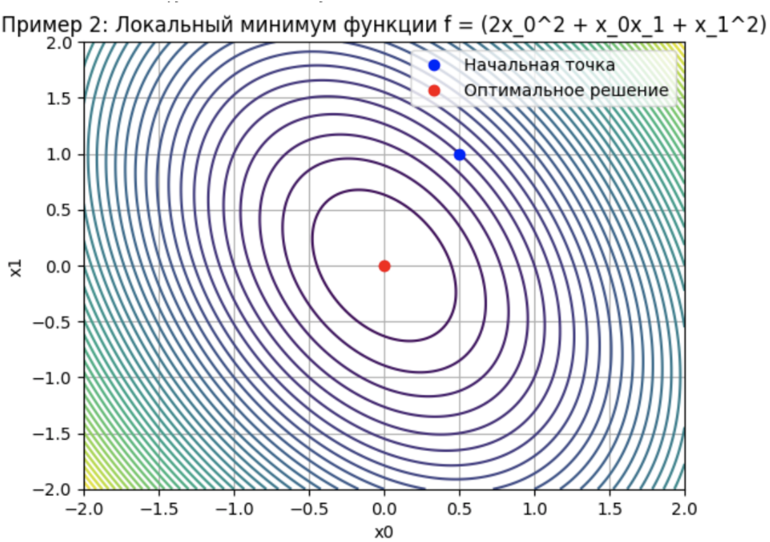
Начальная точка: $(0.5, 1.0)$ Оптимальное решение: $(0.0, 0.0)$ Значение целевой функции: 0.0

```
### Пример 2: Найти локальный минимум функции (f(x) = 2x_0^2 + x_0x_1 + x_1^2)

def objective2(x):
    return 2 * x[0]**2 + x[0] * x[1] + x[1]**2

# Барьерная функция для ограничений x[0] > -1 и x[1] > -1
def barrier(x):
    if x[0] <= -1 or x[1] <= -1:
        return float('inf')
    return -np.log(x[0] + 1) - np.log(x[1] + 1)

# Комбинированная функция, включающая целевую и барьерную функции
def combined_func(x, mu):
    return objective2(x) + mu * barrier(x)
```

Пример 2

```python
x0 = np.array([0.5, 1.0])
eps1 = 0.1
eps2 = 0.15
M = 10
mu_start = 1.0  # Начальное значение барьерного параметра
mu_end = 1e-3  # Конечное значение барьерного параметра
mu_reduction = 0.1  # Фактор уменьшения барьерного параметра

# Итерации метода барьерных функций
mu = mu_start

while mu > mu_end:
    x_sol2 = dfp_method(lambda x: combined_func(x, mu), x0, mu,tol=eps1, max_iter=M)
    mu *= mu_reduction


print("Оптимальное решение:", x_sol2)
print("Значение целевой функции в оптимуме:", objective2(x_sol2))

x_vals = np.linspace(-2, 2, 400)
y_vals = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = 2 * X**2 + X * Y + Y**2

plt.contour(X, Y, Z, levels=50)
plt.plot(x0[0], x0[1], 'bo', label='Начальная точка')
plt.plot(x_sol2[0], x_sol2[1], 'ro', label='Оптимальное решение')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.xlabel('x0')
plt.ylabel('x1')
```

Пример 2


Пример 2: Локальный минимум функции f = (2x_0^2 + x_0x_1 + x_1^2)
Начальная точка
Оптимальное решение
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x0