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

Титушин Александр Дмитриевич

группа 411

16 ноября 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 матрицу $H_0 = I$.
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 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:

$$
\begin{aligned}
r_k &= x_{k+1} - x_k \\
s_k &= \nabla f_{k+1} - \nabla f_k \\
H_{k+1} &= H_k + \frac{r_k (r_k)^T}{(r_k, s_k)} - \frac{(H_k s_k)(H_k s_k)^T}{(H_k s_k, s_k)}
\end{aligned}
$$

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

Реализация основных функций

```python
# Евклидовая норма вектора
def norm(x):
    return sum(i**2 for i in x)**0.5

# Скалярное произведение векторов
def dot_product(a, b):
    return sum(x * y for x, y in zip(a, b))

# Произведение двух векторов
def outer_product(a, b):
    result = [[0] * len(b) for _ in range(len(a))]
    for i in range(len(a)):
        for j in range(len(b)):
            result[i][j] = a[i] * b[j]
    return np.array(result)

# Перемножение двух матриц
def matrix_dot(A, B):
    result = [[0] * len(B[0]) for _ in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                result[i][j] += A[i][k] * B[k][j]
    return np.array(result)

# Транспонирование матрицы
def transpose_matrix(A):
    result = [[0] * len(A) for _ in range(len(A[0]))]
    for i in range(len(A)):
        for j in range(len(A[0])):
            result[j][i] = A[i][j]
    return np.array(result)
```

Реализация основных функций

```python
# Линейный поиск для определения оптимального шага
def line_search(f, x, p, alpha0=1, c1=1e-4, rho=0.9):
    # Инициализация начального шага
    alpha = alpha0
    # Уменьшаем alpha до тех пор, пока условие Армихо не выполняется
    while f(x + alpha * p) > f(x) + c1 * alpha * dot_product(gradient(f, x), p):
        alpha *= rho
    return alpha

# Вычисление 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

        f_x_h = f(x_h)
        f_x = f(x)

        # Избегаем неопределнности доопределением 0
        if np.isfinite(f_x_h) and np.isfinite(f_x):
            grad[i] = (f_x_h - f_x) / epsilon
        else:
            grad[i] = 0

    return grad
```

Реализация барьерной функции

```python
# Барьерная функция для произвольного числа ограничений вида g(x) >= 0
def barrier(x, constraints):
    penalty = 0
    for constraint in constraints:
        value = constraint(x)
        # Если ограничение нарушено, то возвращаем бесконечность
        if value <= 0:
            return float('inf')
        # Прибавляем к штрафу логарифм значения ограничения
        penalty -= np.log(value)
    return penalty

# Целевая функция с учетом штрафа
def combined_func(x, mu, constraints):
    return objective(x) + mu * barrier(x, constraints)
```

Реализация DFP с методом барьерных функций

```python
# Метод Давидона-Флетчера-Пауэлла (DFP) с барьерной функцией
def dfp_method(f, x0, mu, tol=1e-6, max_iter=1000):
    n = len(x0)  # Размерность пространства
    H_k = np.eye(n)  # Инициализация матрицы H как единичной
    x_k = x0  # Начальное приближение решения

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

        # Если градиент достаточно мал, то останавливаемся
        if norm(grad) < tol:
            break

        # Вычисляем направление поиска p_k
        p_k = -matrix_dot(H_k, grad.reshape(-1, 1)).flatten()
        # Определяем оптимальный шаг alpha с помощью линейного поиска
        alpha = line_search(lambda x: combined_func(x, mu, constraints), x_k, p_k)

        # Обновляем шаг
        x_k_next = x_k + alpha * p_k
        # Обновляем градиент
        grad_next = gradient(lambda x: combined_func(x, mu, constraints), x_k_next)

        # Обновление параметров для метода DFP по формулам
        r_k = x_k_next - x_k
        s_k = grad_next - grad

        # Обновление матрицы H_k по формулам метода DFP
        # Подготовка
        s_k_mat = s_k.reshape(-1, 1)

        # Само обновление H_k с учетом изменения градиента
        H_k = H_k + outer_product(r_k, r_k) / dot_product(r_k, s_k) -
            matrix_dot(matrix_dot(H_k, s_k_mat), transpose_matrix(matrix_dot(H_k, s_k_mat))) / dot_product(matrix_dot(H_k, s_k_mat), s_k)

        # Переход к следующей итерации
        x_k = x_k_next

    return x_k
```

Пример 1

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

Ограничения:

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

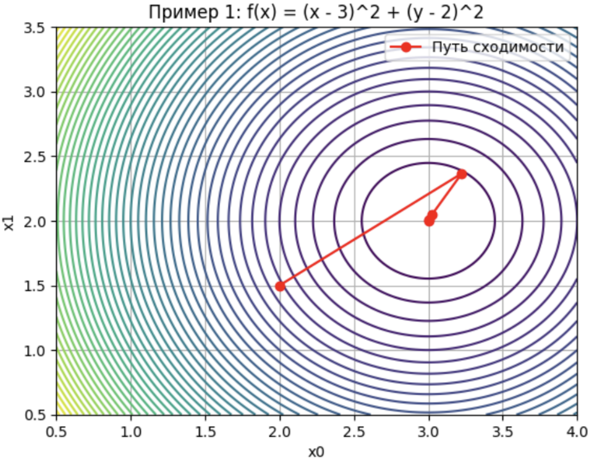
Начальная точка: $(2.0, 1.5)$ Оптимальное решение: $(3.0, 2.0)$ Значение целевой функции: 0.0

Пример 1

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

def constraint1(x):
    return x[0] - 1

def constraint2(x):
    return x[1] - 1

# Начальные данные
x0 = np.array([2.0, 1.5])  # Начальная точка
mu_start = 1.0  # Начальное значение параметра штрафа
mu_end = 1e-3  # Конечное значение параметра штрафа для прекращения процесса
mu_reduction = 0.1  # Коэффициент уменьшения параметра штрафа на каждой итерации

constraints = [constraint1, constraint2]  # Список ограничений

# Начальные значения переменных для итерационного процесса
x_sol = x0
mu = mu_start
history = [x_sol.copy()]

# Цикл сходимости
while mu > mu_end:
    # Запускаем метод DFP для нахождения минимума с текущим значением mu
    x_sol = dfp_method(lambda x: combined_func(x, mu, constraints), x_sol, mu)
    mu *= mu_reduction  # Уменьшаем mu, чтобы постепенно удалить влияние барьерной функции
    history.append(x_sol.copy())

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

# Визуализация пути сходимости
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)  # Контурный график уровней целевой функции
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: f(x) = (x - 3)^2 + (y - 2)^2')
plt.grid(True)
plt.show()
```

Пример 1

```
Оптимальное решение: [3.00024996 2.00049975]
Значение целевой функции в оптимуме: 3.12227196231781e-07
```



Пример 2

$$f(x) = 2x_0^2 + x_0x_1 + x_1^2$$

Ограничения:

$$x_0 <= 2, \quad x_1 <= 2$$

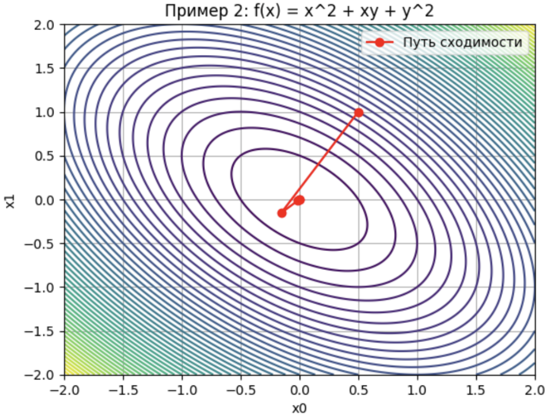
Начальная точка: $(0.5, 1.0)$ Оптимальное решение: $(0.0, 0.0)$ Значение целевой функции: 0.0

Пример 2

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

def constraint1(x):
    return -x[0] + 2

def constraint2(x):
    return -x[1] + 2

x0 = np.array([0.5, 1.0])
mu_start = 1.0
mu_end = 1e-3
mu_reduction = 0.1

constraints = [constraint1, constraint2]

x_sol = x0
mu = mu_start
history = [x_sol.copy()]

while mu > mu_end:
    x_sol = dfp_method(lambda x: combined_func(x, mu, constraints), 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(-2, 2, 400)
y_vals = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = X**2 + X * Y + Y**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(-2, 2)
plt.ylim(-2, 2)
plt.xlabel('x0')
plt.ylabel('x1')
plt.legend()
plt.title('Пример 2: f(x) = x^2 + xy + y^2')
plt.grid(True)
plt.show()
```

Пример 2

```
Оптимальное решение: [-0.00016666 -0.00016666]
Значение целевой функции в оптимуме: 8.33228680265578e-08
```



Пример на выданных функциях

Выданная целевая функция:

$$f(x) = \sum_{i=1}^{4}((x_i - 7)^2 + 42(x_{i+1} - x_i^2)^2)$$

Барьерная функция:

$$g(x) = \sum_{i=1}^{5}(i * x_i^2) \leqslant 72$$

Результат работы стандартных питоновских функций

```python
import numpy as np
from scipy.optimize import minimize

def objective(x):
    return sum([(x[i] - 7)**2 + 42 * (x[i+1] - x[i]**2)**2 for i in range(len(x) - 1)])

def constraint1(x):
    return 72 - sum([i * x[i - 1]**2 for i in range(1, len(x) + 1)])

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

constraints = [{
    'type': 'ineq',
    'fun': constraint1
}]

result = minimize(objective, x0, method='trust-constr', constraints=constraints, options={'verbose': 1})

print("Оптимальное решение:", result.x)
print("Значение целевой функции в оптимуме:", result.fun)
```

```
`xtol` termination condition is satisfied.
Number of iterations: 89, function evaluations: 606, CG iterations: 277, optimality: 1.69e-06, constraint violation: 0.00e+00, execution time: 0.18 s.
Оптимальное решение: [1.12352117 1.20159318 1.36423799 1.7874769  3.14599159]
Значение целевой функции в оптимуме: 127.83667319931448
```

Оптимальное решение: [1.12352117 1.20159318 1.36423799 1.7874769 3.14599159] Значение целевой функции в оптимуме: 127.83667319931448

Результат работы реализованного DFP метода с барьерной функцией

```
def objective(x):
    return sum([(x[i] - 7)**2 + 42 * (x[i+1] - x[i]**2)**2 for i in range(len(x) - 1)])

def constraint1(x):
    return 72 - sum([i * x[i - 1]**2 for i in range(1, len(x) + 1)])

# Начальные данные
x0 = np.array([0.0, 0.0, 0.0, 0.0, 0.0])  # Начальная точка
mu_start = 1.0  # Начальное значение параметра штрафа
mu_end = 1e-3  # Конечное значение параметра штрафа для прекращения процесса
mu_reduction = 0.1  # Коэффициент уменьшения параметра штрафа на каждой итерации

constraints = [constraint1]  # Список ограничений

# Начальные значения переменных для итерационного процесса
x_sol = x0
mu = mu_start
history = [x_sol.copy()]

# Цикл сходимости
while mu > mu_end:
    # Запускаем метод DFP для нахождения минимума с текущим значением mu
    x_sol = dfp_method(lambda x: combined_func(x, mu, constraints), 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))
```

```
<ipython-input-12-de54bedd8332>:65: RuntimeWarning: invalid value encountered in scalar subtract
  grad[i] = (f(x_h) - f(x)) / epsilon
Оптимальное решение: [1.12351745 1.2015847  1.36421716 1.78741893 3.14578244]
Значение целевой функции в оптимуме: 127.83767291054248
```

Результат работы стадартных питоновских функций без ограничений

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

# Определение целевой функции
def objective(x):
    return sum([(x[i] - 7)**2 + 42 * (x[i+1] - x[i]**2)**2 for i in range(len(x) - 1)])


### Оптимизация

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

# Выполнение оптимизации
result = minimize(objective, x0)

# Результаты
optimal_x = result.x
optimal_value = result.fun

print("Оптимальные значения переменных:", optimal_x)
print("Оптимальное значение функции:", optimal_value)
```

```
Оптимальные значения переменных: [ 1.32483905  1.7042027   2.85235099  8.10948754 65.76378813]
Оптимальное значение функции: 78.93878870180826
```

Оптимальные значения переменных: [1.32483905 1.7042027 2.85235099 8.10948754 65.76378813]
Оптимальное значение функции: 78.93878870180826

Результат работы реализованного DFP метода с увеличенной барьерной функцией

```
def objective(x):
    return sum([(x[i] - 7)**2 + 42 * (x[i+1] - x[i]**2)**2 for i in range(len(x) - 1)])

def constraint1(x):
    return 2000000 - sum([i * x[i - 1]**2 for i in range(1, len(x) + 1)])

x0 = np.array([1.0, 1.0, 1.0, 1.0, 1.0])
mu_start = 1.0
mu_end = 1e-3
mu_reduction = 0.1

constraints = [constraint1]

x_sol = x0
mu = mu_start
history = [x_sol.copy()]

while mu > mu_end:
    x_sol = dfp_method(lambda x: combined_func(x, mu, constraints), 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))
```

```
Оптимальное решение: [ 1.3248356   1.70420186  2.85235111  8.10948573 65.76375843]
Значение целевой функции в оптимуме: 78.9387887058618
```

Выводы

Насколько видно из проведенных экспериментов глобальное оптимальное решение xt = [1.32483905 1.7042027 2.85235099 8.10948754 65.76378813] $g(xt) = \sum_{i=1}^{5}(i * x_i^2) = 21918 > 72$ для выданной целевой функции f(x) находится за пределами выданной барьерной функции g(x), поэтому программа выдает с точностью 0.001 локальное оптимальное решение на доступном для нее пространстве (сверяем со стандартной питоновской выдачей). Если увеличить g(x) до <= 21920 и более, то программа выдает с точностью 0.00000001 верное глобальное оптимальное решение (также сверяем со стандартными питоновскими методами оптимизации).