Отчет по решению задачи оптимизации методом Франка-Вульфа

Тишина Ульяна

March 16, 2025

Contents

1 Постановка задачи

Рассмотрим задачу оптимизации:

$$f(x) \to min, \tag{1}$$

где f(x) - нелинейная функция нескольких переменных

С линейными ограничениями:

$$A * x >= b, \tag{2}$$

$$A\_eq * x = b\_eq \tag{3}$$

2 Метод Франка-Вульфа

Метод Франка - Вульфа применяется для задач с нелинейной целевой функцией и линейными ограничениями. Метод основан на разложении нелинейной функции общего вида в ряд Тейлора до членов первого порядка в окрестности некоторой точки

Алгоритм:

1) Вычислим градиент функции в точке x^k:

$$grad(f(x^k)) = (\frac{df(x^k)}{dx_1}, ..., \frac{df(x^k)}{dx_n})$$

2) Составим новую линейную целевую функцию,

$$F_k(y) = \frac{df(x^k)}{dx_1}y_1 + ... + \frac{df(x^k)}{dx_n}y_n \to min$$

Получаем Задачу Линейного Программирования.

3) Решаем полученноую ЗЛП с ограничениями. Например, симплекс-методом. Обозначим решение y^k

4) Найдем минимум функции $f(x^{k+1}) \to min$,

где $x^{k+1} = x^k + h^k(y^k - x^k)$,

$h^k \in (0, 1)$

Например, методом золотого сечения.

5) Проверяем критерии выхода, затем либо останавливаемся, либо k+=1 и переходим к п.1

3 Реализация функции золого сечения и производной

```
def my_grad(f, x, h=1e-6):
    grad = np.zeros_like(x)
    for i in range(len(x)):
        x_plus_h = np.copy(x)
        x_plus_h[i] += h
        grad[i] = (f(x_plus_h) - f(x)) / h
    return grad

def find_xk(f, a,b, tol=1e-6, max_iter = 100):
    hk = 2 - (1 + 5**0.5) / 2

    x1 = a + hk * (b - a)
    x2 = b - hk * (b - a)
    # print(x1,x2)
    f1 = f(x1)
    f2 = f(x2)
    # print(f1,f2)

    for _ in range(max_iter):
        # print(x1,x2)
        if norm(b - a) < tol:
            break
        # print("f1f2",f1,f2)
        if f1 < f2:
            b, x2, f2 = x2, x1, f1
            x1 = a + hk * (b - a)
            f1 = f(x1)
        else:
            a, x1, f1 = x1, x2, f2
            x2 = b - hk * (b - a)
            f2 = f(x2)

    return (a + b) / 2
```

4 Реализация метода Франка-Вульфа

Код на Python, реализующий метод Франка-Вульфа, представлен ниже:

```
def frank_wolf(f, x0, A, b, A_eq, b_eq, max_iter, eps, view=0):
    xk = x0
    k = 0
    opt = {'k': 0,
           'x': x0,
           'f': f,
           'status': -1}

    while (k <= max_iter):
        df = my_grad(f, xk)
        if view: print(f'grad(x{k}) = {df}')

        # minimize
        res = linprog(df, A_ub=-A, b_ub=-b, A_eq=A_eq, b_eq=b_eq)
        yk = res.x
        if view: print(f'y{k} = {yk}')

        # alpha search for xk+1 = xk + ak(yk-xk) for 0<=ak<=1
        prev = xk
        xk = find_xk(f,xk,yk)
        if view: print(f'x{k} = {xk}')

        # fill opt
        opt['k'] = k
        opt['f'] = f(xk)
        opt['x'] = xk
        k += 1
        # break
        if (norm(xk - prev) < eps) and (abs(f(xk) - f(prev)) < eps):
            opt['status'] = 0
            break
    return opt
```

5 Проверка работы

5.1 Пример 1

$$f(x) = x_0^0.25 + (x_1/x_0)^0.25 + (64/x_1)^0.25 \to min,$$

$x_1 >= 1, x_2 - x_1 >= 0, -x2 >= -64$

5.1.1 Реализация

```
def f(x):
    return x[0]**0.25 + (x[1]/x[0])**0.25 + (64/x[1])**0.25

A = np.array([[1, 0],
              [-1, 1],
              [0, -1]])
b = np.array([1, 0, -64])
A_eq, b_eq = None, None
```

5.1.2 Применение метода

```
x0 = np.array([2,10])
max_iter = 100
eps = 0.001
frank_wolf(f, x0, A, b, A_eq, b_eq, max_iter, eps, 0)
```

5.1.3 Вывод функции

```
{'k': 13,
 'x': array([ 3.99993784, 15.99925072]),
 'f': 4.242640687270153,
 'status': 0}
```

6 Список литературы

Лобанов, В. С. Метод линеаризации для задач условной оптимизации. Алгоритм Франка-Вульфа — URL: https://moluch.ru/archive/293/66414/