Отчет по решению задачи оптимизации методом Франка-Вульфа

Тишина Ульяна

26 марта 2025 г.

Содержание

1 Постановка задачи **3**

2 Метод Франка-Вульфа **3**

3 Проверка корректности реализации алгоритма **4**

Список литературы **7**

Приложение **8**

1 Постановка задачи

Рассмотрим задачу оптимизации:

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

где f(x) - нелинейная функция нескольких переменных,
с линейными ограничениями:

$$\begin{gathered} A * x \geq b, \\ A\_eq * x = b\_eq \end{gathered} \tag{2}$$

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 Метод Франка-Вульфа

Входные данные: $f(x)$, x^0, ограничения, критерии выхода

$k \leftarrow 0$
пока Критерии выхода не выполнены **выполняй**
  вычислить градиент: $\nabla f(x^k) = \left(\frac{\partial f(x^k)}{\partial x_1}, \dots, \frac{\partial f(x^k)}{\partial x_n}\right)$;
  составить линейную целевую функцию: $F_k(y) = \nabla f(x^k) \cdot y \to \min$;
  решить ЗЛП с ограничениями (например, симплекс-методом), получить y^k;
  найти минимум функции $f(x^{k+1}) \to min$, где $x^{k+1} = x^k + h^k(y^k - x^k)$, $h^k \in (0, 1)$, применив, например, метод золотого сечения;
  $x^{k+1} \leftarrow x^k + h^k(y^k - x^k)$;
  $k \leftarrow k + 1$.
Конец цикла пока
Вернуть x^k

3 Проверка корректности реализации алгоритма

Для проверки работы алгоритма будем использовать библиотеку scipy.optimize. В ней есть функция minimize, которая находит решение задачи оптимизации с нелинейной или линейной целевой функцией с нелинейными и/или линейными ограничениями-равенствами и/или ограничениями-неравенствами.

Сравним результат работы реализованного алгоритма Франка-Вульфа и функции минимазации из библиотеки: найденные значения целевой функции, количество итераций и статус завершения.

В реализации алгоритма Франка-Вульфа успешному завершению соответствует статус 0, не успешному статус -1. В библиотечной функции успеху также соответствует статус 0, а неуспешному - другое число, значение которого можно найти в документации библиотеки.

В таблицах представлены результаты двух алгоритмов и последний столбец - сравнение этих результатов следующим образом:

- разница значений целевой функции (должна быть меньше заданной точности);
- сравнение кол-ва итераций: у какого метода меньше, тот метод лучше (будем писать для метода Франка-Вульфа (Ф-В): Ф-В лучше или хуже);
- сравнение статусов алгоритмов: если оба алгоритма завершились успехом, то запишем 0, иначе -1.

Тест 1 Рассмотрим задачу оптимизации

$$\begin{gathered} f(x) = x_1^{0.25} + (\tfrac{x_2}{x_1})^{0.25} + (\tfrac{64}{x_2})^{0.25} \to min, \\ x_1 \geq 1, \\ x_2 - x_1 \geq 0, \\ -x_2 \geq -64. \end{gathered} \tag{3}$$

	Метод Франка-Вульфа	Минимизация	Сравнение
F	4.242641	4.242641	0
x	[3.999938, 15.999251]	[3.999027, 16.004494]	-
Количество итераций	13	18	Ф-В лучше
Статус	0	0	0

Таблица 1: 1, $x_0 = (2, 10)$, $\varepsilon = 0.001$.

Тест 2 Рассмотрим задачу оптимизации

$$\begin{gathered} f(x) = (x_1 - 1)^2 + 2(x_2 - 1)^2 - 3 \to min, \\ -x_1 - x_2 \geq -8, \\ -2x_1 + x_2 \geq -12, \\ x_1 \geq 0, \\ x_2 \geq 0. \end{gathered} \tag{4}$$

	Метод Франка-Вульфа	Минимизация	Сравнение
F	-3.0	-3.0	0
x	[0.999987, 0.999818]	[1.000000, 1.000000]	-
Количество итераций	5	3	Ф-В хуже
Статус	0	0	0

Таблица 2: 2, $x_0 = (0,\ 0)$, $\varepsilon = 0.001$.

Тест 3 Рассмотрим задачу оптимизации

$$\begin{gathered} f(x) = -(-x_1^2 + x_1x_2 - 2x_2^2 + 4x_1 + 6x_2) \to min, \\ -x_1 - x_2 \geq -4, \\ x_1 + 2x_2 \geq 2, \\ x_1 \geq 0, \\ x_2 \geq 0. \end{gathered} \tag{5}$$

	Метод Франка-Вульфа	Минимизация	Сравнение
F	-12.25	-12.25	0
x	[2.250000, 1.750000]	[2.250000, 1.750000]	-
Количество итераций	1	2	Ф-В лучше
Статус	0	0	0

Таблица 3: 3, $x_0 = (3,\ 1)$, $\varepsilon = 0.001$.

Тест 4 Рассмотрим задачу оптимизации

$$
\begin{gathered}
f(x) = sin(x_1) + cos(x_2^2) \to min, \\
-x_1 - x_2 \geq -4, \\
2x_1 + 5x_2 \geq 1, \\
x_1 \geq 0, \\
x_2 \geq 0.
\end{gathered} \tag{6}
$$

	Метод Франка-Вульфа	Минимизация	Сравнение
F	-0.999992	-1.0	0.000008
x	[0.000008, 1.772453]	[0.000000, 3.963327]	-
Количество итераций	14	6	Ф-В хуже
Статус	0	0	0

Таблица 4: 4, $x_0 = (0.1,\ 0.1)$, $\varepsilon = 0.0001$.

Во всех примерах значение функции найдено успешно, алгоритм работает корректно, с заданной точностью.

Список литературы

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

2. Алгоритм Франка-Вульфа. (пример1) https://math.semestr.ru/optim/frank.php

3. Алгоритм Франка-Вульфа. (пример2) https://studfile.net/preview/7775095/page:30/

4. Алгоритм Франка-Вульфа. (пример3) https://www.matburo.ru/ex_mp.php?p1=mpnp

Приложение

Реализация метода Франка-Вульфа:

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

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

        # minimize
        if A is None:
            res = linprog(df, A_eq=A_eq, b_eq=b_eq)
        else:
            res = linprog(df, A_ub=-A, b_ub=-b, A_eq=A_eq, b_eq=b_eq)
        yk = res.x
        if yk is None:
            opt['status'] = -1
            if view: print("yk is None: ", res)
            return opt
        if res.status != 0:
            opt['status'] = -1
            if view: print("status of linprog is not 0: ", res)
            return opt
        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(fi,xk,yk)
        if view: print(f'x{k} = {xk}')

        # fill opt
        opt['k'] = k
        opt['f'] = fi(xk)
        opt['x'] = xk
        k += 1
```

```
        # break
        if (norm(xk - prev) < eps) and (abs(fi(xk) - fi(
            prev)) < eps):
            opt['status'] = 0
            break
    return opt
```

Реализация функции, вычисляющей производную

```
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
```

Реализация функции золого сечения

```
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
```

Реализация функции сравнения результатов

```python
def compare_results_df(res_fw, res_min, eps=1e-3, r=4):
    for i in range(len(res_fw)):
        fw = res_fw[i]
        mn = res_min[i]

        fun_comparison = abs(fw['f'] - mn.fun) < eps

        status_comparison = 0 if ((fw['status'] == mn.status) == 0) else -1

        x_comparison = "-"

        if fw['k'] == mn.nit:
            nit_comparison = "eq"
        elif fw['k'] > mn.nit:
            nit_comparison = "F-W␣worse"
        else:
            nit_comparison = "F-W␣better"

        data = {
            "frank_wolf": [fw['f'], fw['status'], round_arr(fw['x']), fw['k']],
            "minimize": [mn.fun, mn.status, round_arr(mn.x), mn.nit],
            "delta": [fun_comparison, status_comparison, x_comparison, nit_comparison]
        }

        df = pd.DataFrame(data, index=["f", "status", "x", "nit"])

        print(f"\n            ␣{i+1}:")
        print(df)
```