

— Линейное программирования

Отчет по лабораторной работе №1

Работу выполнили:

Обиджанов Алишер
Какзаков Андрей
Кузнецов Павел

Преподаватель:

Свинцов М.В.

1 Теория

Задача линейного программирования - максимизировать или минимизировать некоторый линейный функционал на многомерном пространстве при заданных линейных ограничениях.

Каждое из линейных неравенств на переменные ограничивает полупространство в соответствующем линейном пространстве. В результате все неравенства ограничивают некоторый выпуклый многогранник (возможно, бесконечный), называемый также полиэдральным комплексом.

Максимум функционала можно искать в вершинах многогранника. Принцип симплекс-метода состоит в том, что выбирается одна из вершин многогранника, после чего начинается движение по его рёбрам от вершины к вершине в сторону увеличения значения функционала. Когда переход по ребру из текущей вершины в другую вершину с более высоким значением функционала невозможен, считается, что оптимальное значение с найдено.

Последовательность вычислений симплекс-методом можно разделить на две основные фазы:

1. Нахождение исходной вершины множества допустимых решений,
2. Последовательный переход от одной вершины к другой, ведущей к оптимизации значения целевой функции.

2 Постановка задачи

1. Реализуйте возможность ввода данных из файла в формате JSON. Рекомендуемая структура JSON указана ниже.
2. При необходимости добавьте балансирующие переменные для перехода от общей постановки к канонической форме задачи линейного программирования.
3. Реализуйте симплекс-метод для решения задачи.
4. Предусмотрите, что задача как может не иметь решений вообще, так и иметь бесконечное количество решений

3 Реализация

3.1 Подключаем библиотеки

```
import numpy as np
from json import loads
```

3.2 Подготавливаем пример в формате json

```
example = """{"f": [1, 2, 3],
  "goal": "max",
  "constraints": [{"coefs": [1, 0, 0],
    "type": "eq",
    "b": 1},
    {"coefs": [1, 1, 0],
    "type": "gte",
    "b": 2},
    {"coefs": [1, 1, 1],
    "type": "lte",
    "b": 3}]}"""
```

3.3 Парсим json файл

Преобразует json в матрицы f , A и b , которые затем возвращает. Если задача имеет цель "min" то целевая функция домножается на -1. Коэффициенты у всех ограничений в виде неравенств вида "больше" инвертируются, тем самым превращаясь в ограничения вида "меньше". Когда встречается равенство, оно делится на два неравенства, одно с неизменённым знаком, другое - с изменённым.

```

def parse_problem(json: str):
    parsed = loads(json)
    f = np.array(parsed['f'])
    if parsed['goal'] == "min":
        f *= -1
    A = []
    b = []
    for constraint in parsed['constraints']:
        A.append(np.array(constraint['coefs']))
        b.append(constraint['b'])
        if constraint['type'] == "gte":
            A[-1] *= -1
            b[-1] *= -1
        if constraint['type'] == "eq":
            A.append(np.array(constraint['coefs']) * -1)
            b.append(-constraint['b'])
    A = np.array(A)
    A = np.hstack((A, np.eye(A.shape[0])))
    b = np.array(b)
    b = b.reshape((b.shape[0], 1))
    return f, A, b

```

3.4 Симплекс-метод

Сначала создается таблица `tableau`, которая объединяет матрицы `A` и `b`. Если в таблице есть отрицательные элементы в столбце `b`, то выбирается строка `i` с минимальным значением `b` и столбец `l` с минимальным значением в этой строке. Затем находятся отношения между элементами строки `i` и столбцом `l` и выбирается строка `r` с наименьшим отношением. Далее выполняется шаг симплекс-метода для выбранных строк и столбцов.

3.4.1 Шаг симплекс-метода

```

def simplex_step(tableau, r, l):
    print(f"doing simplex step with {r=} and {l=}")
    for i in range(tableau.shape[0]):
        if i == r:
            tableau[i] /= tableau[i, l]
            continue
        tableau[i] -= tableau[r] * tableau[i, l] / tableau[r, l]

```

3.4.2 Симплекс-метод

```
def simplex(f, A, b: np.ndarray, isMin):
    tableau = np.hstack((b.reshape(-1, 1), A))
    tableau = np.vstack((tableau, np.hstack((np.zeros((1,)), f, np.zeros((A.shape[0]))))))
    print(tableau)
    basis = np.arange(A.shape[0], A.shape[0] * 2) - 1
    print(basis)
    while (tableau[-1, 0].min() < 0):
        i = tableau[-1, 0].argmin()
        l = tableau[i, 1:].argmin() + 1
        if tableau[i, l] >= 0:
            raise Exception("No solution.")
        ratios = tableau[-1, 0] / tableau[-1, l]
        r = np.where(ratios > 0, ratios, np.inf).argmin()
        simplex_step(tableau, r, l)
        basis[r] = l
        print(tableau)
    s = (tableau[-1, 1:].argmax() if isMin else tableau[-1, 1:].argmin()) + 1
    last_max = -1
    while tableau[-1, 1:].min() < 0 if isMin else tableau[-1, 1:].max() > 0:
        # print(tableau[-1, s])
        temp = tableau[-1, s]
        temp[temp == 0] = -1
        ratios = tableau[-1, 0] / temp
        print(f"ratios:\n{ratios}")
        j = ratios.argmin()
        simplex_step(tableau, j, s)
        basis[j] = s
        print(basis)
        print(tableau)
        s = (tableau[-1, 1:].argmax() if isMin else tableau[-1, 1:].argmin()) + 1
        if tableau[-1, s] == last_max:
            break
        last_max = tableau[-1, s]
    return -tableau[-1, 0]
```

4 Результат

```
print(simplex(*parse_problem(example)))
```

Output:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 2 & 0 & 1 & 0 \\ 0 & -2 & 1 & 1 & -3 & 0 & 0 & 1 \\ 0 & 1 & 2 & 3 & 0 & 0 & 0 & 0 \end{pmatrix}$$
$$(0 \ 0 \ 0 \ 1)$$

$$\begin{pmatrix} -1 & -1 & -0 & 1 & -1 & -0 & -0 & -0 \\ -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 \\ 1 & -1 & 1 & 0 & -2 & 0 & 0 & 1 \\ 3 & 4 & 2 & 0 & 3 & 0 & 0 & 0 \end{pmatrix}$$

$$(-1 \quad -1 \quad 0 \quad -1)$$

$$\begin{pmatrix} -2 & 0 & -1 & 1 & 1 & -0 & -0 & -1 \\ -2 & 0 & -1 & 0 & 2 & 1 & 0 & -1 \\ -2 & 0 & -2 & 0 & 3 & 0 & 1 & -1 \\ -1 & 1 & -1 & -0 & 2 & -0 & -0 & -1 \\ 7 & 0 & 6 & 0 & -5 & 0 & 0 & 4 \end{pmatrix}$$

$$7.0$$

5 Тестирование

5.1 Задачи для тестирования

```
[
  {
    "task": {
      "f": [1, 2, 3],
      "goal": "max",
      "constraints": [{"coefs": [1, 0, 0],
        "type": "eq",
        "b": 1},
        {"coefs": [1, 1, 0],
        "type": "gte",
        "b": 2},
        {"coefs": [1, 1, 1],
        "type": "lte",
        "b": 3}],
      "correct_answer": 3.0
    },
    {
      "task": {
        "f": [3, 4],
        "goal": "max",
        "constraints": [{"coefs": [-1, 1],
          "type": "lte",
          "b": 3},
          {"coefs": [5, 3],
          "type": "lte",
          "b": 97},
          {"coefs": [1, 7],
          "type": "gte",
          "b": 74}],
        "correct_answer": 58.375
      },
      {
        "task": {
          "f": [5.2, -1],
          "goal": "min",
          "constraints": [{"coefs": [2, 5],
            "type": "gte",
            "b": 10},
            {"coefs": [1, -3],
            "type": "lte",
            "b": 3},
            {"coefs": [-1, 1],
            "type": "lte",
            "b": 1}],
          "correct_answer": 2
        },
        {
          "task": {
            "f": [-3, 1, 4, 0, 0],
            "goal": "max",
            "constraints": [{"coefs": [0, -1, 1, 1, 0],
              "type": "eq",
              "b": 1},
              {"coefs": [-5, 1, 1, 0, 0],
              "type": "eq",
              "b": 2},
              {"coefs": [-8, 1, 2, -1, 0],
              "type": "eq",
              "b": 3}],
            "correct_answer": 10
          }
        }
      }
    }
  ]
```

5.2 Результат тестов

task number	correct answer	our answer	equal
0	3.0	3.0	true
1	58.375	58.375	true
2	2	2.0	true
3	10	10.0	true

Рис. 1:

6 Вывод

В ходе данной лабораторной работы был реализован симплекс-метод для решения задачи линейного программирования. Была предусмотрена возможность ввода данных из файла в формате JSON. Были добавлены балансирующие переменные для перехода от общей постановки к канонической форме задачи линейного программирования.