Прикладная математика
Лабораторная работа №6
Симплекс-метод

M33001
ВОЕВОДСКИЙ ДМИТРИЙ
ЕВТУШЕНКО ИВАН
БЛАЖКОВ АЛЕКСАНДР

2022 год, университет ИТМО

1 Задача

Имеется задача линейного программирования в произвольной форме, необходимо найти ее решение

2 Теория

2.1 Формы задачи линеного программирования

Задача линейного программирования состоит из формы для минимизации(максимизации) вида

$$\sum_{i=1}^{n} c_i \cdot x_i \to \min(\max)$$

и набора ограницений вида

$$\sum_{i=1}^{n} a_{ij} \cdot x_i \le b_j, j = \overline{1, m}$$

$$\sum_{i=1}^{n} a_{ij} \cdot x_i \ge b_j, j = \overline{(m+1), k}$$

$$\sum_{i=1}^{n} a_{ij} \cdot x_i = b_j, j = \overline{(k+1), p}$$

$$x_i > 0, i = \overline{1, n}$$

Форма задачи, описанная выше, называется произвольной

2.1.1 Каноническая форма

$$\begin{cases} \sum_{i=1}^{n} c_i \cdot x_i \to \min(\max) \\ \sum_{i=1}^{n} a_{ij} \cdot x_i = b_j, j = \overline{1, m} \\ x_i > 0, i = \overline{1, n} \end{cases} \tag{1}$$

Для приведения задачи в каноническую форму вводятся новые переменные

$$A \cdot \overline{x} \le \overline{b}$$

$$A \cdot \overline{x} + B \cdot \overline{y} = \overline{b}, \overline{x} \ge 0, \overline{y} \ge 0$$

2.2 метод искусственного базиса

Введем m новых переменных, $u_1..u_m$

$$\begin{cases} \sum_{i=1}^{n} c_i \cdot x_i + \sum_{i=1}^{m} M \cdot u_i \to \min(\max) \\ u_j = b_j - \sum_{i=1}^{n} a_{ij} \cdot x_i, j = \overline{1, m} \\ x_i > 0, i = \overline{1, n} u_i > 0, i = \overline{1, m} \end{cases} \tag{2}$$

3 Код

```
# %%
import numpy as np

# %%
def read_file(path):
    f = open(path, "r")
    func = f.readline().replace('\n', '')
    limitations = []
    f.readline()
    line = f.readline().replace('\n', '')
    while line != '':
        limitations.append(line)
        line = f.readline().replace('\n', '')

    cnt = int(f.readline().replace('\n', ''))
    f.close()
    return (func, limitations, cnt)

# %%
class Equation():
    def __init__(self, coefs, sign, bias):
        self.coefs = coefs
        self.sign = sign
        self.bias = bias

    @staticmethod
    def parse(eq, res_sz):
        sign = None
        sp = eq.split('=')
        b = 0
        if '>=' in eq:
            sign = '>='
            sp = eq.split(sign)
            b = int(sp[1])
            eq = sp[0]
        elif '<=' in eq:
            sign = '<='
            sp = eq.split(sign)
            b = int(sp[1])
            eq = sp[0]
        elif '=' in eq:
            sign = '='
            sp = eq.split(sign)
            b = int(sp[1])
            eq = sp[0]

        eq = eq.replace('-', '+-').replace('*', '')
        eq = [x for x in eq.split('+') if x]

        res = [0] * res_sz
        for sub_eq in eq:
            sub_eq = sub_eq.split('x')
            if len(sub_eq) == 1:
                b -= int(sub_eq[0])
            else:
                pos = int(sub_eq[1]) - 1
                if (sub_eq[0] == '-'):
                    val = -1
```

```
                elif not sub_eq[0]:
                    val = 1
                else:
                    val = int(sub_eq[0])

                res[pos] += val
        return Equation(res, sign, b)

    def __str__(self):
        return f'{self.coefs} {self.sign} {self.bias}'

# %%
def to_canonical(fx, limits, n = None):
    if n is None:
        n = max([len(x.coefs) for x in limits])
    addition = 0
    for eq in limits:
        if len(eq.coefs) < n + addition:
            eq.coefs.extend([0] * (n + addition - len(eq.coefs)))
        if eq.sign == '<=':
            eq.coefs.append(1)
            addition += 1
            eq.sign = '='
        elif eq.sign == '>=':
            eq.coefs.append(-1)
            addition += 1
            eq.sign = '='

    for eq in limits:
        if len(eq.coefs) < n + addition:
            eq.coefs.extend([0] * (n + addition - len(eq.coefs)))
        if eq.bias < 0:
            eq.bias *= -1
            eq.coefs = list(map(lambda x: x * -1, eq.coefs))
    fx.coefs.extend([0] * (n + addition - len(fx.coefs)))
    return fx, limits

# %%
def generate_table(canonical_func, canonical_limits):
    limits_matrix = [[x.bias] + x.coefs for x in canonical_limits]
    func = [-canonical_func.bias] + canonical_func.coefs
    limits_matrix = np.array(limits_matrix, dtype=float)
    func = np.array(func, dtype=float)
    diag = np.zeros((limits_matrix.shape[0], limits_matrix.shape[0]), dtype=float)
    np.fill_diagonal(diag, 1)
    simplex_table = np.hstack((limits_matrix, diag))
    penalty = np.sum(limits_matrix, axis=0)
    func = np.hstack((func, [0] * limits_matrix.shape[0]))
    penalty = np.hstack((penalty, [0] * limits_matrix.shape[0]))
    simplex_table = np.vstack((simplex_table, -func, penalty))
    return simplex_table

# %%
def find_col(simplex_table, *, debug=False):
    col = 1
    for i in range (2, simplex_table.shape[1]):
        if simplex_table[-1, i] > simplex_table[-1, col] or (simplex_table[-1, i] == simplex_table[-1, col] and simple
            col = i
    return col

# %%
```

```python
def check_stop(simplex_table):
    for i in range(1, simplex_table.shape[1]):
        if simplex_table[-1, i] > 0:
            return True
        if simplex_table[-2, i] > 0  and abs(simplex_table[-1, i]) == 0:
            return True
    return False

# %%
def iterate(simplex_table, base, *, eps=1e-9, debug=False):
    simplex_table_old = np.copy(simplex_table)
    column = find_col(simplex_table, debug=debug)

    with np.errstate(divide='ignore'):
        d = simplex_table[:-2, 0] / simplex_table[: -2, column]
    d[simplex_table[:-2, column] <= 0] = np.NAN

    try:
        row = np.nanargmin(d)
    except ValueError:
        raise RuntimeError('None or infinity solutions')

    base[row] = column

    if debug:
        with np.printoptions(precision=3, suppress=True):
            print(f'd: {d}')
            print(f'column: {column}, row: {row}, a_rl: {simplex_table[row, column]}')
            print(simplex_table)

    simplex_table[row, :] /= simplex_table_old[row, column]
    simplex_table[:, column] = 0
    simplex_table[row, column] = 1

    for i in range(simplex_table.shape[0]):
        for j in range(simplex_table.shape[1]):
            if i == row or j == column:
                continue
            simplex_table[i, j] = simplex_table_old[i, j] - (simplex_table_old[row, j] * simplex_table_old[i, column])

    simplex_table[abs(simplex_table) < eps] = 0
    return simplex_table, base

# %%
def solve(canonical_func, canonical_limits, *, debug=False):
    simplex_table = generate_table(canonical_func, canonical_limits)
    base_n = simplex_table.shape[0] - 2
    base = list(range(simplex_table.shape[1] - base_n, simplex_table.shape[1]))

    while(check_stop(simplex_table)):
        simplex_table, base = iterate(simplex_table, base, debug=debug)

    res = np.zeros(simplex_table.shape[1] - base_n)
    if max(base) >= len(res):
         raise RuntimeError('None solutions')
    res[base] = simplex_table[range(base_n), 0]
    res[0] = simplex_table[-2, 0]

    return res

# %%
```

```python
from os import path
def test(file, base_path):
    func, limits, cnt = read_file(path.join(base_path, file))
    func = Equation.parse(func, cnt)
    limits = list(map(lambda x: Equation.parse(x, cnt), limits))
    canonical_func, canonical_limits = to_canonical(func, limits, cnt)
    with np.printoptions(precision=3, suppress=True):
        print(solve(canonical_func, canonical_limits))

# %%
test_files = ['1.txt', '2.txt', '3.txt', '4.txt', '5.txt', '6.txt', '7.txt']
base_path = r'C:\Users\Bill\YandexDisk\Primat-5th-Semester\lab 1\tests'
for f in test_files:
    test(f, base_path)

# %%
test('8.txt', base_path)
```