Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

Методы оптимизации

Лабораторная работа №1 на тему:

«Постановка задачи линейного программирования»

Вариант 10

Преподаватель:

Коннова Н.С.

Студент:

Катков Е.И

Группа:
ИУ8-34

Москва 2024

Цель работы

Изучение симплекс-метода решения задачи линейного программирования (ЛП).

Постановка задачи

Требуется найти решение следующей задачи линейного программирования.

F=cx -> max

Ax ⩽ b

x ⩾ 0

c:[7 3 4]

A:[[2 1 1]

[1 4 0]

[0 0,5 2]]

b:[3 6 1],

где c - вектор коэффициентов целевой функции F;

A - матрица системы ограничений;

b - вектор правой части системы ограничений.

Ход работы

Определим начальную симплекс-таблицу для последующих преобразований, добавив соответствующие фиктивные переменные

Базис	Sio	X1	X2	X3
X4	3	2	1	1
X5	6	1	4	0
X6	1	0	0.5	2
F	0	7	3	4

Опорный элемент находится в пересечении (x4,x1) - 2

Проведем перерасчет симплекс-таблицы по правилу прямоугольника

Ni = N_old_i - (A∗B)/Hi

N_old_i - элемент исходного плана, Hi - разрешающий элемент (2), A и B - элементы старого плана, образующие прямоугольник с элементами N_old_i и Hi.

Базис	Sio	X4	X2	X3
X1	3/2	1/2	1/2	1/2
X5	9/2	-1/2	7/2	-1/2
X6	1	0	1/2	2
F	21/2	-7/2	-1/2	1/2

Проверим строку F на оптимальность
Пересечение (F,x3) > 0 => можно оптимизировать далее.

Опорный элемент находится в пересечении (x3,x6) - 2

Пересчитаем симплекс-таблицу

Базис	Sio	X4	X2	X6
X1	1.25	0.5	0.38	-0.25
X5	4.75	-0.5	3.62	0.25
X3	0.5	0	0.25	0.5
F	10.75	-3.5	-0.62	-0.25

Критерий оптимальности по строке F пройден
Ответ: x1 = 1.25, x2 = 0, x3= 0.5. При этом максимизированная функция равна 10.75

Проверим решение подставив значения в исходную систему

1) 1.25*2+0.5*1 ⩽ 3 – выполняется (3 ⩽ 3)

2) 1.25 ⩽ 6 – выполняется (1.25 ⩽ 6)

3) 2*0.5 ⩽ 1 – выполняется (1 ⩽ 1)

Приложение А.

Файл ‘simplex.py’.

```
import numpy as np
class SimplexMethod:
    def __init__(self, c, A, b, minormax):
        if A.shape[0] != len(b) or A.shape[1] != len(c):
            raise Exception("Неверное соотношение размеров матриц!")
        self.c = -c if minormax == "max" else c
        self.A = A
        self.b = b
        self.minormax = minormax
        self.table = None
        self.FreeX = [f"X{i+1}" for i in range(len(c))]
        self.DependX = [f"X{i+len(c)+1}" for i in range(len(b))]
        self.fill_table()
    def fill_table(self):
        num_vars = len(self.c)
        num_constraints = len(self.b)
        self.table = np.zeros((num_vars + 1, num_constraints + 1))
        self.table[0, :-1] = self.b
        self.table[1:, :-1] = self.A.T
        self.table[1:, -1] = -self.c
        self.print_table()
    def solution(self):
        if self.find_opt_solve():
            if self.minormax == "max":
                self.table[0, -1] *= -1
            return True
        return False
    def find_opt_solve(self):
        if self.find_opr_solve():
            self.print_table()
            print("-"*50 + "#")
            if all(self.table[1:, -1] <= 0):
                return True
        for i in range(1, len(self.c) + 1):
            if self.table[i, -1] > 0:
                pivot_col = i
                min_ratio = float('inf')
                pivot_row = -1
                for z in range(len(self.b)):
                    if self.table[pivot_col, z] != 0:
                        ratio = self.table[0, z] / self.table[pivot_col, z]
                        if 0 < ratio < min_ratio:
                            min_ratio = ratio
                            pivot_row = z
```

```
                if pivot_row != -1:
                    self.fix_table(pivot_row, pivot_col)
                    self.print_table()
                    print("-"*50 + "#")
                    if not self.find_opt_solve():
                        return False
                    return True
        return False
    def find_opr_solve(self):
        if all(self.table[0, :-1] >= 0):
            return True
        for i in range(len(self.b) + 1):
            if self.table[0, i] < 0:
                for j in range(1, len(self.c) + 1):
                    if self.table[j, i] < 0:
                        pivot_col = j
                        min_ratio = float('inf')
                        pivot_row = -1
                        for z in range(len(self.b)):
                            ratio = self.table[0, z] / self.table[pivot_col, z]
                            if 0 < ratio < min_ratio:
                                min_ratio = ratio
                                pivot_row = z
                        if pivot_row != -1:
                            self.fix_table(pivot_row, pivot_col)
                            self.print_table()
                            print("/n")
                            return self.find_opr_solve()
        return False
    def fix_table(self, pivot_row, pivot_col):
        r_e = self.table[pivot_col, pivot_row]
        self.FreeX[pivot_col - 1], self.DependX[pivot_row] = self.DependX[pivot_row],
self.FreeX[pivot_col - 1]
        new_table = np.zeros_like(self.table)
        new_table[pivot_col, pivot_row] = 1 / r_e
        for i in range(self.table.shape[0]):
            if i != pivot_col:
                new_table[i, pivot_row] = self.table[i, pivot_row] / r_e
        for i in range(self.table.shape[1]):
            if i != pivot_row:
                new_table[pivot_col, i] = -self.table[pivot_col, i] / r_e
        for i in range(self.table.shape[0]):
            for j in range(self.table.shape[1]):
                if i != pivot_col and j != pivot_row:
                    new_table[i, j] = self.table[i, j] - (self.table[pivot_col, j] * self.table[i, pivot_row]) / r_e
        self.table = new_table
```

```
def print_table(self):
    col_width = 10

    print(f"{'':<{col_width}}", end="")
    print(f"{'Sio':>{col_width}}", end="")
    for x in self.FreeX:
        print(f"{x:>{col_width}}", end="")
    print()
    for i in range(len(self.b) + 1):
        if i != len(self.b):
            print(f"{self.DependX[i]:<{col_width}}", end="")
        else:
            print(f"{'F':<{col_width}}", end="")
        for j in range(len(self.c) + 1):
            value = self.table[j, i]
            if isinstance(value, (int, float)):
                print(f"{value:>{col_width}.2f}", end="")
            else:
                print(f"{value:<{col_width}}", end="")
        print()
    print()
def get_solution(self):
    solution = np.zeros(len(self.c))
    for i, var in enumerate(self.FreeX):
        if var.startswith('X'):
            index = int(var[1:]) - 1
            if index < len(solution):
                solution[index] = self.table[i+1, -1]
    for i in range(len(solution)):
        if f'X{i+1}' not in self.FreeX and f'X{i+1}' not in self.DependX:
            solution[i] = 0
    for i, var in enumerate(self.DependX):
        if var.startswith('X'):
            index = int(var[1:]) - 1
            if index < len(solution):
                solution[index] = self.table[0, i]
    return solution
def print_solution(self):
    solution = self.get_solution()
    print("Решение:")
    print(f" x = [ {self.table[0,0]:.2f} , {abs(self.table[1,2]):.2f} , {self.table[0,2]:.2f} ]")
    print()
    f_value = self.table[0, -1] if self.minormax == "min" else -self.table[0, -1]
    print(f"Значение целевой функции: {-f_value:.2f}")
def check_solution(self):
    solution = self.get_solution()
```

```
        for i in range(len(self.b)):
            left_side = np.dot(self.A[i], solution)
            right_side = self.b[i]
            print(f"Ограничение {i+1}: ", end="")
            print(" + ".join([f"{(self.A[i][j] * solution[j]):.2f}" for j in range(len(solution))]), end="")
            print(f" <= {right_side:.2f} ({left_side:.2f} <= {right_side:.2f})")
            if not np.isclose(left_side, right_side, atol=100):
                print(f"Ошибка в уравнении {i+1}: {left_side:.6f} != {right_side:.6f}")
                return False
        print("Решение удовлетворяет всем ограничениям.")
        return True
c = np.array([7, 3, 4])
A = np.array([
    [2, 1, 1],
    [1, 4, 0],
    [0, 0.5, 2]
])
b = np.array([3, 6, 1])
simplex = SimplexMethod(c, A, b, "max")
if simplex.solution():
    simplex.print_table()
    simplex.print_solution()
    simplex.check_solution()
```

Python Terminal:

```
            Sio       X1       X2       X3
X4         3.00     2.00     1.00     1.00
X5         6.00     1.00     4.00     0.00
X6         1.00     0.00     0.50     2.00
F          0.00     7.00     3.00     4.00

-----------------------------------------------------#
            Sio       X4       X2       X3
X1         1.50     0.50     0.50     0.50
X5         4.50    -0.50     3.50    -0.50
X6         1.00    -0.00     0.50     2.00
F        -10.50    -3.50    -0.50     0.50

-----------------------------------------------------#
            Sio       X4       X2       X3
X1         1.50     0.50     0.50     0.50
X5         4.50    -0.50     3.50    -0.50
X6         1.00    -0.00     0.50     2.00
F        -10.50    -3.50    -0.50     0.50
```

```
-------------------------------------------------#
          Sio      X4      X2      X6
X1         1.25    0.50    0.38   -0.25
X5         4.75   -0.50    3.62    0.25
X3         0.50   -0.00    0.25    0.50
F        -10.75   -3.50   -0.62   -0.25

-------------------------------------------------#
          Sio      X4      X2      X6
X1         1.25    0.50    0.38   -0.25
X5         4.75   -0.50    3.62    0.25
X3         0.50   -0.00    0.25    0.50
F        -10.75   -3.50   -0.62   -0.25
```

Решение:
x = [1.25 , 0.00 , 0.50]

Значение целевой функции: 10.75
Ограничение 1: 2.50 + 0 + 0.50 <= 3.00 (3.00 <= 3.00)
Ограничение 2: 1.25 + 0 + 0.00 <= 6.00 (1.25 <= 6.00)
Ограничение 3: 0.00 + 0 + 1.00 <= 1.00 (1 <= 1.00)
Решение удовлетворяет всем ограничениям.

Вывод:

В ходе выполнения лабораторной работы был написан унифицированный класс для Python, который решает ПЗЛП с помощью симплекс-метода.