Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа № 1
«Симлекс метод»
по курсу
«Теория игр»

Студент группы ИУ9-31М Белогуров А.А.

Преподаватель Басараб М.А.

Москва, 2019

Содержание

1 Цель работы **3**

2 Постановка задачи **4**

3 Практическая реализация **5**

4 Результаты **12**

1 Цель работы

Изучение симплекс-метода решения задачи линеиного программирования.

2 Постановка задачи

Требуется найти решение следующей задачи:

$$F = cx \to max, \quad Ax \leq b, \quad x \geq 0.$$

Где x - искомый вектор решения, c - вектор коэффициентов целевой функции F, A - матрица системы ограничений, b - вектор правой части системы ограничений.

Использовать используя каноническую форму ЛП:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, 2, ..., m,$$

$$x_j \geq 0, \quad j = 1, 2, ..., n.$$

то надо найти минимум функции:

$$F = \sum_{j=1}^{n} \lambda_j x_j$$

Исходная симлекс-таблица, вариант 2:

$$c = \begin{pmatrix} 2 & 5 & 3 \end{pmatrix} \quad b = \begin{pmatrix} 6 \\ 6 \\ 2 \end{pmatrix} \quad A = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 0 \\ 0 & 0.5 & 1 \end{pmatrix}$$

3 Практическая реализация

SimplexMatrix.py

```python
from copy import deepcopy

import numpy as np
import pandas as pd

from Lab1.Conditions import FCondition, AConditionB


class SimplexMatrix:
    def __init__(self, A, b, c, f_condition=FCondition.MAX,
    ↪  a_condition_b=AConditionB.LESS_OR_EQUAL):
        """
        1) F = cx -> max, Ax <= b
        2) F = cx -> max, Ax >= b
        3) F = cx -> min, Ax <= b
        4) F = cx -> min, AX >= b
        x_i >= 0
        """
        self.A = np.array(A, dtype='float64')
        self.b = np.array(b, dtype='float64')
        self.c = np.array(c, dtype='float64')
        self.f_condition = f_condition
        self.a_condition_b = a_condition_b

    def prepare_variables(self):
        # По умолчанию : F = cx -> max, Ax <= b
        # Канонический вид x = b - A

        if self.f_condition is FCondition.MIN:
            self.c = -self.c

        if self.a_condition_b is AConditionB.GREATER_OR_EQUAL:
            self.A = -self.A
            self.b = -self.b

        self.canonic = np.hstack((self.b, self.A))
        self.f = np.insert(-self.c, 0, 0)
```

```
37          self.canonic_with_f = np.vstack((self.canonic, self.f))
38
39          self.rows, self.cols = self.canonic_with_f.shape
40
41          self.col = ['x' + str(self.cols - 1 + i) for i in range(1,
            ↪  self.rows)] + ['f']  # Свободные
42          self.row = ['s'] + ['x' + str(i) for i in range(1,
            ↪  self.cols)]  # Базис
43
44          self.resolving_row_idx = None  # Индекс разрещающей строки
45          self.resolving_col_idx = None  # Индекс разрещающего
            ↪  столбца
46          self.resolving_element = None
47
48          self.iteration = 0
49
50      def set_resolving_elements(self, resolving_row_idx,
        ↪  resolving_col_idx):
51          self.resolving_row_idx = resolving_row_idx
52          self.resolving_col_idx = resolving_col_idx
53          self.resolving_element =
            ↪  self.canonic_with_f[resolving_row_idx,
            ↪  resolving_col_idx]
54
55      def clear_resolving_elements(self):
56          self.resolving_row_idx = None
57          self.resolving_col_idx = None
58          self.resolving_element = None
59
60      def create_new_matrix(self):
61          new_matrix = deepcopy(self)
62          new_matrix.clear_resolving_elements()
63
64          for i in range(self.rows):
65              for j in range(self.cols):
66                  if (i, j) == (self.resolving_row_idx,
                    ↪  self.resolving_col_idx):
67                      new_matrix.canonic_with_f[
68                          self.resolving_row_idx,
                            ↪  self.resolving_col_idx] = 1 /
                            ↪  self.resolving_element
69                  elif i == self.resolving_row_idx:
```

```
                    new_matrix.canonic_with_f[i, j] =
                    ↪ self.canonic_with_f[i, j] /
                    ↪ self.resolving_element
                elif j == self.resolving_col_idx:
                    new_matrix.canonic_with_f[i, j] = -
                    ↪ self.canonic_with_f[i, j] /
                    ↪ self.resolving_element
                else:
                    new_matrix.canonic_with_f[i, j] =
                    ↪ self.canonic_with_f[i, j] -
                    ↪ self.canonic_with_f[i,
                    ↪ self.resolving_col_idx] *
                    ↪ self.canonic_with_f[self.resolving_row_idx,
                    ↪ j] \ self.resolving_element

        new_matrix.canonic =
        ↪ new_matrix.canonic_with_f[0:new_matrix.rows - 1, :]
        new_matrix.f = new_matrix.canonic_with_f[new_matrix.rows -
        ↪ 1, :]

        new_matrix.col[self.resolving_row_idx],
        ↪ new_matrix.row[self.resolving_col_idx] =
        ↪ self.row[self.resolving_col_idx],
        ↪ \self.col[self.resolving_row_idx]
        new_matrix.iteration += 1

        print(new_matrix)

        return new_matrix

    def __repr__(self):
        df = pd.DataFrame(self.canonic_with_f, columns=self.row,
        ↪ index=self.col)
        return str(df) + "\n"
```

SimplexMethod.py

```
import numpy as np

from Lab1.SimplexMatrix import SimplexMatrix, FCondition
```

```
4
5   INF = float('inf')
6
7
8   def find_idx_of(condition):
9       idx_list = np.where(condition)[0]
10      if len(idx_list) == 0:
11          return -1
12      else:
13          return idx_list[0]
14
15
16  def replace_positive(array, replace_element=INF):
17      return np.where(array >= 0, INF, array)
18
19
20  def replace_negative(array, replace_element=INF):
21      return np.where(array < 0, INF, array)
22
23
24  class SimplexMethod:
25      def __init__(self, matrix: SimplexMatrix):
26          self.iterations = [matrix]
27          self.result = None
28
29      def start(self):
30          current_matrix = self.iterations[0]
31          current_matrix.prepare_variables()
32          print("Start simplex method \n{}".format(current_matrix))
33
34          # Если в столбце S нет отрицательных элементов, то имеем
            ↪  опорное решение
35          # В последней строке ищем первый отрицательный элемент
36          while True:
37              first_col = current_matrix.canonic[:, 0]
38              negative_element_idx = find_idx_of(first_col < 0)
39              remove_positive = False
40
41              if negative_element_idx != -1:
42                  resolving_col_idx =
                    ↪  find_idx_of(current_matrix.canonic[negative_element_idx][1:]
                    ↪  < 0)
```

```
43                      remove_positive = True
44                      if resolving_col_idx == -1:
45                          raise RuntimeError("Не найден отрицательный
                            ↪  элемент в строке
                            ↪  {}".format(negative_element_idx))
46                  else:
47                      resolving_col_idx =
                        ↪  find_idx_of(current_matrix.f[1:] < 0)
48                      if resolving_col_idx == -1:
49                          # Отрицательных элементов нет, значит нашли
                            ↪  оптимальное решение
50                          break
51                      else:
52                          # Находим отрицательный элемент, который будет
                            ↪  минимален по модулю
53                          resolving_col_idx =
                            ↪  np.argmin(abs(replace_positive(current_matrix.f[1:])))
54
55                          # Необходимо добавить столбец s
56                  resolving_col_idx += 1
57
58                  # Найти минимальное положительное отношение элемента
                    ↪  свободных членов
59                  # к соответствующему элементу в разрешающем столбце
60                  resolving_col = current_matrix.canonic[:,
                    ↪  resolving_col_idx]
61                  if remove_positive:
62                      divison = replace_positive(first_col) /
                        ↪  resolving_col
63                  else:
64                      divison = first_col / resolving_col
65
66                  resolving_row_idx =
                    ↪  np.argmin(replace_negative(divison))
67
68                  print('Замена базисной переменной {} на свободную
                    ↪  {}'.format(current_matrix.row[resolving_col_idx],
                    ↪  current_matrix.col[resolving_row_idx]))
69
                    ↪  current_matrix.set_resolving_elements(resolving_row_idx,
                    ↪  resolving_col_idx)
70
```

```
            new_matrix = current_matrix.create_new_matrix()
            self.iterations.append(new_matrix)

            current_matrix = new_matrix

        self.result = current_matrix.f[0]

        # Данная реализация всегда ищет максимум, если необходимо найти минимум
        # то умножаем результат на -1
        if current_matrix.f_condition == FCondition.MIN:
            self.result *= -1
            print('Так как ищем MIN, то умножаем резульат на -1 -> {}'.format(self.result))

        return self.result
```

Conditions.py

```
from enum import Enum


class FCondition(Enum):
    MIN = 1
    MAX = 2


def opposite_f_condition(cond: FCondition):
    if cond is FCondition.MIN:
        return FCondition.MAX
    elif cond is FCondition.MAX:
        return FCondition.MIN
    else:
        raise ValueError("Unknown f condition")


class AConditionB(Enum):
    LESS_OR_EQUAL = 1, "<="
    GREATER_OR_EQUAL = 2, ">="

```

```
22
23  def opposite_a_condition_b(cond: AConditionB):
24      if cond is AConditionB.LESS_OR_EQUAL:
25          return AConditionB.GREATER_OR_EQUAL
26      elif cond is AConditionB.GREATER_OR_EQUAL:
27          return AConditionB.LESS_OR_EQUAL
28      else:
29          raise ValueError("Unknown a condition b")
```

4 Результаты

Промежуточные симлекс-таблицы:

```
Start simplex method
      s   x1   x2   x3
x4  6.0  2.0  1.0  2.0
x5  6.0  1.0  2.0  0.0
x6  2.0  0.0  0.5  1.0
f   0.0 -2.0 -5.0 -3.0

Замена базисной переменной x1 на свободную x4
      s   x4   x2   x3
x1  3.0  0.5  0.5  1.0
x5  3.0 -0.5  1.5 -1.0
x6  2.0 -0.0  0.5  1.0
f   6.0  1.0 -4.0 -1.0

Замена базисной переменной x3 на свободную x6
      s   x4   x2   x6
x1  1.0  0.5  0.0 -1.0
x5  5.0 -0.5  2.0  1.0
x3  2.0 -0.0  0.5  1.0
f   8.0  1.0 -3.5  1.0

Замена базисной переменной x2 на свободную x5
        s     x4    x5    x6
x1   1.00  0.500 -0.00 -1.00
x2   2.50 -0.250  0.50  0.50
x3   0.75  0.125 -0.25  0.75
f   16.75  0.125  1.75  2.75
```

Ответ:

$$F = 16.75$$