

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

Лабораторная работа № 2
«Двойственность в линейном программировании»
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	6
4	Результаты	13

1 Цель работы

Научиться по прямой задаче ЛП формулировать и решать соответствующую двойственную задачу.

2 Постановка задачи

Пусть исходная ПЗ ЛП имеет вид:

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

Где x - искомый вектор решения, c - вектор коэффициентов целевой функции ЦФ F , A - матрица системы ограничений, b - вектор правой части системы ограничений. Требуется по ПЗ ЛП сформулировать двойственную задачу ЛП и решить ее симплекс-методом.

Использовать каноническую форму ЛП:

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad i = 1, 2, \dots, m,$$
$$x_j \geq 0, \quad j = 1, 2, \dots, n.$$

то надо найти минимум функции:

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

Двойственная задача (ДЗ) ЛП будет выглядеть следующим образом:

- количество неизвестных (y_i) ДЗ равно количеству ограничений ПЗ;
- количество ограничений ДЗ равно количеству неизвестных (x_j) ПЗ;
- минимизация ЦФ F ПЗ соответствует максимизации ЦФ W ДЗ и наоборот;
- коэффициенты при ЦФ W ДЗ равны свободным членам ограничений ПЗ;

- свободные члены ограничений ДЗ равны коэффициентам при ЦФ F ПЗ;
- коэффициенты любого ограничения ДЗ равны коэффициентам при одной переменной из всех ограничений ПЗ;
- ограничения вида (\leq) ПЗ переходят в ограничения вида (\geq) ДЗ;
- все неизвестные (y_i) ДЗ неотрицательны.

$$\begin{aligned} W &= b^t y \rightarrow \max, \\ A^T y &\geq c^T, \\ y &\geq 0, \end{aligned}$$

где $y = [y_1, y_2, \dots, y_m]^T$

Исходная симплекс-таблица, вариант 2:

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

3 Практическая реализация

SimplexMatrix.py

```
1  from copy import deepcopy
2
3  import numpy as np
4  import pandas as pd
5
6  from Lab1.Conditions import FCondition, AConditionB
7
8
9  class SimplexMatrix:
10     def __init__(self, A, b, c, f_condition=FCondition.MAX,
11         ↪ a_condition_b=AConditionB.LESS_OR_EQUAL):
12         """
13         1)  $F = cx \rightarrow \max, Ax \leq b$ 
14         2)  $F = cx \rightarrow \max, Ax \geq b$ 
15         3)  $F = cx \rightarrow \min, Ax \leq b$ 
16         4)  $F = cx \rightarrow \min, Ax \geq b$ 
17          $x_i \geq 0$ 
18         """
19         self.A = np.array(A, dtype='float64')
20         self.b = np.array(b, dtype='float64')
21         self.c = np.array(c, dtype='float64')
22         self.f_condition = f_condition
23         self.a_condition_b = a_condition_b
24
25     def prepare_variables(self):
26         # По умолчанию :  $F = cx \rightarrow \max, Ax \leq b$ 
27         # Канонический вид  $x = b - A$ 
28
29         if self.f_condition is FCondition.MIN:
30             self.c = -self.c
31
32         if self.a_condition_b is AConditionB.GREATER_OR_EQUAL:
33             self.A = -self.A
34             self.b = -self.b
35
36         self.canonic = np.hstack((self.b, self.A))
37         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,
42             ↪ self.rows)] + ['f'] # Свободные
43         self.row = ['s'] + ['x' + str(i) for i in range(1,
44             ↪ self.cols)] # Базис
45
46         self.resolving_row_idx = None # Индекс разрешающей строки
47         self.resolving_col_idx = None # Индекс разрешающего
48             ↪ столбца
49         self.resolving_element = None
50
51         self.iteration = 0
52
53     def set_resolving_elements(self, resolving_row_idx,
54         ↪ resolving_col_idx):
55         self.resolving_row_idx = resolving_row_idx
56         self.resolving_col_idx = resolving_col_idx
57         self.resolving_element =
58             ↪ self.canonic_with_f[resolving_row_idx,
59             ↪ resolving_col_idx]
60
61     def clear_resolving_elements(self):
62         self.resolving_row_idx = None
63         self.resolving_col_idx = None
64         self.resolving_element = None
65
66     def create_new_matrix(self):
67         new_matrix = deepcopy(self)
68         new_matrix.clear_resolving_elements()
69
70         for i in range(self.rows):
71             for j in range(self.cols):
72                 if (i, j) == (self.resolving_row_idx,
73                     ↪ self.resolving_col_idx):
74                     new_matrix.canonic_with_f[
75                         self.resolving_row_idx,
76                         ↪ self.resolving_col_idx] = 1 /
77                         ↪ self.resolving_element
78                 elif i == self.resolving_row_idx:

```

```

70         new_matrix.canonic_with_f[i, j] =
            ↪ self.canonic_with_f[i, j] /
            ↪ self.resolving_element
71     elif j == self.resolving_col_idx:
72         new_matrix.canonic_with_f[i, j] = -
            ↪ self.canonic_with_f[i, j] /
            ↪ self.resolving_element
73     else:
74         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
75
76     new_matrix.canonic =
            ↪ new_matrix.canonic_with_f[0:new_matrix.rows - 1, :]
77     new_matrix.f = new_matrix.canonic_with_f[new_matrix.rows -
            ↪ 1, :]
78
79     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]
80     new_matrix.iteration += 1
81
82     print(new_matrix)
83
84     return new_matrix
85
86     def __repr__(self):
87         df = pd.DataFrame(self.canonic_with_f, columns=self.row,
            ↪ index=self.col)
88         return str(df) + "\n"

```

DualSimplexMethod.py

```

1     import numpy as np
2
3     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
35         ↪ опорное решение
36         # В последней строке ищем первый отрицательный элемент
37         while True:
38             first_col = current_matrix.canonic[:, 0]
39             negative_element_idx = find_idx_of(first_col < 0)
40             remove_positive = False
41
42             if negative_element_idx != -1:
43                 resolving_col_idx =
44                     ↪ find_idx_of(current_matrix.canonic[negative_element_idx][1:]
45                     ↪ < 0)

```

```

43         remove_positive = True
44         if resolving_col_idx == -1:
45             raise RuntimeError("Не найден отрицательный
46                 ↪ элемент в строке
47                 ↪ {}".format(negative_element_idx))
48         else:
49             resolving_col_idx =
50                 ↪ find_idx_of(current_matrix.f[1:] < 0)
51             if resolving_col_idx == -1:
52                 # Отрицательных элементов нет, значит нашли
53                 ↪ оптимальное решение
54                 break
55             else:
56                 # Находим отрицательный элемент, который будет
57                 ↪ минимален по модулю
58                 resolving_col_idx =
59                 ↪ np.argmin(abs(replace_positive(current_matrix.f[1:])))
60
61                 # Необходимо добавить столбец s
62             resolving_col_idx += 1
63
64             # Найти минимальное положительное отношение элемента
65             ↪ свободных членов
66             # к соответствующему элементу в разрешающем столбце
67             resolving_col = current_matrix.canonic[:,
68                 ↪ resolving_col_idx]
69             if remove_positive:
70                 divison = replace_positive(first_col) /
71                 ↪ resolving_col
72             else:
73                 divison = first_col / resolving_col
74
75             resolving_row_idx =
76                 ↪ np.argmin(replace_negative(divison))
77
78             print('Замена базисной переменной {} на свободную
79                 ↪ {}'.format(current_matrix.row[resolving_col_idx],
80                 ↪ current_matrix.col[resolving_row_idx]))
81
82             ↪ current_matrix.set_resolving_elements(resolving_row_idx,
83             ↪ resolving_col_idx)
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

71         new_matrix = current_matrix.create_new_matrix()
72         self.iterations.append(new_matrix)
73
74         current_matrix = new_matrix
75
76         self.result = current_matrix.f[0]
77
78         # Данная реализация всегда ищет максимум, если необходимо
79         ↪ найти минимум
80         # то умножаем результат на -1
81         if current_matrix.f_condition == FCondition.MIN:
82             self.result *= -1
83             print('Так как ищем MIN, то умножаем результат на -1 ->
84                   ↪ {}'.format(self.result))
85
86         return self.result

```

Conditions.py

```

1  from enum import Enum
2
3
4  class FCondition(Enum):
5      MIN = 1
6      MAX = 2
7
8
9  def opposite_f_condition(cond: FCondition):
10     if cond is FCondition.MIN:
11         return FCondition.MAX
12     elif cond is FCondition.MAX:
13         return FCondition.MIN
14     else:
15         raise ValueError("Unknown f condition")
16
17
18  class AConditionB(Enum):
19     LESS_OR_EQUAL = 1, "<="
20     GREATER_OR_EQUAL = 2, ">="
21

```

```
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

Решение прямой задачи ЛП:

```
1 Start simplex method
2      s    x1    x2    x3
3 x4  6.0  2.0  1.0  2.0
4 x5  6.0  1.0  2.0  0.0
5 x6  2.0  0.0  0.5  1.0
6 f   0.0 -2.0 -5.0 -3.0
7
8 Замена базисной переменной x1 на свободную x4
9      s    x4    x2    x3
10 x1  3.0  0.5  0.5  1.0
11 x5  3.0 -0.5  1.5 -1.0
12 x6  2.0 -0.0  0.5  1.0
13 f   6.0  1.0 -4.0 -1.0
14
15 Замена базисной переменной x3 на свободную x6
16      s    x4    x2    x6
17 x1  1.0  0.5  0.0 -1.0
18 x5  5.0 -0.5  2.0  1.0
19 x3  2.0 -0.0  0.5  1.0
20 f   8.0  1.0 -3.5  1.0
21
22 Замена базисной переменной x2 на свободную x5
23      s    x4    x5    x6
24 x1  1.00  0.500 -0.00 -1.00
25 x2  2.50 -0.250  0.50  0.50
26 x3  0.75  0.125 -0.25  0.75
27 f  16.75  0.125  1.75  2.75
```

Решений двойственной задачи ЛП:

```
1 Start simplex method
2      s    x1    x2    x3
3 x4  6.0  2.0  1.0  2.0
4 x5  6.0  1.0  2.0  0.0
5 x6  2.0  0.0  0.5  1.0
```

```

6   f    0.0 -2.0 -5.0 -3.0
7
8   Замена базисной переменной x1 на свободную x4
9       s    x4    x2    x3
10  x1    3.0  0.5  0.5  1.0
11  x5    3.0 -0.5  1.5 -1.0
12  x6    2.0 -0.0  0.5  1.0
13  f     6.0  1.0 -4.0 -1.0
14
15  Замена базисной переменной x3 на свободную x6
16       s    x4    x2    x6
17  x1    1.0  0.5  0.0 -1.0
18  x5    5.0 -0.5  2.0  1.0
19  x3    2.0 -0.0  0.5  1.0
20  f     8.0  1.0 -3.5  1.0
21
22  Замена базисной переменной x2 на свободную x5
23       s    x4    x5    x6
24  x1    1.00  0.500 -0.00 -1.00
25  x2    2.50 -0.250  0.50  0.50
26  x3    0.75  0.125 -0.25  0.75
27  f    16.75  0.125  1.75  2.75
28
29  Start simplex method
30       s    x1    x2    x3
31  x4 -2.0 -2.0 -1.0 -0.0
32  x5 -5.0 -1.0 -2.0 -0.5
33  x6 -3.0 -2.0 -0.0 -1.0
34  f    0.0  6.0  6.0  2.0
35
36  Замена базисной переменной x1 на свободную x4
37       s    x4    x2    x3
38  x1    1.0 -0.5  0.5  0.0
39  x5 -4.0 -0.5 -1.5 -0.5
40  x6 -1.0 -1.0  1.0 -1.0
41  f    -6.0  3.0  3.0  2.0
42
43  Замена базисной переменной x4 на свободную x6
44       s    x6    x2    x3
45  x1    1.5 -0.5  0.0  0.5
46  x5 -3.5 -0.5 -2.0  0.0
47  x4    1.0 -1.0 -1.0  1.0

```

```

48 f -9.0 3.0 6.0 -1.0
49
50 Замена базисной переменной x6 на свободную x5
51 s x5 x2 x3
52 x1 5.0 -1.0 2.0 0.5
53 x6 7.0 -2.0 4.0 -0.0
54 x4 8.0 -2.0 3.0 1.0
55 f -30.0 6.0 -6.0 -1.0
56
57 Замена базисной переменной x3 на свободную x4
58 s x5 x2 x4
59 x1 1.0 0.0 0.5 -0.5
60 x6 7.0 -2.0 4.0 0.0
61 x3 8.0 -2.0 3.0 1.0
62 f -22.0 4.0 -3.0 1.0
63
64 Замена базисной переменной x2 на свободную x6
65 s x5 x6 x4
66 x1 0.125 0.25 -0.125 -0.5
67 x2 1.750 -0.50 0.250 0.0
68 x3 2.750 -0.50 -0.750 1.0
69 f -16.750 2.50 0.750 1.0
70
71 Так как ищем MIN, то умножаем результат на -1 -> 16.75

```

Результаты ПЗ и ДЗ совпадают:

$$F = W = 16.75$$