Министерство образования Российской Федерации

МОСКВОСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э.БАУМАНА

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

МЕТОДЫ ОПТИМИЗАЦИИ

Лабораторная работа №5 на тему:

«Матричные игры с ненулевой суммой. Смешанные стратегии».

Вариант – 1

Преподаватель:
Коннова Н.С.

Студент:
Александров А. Н.

Группа:
ИУ8-34

Москва, 2020

Цель работы:

Изучить постановку антагонистической игры двух лиц в нормальной форме; получить навыки нахождения решения игры в смешанных стратегиях (стратегическую седловую точку) обоих игроков.

Постановка задачи:

Решить матричную игру в смешанных стратегиях за обоих игроков (строки матриц соответствуют стратегиям игрока *A*, столбцы – стратегиям игрока *B*):

$$\begin{pmatrix} 1 & 11 & 12 & 11 \\ 7 & 5 & 7 & 7 \\ 16 & 6 & 13 & 2 \\ 9 & 9 & 16 & 13 \\ 17 & 18 & 15 & 7 \end{pmatrix}$$

Ход решения:

Для решения задачи, воспользовался собственной программой, написанной на языке программирования **Python**(см. Приложение А):

За основу взята программа для решения лабораторной работы №2 «Двойственность в линейном программировании».

Добавлен файл ***strategic.py***, содержащий функции ***StrategyA*** и ***StrategyB***, аргументами которых выступают конечные симплекс-таблицы, полученные в результате решения прямой и двойственной задач линейного программирования, к которым мы сводим матричную игру.

Итак, матрица стратегий имеет вид:

Таблица 1 Матрица стратегий игроков. Нахождение верхней и нижней цен игры.

Стратегии	b_1	b_2	b_3	b_4	α_i
a_1	1	11	12	11	1
a_2	7	5	7	7	5
a_3	16	6	13	2	2
a_4	9	9	16	13	9
a_5	17	18	15	7	7
β_i	17	18	16	13	

<u>Цена игры:</u> $9 \leqslant V \leqslant 13$

Найдём смешанные стратегии игрока *А*. Для этого составим систему уравнений:

$$
\begin{cases}
x_1+7x_2+16x_3+9x_4+17x_5 \geqslant g \\
11x_1+5x_2+6x_3+9x_4+18x_5 \geqslant g \\
12x_1+7x_2+13x_3+16x_4+15x_5 \geqslant g \\
11x_1+7x_2+2x_3+13x_4+7x_5 \geqslant g \\
x_1+x_2+x_3+x_4+x_5=1
\end{cases}
$$

где g — минимальный выигрыш.

Разделим систему на функцию g:

$$
\begin{cases}
u_1+7u_2+16u_3+9u_4+17u_5 \geqslant 1 \\
11u_1+5u_2+6u_3+9u_4+18u_5 \geqslant 1 \\
12u_1+7u_2+13u_3+16u_4+15u_5 \geqslant 1 \\
11u_1+7u_2+2u_3+13u_4+7u_5 \geqslant 1 \\
u_1+u_2+u_3+u_4+u_5=1/g
\end{cases}
$$

Сформулируем задачу для решения симплекс-методом:

```
/usr/bin/python3.6 /home/aaaaaaaaalesha/github/moLabs/03-lab-05-non_zero_sum_matrix_game/main.py
    Найдём смешанные стратегии для игрока А. Сформулируем задачу для решения симплекс-методом:
------------------------------------------------------------
F = cx -> min,
Ax >= 1,
x1,x2, ..., xn >= 0
C = [1 1 1 1 1],
A =
[[ 1  7 16  9 17]
 [11  5  6  9 18]
 [12  7 13 16 15]
 [11  7  2 13  7]],
b^T = [1 1 1 1].
------------------------------------------------------------
Процесс решения:
1) Поиск опорного решения:
Исходная симплекс-таблица:
+----+------+-------+------+-------+-------+-------+
|    | Si0  |   x1  |  x2  |   x3  |   x4  |   x5  |
+----+------+-------+------+-------+-------+-------+
| x6 | -1.0 |  -1.0 | -7.0 | -16.0 |  -9.0 | -17.0 |
| x7 | -1.0 | -11.0 | -5.0 |  -6.0 |  -9.0 | -18.0 |
| x8 | -1.0 | -12.0 | -7.0 | -13.0 | -16.0 | -15.0 |
| x9 | -1.0 | -11.0 | -7.0 |  -2.0 | -13.0 |  -7.0 |
| F  | 0.0  |  -1.0 | -1.0 |  -1.0 |  -1.0 |  -1.0 |
+----+------+-------+------+-------+-------+-------+
```

Рисунок 1 Формулировка задачи ЛП. Начальная симплекс-таблица.

$$W = u_1 + u_2 + u_3 + u_4 + u_5 \rightarrow min;$$

$$\begin{cases} u_1 + 7u_2 + 16u_3 + 9u_4 + 17u_5 \geqslant 1 \\ 11u_1 + 5u_2 + 6u_3 + 9u_4 + 18u_5 \geqslant 1 \\ 12u_1 + 7u_2 + 13u_3 + 16u_4 + 15u_5 \geqslant 1 \\ 11u_1 + 7u_2 + 2u_3 + 13u_4 + 7u_5 \geqslant 1 \end{cases}$$

$$u_i \geqslant 0,\ i = 1, 2, 3, 4, 5.$$

Находим оптимальное решение. Конечная симплекс таблица и найденное оптимальное решение:

```
+----+--------+---------+---------+---------+---------+----------+
|    |  Si0   |    x6   |    x9   |    x2   |    x3   |    x1    |
+----+--------+---------+---------+---------+---------+----------+
| x4 | 0.0634 |  0.0442 | -0.1077 |  0.4428 | -0.4938 |  1.1396  |
| x8 | 0.3925 | -0.5255 | -0.8671 |  2.7459 | -2.8634 | -1.9344  |
| x5 | 0.0252 | -0.0822 |  0.0569 |  0.1772 |  1.2016 | -0.5436  |
| x7 | 0.0246 | -1.0817 |  0.0571 |  2.1781 | 11.1982 | -10.5431 |
| F  | 0.0887 |  -0.038 | -0.0505 | -0.3797 | -0.2912 |  -0.405  |
+----+--------+---------+---------+---------+---------+----------+
-----------
Оптимальное решение найдено!
x6 = x9 = x2 = x3 = x1 = 0, x4 =  0.063, x8 =  0.392, x5 =  0.025, x7 =  0.025,
Целевая функция: F = 0.089
-----------
```

Рисунок 2 Конечная симплекс-таблица. Оптимальное решение задачи ЛП.

Решение для смешанной стратегии игрока А:

$$u_1 = 0,\ u_2 = 0,\ u_3 = 0,\ u_4 = 0.063,\ u_5 = 0.025;$$
$$W = 0.089.$$
$$g = 1/W = 11.236$$

Цена игры: $9 \leqslant 11.236 \leqslant 13$ – верно.

Оптимальные стратегии:

$$x_1 = u_1 \cdot g = 0 \cdot 11.236 = 0;$$
$$x_2 = u_2 \cdot g = 0 \cdot 11.236 = 0;$$
$$x_3 = u_3 \cdot g = 0 \cdot 11.236 = 0;$$
$$x_4 = u_4 \cdot g = 0.063 \cdot 11.236 = 0.708;$$
$$x_5 = u_5 \cdot g = 0.025 \cdot 11.236 = 0.281;$$

Таким образом, оптимальная смешанная стратегия игрока А имеет вид:

$$(0, 0, 0, 0.708, 0.281).$$

```
Решение для смешанной стратегии игрока A:
u1 = 0, u2 = 0, u3 = 0, u4 = 0.063, u5 = 0.025;
W = 0.089;
g = 1/W = 11.236.
Частоты выбора стратегий:
x1 = u1·g = 0·11.236 = 0.0;
x2 = u2·g = 0·11.236 = 0.0;
x3 = u3·g = 0·11.236 = 0.0;
x4 = u4·g = 0.063·11.236 = 0.708;
x5 = u5·g = 0.025·11.236 = 0.281;
Таким образом, оптимальная смешанная стратегия игрока A имеет вид
 [0.0, 0.0, 0.0, 0.708, 0.281].
```

Рисунок 3 Решение для смешанной стратегии игрока A.

Найдём смешанные стратегии игрока *B*. Для этого составим систему уравнений:

$$\begin{cases} y_1+11y_2+12y_3+11y_4 \leqslant h \\ 7y_1+5y_2+7y_3+7y_4 \leqslant h \\ 16y_1+6y_2+13y_3+2y_4 \leqslant h \\ 9y_1+9y_2+16y_3+13y_4 \leqslant h \\ 17y_1+18y_2+15y_3+7y_4 \leqslant h \\ y_1+y_2+y_3+y_4=1 \end{cases}$$

где h — максимальный проигрыш игрока *B*.

Разделим систему на h:

$$\begin{cases} v_1+11v_2+12v_3+11v_4 \leqslant 1 \\ 7v_1+5v_2+7v_3+7v_4 \leqslant 1 \\ 16v_1+6v_2+13v_3+2v_4 \leqslant 1 \\ 9v_1+9v_2+16v_3+13v_4 \leqslant 1 \\ 17v_1+18v_2+15v_3+7v_4 \leqslant 1 \\ v_1+v_2+v_3+v_4=1/h \end{cases}$$

Сформулируем задачу для решения симплекс-методом:

$$Z=v_1+v_2+v_3+v_4 \rightarrow max;$$
$$\begin{cases} v_1+11v_2+12v_3+11v_4 \leqslant 1 \\ 7v_1+5v_2+7v_3+7v_4 \leqslant 1 \\ 16v_1+6v_2+13v_3+2v_4 \leqslant 1 \\ 9v_1+9v_2+16v_3+13v_4 \leqslant 1 \\ 17v_1+18v_2+15v_3+7v_4 \leqslant 1 \end{cases}$$
$$v_i \geqslant 0,\ i=1,2,3,3,4.$$

```
    Найдём смешанные стратегии для игрока B. Сформулируем задачу для решения симплекс-методом:
Условие задачи:
------------------------------------------------------------
Найти вектор x = (x1,x2,..., xn)^T как решение след. задачи:
F = cx -> max,
Ax <= 1,
x1,x2, ..., xn >= 0
C = [1 1 1 1],
A =
[[ 1 11 12 11]
 [ 7  5  7  7]
 [16  6 13  2]
 [ 9  9 16 13]
 [17 18 15  7]],
b^T = [1 1 1 1 1].
------------------------------------------------------------
Процесс решения:
1) Поиск опорного решения:
Исходная симплекс-таблица:
+----+-----+------+------+------+------+
|    | Si0 |  x1  |  x2  |  x3  |  x4  |
+----+-----+------+------+------+------+
| x5 | 1.0 | 1.0  | 11.0 | 12.0 | 11.0 |
| x6 | 1.0 | 7.0  | 5.0  | 7.0  | 7.0  |
| x7 | 1.0 | 16.0 | 6.0  | 13.0 | 2.0  |
| x8 | 1.0 | 9.0  | 9.0  | 16.0 | 13.0 |
| x9 | 1.0 | 17.0 | 18.0 | 15.0 | 7.0  |
| F  | 0.0 | 1.0  | 1.0  | 1.0  | 1.0  |
+----+-----+------+------+------+------+
```

Рисунок 4 Формулировка задачи ЛП для игрока B. Начальная симплекс-таблица.

Находим оптимальное решение. Конечная симплекс таблица и найденное оптимальное решение:

```
+----+---------+--------+---------+--------+--------+
|    |   Si0   |   x9   |   x2    |   x8   |   x3   |
+----+---------+--------+---------+--------+--------+
| x5 |  0.4051 |  0.5443 | 10.5443  | -1.1393 |  1.9367 |
| x6 |  0.3797 | -0.1772 | -2.1772  | -0.4431 | -2.7468 |
| x7 |  0.2912 | -1.2026 | -11.2026 |  0.4937 |  2.8607 |
| x4 |  0.0506 |  -0.057 |  -0.057  |  0.1076 |  0.8671 |
| x1 |  0.038  |  0.0823 |  1.0823  | -0.0443 |  0.5254 |
| F  | -0.0886 | -0.0253 | -0.0253  | -0.0633 | -0.3924 |
+----+---------+--------+---------+--------+--------+
-----------
Оптимальное решение найдено!
x9 = x2 = x8 = x3 = 0, x5 =  0.405, x6 =  0.38, x7 =  0.291, x4 =  0.051, x1 =  0.038,
Целевая функция: F = 0.089
```

Рисунок 5 Конечная симплекс-таблица. Оптимальное решение задачи ЛП для игрока B.

Решение для смешанной стратегии игрока *B*:

$$v_1 = 0.038,\ v_2 = 0,\ v_3 = 0,\ v_4 = 0.051\,;$$
$$Z = 0.089\,.$$
$$h = 1/Z = 11.236$$

Цена игры: $9 \leqslant 11.236 \leqslant 13$ – верно

Частоты выбора стратегий:

$$y_1 = v_1 \cdot h = 0.038 \cdot 11.236 = 0.427\,;$$
$$y_2 = v_2 \cdot h = 0 \cdot 11.236 = 0\,;$$
$$y_3 = v_3 \cdot h = 0 \cdot 11.236 = 0\,;$$
$$y_4 = v_4 \cdot h = 0.051 \cdot 11.236 = 0.573\,;$$

Таким образом, оптимальная смешанная стратегия игрока *B* имеет вид:

$$(0.427\,,\ 0\,,\ 0\,,\ 0\,,\ 0.573).$$

```
Решение для смешанной стратегии игрока B:
v1 = 0.038, v2 = 0, v3 = 0, v4 = 0.051;
Z = 0.089;
h = 1/Z = 11.236.
Частоты выбора стратегий:
y1 = v1·h = 0.038·11.236 = 0.427;
y2 = v2·h = 0·11.236 = 0.0;
y3 = v3·h = 0·11.236 = 0.0;
y4 = v4·h = 0.051·11.236 = 0.573;
Таким образом, оптимальная смешанная стратегия игрока B имеет вид
 [0.427, 0.0, 0.0, 0.573].
```

Рисунок 6 Конечная симплекс-таблица. Оптимальное решение задачи ЛП.

Для проверки сложим полученные вероятности и убедимся, что сумма равна единице (с учётом округления до трёх цифр после точки):

$$\sum_{i=1}^{5} x_i = 0 + 0 + 0 + 0.708 + 0.281 \approx 1$$

$$\sum_{i=1}^{4} y_i = 0.427 + 0 + 0 + 0.573 \approx 1$$

Вывод:

В данной работе была изучена постановка антагонистической игры двух лиц в нормальной форме. Были получены навыки нахождения решения игры в смешанных стратегиях.

Проанализировав матрицу стратегий, проверил, что седловой точки нет, поэтому применил смешанные стратегии, приступил к нахождению стратегической седловой точки за обоих игроков.

Для решения задачи потребовалось немного модифицировать программу для лабораторной работы №2 «Двойственность в линейном программировании», так как прямоугольная игра с нулевой суммой и данной матрицей стратегий сводилась к прямой задаче ЛП для игрока *A*, и к двойственной задаче ЛП для игрока *B*.

По результатам видим, что полученная цена игры $1/W=1/Z=g=h=11.236$ лежит в найденном в начале диапазоне нижней и верхней цен игры: $9\leqslant V\leqslant 13$, что согласуется с теорией.

Приложение А

Код программы

Файл «main.py»

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

"""
Лабораторная работа № 5
Матричные игры с нулевой суммой. Смешанные стратегии.
Цель работы: изучить постановку антагонстической игры двух лиц в нормальной форме; получить навыки нахождения решения игры в смешанных стратегиях (стратегическую седловую точку) за обоих игроков.
Вариант 1.
"""
import dual_problem
from simplex import Simplex
import strategic
if __name__ == '__main__':
    print("\tНайдём смешанные стратегии для игрока A. Сформулируем задачу для решения симплекс-методом:")
    dual_p = dual_problem.DualProblem("input_data.json")
    # Находим опорное решение.
    dual_p.reference_solution()
    # Находим оптимальное решение.
    dual_p.optimal_solution()
    #
    print(strategic.StrategyA(dual_p.simplex_table_))
    print("\tНайдём смешанные стратегии для игрока B. Сформулируем задачу для решения симплекс-методом:")
    problem = Simplex("input_data.json")
    print(problem)
    # Находим опорное решение.
    problem.reference_solution()
    # Находим оптимальное решение.
    problem.optimal_solution()
    print(strategic.StrategyB(problem.simplex_table_))
```

Файл «simplex.py»

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

from simplex_table import *
import json
class Simplex:
    """
    Класс для решения задачи ЛП симплекс-методом.
    """
    def __init__(self, path_to_file):
        """
        Переопределённый метод __init__. Регистрирует входные данные из JSON-файла.
```

```
        Определяем условие задачи.
        :param path_to_file: путь до JSON-файла с входными данными.
        """
        # Парсим JSON-файл с входными данными
        with open(path_to_file, "r") as read_file:
            json_data = json.load(read_file)
            self.obj_func_coffs_ = np.array(json_data["obj_func_coffs"])  # вектор-строка с
- коэффициенты ЦФ
            self.constraint_system_lhs_ = np.array(json_data["constraint_system_lhs"])  #
матрица ограничений А
            self.constraint_system_rhs_ = np.array(json_data["constraint_system_rhs"])
# вектор-столбец ограничений b
            self.func_direction_ = json_data["func_direction"]  # направление задачи
(min или max)
            if len(self.constraint_system_rhs_) != self.constraint_system_rhs_.shape[0]:
                raise SimplexException(
                    "Ошибка при вводе данных. Число строк в матрице и
столбце ограничений не совпадает.")
            # Если задача на max, то меняем знаки ЦФ и направление задачи (в конце
возьмем решение со знаком минус и
            # получим искомое).
            if self.func_direction_ == "max":
                self.obj_func_coffs_ *= -1
            # Инициализация симплекс-таблицы.
            self.simplex_table_ = SimplexTable(self.obj_func_coffs_,
self.constraint_system_lhs_,
                                               self.constraint_system_rhs_)
    def __str__(self):
        """
        Переопренный метод __str__ для условия задачи.
        :return: Строка с выводом условия задачи.
        """
        output = """Условие задачи:
------------------------------------------------------------
Найти вектор x = (x1,x2,..., xn)^T как решение след. задачи:"""
        output += f"\nF = cx -> {self.func_direction_},"
        output += "\nAx <= 1,\nx1,x2, ..., xn >= 0"
        output += f"\nC = {-self.obj_func_coffs_},"
        output += f"\nA =\n{self.constraint_system_lhs_},"
        output += f"\nb^T = {self.constraint_system_rhs_}."
        output += "\n------------------------------------------------------------"
        return output
    # Этап 1. Поиск опорного решения.
    def reference_solution(self):
        """
        Метод производит отыскание опорного решения.
        """
        print("Процесс решения:\n1) Поиск опорного решения:")
        print("Исходная симплекс-таблица:", self.simplex_table_, sep="\n")
        while not self.simplex_table_.is_find_ref_solution():
            self.simplex_table_.search_ref_solution()
        print("-----------")
        print("Опорное решение найдено!")
        self.output_solution()
        print("-----------")
```

```python
    # Этап 2. Поиск оптимального решения.
    def optimal_solution(self):
        """
        Метод производит отыскание оптимального решения.
        """
        print("2) Поиск оптимального решения:")
        while not self.simplex_table_.is_find_opt_solution():
            self.simplex_table_.optimize_ref_solution()
        # Если задача на max, то в начале свели задачу к поиску min, а теперь
        # возьмём это решение со знаком минус и получим ответ для мак.
        if self.func_direction_ == "max":
            self.simplex_table_.main_table_[self.simplex_table_.main_table_.shape[0] - 1][0] *= -1
        print("----------")
        print("Оптимальное решение найдено!")
        self.output_solution()
        print("----------")
    def output_solution(self):
        """
        Метод выводит текущее решение, используется для вывода опорного и оптимального решений.
        """
        fict_vars = self.simplex_table_.top_row_[2:]
        last_row_ind = self.simplex_table_.main_table_.shape[0] - 1
        for var in fict_vars:
            print(var, "= ", end="")
        print(0, end=", ")
        for i in range(last_row_ind):
            print(self.simplex_table_.left_column_[i], "= ", round(self.simplex_table_.main_table_[i][0], 3), end=", ")
        print("\nЦелевая функция: F =", round(self.simplex_table_.main_table_[last_row_ind][0], 3))
```

Файл «simplex_table.py»

```python
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

import numpy as np
import warnings
from prettytable import PrettyTable
ROUND_CONST = 4
class SimplexException(Exception):
    """Пользовательское исключение для решения задач симплекс-методом."""
    def __init__(self, text):
        self.txt = text
class SimplexTable:
    """
    Класс симплекс-таблицы.
    """
    def __init__(self, obj_func_coffs, constraint_system_lhs, constraint_system_rhs):
        """
        Переопределённый метод __init__ для создания экземпляра класса SimplexTable.
        :param obj_func_coffs: коэффициенты ЦФ.
        :param constraint_system_lhs: левая часть системы ограничений.
        :param constraint_system_rhs: правая часть системы ограничений.
```

```
        """
        var_count = len(obj_func_coffs)
        constraint_count = constraint_system_lhs.shape[0]
        # Заполнение верхнего хедера.
        self.top_row_ = ["  ", "Si0"]
        for i in range(var_count):
            self.top_row_.append("x" + str(i + 1))
        # Заполнение левого хедера.
        self.left_column_ = []
        ind = var_count
        for i in range(constraint_count):
            ind += 1
            self.left_column_.append("x" + str(ind))
        self.left_column_.append("F ")
        self.main_table_ = np.zeros((constraint_count + 1, var_count + 1))
        # Заполняем столбец Si0.
        for i in range(constraint_count):
            self.main_table_[i][0] = round(constraint_system_rhs[i], ROUND_CONST)
        # Заполняем строку F.
        for j in range(var_count):
            self.main_table_[constraint_count][j + 1] = -round(obj_func_coffs[j],
ROUND_CONST)
        # Заполняем A.
        for i in range(constraint_count):
            for j in range(var_count):
                self.main_table_[i][j + 1] = round(constraint_system_lhs[i][j],
ROUND_CONST)
    def __str__(self):
        """
        Переопренный метод __str__ для симплекс-таблицы.
        :return: Строка с выводом симплекс-таблицы.
        """
        table = PrettyTable()
        table.field_names = self.top_row_
        for i in range(self.main_table_.shape[0]):
            row = [self.left_column_[i]] + list(self.main_table_[i])
            table.add_row(row)
        return table.__str__()
    def is_find_ref_solution(self):
        """
        Функция проверяет, найдено ли опорное решение по свободным в симплекс-
таблице.
        :return: True - опорное решение уже найдено. False - полученное решение пока
не является опорным.
        """
        # Проверяем все, кроме коэффициента ЦФ
        for i in range(self.main_table_.shape[0] - 1):
            if self.main_table_[i][0] < 0:
                return False
        return True
    def search_ref_solution(self):
        """
        Функция производит одну итерацию поиска опорного решения.
        """
        res_row = None
```

```
        for i in range(self.main_table_.shape[0] - 1):
            if self.main_table_[i][0] < 0:
                res_row = i
                break
        # Если найден отрицательный элемент в столбце свободных членов, то ищем
первый отрицательный в строке с ней.
        res_col = None
        if res_row is not None:
            for j in range(1, self.main_table_.shape[1]):
                if self.main_table_[res_row, j] < 0:
                    res_col = j
                    break
        # Если найден разрешающий столбец, то находим в нём разрешающий
элемент.
        res_element = None
        if res_col is not None:
            # Ищем минимальное положительное отношение Si0 / x[res_col]
            minimum = None
            ind = -1
            for i in range(self.main_table_.shape[0] - 1):
                # Ищем минимальное отношение -- разрешающую строку.
                curr = self.main_table_[i][res_col]
                s_i0 = self.main_table_[i][0]
                if curr == 0:
                    continue
                elif (s_i0 / curr) > 0 and (minimum is None or (s_i0 / curr) < minimum):
                    minimum = (s_i0 / curr)
                    ind = i
            if minimum is None:
                raise SimplexException("Решения не существует! При нахождении
опорного решения не нашлось минимального "
                                       "положительного отношения.")
            else:
                res_row = ind
            # Разрешающий элемент найден.
            res_element = self.main_table_[res_row][res_col]
            print("Разрешающая строка: {}".format(self.left_column_[res_row]))
            print("Разрешающий столбец: {}".format(self.top_row_[res_col + 1]))
            # Пересчёт симплекс-таблицы.
            self.recalc_table(res_row, res_col, res_element)
        else:
            raise SimplexException("Задача не имеет допустимых решений! При
нахождении опорного решения не нашлось "
                                   "отрицательного элемента в строке с
отрицательным свободным членом.")
    def is_find_opt_solution(self):
        """
        Функция проверяет, найдено ли оптимальное решение по коэффициентам ЦФ в
симплекс-таблице.
        :return: True - оптимальное решение уже найдено. False - полученное решение
пока не оптимально.
        """
        for i in range(1, self.main_table_.shape[1]):
            if self.main_table_[self.main_table_.shape[0] - 1][i] > 0:
                return False
```

```
        # Если положительных не нашлось, то оптимальное решение уже найдено.
        return True
    def optimize_ref_solution(self):
        """
        Функция производит одну итерацию поиска оптимального решения на основе
        уже полученного опорного решения.
        """
        res_col = None
        ind_f = self.main_table_.shape[0] - 1
        # В строке F ищем первый положительный.
        for j in range(1, self.main_table_.shape[1]):
            curr = self.main_table_[ind_f][j]
            if curr > 0:
                res_col = j
                break
        minimum = None
        res_row = None
        # Идём по всем, кроме ЦФ ищём минимальное отношение.
        for i in range(self.main_table_.shape[0] - 1):
            # Ищем минимальное отношение -- разрешающую строку.
            curr = self.main_table_[i][res_col]
            s_i0 = self.main_table_[i][0]
            if curr < 0:
                continue
            elif (s_i0 / curr) >= 0 and (minimum is None or (s_i0 / curr) < minimum):
                minimum = (s_i0 / curr)
                res_row = i
        if res_row is None:
            raise SimplexException("Функция не ограничена! Оптимального
решения не существует.")
        else:
            # Разрешающий элемент найден.
            res_element = self.main_table_[res_row][res_col]
            print("Разрешающая строка: {}".format(self.left_column_[res_row]))
            print("Разрешающий столбец: {}".format(self.top_row_[res_col + 1]))
            # Пересчёт симплекс-таблицы.
            self.recalc_table(res_row, res_col, res_element)
    def recalc_table(self, res_row, res_col, res_element):
        """
        Функция по заданным разрешающим строке, столбцу и элекменту производит
перерасчёт
        симплекс-таблицы методом жордановых искоючений.
        :param res_row: индекс разрешающей строки
        :param res_col: индекс разрешающего столбца
        :param res_element: разрешающий элемент
        """
        recalced_table = np.zeros((self.main_table_.shape[0], self.main_table_.shape[1]))
        # Пересчёт разрешающего элемента.
        recalced_table[res_row][res_col] = round(1 / res_element, ROUND_CONST)
        # Пересчёт разрешающей строки.
        for j in range(self.main_table_.shape[1]):
            if j != res_col:
                recalced_table[res_row][j] = round(self.main_table_[res_row][j] /
res_element, ROUND_CONST)
        # Пересчёт разрешающего столбца.
```

```
        for i in range(self.main_table_.shape[0]):
            if i != res_row:
                recalced_table[i][res_col] = -round((self.main_table_[i][res_col] /
res_element), ROUND_CONST)
        # Пересчёт оставшейся части таблицы.
        for i in range(self.main_table_.shape[0]):
            for j in range(self.main_table_.shape[1]):
                if (i != res_row) and (j != res_col):
                    recalced_table[i][j] = round(self.main_table_[i][j] - (
                            (self.main_table_[i][res_col] * self.main_table_[res_row][j]) /
res_element), ROUND_CONST)
        self.main_table_ = recalced_table
        self.swap_headers(res_row, res_col)
        print(self.__str__())
    def swap_headers(self, res_row, res_col):
        """
        Функция меняет меняет переменные в строке и столбце местами.
        :param res_row: разрешающая строка
        :param res_col: разрешающий столбец
        """
        temp = self.top_row_[res_col + 1]
        self.top_row_[res_col + 1] = self.left_column_[res_row]
        self.left_column_[res_row] = temp
```

Файл «dual_problem.py»

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

from simplex import *
class DualProblem(Simplex):
    """
    Класс унаследован от Simplex и нужен для переформулирования задачи из ПЗ в ДЗ.
    """
    def __init__(self, path_to_file):
        """
        Переопределённый метод __init__. Регистрирует входные данные из JSON-
файла.
        Определяем условие двойственной задачи.
        :param path_to_file: путь до JSON-файла с входными данными.
        """
        # Парсим JSON-файл с входными данными
        with open(path_to_file, "r") as read_file:
            json_data = json.load(read_file)
            # Коэффициенты при ЦФ в ДЗ равны свободным членам ограничений в ПЗ.
            self.obj_func_coffs_ = np.array(json_data["constraint_system_rhs"])
            # Свободные члены ограничений в ДЗ равны коэффициентам при ЦФ в ПЗ.
            self.constraint_system_lhs_ = np.array(
                json_data["constraint_system_lhs"]).transpose()
            # Коэффициенты  любого ограничения ДЗ равны коэффициентам при
одной переменной из всех ограничений ПЗ.
            self.constraint_system_rhs_ = np.array(json_data["obj_func_coffs"])
            # Минимизация ЦФ в ПЗ соответвстует максимизации ЦФ в ДЗ.
            self.func_direction_ = "max" if json_data[
                                                          "func_direction"] == "min" else
"min"
```

```
            print(self.__str__())
            # Ограничения вида (<=) ПЗ переходят в ограничения вида (>=) ДЗ.
            self.constraint_system_lhs_ *= -1
            self.constraint_system_rhs_ *= -1
            if len(self.constraint_system_rhs_) != self.constraint_system_rhs_.shape[0]:
                raise Exception("Ошибка при вводе данных. Число строк в матрице
и столбце ограничений не совпадает.")
            if len(self.constraint_system_rhs_) > len(self.obj_func_coffs_):
                raise Exception("СЛАУ несовместна! Число уравнений больше
числа переменных.")
            # Если задача на max, то меняем знаки ЦФ и направление задачи (в конце
возьмем решение со знаком минус и
            # получим искомое).
            if self.func_direction_ == "max":
                self.obj_func_coffs_ *= -1
            # Инициализация симплекс-таблицы.
            self.simplex_table_ = SimplexTable(self.obj_func_coffs_,
self.constraint_system_lhs_,
                                               self.constraint_system_rhs_)
    def __str__(self):
        """
        Переопренный метод __str__ для условия двойственной задачи.
        :return: Строка с выводом условия двойственной задачи.
        """
        output = """------------------------------------------------------------"""
        output += f"\nF = cx -> {self.func_direction_},"
        output += "\nAx >= 1,\nx1,x2, ..., xn >= 0"
        if self.func_direction_ == "max":
            output += f"\nC = {-self.obj_func_coffs_},"
        else:
            output += f"\nC = {self.obj_func_coffs_},"
        output += f"\nA =\n{self.constraint_system_lhs_},"
        output += f"\nb^T = {self.constraint_system_rhs_}."
        output += "\n------------------------------------------------------------"
        return output
```

Файл «strategic.py»

```
# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>

from simplex_table import SimplexTable
ROUND_CONST = 3
def StrategyA(simplex_table: SimplexTable):
    """
    Решение для смешанной стратегии игрока A.
    :param simplex_table: конечная симплекс таблица.
    :return: консольный вывод.
    """
    d = {"x1": 0, "x2": 0, "x3": 0, "x4": 0, "x5": 0, "W": 0, 'g': 0}
    for i in range(len(simplex_table.left_column_)):
        if simplex_table.left_column_[i] in ["x1", "x2", "x3", "x4", "x5"]:
            d[simplex_table.left_column_[i]] = round(simplex_table.main_table_[i][0],
ROUND_CONST)
    d["W"] = round(simplex_table.main_table_[-1][0], ROUND_CONST)
    d["g"] = round(1 / d["W"], ROUND_CONST)
```

```
    strategy_a = [round(d['x1'] * d['g'], ROUND_CONST), round(d['x2'] * d['g'], ROUND_CONST),
                  round(d['x3'] * d['g'], ROUND_CONST), round(d['x4'] * d['g'], ROUND_CONST),
                  round(d['x5'] * d['g'], ROUND_CONST)]
    output = f"Решение для смешанной стратегии игрока A:\nu1 = {d['x1']}, u2 = {d['x2']}, " \
             f"u3 = {d['x3']}, u4 = {d['x4']}, u5 = {d['x5']};\n" \
             f"W = {d['W']};\ng = 1/W = {d['g']}.\n" \
             f"Частоты выбора стратегий:\n" \
             f"x1 = u1·g = {d['x1']}·{d['g']} = {strategy_a[0]};\nx2 = u2·g = {d['x2']}·{d['g']} = {strategy_a[1]};\n" \
             f"x3 = u3·g = {d['x3']}·{d['g']} = {strategy_a[2]};\nx4 = u4·g = {d['x4']}·{d['g']} = {strategy_a[3]};\n" \
             f"x5 = u5·g = {d['x5']}·{d['g']} = {strategy_a[4]};\n" \
             f"Таким образом, оптимальная смешанная стратегия игрока A имеет вид\n {strategy_a}.\n\n"
    return output
def StrategyB(simplex_table: SimplexTable):
    """
    Решение для смешанной стратегии игрока B.
    :param simplex_table: конечная симплекс таблица.
    :return: консольный вывод.
    """
    d = {"x1": 0, "x2": 0, "x3": 0, "x4": 0, "Z": 0, 'h': 0}
    for i in range(len(simplex_table.left_column_)):
        if simplex_table.left_column_[i] in ["x1", "x2", "x3", "x4"]:
            d[simplex_table.left_column_[i]] = round(simplex_table.main_table_[i][0], ROUND_CONST)
    d["Z"] = round(simplex_table.main_table_[-1][0], ROUND_CONST)
    d["h"] = round(1 / d["Z"], ROUND_CONST)
    strategy_b = [round(d['x1'] * d['h'], ROUND_CONST), round(d['x2'] * d['h'], ROUND_CONST),
                  round(d['x3'] * d['h'], ROUND_CONST), round(d['x4'] * d['h'], ROUND_CONST)]
    output = f"Решение для смешанной стратегии игрока B:\nv1 = {d['x1']}, " \
             f"v2 = {d['x2']}, v3 = {d['x3']}, v4 = {d['x4']};\n" \
             f"Z = {d['Z']};\nh = 1/Z = {d['h']}.\n" \
             f"Частоты выбора стратегий:\n" \
             f"y1 = v1·h = {d['x1']}·{d['h']} = {strategy_b[0]};\ny2 = v2·h = {d['x2']}·{d['h']} = {strategy_b[1]};\n" \
             f"y3 = v3·h = {d['x3']}·{d['h']} = {strategy_b[2]};\ny4 = v4·h = {d['x4']}·{d['h']} = {strategy_b[3]};\n" \
             f"Таким образом, оптимальная смешанная стратегия игрока B имеет вид\n {strategy_b}."
    return output
```

Файл «input_data.json»

```
{
  "obj_func_coffs": [1, 1, 1, 1],
  "constraint_system_lhs": [
```

```
        [1, 11, 12, 11],
        [7, 5, 7, 7],
        [16, 6, 13, 2],
        [9, 9, 16, 13],
        [17, 18, 15, 7]
    ],
    "constraint_system_rhs": [1, 1, 1, 1, 1],
    "func_direction": "max"
}
```