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 №2 на тему:

«Двойственность в линейном программировании»

Вариант – 1

Преподаватель:
Коннова Н.С.

Студент:
Александров А. Н.

Группа:
ИУ8-34

Москва, 2020

Цель работы:

изучить постановку двойственной задачи (ДЗ) линейного программирования по прямой задаче (ПЗ); получить навыки решения соответствующей ДЗ по прямой задаче.

Постановка задачи:

Пусть исходная ПЗ ЛП имеет вид:

$$F = cx \to max,$$

$$Ax \le b,$$

$$x_1, x_2, x_3 \ge 0$$

Сформулировать двойственную задачу и найти ее решение.

По условию:

$$c = (5\ 6\ 4)$$

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 0 \\ 0 & 0.5 & 4 \end{pmatrix}$$

$$b = \begin{pmatrix} 7 \\ 8 \\ 6 \end{pmatrix}$$

Ход решения:

Решение исходной ПЗ ЛП симплекс-методом, полученное мной в предыдущей лабораторной работе имеет вид:

$$x_3 = x_4 = x_5 = 0, x_1 = 6.5, x_2 = 0.5, x_6 = 5.8; F = 35.5$$

Двойственная задача ЛП имеет вид:

$$\Phi = b^T y \to min;$$

$$A^T y \ge c^T;$$

$$y \ge 0;$$

Сформулируем двойственную задачу ЛП:

$$\Phi = 7y_1 + 8y_2 + 6y_3 \to min;$$

$$\begin{cases} y_1 + y_2 \ge 5 \\ y_1 + 3y_2 + 0.5y_3 \ge 6; \\ y_1 + 4y_3 \ge 4 \end{cases}$$

$$y_1, y_2, y_3 \geq 0$$

Запишем в каноническом виде и выразим базисные переменные:

$$\Phi = -\left(-7\,y_1 - 8\,y_2 - 6\,y_3\right) \rightarrow min;$$

$$\begin{cases} y_4 = -5 - \left(-y_1 - y_2\right) \\ y_5 = -6 - \left(-y_1 - 3\,y_2 - 0.5\,y_3\right); \\ y_6 = -4 - \left(-y_1 - 4\,y_3\right) \end{cases}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \geq 0$$

Построим исходную симплекс-таблицу ДЗ ЛП. За базисные переменные принимаем дополнительные (для удобства), Si0 – столбец свободных переменных:

Таблица 1 Начальная симплекс-таблица

	Si0	**x1**	**x2**	**x3**
x4	-5	-1	-1	0
x5	-6	-1	-3	-0.5
x6	-4	-1	0	-4
F	0	-7	-8	-6

Для решения задачи воспользовался дополненной версией программы на языке программирования **Python**, написанной мной для предыдущей лабораторной работы (см. Приложение А):

Добавлен файл ***dual_problem.py***:

В данном классе был реализован класс ***DualProplem***, унаследованный от класса ***Simplex*** (из файла ***simplex.py***). Ещё раз напомню те поля и методы, которые унаследовались от родительского класса:

Поля класса ***Simplex*** :

Поле **obj_func_coffs_**: хранит коэффициенты целевой функции (ЦФ).

Поле **obj_constraint_system_lhs_**: хранит левую часть системы ограничений.

Поле **obj_constraint_system_rhs_**: хранит правую часть системы ограничений.

Поле **func_direction_**: хранит в себе направление целевой функции(поиск минимума или поиск максимума)

Поле **simplex_table_**: хранит в себе симплекс таблицу, динамически меняющуюся на каждой итерации жардановых исключений (объект класса ***SimplexTable***)

Методы класса ***Simplex***:

Метод **__init__(self, path_to_file)**: считывает данные из JSON-файла, находящегося по пути path_to_file и инициализирует ими поля созданного объекта класса Simplex.

Метод **__str__(self)**: переопределяет строковое представление нашего объекта класса, для вывода условия задачи в стандартный поток вывода.

Метод **reference_solution(self)**: производит отыскание опорного решения в симплекс-методе.

Метод **optimal_soputions(self)**: производит отыскание оптимального решения на основе полученного опорного решения.

Метод **output_solution(self)**: метод выводит текущее решение в стандартный

поток вывода. Используется для вывода опорного и оптимального решений.

Для переформулировки задачи переопределим следующие поля дочернего класса:

Метод **__init__(self, path_to_file)**: также считывает данные с JSON-файла, однако теперь мы руководствуемся правилами формулировки ДЗ ЛП, а именно:

- поле **self.obj_func_coffs_** заполняется значениями из **constraint_system_rhs**;
- поле – есть транспонированная матрица ограничений для ПЗ ЛП;
- поле **self.constraint_system_rhs** заполняется значениями из **obj_func_coffs**;
- в поле **self.func_direction_** минимизация (максимизация) ЦФ в ПЗ соответствует максимизации (минимизации) ДЗ;
- Домножем поля **self.constraint_system_lhs_** и **self.constraint_system_rhs_** на -1, тем самым меняя знаки неравенств;

Метод **__str__(self)**: переопределяем строковое представление нашего класса, дабы получть хороший вывод ДЗ в соответствующий поток.

Пронаблюдаем ход выполнения программы. Для начала идёт отыскание оптимального решения ПЗ ЛП симплекс-методом:

Рисунок 1 Поиск опорного решения ПЗ

```
ЛР2 по МО. "ДЗ в линейном программировании.
    Прямая задача ЛП:
Условие задачи:
------------------------------------------------------------
Найти вектор x = (x1,x2,..., xn)^T как решение след. задачи:
F = cx -> max,
Ax <= b,
x1,x2, ..., xn >= 0
C = [-5 -6 -4],
A =
[[1.  1.  1. ]
 [1.  3.  0. ]
 [0.  0.5 4. ]],
b^T = [7 8 6].
------------------------------------------------------------
Процесс решения:
1) Поиск опорного решения:
Исходная симплекс-таблица:
         Si0    x1    x2    x3
    x4   7.0   1.0   1.0   1.0
    x5   8.0   1.0   3.0   0.0
    x6   6.0   0.0   0.5   4.0
    F    0.0   5.0   6.0   4.0

-----------
Опорное решение найдено!
x1 = x2 = x3 = 0, x4 =  7.0, x5 =  8.0, x6 =  6.0,
Целевая функция: F = 0.0
```

Рисунок 2 Поиск оптимального решения ПЗ

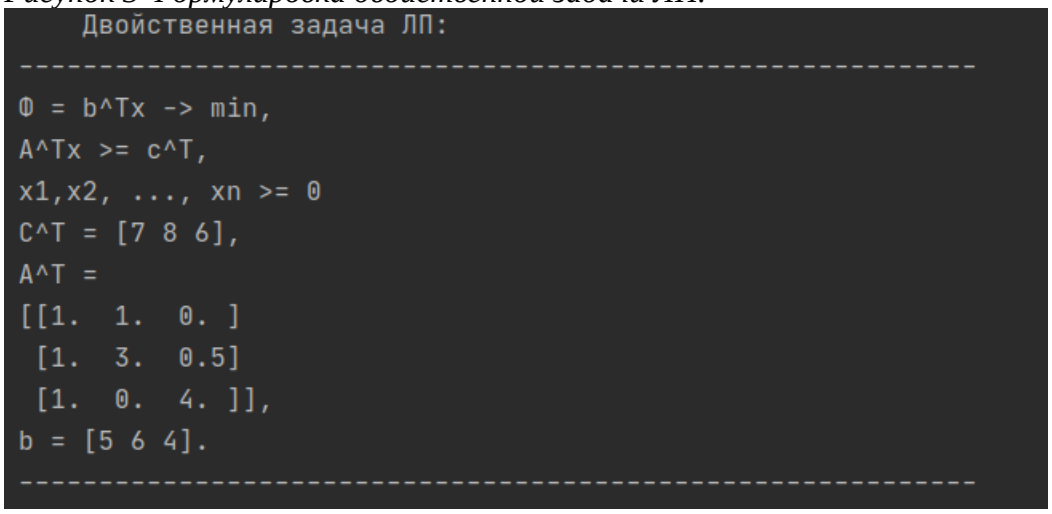
```
2) Поиск оптимального решения:
Разрешающая строка: x1
Разрешающий столбец: x4
         Si0    x4    x2    x3
    x1   7.0   1.0   1.0   1.0
    x5   1.0  -1.0   2.0  -1.0
    x6   6.0  -0.0   0.5   4.0
    F  -35.0  -5.0   1.0  -1.0

Разрешающая строка: x2
Разрешающий столбец: x5
         Si0    x4    x5    x3
    x1   6.5   1.5  -0.5   1.5
    x2   0.5  -0.5   0.5  -0.5
    x6   5.8   0.2  -0.2   4.2
    F  -35.5  -4.5  -0.5  -0.5

-----------
Оптимальное решение найдено!
x4 = x5 = x3 = 0, x1 =  6.5, x2 =  0.5, x6 =  5.8,
Целевая функция: F = 35.5
```

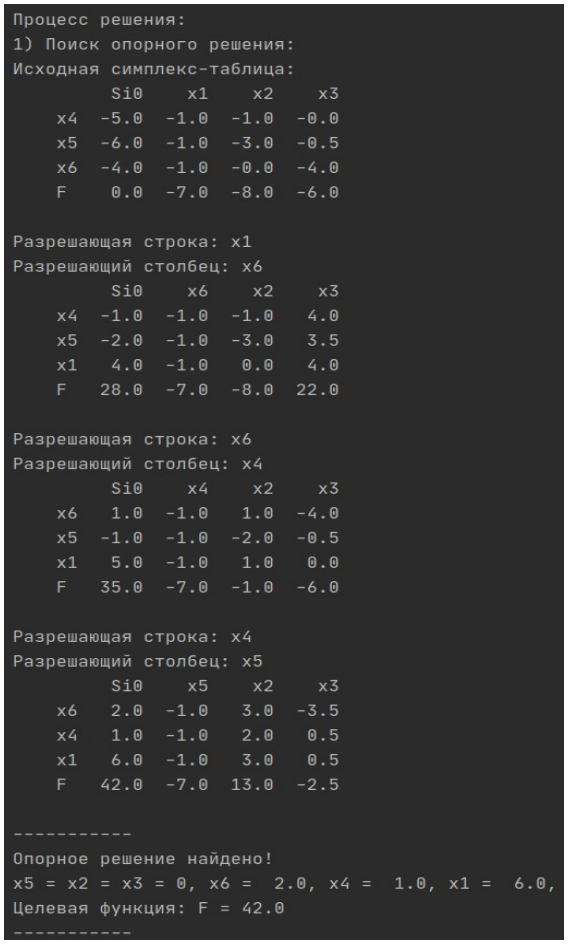
Далее переходим к переформулировке ПЗ в ДЗ:

Рисунок 3 Формулировка двойственной задачи ЛП.



При поиске опороного решения происходит 3 итерации пересчёта симплекс-таблицы:

Рисунок 4 Поиск опорного решения в ДЗ ЛП.



Получили опорное решение:

$$y_2 = y_3 = y_5 = 0, y_1 = 6, y_4 = 1, y_6 = 2;\ \Phi = 42$$

Поиск оптимального решения произведём за 1 итерацию и получим ответ:

Рисунок 5 Оптимальное решение ДЗ ЛП

```
2) Поиск оптимального решения:
Разрешающая строка: x2
Разрешающий столбец: x4
         Si0    x5    x4    x3
    x6   0.5   0.5  -1.5  -4.2
    x2   0.5  -0.5   0.5   0.2
    x1   4.5   0.5  -1.5  -0.2
    F   35.5  -0.5  -6.5  -5.8

-----------
Оптимальное решение найдено!
x5 = x4 = x3 = 0, x6 =  0.5, x2 =  0.5, x1 =  4.5,
Целевая функция: F = 35.5
-----------

Process finished with exit code 0
```

Таким образом получили оптимальное решение:

$$y_3 = y_4 = y_5 = 0, y_1 = 4.5, y_2 = 0.5, y_6 = 0.5;\ \Phi = 35.5$$

Ответ: $y_3 = y_4 = y_5 = 0, y_1 = 4.5, y_2 = 0.5, y_6 = 0.5;\ \Phi = 35.5$

Сравнение решений ПЗ и ДЗ:

Получили, что прямая и двойственная задачи ЛП имеют одно и то же оптимальное решение: $F = Ф = 35.5$, что согласуется с принципом двойственности.

Вывод:

В данной работе была изучена постановка двойственной задачи линейного программирования по соответсвующей прямой задаче. Были получены навыки в переформулировнии ПЗ ЛП в ДЗ ЛП. Стало известно, что оптимальные решения прямой и двойственной задач совпадают, что можно использовать для проверки результатов.

Программу, написанную для решения ПЗ ЛП, было нетрудно модифицировать для решения ДЗ ЛП, программа изначально делалась с учётом на дальнейшую расширяемость, что помогло в решении этой работы.

Приложение А

Код программы

Файл “main.py”

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

"""
Лабораторная работа № 2
Двойственность в линейном программировании.
Цель: Изучить постановку двойственной задачи (ДЗ);
ЛП по прямой задаче (ПЗ); получить навыки решения соответствующей
ДЗ по прямой задаче.

Вариант 1.
"""

from simplex import *
import dual_problem

if __name__ == '__main__':
    print('ЛР2 по МО. "ДЗ в линейном программировании.', )

    print("\tПрямая задача ЛП:")
    problem = Simplex("input_data.json")
    print(problem)

    # Находим опорное решение.
    problem.reference_solution()
    # Находим оптимальное решение.
    problem.optimal_solution()

    print("\tДвойственная задача ЛП:")
    dual_p = dual_problem.DualProblem("input_data.json")

    # Находим опорное решение.
    dual_p.reference_solution()
    # Находим оптимальное решение.
    dual_p.optimal_solution()
```

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 A
            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 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 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 <= b,\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):
        """
        Функция 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("-----------")

    # Этап 2. Поиск оптимального решения.
    def optimal_solution(self):
        """
        Функция 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):
        """
        Функция выводит текущее решение, используется для вывода
опорного и потимального 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], 1), end=", ")

        print("\nЦелевая функция: F =",
round(self.simplex_table_.main_table[last_row_ind][0], 1))
```

Файл “simplex_table.py”

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

import numpy as np


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] = constraint_system_rhs[i]
        # Заполняем строку F.
        for j in range(var_count):
            self.main_table[constraint_count][j + 1] = -obj_func_coffs[j]

        # Заполняем A.
        for i in range(constraint_count):
            for j in range(var_count):
```

```
                self.main_table[i][j + 1] = constraint_system_lhs[i][j]

    def __str__(self):
        """
        Переопренный метод __str__ для симплекс-таблицы.
        :return: Строка с выводом симплекс-таблицы.
        """
        output = ""
        for header in self.top_row:
            output += '{:>6}'.format(header)
        output += "\n"

        for i in range(self.main_table.shape[0]):
            output += '{:>6}'.format(self.left_column[i])
            for j in range(self.main_table.shape[1]):
                output += '{:>6}'.format(round(self.main_table[i][j], 1))
            output += "\n"

        return output

    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):
                curr = self.main_table[i][0] / self.main_table[i][res_col]
                if curr < 0 or (self.main_table[i][res_col] == 0):
                    continue
                elif minimum is None:
                    minimum = curr
                    ind = i
                elif curr < minimum:
                    minimum = curr
                    ind = i

            if minimum is None:
                raise Exception("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_col]))
            print("Разрешающий столбец: {}".format(self.top_row[res_row +
1]))

            # Пересчёт симплекс-таблицы.
            self.recalc_table(res_row, res_col, res_element)
        else:
            raise Exception("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.
        """
        ind_f = self.main_table.shape[0] - 1

        for i in range(1, self.main_table.shape[1]):
            curr = self.main_table[ind_f][i]
            if curr > 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 положительное
отношение.
        for i in range(self.main_table.shape[0] - 1):
            # Ищем минимальное положительное 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 Exception("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_col]))
            print("Разрешающий столбец: {}".format(self.top_row[res_row +
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] = 1 / res_element

        # Пересчёт разрешающей строки.
        for j in range(self.main_table.shape[1]):
            if j != res_col:
                recalced_table[res_row][j] = self.main_table[res_row][j] /
res_element

        # Пересчёт разрешающего столбца.
        for i in range(self.main_table.shape[0]):
            if i != res_row:
                recalced_table[i][res_col] = -(self.main_table[i][res_col] /
res_element)

        # 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] = self.main_table[i][j] - (
                            (self.main_table[i][res_col] * self.main_table[res_row][j]) /
res_element)

        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"\nΦ = b^Tx -> {self.func_direction_},"
        output += "\nA^Tx >= c^T,\nx1,x2, ..., xn >= 0"

        if self.func_direction_ == "max":
            output += f"\nC^T = {-self.obj_func_coffs_},"
        else:
            output += f"\nC^T = {self.obj_func_coffs_},"

        output += f"\nA^T =\n{self.constraint_system_lhs_},"
        output += f"\nb = {self.constraint_system_rhs_}."
        output += "\n----------------------------------------------------------"

        return output
```

Файл “input_data.json”

```
{
  "obj_func_coffs": [5, 6, 4],
  "constraint_system_lhs": [
    [1, 1, 1],
    [1, 3, 0],
    [0, 0.5, 4]
  ],
  "constraint_system_rhs": [7, 8, 6],
  "func_direction": "max"
}
```