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 №1 на тему:
«Линейное программирование. Симплекс-метод»

Вариант 1

Преподаватель:

Коннова Н.С.

Студент:

Александров А.Н.

Группа:

ИУ8-34

Москва, 2020

Цель работы:

изучить постановку задачи линейного программирования (ЛП); овладеть навыками решения задач ЛП с помощью симплекс-метода.

Постановка задачи:

требуется найти решение следующей задачи:

$$F = cx \to max\,,$$

$$Ax \le b\,,$$

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

Необходимо найти вектор $x = (x_1, x_2, x_3)^T$. Здесь:

c – вектор коэффициентов целевой функции(ЦФ),

A – матрица системы ограничений,

b – вектор правой части системы ограничений.

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

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

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

Ход работы:

Начнём решение с записи задачи ЛП в каноническом виде. По заданию нам необходимо найти максимум целевой функции, поэтому требуется изменить знаки коэффициентов целевой функции, при этом задача сводится к поиску минимума. После нахождения минимума, мы вернёмся к искомому:

$$max\, F(x) = -min\{-F(x)\}$$

Преобразуем систему неравенств в СЛАУ, добавив дополнительные (фиктивные) переменные.

Получаем запись задачи в каноническом виде:

$$F = -5x_1 - 6x_2 - 4x_3 \rightarrow min\,;$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 7 \\ x_1 + 3x_2 + x_5 = 8 \\ 0.5x_2 + x_3 + x_6 = 6 \end{cases};$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

Далее выражаем базисные переменные в СЛАУ и ЦФ:

$$\begin{cases} x_4 = 7 - (x_1 + x_2 + x_3) \\ x_5 = 8 - (x_1 + 3x_2) \\ x_6 = 6 - (0.5x_2 + 4x_3) \end{cases}$$

$$F = -(5x_1 + 6x_2 + 4x_3)$$

Построим начальную симплекс-таблицу. Удобно взять дополнительные переменные в качестве базисных, **si0** – столбец свободных переменных:

Таблица 1 Начальная симплекс-таблица

	si0	**x1**	**x2**	**x3**
x4	7	1	1	1
x5	8	1	3	0
x6	6	0	0.5	4
F	0	5	6	4

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

1. simplex.py

В данном файле реализован класс ***Simplex***, который регистрирует входные данные, определяет данные задачи и, посредством своих методов производит поиск опроного и оптимального решений. Входные данные берутся из файла ***input_data.json*** посредством парсинга с помощью подключенной библиотеки **json.** Сюда же подключен файл ***simplex_table.py***, описанный ниже.

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_sopution(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.

2. simplex_table.py

В данном файле реализован класс ***SimplexTable***, отвечающий за за заполнение симплекс-таблиц, их ведение и пересчёт. Для хранения матриц, массивов и удобной работы с ними была подключена библиотека **numpy**.

Поля класса ***SimplexTable***:

Поле **top_row_**: верхняя строка таблицы, содержащая названия столбцов в виде массива (свободные переменные).

Поле **left_column_**: левый столбец таблицы, содержащий названия строк в виде массива (базисные переменные и F).

Поле **main_table_**: содержит значения коэффициентов при ЦФ и при переменных в виде двумерной матрицы.

Методы класса **SimplexTable**:

Метод **__init__(self, obj_func_coffs, constraint_system_lhs, constraint_system_rhs)**: заполняет симплекс-таблицу по заданным коэффициентам ЦФ и по системе ограничений.

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

Метод **is_find_ref_solution(self):** проверяет по столбцу свободных членов, найдено ли опорное решение.

Метод **search_ref_solution(self)**: производит одну итерацию поиска опорного решения в симплекс-таблице.

Метод **is_find_opt_solution(self)**: проверяет, найдено ли оптимальное решение.

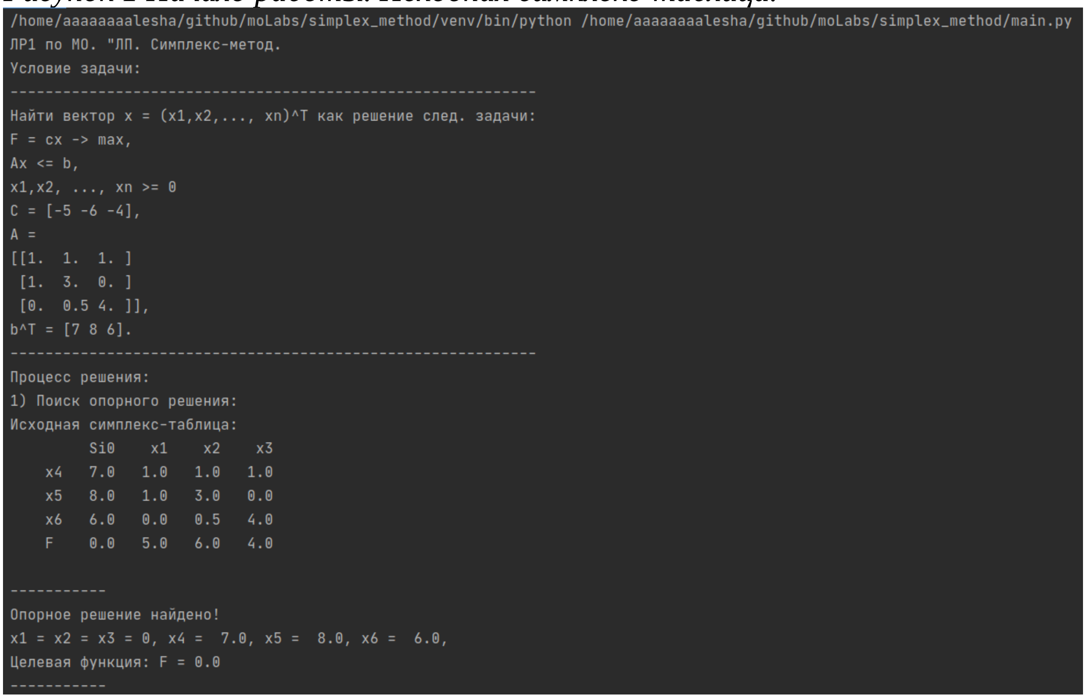
Метод **optimize_ref_solution(self)**: производит одну итерацию оптимизации на основе опорного решения.

Метод **recalc_table(self, res_row, res_col, res_element)**: по заданным разрешающим строке, столбцу и элементу производит пересчёт симплекс-таблицы методом жардановых исключений.

Метод **swap_headers(self, res_row, res_col)**: меняет базисную и свободную переменную местами в разрешающих строке и столбце.

Прокомментирую ход выполнения программы. Для начала мы выводим постановку задачи. Далее задача сводится к каноническому виду и строится начальная симплекс-таблица (рис. 1).

Рисунок 1 Начало работы. Исходная симплекс-таблица.



```
/home/aaaaaaaalesha/github/moLabs/simplex_method/venv/bin/python /home/aaaaaaaalesha/github/moLabs/simplex_method/main.py
ЛР1 по МО. "ЛП. Симплекс-метод.
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
-----------
```

Видим, что столбец свободных членов неотрицателен, что сразу же даёт нам опорное решение:

$$x_1 = x_2 = x_3 = 0, x_4 = 7, x_5 = 8, x_6 = 6; F = 0$$

Переходим ко второму этапу – оптимизации полученного решения. (рис. 2)

На первой итерации оптимизации получаем решение:

Рисунок 2 Нахождение оптимального решения.

```
2) Поиск оптимального решения:
Разрешающая столбец: x4
Разрешающий строка: x1
         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 столбец: x5
Разрешающий строка: x2
         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
-----------

Process finished with exit code 0
```

$$x_4 = x_2 = x_3 = 0, x_1 = 7, x_5 = 1, x_6 = 6; F = -35.0$$

Однако оно не оптимально – в строке F всё ещё имеется положительный элемент, поэтому, проведя ещё одну итерацию, получаем решение:

$$x_4 = x_2 = x_3 = 0, x_1 = 7, x_5 = 1, x_6 = 6; F = -35.5$$

Осталось лишь поменять знак и вернуться к искомому максимуму $max\,F(x) = -min\{-F(x)\}$. Получаем оптимальное решение.

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

Проверка решения на допустимость:

$$F(6.5, 0.5, 0)=5\cdot 6.5+6\cdot 0.5+4\cdot 0=35.5$$

$$\begin{cases} 6.5+0.5+0\leqslant 7 \\ 6.5+3\cdot 0.5+0\leqslant 8 \\ 0\cdot 6.5+0.5\cdot 0.5+4\cdot 0\leqslant 6 \end{cases};$$

$$6.5, 0.5, 0\geq 0$$

Решение действиетельно найдено верно и оно оптимльно.

Вывод:

В данной лабораторной работе была рассмотрена задача линейного программирования. Мною была изучена постановка задачи ЛП, были получены необходимые навыки решения задачи ЛП с помощью симплекс-метода.

В ходе решения была написана программа на языке Python, что помогло также получить необходимый опыт как в программировании, так и в декомпозиции задач.

Изученный метод оптимизация с помощью симплекс-таблиц весомо ускоряет процесс решения задачи линейного программирования и является крайне эффективным.

Приложение А

Код программы:

Файл “main.py”

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

"""
Лабораторная работа № 1
Линейное программирование. Симплекс-метод.
Цель: Изучить постановку задачи линейного программирования (ЛП);
овладеть навыками решения задач ЛП с помощью симплекс-метода.

Вариант 1.
"""

from simplex import *

if __name__ == '__main__':
    problem = Simplex("input_data.json")

    print('ЛР1 по МО. "ЛП. Симплекс-метод.', )
    print(problem)

    problem.reference_solution()

    problem.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 c - 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 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):
        """
        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("-----------")

    # 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 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.h”

```
# 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 и для её заполнения.
        :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
```

```python
        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):
        """
        Метод 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):
        """
        Метод 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 столбец:
{}".format(self.left_column_[res_row]))
            print("Разрешающий строка: {}".format(self.top_row_[res_col + 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):
        """
        Метод проверяет, найдено ли оптимальное решение по строке 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):
        """
        Метод 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 столбец:
{}".format(self.left_column_[res_row]))
            print("Разрешающий строка: {}".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):
        """
        Метод по заданным разрешающим строке, столбцу и элементу
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
```

```python
        # 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):
        """
        Метод меняет меняет переменные в строке и столбце местами.
        :param res_row: разрешающая строка
        :param res_col: разрешающий столбец
        """

        self.top_row_[res_col + 1], self.left_column_[res_row] =
self.left_column_[res_row], self.top_row_[res_col + 1]
```

Файл “input_data.json”

```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"
}
```