**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 *А*, столбцы – стратегиям игрока *B*):

**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 |  |  |  |  |  |
|  | 1 | 11 | 12 | 11 | 1 |
|  | 7 | 5 | 7 | 7 | 5 |
|  | 16 | 6 | 13 | 2 | 2 |
|  | 9 | 9 | 16 | 13 | 9 |
|  | 17 | 18 | 15 | 7 | 7 |
|  | 17 | 18 | 16 | 13 |  |

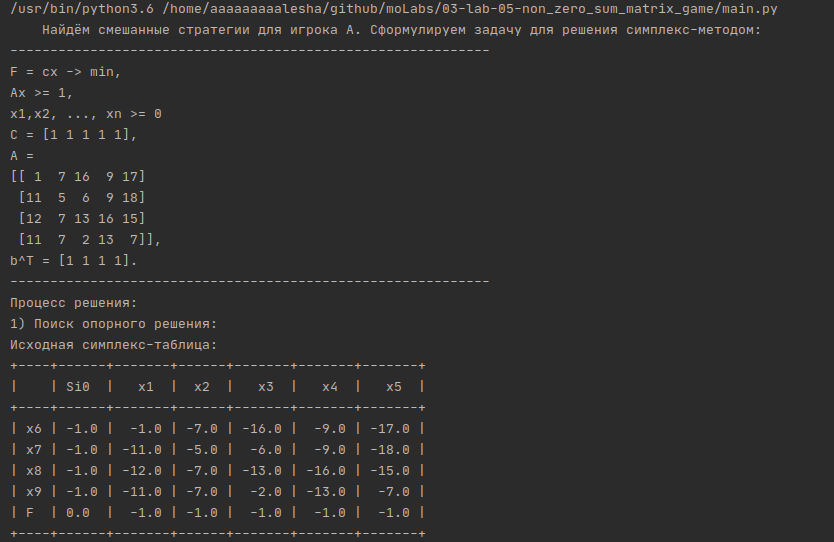
Цена игры:

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

где *g* — минимальный выигрыш.

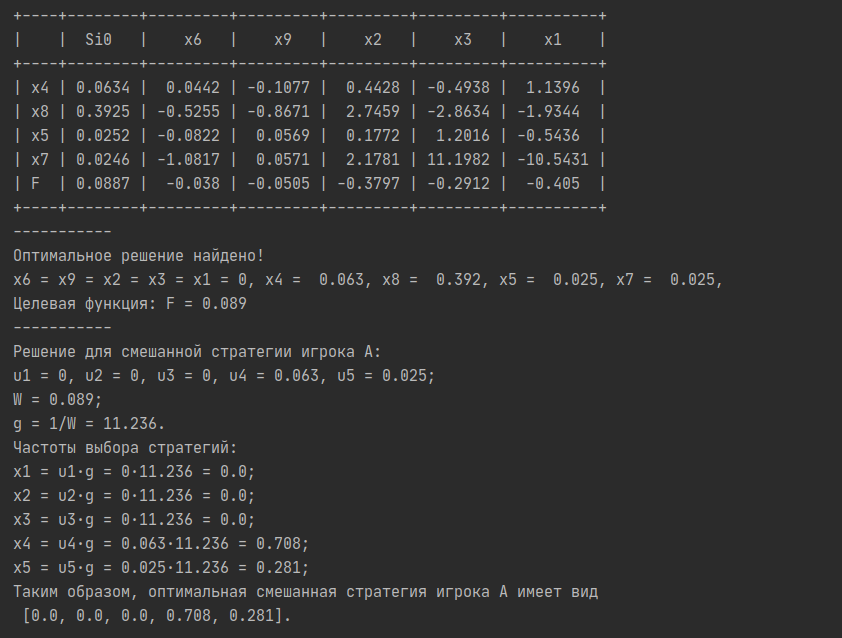
Разделим систему на функцию *g*:

Сформулируем задачу для решения симплекс-методом:

  
Рисунок 1 Формулировка задачи ЛП. Начальная симплекс-таблица.

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

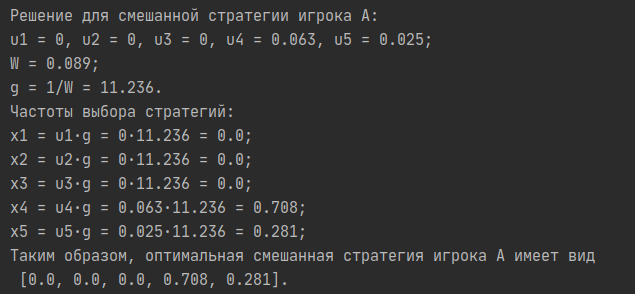
Решение для смешанной стратегии игрока А:

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

Цена игры: – верно.

Оптимальные стратегии:

Таким образом, оптимальная смешанная стратегия игрока А имеет вид:

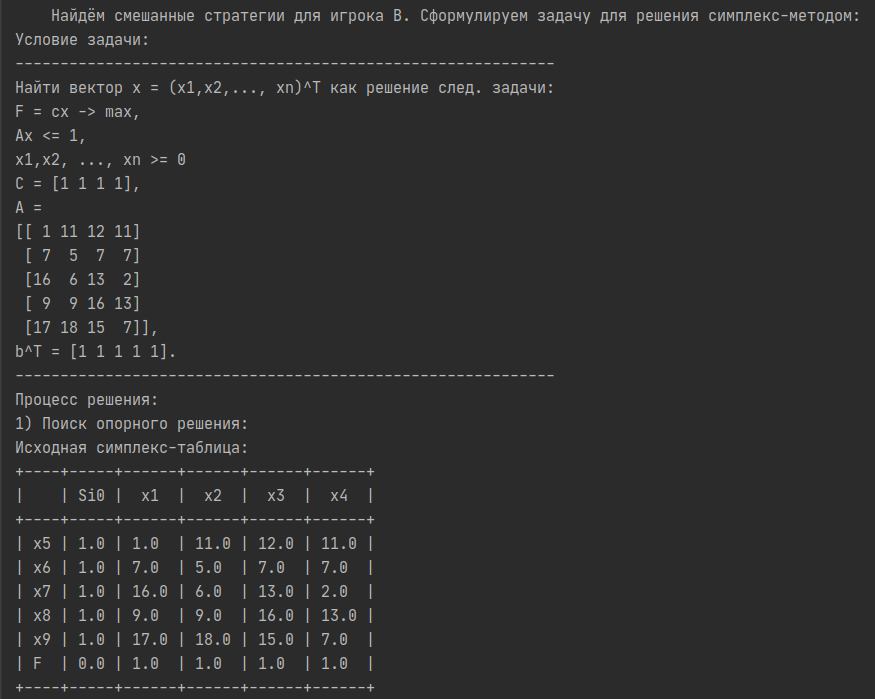
  
Рисунок 3 Решение для смешанной стратегии игрока А.

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

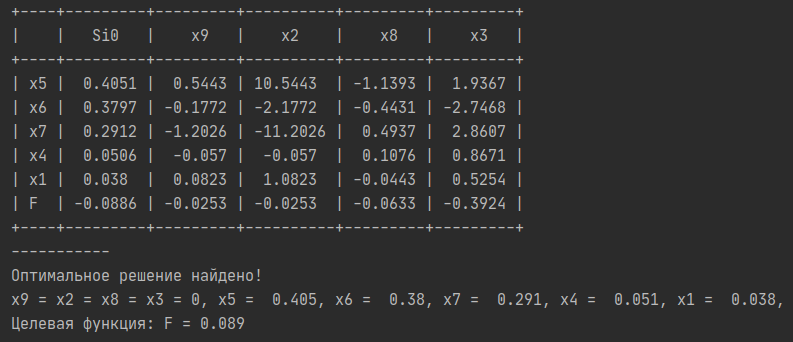
где *h* — максимальный проигрыш игрока *B*.

Разделим систему на *h*:

Сформулируем задачу для решения симплекс-методом:

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

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

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

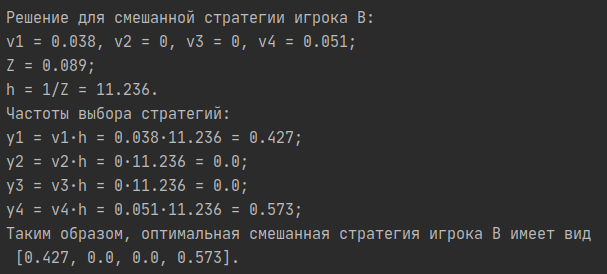
Решение для смешанной стратегии игрока *B*:

Цена игры: – верно

Частоты выбора стратегий:

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

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

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

**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 *А*, и к двойственной задаче ЛП для игрока *B.*

По результатам видим, что полученная цена игры лежит в найденном в начале диапазоне нижней и верхней цен игры: 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(**"**\t**Найдём смешанные стратегии для игрока А. 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(**"**\t**Найдём 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"**\n**F = cx ->** {self.func\_direction\_}**,"**

output += **"**\n**Ax <= 1,**\n**x1,x2, ..., xn >= 0"**

output += **f"**\n**C =** {-self.obj\_func\_coffs\_}**,"**

output += **f"**\n**A =**\n{self.constraint\_system\_lhs\_}**,"**

output += **f"**\n**b^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:**\n**1) 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 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(**"**\n**Целевая функция: F ="**, round(self.simplex\_table\_.main\_table\_[last\_row\_ind][0], 3))

***Файл «simplex\_table.py»***

***# 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 А.*

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"**\n**F = cx ->** {self.func\_direction\_}**,"**

output += **"**\n**Ax >= 1,**\n**x1,x2, ..., xn >= 0"**

if self.func\_direction\_ == **"max"**:

output += **f"**\n**C =** {-self.obj\_func\_coffs\_}**,"**

else:

output += **f"**\n**C =** {self.obj\_func\_coffs\_}**,"**

output += **f"**\n**A =**\n{self.constraint\_system\_lhs\_}**,"**

output += **f"**\n**b^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 А.*

*: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 А:**\n**u1 =** {d[**'x1'**]}**, u2 =** {d[**'x2'**]}**, "** \

**f"u3 =** {d[**'x3'**]}**, u4 =** {d[**'x4'**]}**, u5 =** {d[**'x5'**]}**;**\n**"** \

**f"W =** {d[**'W'**]}**;**\n**g = 1/W =** {d[**'g'**]}**.**\n**"** \

**f"Частоты выбора стратегий:**\n**"** \

**f"x1 = u1·g =** {d[**'x1'**]}**·**{d[**'g'**]} **=** {strategy\_a[0]}**;**\n**x2 = u2·g =** {d[**'x2'**]}**·**{d[**'g'**]} **=** {strategy\_a[1]}**;**\n**"** \

**f"x3 = u3·g =** {d[**'x3'**]}**·**{d[**'g'**]} **=** {strategy\_a[2]}**;**\n**x4 = 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 В:**\n**v1 =** {d[**'x1'**]}**, "** \

**f"v2 =** {d[**'x2'**]}**, v3 =** {d[**'x3'**]}**, v4 =** {d[**'x4'**]}**;**\n**"** \

**f"Z =** {d[**'Z'**]}**;**\n**h = 1/Z =** {d[**'h'**]}**.**\n**"** \

**f"Частоты выбора стратегий:**\n**"** \

**f"y1 = v1·h =** {d[**'x1'**]}**·**{d[**'h'**]} **=** {strategy\_b[0]}**;**\n**y2 = v2·h =** {d[**'x2'**]}**·**{d[**'h'**]} **=** {strategy\_b[1]}**;**\n**"** \

**f"y3 = v3·h =** {d[**'x3'**]}**·**{d[**'h'**]} **=** {strategy\_b[2]}**;**\n**y4 = 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"

}