Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)
КАФЕДРА Информационная безопасность (ИУ8)

Теория игр и исследование операций

Расчётно-графическое домашнее задание

Вариант: 1

Студент:
Александров Алексей Николаевич, группа ИУ8-104
(5 курс)

____
(подпись, дата)

Преподаватель:
к.т.н., доцент кафедры ИУ8
Коннова Наталья Сергеевна

____
(подпись, дата)

Москва, 2024 г.

Оглавление

Цель работы

Изучить постановку антагонистической игры двух лиц с нулевой суммой. Получить практические навыки нахождения решения игры за обоих игроков в смешанных стратегиях следующими методами:

- графоаналитическим;
- аналитическим (матичным);
- графическим (задача линейного программирования);
- симплекс-методом (задача линейного программирования).

Задание

Задана платежная матрица прямоугольной игры с нулевой суммой (на рисунке 1 приведена матрица по варианту).

4	-3	5	6	4
6	5	-3	4	7
6	5	-3	-3	5
-3	-3	4	4	4
7	6	4	5	6

Рисунок 1 – Матрица игры с нулевой суммой для варианта 1

1. Нормализовать матрицу (привести к матрице с неотрицательными элементами) и свести исходную игру к матричной игре размерности 2×2 следующими способами:
    1. путем поглощения доминируемых стратегий;
    2. путем удаления NBR-стратегий.
2. Найти смешанные стратегии игроков следующими методами:
    1. графоаналитическим;
    2. аналитическим (матричным);
    3. графически метод (задача линейного программирования);
    4. симплекс-методом (задача линейного программирования).
3. Рассчитать цену игры для исходной матрицы.

Ход работы

Для реализации решения расчётно-графического домашнего задания был использован язык программирования Python. К защите представляется интерактивный блокнот Jupyter Notebook в файле rgz.ipynb (см. приложение А).

В исполняемом «ноутбуке» импортируются реализованные модули *matrix_games* и *simplex*, инкапсулирующие логику и алгоритмы для матричных игр и симплекс-метода соответственно. Ознакомиться со всем исходным кодом данной работы можно, посетив репозиторий, ссылка на который представлена в приложении Б. Инициализация матрицы представлена на рисунке 2. Нижняя и верхняя цены игры: 4 и 5. Седловой точки нет. Для идентификации назовём игроков A и B.

```
[1]: import json
     import logging
     from pathlib import Path

     import matplotlib.pyplot as plt
     import numpy as np

     from game_theory.utils.matrix_games.analytical import AnalyticalSolver
     from game_theory.utils.matrix_games.game_matrix import GameMatrix
     from game_theory.utils.matrix_games.mixed_strategies import (
         check_resulted_game_price,
         get_resulted_mixed_strategies,
     )
     from game_theory.utils.simplex.dual_problem import DualProblem
     from game_theory.utils.simplex.simplex_problem import SimplexProblem

     logging.basicConfig(level=logging.INFO, format='%(message)s')
```

```
[2]: # Входная матрица прямоугольной игры с нулевой суммой.
     matrix = np.array(
         [
             [4, -3, 5, 6, 4],
             [6, 5, -3, 4, 7],
             [6, 5, -3, -3, 5],
             [-3, -3, 4, 4, 4],
             [7, 6, 4, 5, 6],
         ],
         dtype=int,
     )

     game_matrix = GameMatrix(matrix)
     game_matrix
```

```
[2]: +------------------------------------------------------------+
     |                  Таблица стратегий (игрока A)              |
     +----------------+----+----+----+----+----+------------------+
     |   Стратегии    | b1 | b2 | b3 | b4 | b5 | MIN выигрыш A    |
     +----------------+----+----+----+----+----+------------------+
     |       a1       | 4  | -3 | 5  | 6  | 4  |        -3        |
     |       a2       | 6  | 5  | -3 | 4  | 7  |        -3        |
     |       a3       | 6  | 5  | -3 | -3 | 5  |        -3        |
     |       a4       | -3 | -3 | 4  | 4  | 4  |        -3        |
     |       a5       | 7  | 6  | 4  | 5  | 6  |        4         |
     | MAX проигрыш B | 7  | 6  | 5  | 6  | 7  |                  |
     +----------------+----+----+----+----+----+------------------+
```

Рисунок 2 – Задание исходной матрицы игры с нулевой суммой

1. Нормализация матрицы

Для нормализации матрицы достаточно прибавить ко всем элементам матрицы максимальное по модулю отрицательное число (если оно существует). В нашем случае это слагаемое 3 (см. рисунок 3). Нижняя и верхняя цены игры: 7 и 8.

1. Нормализация матрицы. Уменьшение размерности исходной матричной игры

```
[4]: normalizer: int = game_matrix.normalize_matrix()
     game_matrix
```

```
Прибавили ко всем элементам исходной матрицы 3
```

```
[4]: +---------------------------------------------------------+
     |                  Таблица стратегий (игрока A)           |
     +---------------+----+----+----+----+----+----------------+
     |   Стратегии   | b1 | b2 | b3 | b4 | b5 | MIN выигрыш A  |
     +---------------+----+----+----+----+----+----------------+
     |       a1      | 7  | 0  | 8  | 9  | 7  |       0        |
     |       a2      | 9  | 8  | 0  | 7  | 10 |       0        |
     |       a3      | 9  | 8  | 0  | 0  | 8  |       0        |
     |       a4      | 0  | 0  | 7  | 7  | 7  |       0        |
     |       a5      | 10 | 9  | 7  | 8  | 9  |       7        |
     | MAX проигрыш B | 10 | 9  | 8  | 9  | 10 |                |
     +---------------+----+----+----+----+----+----------------+
```

```
[5]: print(f"Нижняя цена игры:  {game_matrix.lowest_game_price[1]}\n"
           f"Верхняя цена игры: {game_matrix.highest_game_price[1]}")
```

```
Нижняя цена игры:  7
Верхняя цена игры: 8
```

Рисунок 3 – Нормализация прямоугольной матрицы игры

2. Уменьшение размерности матричной игры

Для упрощения, сведём матричную игру 5x5 к матричной игре 2x2 двумя способами: поглощением доминируемых стратегий и удалением NBR-стратегий (Never Best Response). Но для начала следует проверить наличие дублирующихся стратегий (их тоже следует удалить). В нашем случае таких не нашлось, так что сразу переходим к основным методом уменьшения размерности.

Доминирующая строка (столбец) содержит элементы, поразрядно не ме́ньшие (не бо́льшие) элементов доминируемой строки (столбца). Итеративным обходом таблицы стратегий поглощаются стратегии a_2, a_3, a_4, b_1, b_4, b_5. (см. рисунок 4).

```
[6]: dominant_reduced_game: GameMatrix = game_matrix.reduce_dimension(method='dominant_absorption')
     dominant_reduced_game
```

```
a2 > a3: поглощение стратегии a3 доминирующей стратегией a2
a1 > a4: поглощение стратегии a4 доминирующей стратегией a1
b2 > b1: поглощение стратегии b1 доминирующей стратегией b2
b3 > b4: поглощение стратегии b4 доминирующей стратегией b3
b2 > b5: поглощение стратегии b5 доминирующей стратегией b2
a5 > a2: поглощение стратегии a2 доминирующей стратегией a5
```

```
[6]: +-----------------------------------------+
     |       Таблица стратегий (игрока A)      |
     +----------------+----+----+--------------+
     |   Стратегии    | b2 | b3 | MIN выигрыш A |
     +----------------+----+----+--------------+
     |       a1       | 0  | 8  |       0      |
     |       a5       | 9  | 7  |       7      |
     | MAX проигрыш B | 9  | 8  |              |
     +----------------+----+----+--------------+
```

```
[7]: print(f"Нижняя цена игры:  {dominant_reduced_game.lowest_game_price[1]}\n"
           f"Верхняя цена игры: {dominant_reduced_game.highest_game_price[1]}")

     Нижняя цена игры:  7
     Верхняя цена игры: 8
```

Рисунок 4 – Снижение размерности методом доминируемых стратегий

Аналогичное проделаем методом удаление NBR-стратегий. NBR-строкой (столбцом) назовём такую строку (столбец), которая объективно не будет разыгрываться игроком A (B) для всех наперёд фиксированных стратегий B (A). На рисунке 5 легко убедиться, что алгоритм, реализующий данный метод, даёт на выходе ту же матрицу игры размера 2x2.

```
[8]: # Вычеркиваем столбцы и строки, которые мы точно не выберем при фиксированной стратегии.
     nbr_reduced_game: GameMatrix = game_matrix.reduce_dimension(method='nbr_drop')
     nbr_reduced_game
```

```
Удаление NBR-стратегий ['a3', 'a4']
Удаление NBR-стратегий ['b1', 'b4', 'b5']
Удаление NBR-стратегий ['a2']
```

```
[8]: +-----------------------------------------+
     |       Таблица стратегий (игрока A)      |
     +----------------+----+----+--------------+
     |   Стратегии    | b2 | b3 | MIN выигрыш A |
     +----------------+----+----+--------------+
     |       a1       | 0  | 8  |       0      |
     |       a5       | 9  | 7  |       7      |
     | MAX проигрыш B | 9  | 8  |              |
     +----------------+----+----+--------------+
```

```
[9]: print(f"Нижняя цена игры:  {nbr_reduced_game.lowest_game_price[1]}\n"
           f"Верхняя цена игры: {nbr_reduced_game.highest_game_price[1]}")

     Нижняя цена игры:  7
     Верхняя цена игры: 8
```

```
[10]: assert dominant_reduced_game == nbr_reduced_game
      reduced_game: GameMatrix = nbr_reduced_game
```

Рисунок 5 – Снижение размерности удалением NBR-стратегий

3. Графоаналитический метод

Пусть x_1 – вероятность выбора игроком А стратегии a_1. $x_5 = 1 - x_1$ – вероятность выбора игроком А стратегии a_5.

Ожидаемый выигрыш А при реализации стратегии b_2:

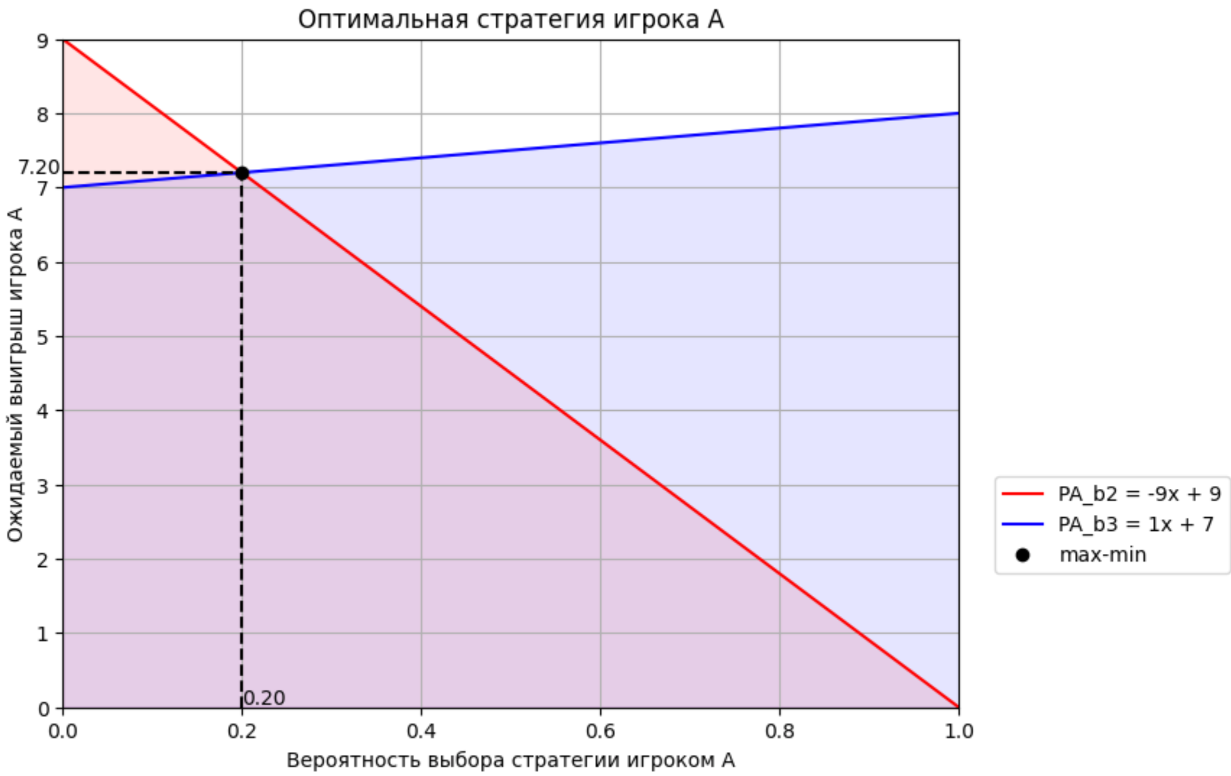
$$PA_{b2} = c_{12}x_1 + c_{52}x_5 = (c_{12} - c_{52})x_5 + c_{52}$$

Ожидаемый выигрыш А при реализации стратегии b_3:

$$PA_{b3} = c_{13}x_1 + c_{53}x_5 = (c_{13} - c_{53})x_5 + c_{53}$$

Оптимальная стратегия А: $PA_{b2} = PA_{b3}$.

На рисунке 6 найдено решение для оптимальной стратегии игрока А. Таким образом смешанные стратегии игрока А: *(0,2; 0; 0; 0; 0,8)*. Цена игры: *7,2*.




```
# Смешанные стратегии игрока А и цена игры. •••

Цена игры: 7 <= 7.200 <= 8
+-------------------------------------+
|    Смешанные стратегии игрока А     |
+-------+-------+-------+-------+-------+
|  a1   |  a2   |  a3   |  a4   |  a5   |
+-------+-------+-------+-------+-------+
| 0.20  | 0.00  | 0.00  | 0.00  | 0.80  |
+-------+-------+-------+-------+-------+
```

Рисунок 6 – Графоаналитический метод поиска смешанных стратегий игрока А

Пусть y_2 – вероятность выбора игроком B стратегии b_2. $y_3 = 1 - y_2$ – вероятность выбора игроком B стратегии b_3.

Ожидаемый проигрыш B при реализации стратегии a_1:

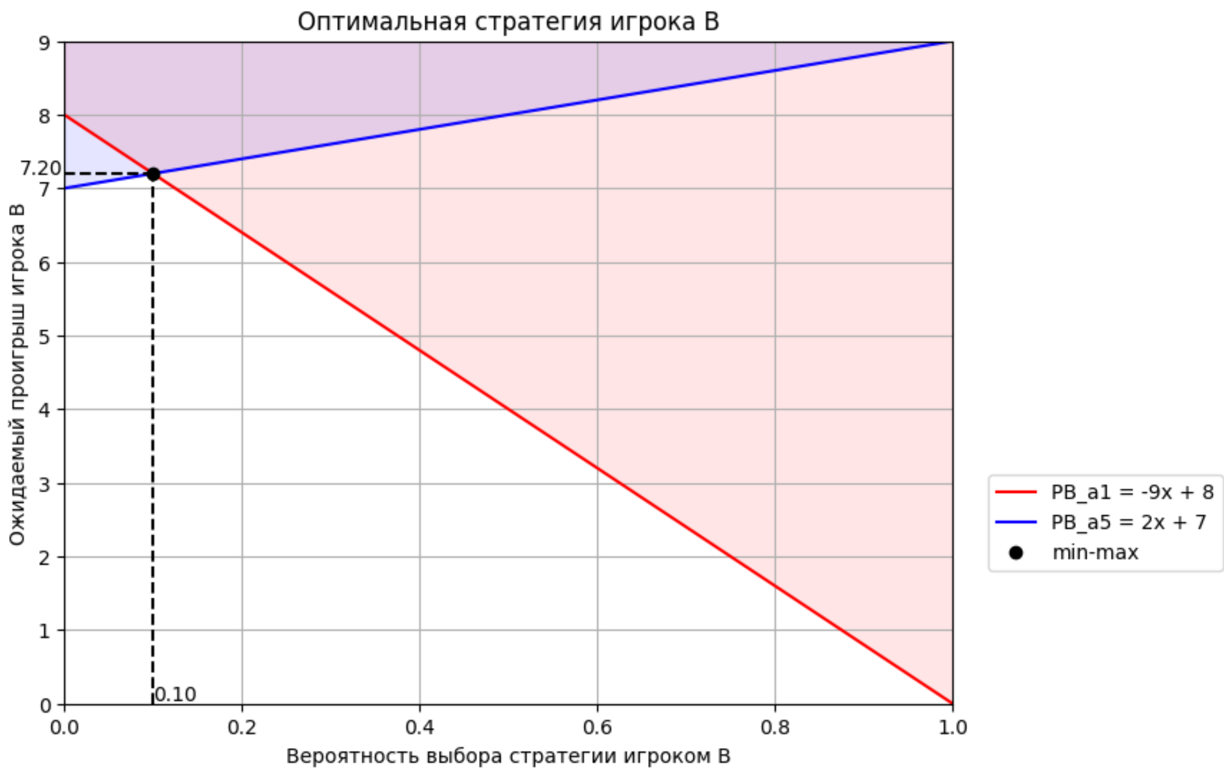
$$PB_{a1} = c_{12}y_2 + c_{13}y_3 = (c_{12} - c_{13})y_2 + c_{13}$$

Ожидаемый проигрыш B при реализации стратегии a_5:

$$PB_{a5} = c_{52}y_2 + c_{53}y_3 = (c_{52} - c_{53})y_2 + c_{53}$$

Оптимальная стратегия B: $PB_{a1} = PB_{a5}$.

На рисунке 7 найдено решение для оптимальной стратегии игрока B. Таким образом смешанные стратегии игрока B: *(0; 0,1; 0,9; 0; 0)*. Цена игры: *7,2*.




```
# Смешанные стратегии игрока B и цена игры. •••

Цена игры: 7 <= 7.200 <= 8
+---------------------------------------+
|     Смешанные стратегии игрока B      |
+-------+-------+-------+-------+-------+
|  b1   |  b2   |  b3   |  b4   |  b5   |
+-------+-------+-------+-------+-------+
| 0.00  | 0.10  | 0.90  | 0.00  | 0.00  |
+-------+-------+-------+-------+-------+
```

Рисунок 7 – Графоаналитический метод поиска смешанных стратегий игрока B

4. Аналитический (матричный) метод

Для игрока А (h – цена игры; $y_1, \dots, y_m$ – смешанные стратегии игрока А) задача сводится к решению следующей СЛАУ матричным методом:

$$\begin{cases} c_{11}y_1 & + & \dots & + & c_{m1}y_m & = & h \\ c_{12}y_1 & + & \dots & + & c_{m2}y_m & = & h \\ \dots & & \dots & & & & \dots \\ c_{1n}y_1 & + & \dots & + & c_{mn}y_m & = & h \\ y_1 & + & \dots & + & y_m & = & 1 \end{cases}$$

$$\begin{cases} c_{11}y_1 & + & \dots & + & c_{m1}y_m & - & h & = & 0 \\ c_{12}y_1 & + & \dots & + & c_{m2}y_m & - & h & = & 0 \\ \dots & & \dots & & & & \dots & & \\ c_{1n}y_1 & + & \dots & + & c_{mn}y_m & - & h & = & 0 \\ y_1 & + & \dots & + & y_m & & & = & 1 \end{cases}$$

$$\begin{pmatrix} c_{11} & \dots & c_{m1} & -1 \\ c_{12} & \dots & c_{m2} & -1 \\ \dots & \dots & \dots & \dots \\ c_{1n} & \dots & c_{mn} & -1 \\ 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_m \\ h \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_m \\ h \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{m1} & -1 \\ c_{12} & \dots & c_{m2} & -1 \\ \dots & \dots & \dots & \dots \\ c_{1n} & \dots & c_{mn} & -1 \\ 1 & \dots & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{pmatrix}.$$

Для игрока В (g – цена игры (h=g); $x_1, \dots, x_n$ – смешанные стратегии игрока А) задача сводится к решению следующей аналогичной СЛАУ (с точностью до транспонирования матрицы игры):

$$\begin{cases} c_{11}x_1 & + & \dots & + & c_{1n}x_n & = & g \\ c_{21}x_1 & + & \dots & + & c_{2n}x_n & = & g \\ \dots & & \dots & & & & \dots \\ c_{m1}x_1 & + & \dots & + & c_{mn}x_n & = & g \\ x_1 & + & \dots & + & x_n & = & 1 \end{cases}$$

$$\dots$$

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \\ g \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1n} & -1 \\ c_{21} & \dots & c_{2n} & -1 \\ \dots & \dots & \dots & \dots \\ c_{m1} & \dots & c_{mn} & -1 \\ 1 & \dots & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{pmatrix}.$$

На рисунках 8 и 9 представлены решения матричной игры в смешанных стратегиях для игроков А и В соответственно.

```
[17]: analytical_solver = AnalyticalSolver(reduced_game)
      first_mixed_strategy, second_mixed_strategy, game_price_value = analytical_solver.player_a_solve()

      # Смешанные стратегии игрока А и цена игры. •••

      Цена игры: 7 <= 7.200 <= 8
      +-----------------------------------+
      |    Смешанные стратегии игрока А   |
      +------+------+------+------+------+
      |  a1  |  a2  |  a3  |  a4  |  a5  |
      +------+------+------+------+------+
      | 0.20 | 0.00 | 0.00 | 0.00 | 0.80 |
      +------+------+------+------+------+
```

Рисунок 8 – Аналитический метод поиска смешанных стратегий игрока А

```
[19]: analytical_solver = AnalyticalSolver(reduced_game)
      first_mixed_strategy, second_mixed_strategy, game_price_value = analytical_solver.player_b_solve()

      # Смешанные стратегии игрока B и цена игры. •••

      Цена игры: 7 <= 7.200 <= 8
      +-----------------------------------+
      |    Смешанные стратегии игрока B   |
      +------+------+------+------+------+
      |  b1  |  b2  |  b3  |  b4  |  b5  |
      +------+------+------+------+------+
      | 0.00 | 0.10 | 0.90 | 0.00 | 0.00 |
      +------+------+------+------+------+
```

Рисунок 9 – Аналитический метод поиска смешанных стратегий игрока В

Полученные цена игры и смешанные стратегии совпали со значениями, полученными предыдущим методом: *(0,2; 0; 0; 0; 0,8)* для игрока А и *(0; 0,1; 0,9; 0; 0)* – для игрока В; цена игры – *7,2*.

5. Графический метод (задача ЛП)

Задача линейного программирования (далее – ЗЛП) формулируется для матричной игры так, как показано на рисунке 10. Таким образом для игрока А будем решать двойственную задачу (ДЗ ЛП), а для В – прямую (ПЗ ЛП).

Задача линейного программирования с целевой функцией

$$F(\boldsymbol{\xi}) = \sum_{i=1}^{n} \xi_i \to \mathrm{MIN}$$

так как необходимо

$$\omega \to \mathrm{MAX}$$

$$\begin{cases} c_{11}q_1 + c_{12}q_2 + \ldots + c_{1n}q_n \ge \omega \\ c_{21}q_1 + c_{22}q_2 + \ldots + c_{2n}q_n \ge \omega \\ \ldots \\ c_{m1}q_1 + c_{m2}q_2 + \ldots + c_{mn}q_n \ge \omega \end{cases}$$

где
ω – гарантированный MIN выигрыш

Пусть $\xi_1 = \dfrac{q_1}{\omega}, \quad \xi_2 = \dfrac{q_2}{\omega}, \quad \ldots \quad \xi_n = \dfrac{q_n}{\omega}$

тогда

$$\begin{cases} c_{11}\xi_1 + c_{12}\xi_2 + \ldots + c_{1n}\xi_n \ge 1 \\ c_{21}\xi_1 + c_{22}\xi_2 + \ldots + c_{2n}\xi_n \ge 1 \\ \ldots \\ c_{m1}\xi_1 + c_{m2}\xi_2 + \ldots + c_{mn}\xi_n \ge 1 \end{cases}$$

где $\xi_i \ge 0, \quad \sum_{i=1}^{n} \xi_i = \dfrac{1}{\omega}$

Рисунок 10 – Сведение матричной игры с нулевой суммой к ЗЛП

Для игрока А определим полуплоскости на основе следующих ограничений:

$$\Phi = y_1 + y_2 \rightarrow min$$

$$\begin{cases} & & 9y_2 & \geq & 1 \\ 8y_1 & + & 7y_2 & \geq & 1 \end{cases}$$

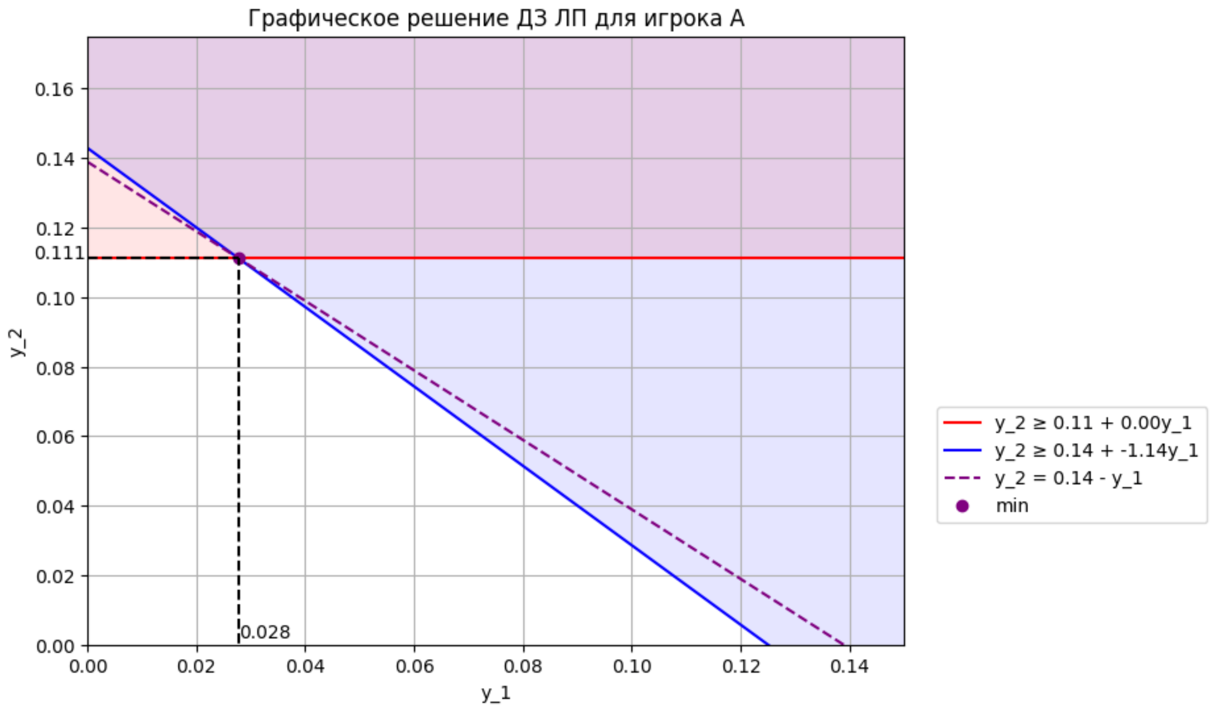
$$y_1, y_2 \geq 0$$

.

В системе координат Oy_1y_2 были построены полуплоскости ограничений и найдена точка (0,028; 0,111), через которую проходит линия уровня:

$$y_2 = \Phi - y_1 \text{ при } \Phi = 0{,}028 + 0{,}111 \approx 0{,}14.$$

Чтобы из полученных значений восстановить цену игры, произведём действия, обратные тем, что были выполнены при сведении матричной игры к ЗЛП: $h = {}^{1}/_{\Phi}$ – цена игры, $(h \cdot y_1; 0; 0; 0; h \cdot y_2)$ – смешанные стратегии игрока А (см. рисунок 11).




```
# Смешанные стратегии игрока А и цена игры.
Цена игры: 7 <= 7.200 <= 8
+--------------------------------------+
|     Смешанные стратегии игрока А     |
+-------+-------+-------+-------+------+
|  a1   |  a2   |  a3   |  a4   |  a5  |
+-------+-------+-------+-------+------+
| 0.20  | 0.00  | 0.00  | 0.00  | 0.80 |
+-------+-------+-------+-------+------+
```

Рисунок 11 – Графическое решение ДЗ ЛП для игрока А

Для игрока B аналогично определим полуплоскости на основе следующих ограничений:

$$F = x_1 + x_2 \to max$$

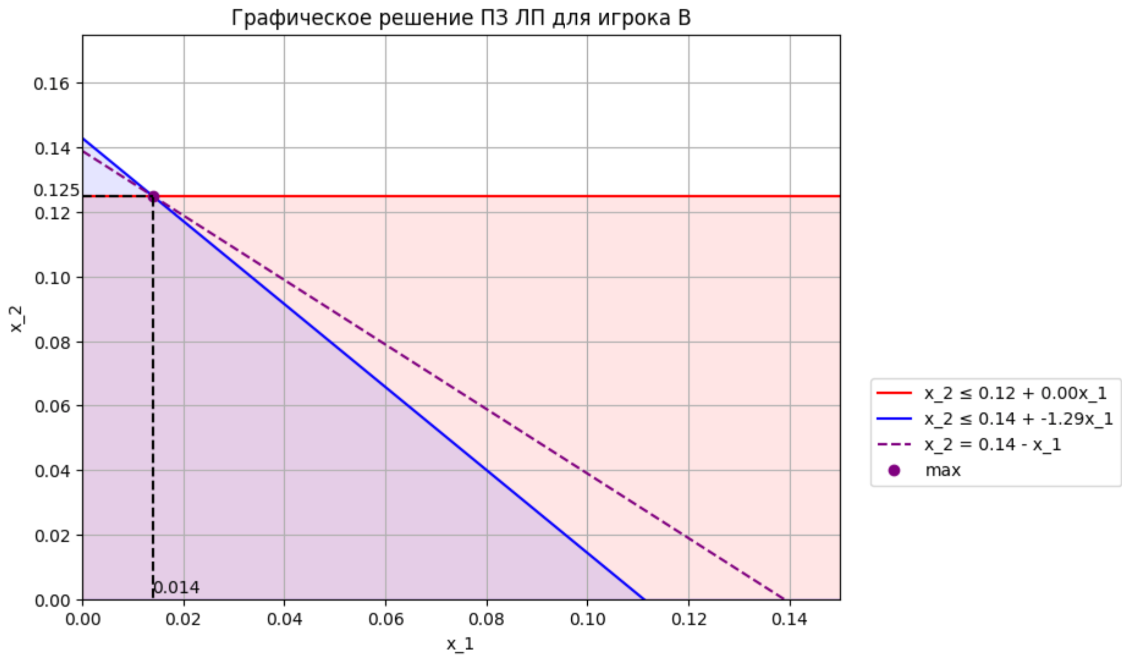
$$\begin{cases} & & 8x_2 & \leq & 1 \\ 9x_1 & + & 7y_2 & \leq & 1 \\ \end{cases}$$

$$x_1, x_2 \geq 0$$

В системе координат Ox_1x_2 были построены полуплоскости ограничений и найдена точка (0,014; 0,125), через которую проходит линия уровня:

$$x_2 = F - x_1 \text{ при } F = 0{,}014 + 0{,}125 \approx 0{,}14.$$

Чтобы из полученных значений восстановить цену игры, произведём действия, обратные тем, что были выполнены при сведении матричной игры к ЗЛП: $g = {}^1/_F$ – цена игры, $(0; g \cdot x_1; g \cdot x_2; 0; 0)$ – смешанные стратегии игрока B (см. рисунок 12).




```
# Смешанные стратегии игрока В и цена игры. •••

Цена игры: 7 <= 7.200 <= 8
+----------------------------------+
|   Смешанные стратегии игрока В   |
+------+------+------+------+------+
|  b1  |  b2  |  b3  |  b4  |  b5  |
+------+------+------+------+------+
| 0.00 | 0.10 | 0.90 | 0.00 | 0.00 |
+------+------+------+------+------+
```

Рисунок 12 – Графическое решение ПЗ ЛП для игрока B

Полученные цена игры и смешанные стратегии совпали со значениями, полученными предыдущими методами: (0,2; 0; 0; 0; 0,8) для игрока А и (0; 0,1; 0,9; 0; 0) – для игрока В; цена игры – 7,2.

6. Симплекс-метод (задача ЛП)

Прямая и двойственная задачи ЛП уже были сформулированы в предыдущем пункте. Осталось применить уже известный симплекс-метод для решения задач ДЗ и ПЗ ЛП для игроков А и В соответственно.

Решение для игрока А представлено на рисунке 13. Из полученных значений по аналогии с предыдущим методом получаем цену игры $h = {}^{1}/_{\Phi}$ и смешанные стратегии игрока А $(h \cdot y_1; 0; 0; 0; h \cdot y_2)$.

2.4.1. Двойственная задача ЛП для игрока А

```
[28]: player_a_problem = DualProblem(input_path)
```

```
F = b^T·x -> min,
A^T·x >= c^T,
x1, x2, ..., xn >= 0
b^T = [1 1]
A^T =
[[0 9]
 [8 7]],
c^T = [1 1].
```

```
[29]: player_a_solution = player_a_problem.solve()
      player_a_var_values, player_a_target_function_value = player_a_solution
```

```
Процесс решения:
Поиск опорного решения:
Исходная симплекс-таблица:
+----+---------+---------+---------+
|    |   Si0   |    x1   |    x2   |
+----+---------+---------+---------+
| x3 | -1.0000 |  0.0000 | -9.0000 |
| x4 | -1.0000 | -8.0000 | -7.0000 |
| F  |  0.0000 | -1.0000 | -1.0000 |
+----+---------+---------+---------+
Разрешающая строка: x3
Разрешающий столбец: x2
+----+---------+---------+---------+
|    |   Si0   |    x1   |    x3   |
+----+---------+---------+---------+
| x2 |  0.1111 | -0.0000 | -0.1111 |
| x4 | -0.2222 | -8.0000 | -0.7778 |
| F  |  0.1111 | -1.0000 | -0.1111 |
+----+---------+---------+---------+
Разрешающая строка: x4
Разрешающий столбец: x1
+----+--------+---------+---------+
|    |  Si0   |    x4   |    x3   |
+----+--------+---------+---------+
| x2 | 0.1111 | -0.0000 | -0.1111 |
| x1 | 0.0278 | -0.1250 |  0.0972 |
| F  | 0.1389 | -0.1250 | -0.0139 |
+----+--------+---------+---------+
Опорное решение найдено!
x4 = x3 = 0,
x2 = 0.111, x1 = 0.028
Целевая функция: F = 0.139
Поиск оптимального решения:
Оптимальное решение найдено!
x4 = x3 = 0,
x2 = 0.111, x1 = 0.028
Целевая функция: F = 0.139
```

Рисунок 13 – Решение ДЗ ЛП симплекс-методом для игрока А

Решение для игрока B представлено на рисунке 13. Из полученных значений по аналогии с предыдущим методом получаем цену игры $g = {}^{1}/_{F}$ и смешанные стратегии игрока B $(0; g \cdot x_1; g \cdot x_2; 0; 0)$.

2.4.2. Прямая задача ЛП для игрока B

```
[31]: player_b_problem = SimplexProblem(input_path)
      player_b_problem

[31]: F = c·x -> max,
      Ax <= b,
      x1,x2, ..., xn >= 0
      c^T = [-1 -1],
      A =
      [[0 8]
       [9 7]],
      b^T = [1 1].

[32]: player_b_solution = player_b_problem.solve()
      player_b_var_values, player_b_target_function_value = player_b_solution

Процесс решения:
Поиск опорного решения:
Исходная симплекс-таблица:
+----+--------+--------+--------+
|    |  Si0   |   x1   |   x2   |
+----+--------+--------+--------+
| x3 | 1.0000 | 0.0000 | 8.0000 |
| x4 | 1.0000 | 9.0000 | 7.0000 |
| F  | 0.0000 | 1.0000 | 1.0000 |
+----+--------+--------+--------+
Опорное решение найдено!
x1 = x2 = 0,
x3 = 1.000, x4 = 1.000
Целевая функция: F = 0.000
Поиск оптимального решения:
Разрешающая строка: x4
Разрешающий столбец: x1
+----+---------+---------+--------+
|    |   Si0   |    x4   |   x2   |
+----+---------+---------+--------+
| x3 |  1.0000 | -0.0000 | 8.0000 |
| x1 |  0.1111 |  0.1111 | 0.7778 |
| F  | -0.1111 | -0.1111 | 0.2222 |
+----+---------+---------+--------+
Разрешающая строка: x3
Разрешающий столбец: x2
+----+---------+---------+---------+
|    |   Si0   |    x4   |    x3   |
+----+---------+---------+---------+
| x2 |  0.1250 | -0.0000 |  0.1250 |
| x1 |  0.0139 |  0.1111 | -0.0972 |
| F  | -0.1389 | -0.1111 | -0.0278 |
+----+---------+---------+---------+
Оптимальное решение найдено!
x4 = x3 = 0,
x2 = 0.125, x1 = 0.014
Целевая функция: F = 0.139
```

Рисунок 14 – Решение ПЗ ЛП симплекс-методом для игрока B

Полученные цена игры и смешанные стратегии совпали со значениями, полученными предыдущими методами: (0,2; 0; 0; 0; 0,8) для игрока A и (0; 0,1; 0,9; 0; 0) – для игрока B; цена игры – 7,2.

7. Расчёт цены игры исходной матрицы

Вернёмся к исходной матричной игре, вычитанием скаляра 3, который мы добавляли ко всем элементам матрицы для нормализации (см. рисунок 15). Смешанные стратегии игроков останутся неизменными (как вероятности), а итоговая цена игры уменьшится:

$$g - 3 = h - 3 = 7{,}2 - 3 = 4{,}2.$$

Задача решена.

3. Расчёт цены игры исходной матрицы

Вспомним изначальную заданную матрицу игры, до нормализации и найдём её цену игры и смешанные стратегии игроков.

```
# Вычитаем ранее добавленное слагаемое нормализации.
game_matrix.matrix -= normalizer
game_matrix
```

```
+-----------------------------------------------------------+
|               Таблица стратегий (игрока A)                |
+----------------+----+----+----+----+----+----------------+
|   Стратегии    | b1 | b2 | b3 | b4 | b5 | MIN выигрыш A  |
+----------------+----+----+----+----+----+----------------+
|       a1       | 4  | -3 | 5  | 6  | 4  |       -3       |
|       a2       | 6  | 5  | -3 | 4  | 7  |       -3       |
|       a3       | 6  | 5  | -3 | -3 | 5  |       -3       |
|       a4       | -3 | -3 | 4  | 4  | 4  |       -3       |
|       a5       | 7  | 6  | 4  | 5  | 6  |       4        |
| MAX проигрыш B | 7  | 6  | 5  | 6  | 7  |                |
+----------------+----+----+----+----+----+----------------+
```

```
print(f"Нижняя цена игры:  {game_matrix.lowest_game_price[1]}\n"
      f"Верхняя цена игры: {game_matrix.highest_game_price[1]}")
```

```
Нижняя цена игры:  4
Верхняя цена игры: 5
```

```
original_game_price: float = game_price_value - normalizer
print(f"Цена игры исходной матрицы: {original_game_price:.2f}")
```

```
Цена игры исходной матрицы: 4.20
```

Рисунок 15 – Нахождение цены игры исходной матричной игры

Вывод

В данном расчётно-графическом задании была исследована антагонистическая игра двух лиц с нулевой суммой. Для этого матрица 5x5 игроков А и В была нормализована и сведена к игре меньшей размерности (2x2) методами поглощения доминируемых стратегий и удалением NBR-стратегий (полученные матрицы совпали).

Далее для полученной матричной игры 2x2 были реализованы методы для нахождения смешанных стратегий обоих игроков. Примечательно, что графоаналитический и графический (ЛП) методы имели хорошую наглядность, что в частном случае будет менее предпочтительно в пространствах высших размерностей. Универсальными показали себя аналитический (матричный) и симплекс- методы.

В конечном итоге была найдена цена игры исходной матричной игры. Частоты выбора стратегий игроками А и В не претерпели изменений, что согласуется с теорией. В итоге была получена цена игры 4,2 и смешанные стратегии игроков:

(0,2; 0; 0; 0; 0,8) для игрока А;

(0; 0,1; 0,9; 0; 0) – для игрока В.

ПРИЛОЖЕНИЕ А
Листинг интерактивного блокнота rgz.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # Расчетно-графическое домашнее задание
#
# **Выполнил: Александров А. Н., ИУ8-104**
#
# **Вариант: 1**
#
# ## Задание
# Задана платежная матрица прямоугольной игры с нулевой суммой.
#
#
# | 4  | -3 | 5  | 6  | 4 |
# |:--:|:--:|:--:|:--:|:-:|
# | 6  | 5  | -3 | 4  | 7 |
# | 6  | 5  | -3 | -3 | 5 |
# | -3 | -3 | 4  | 4  | 4 |
# | 7  | 6  | 4  | 5  | 6 |
#
#
# 1. Нормализовать матрицу (привести к матрице с неотрицательными элементами) и свести исходную игру к матричной игре 2×2 следующими способами:
#     - [x] поглощением доминируемых стратегий;
#     - [x] удалением NBR-стратегий (Never Best Response).
# 2. Найти смешанные стратегии игроков следующими методами:
#     - [x] графоаналитическим;
#     - [x] аналитическим (матричным);
#     - [x] графически (задача ЛП);
#     - [x] симплекс-методом (задача ЛП).
# 3. Рассчитать цену игры для исходной матрицы.

# In[1]:


import json
import logging
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np

from game_theory.utils.matrix_games.analytical import AnalyticalSolver
from game_theory.utils.matrix_games.game_matrix import GameMatrix
from game_theory.utils.matrix_games.mixed_strategies import (
    check_resulted_game_price,
    get_resulted_mixed_strategies,
)
```

```python
from game_theory.utils.simplex.dual_problem import DualProblem
from game_theory.utils.simplex.simplex_problem import SimplexProblem

logging.basicConfig(level=logging.INFO, format='%(message)s')


# In[2]:


# Входная матрица прямоугольной игры с нулевой суммой.
matrix = np.array(
    [
        [4, -3, 5, 6, 4],
        [6, 5, -3, 4, 7],
        [6, 5, -3, -3, 5],
        [-3, -3, 4, 4, 4],
        [7, 6, 4, 5, 6],
    ],
    dtype=int,
)

game_matrix = GameMatrix(matrix)
game_matrix


# In[3]:


print(f"Нижняя цена игры:  {game_matrix.lowest_game_price[1]}\n"
      f"Верхняя цена игры: {game_matrix.highest_game_price[1]}")


# ## 1. Нормализация матрицы. Уменьшение размерности исходной матричной игры

# In[4]:


normalizer: int = game_matrix.normalize_matrix()
game_matrix


# In[5]:


print(f"Нижняя цена игры:  {game_matrix.lowest_game_price[1]}\n"
      f"Верхняя цена игры: {game_matrix.highest_game_price[1]}")


# ### 1.1. Поглощение доминируемых стратегий
# <u>Опр. **Доминирующая (поглощающая) строка**</u> содержит элементы $\geq$ элементам другой строки (поглощаемой).
#
```

```
# <u>Опр. **Доминирующий (поглощающая) столбец**</u> содержит элементы $\leq$ элементам другого столбца (поглощаемого).

# In[6]:


dominant_reduced_game: GameMatrix = game_matrix.reduce_dimension(method='dominant_absorption')
dominant_reduced_game


# In[7]:


print(f"Нижняя цена игры:  {dominant_reduced_game.lowest_game_price[1]}\n"
      f"Верхняя цена игры: {dominant_reduced_game.highest_game_price[1]}")


# ### 1.2. Удаление NBR-стратегий
#
# <u>Опр. **NBR-строка**</u> - строка, которая объективно не будет разыгрываться игроком А для всех наперёд фиксированных стратегий В.
#
# <u>Опр. **NBR-столбец**</u> - столбец, который объективно не будет разыгрываться игроком В для всех наперёд фиксированных стратегий А.

# In[8]:


# Вычеркиваем столбцы и строки, которые мы точно не выберем при фиксированной стратегии.
nbr_reduced_game: GameMatrix = game_matrix.reduce_dimension(method='nbr_drop')
nbr_reduced_game


# In[9]:


print(f"Нижняя цена игры:  {nbr_reduced_game.lowest_game_price[1]}\n"
      f"Верхняя цена игры: {nbr_reduced_game.highest_game_price[1]}")


# In[10]:


assert dominant_reduced_game == nbr_reduced_game
reduced_game: GameMatrix = nbr_reduced_game


# ## 2. Нахождение смешанных стратегий и цены игры

# ### 2.1. Графоаналитический метод
```

```
# #### 2.1.1. Для игрока A
# Пусть
# - $x_1$ - вероятность выбора игроком A стратегии $a_1$.
# - $x_5 = 1 - x_1$ - вероятность выбора игроком A стратегии $a_5$.
#
# Ожидаемый выигрыш A при реализации стратегии $b_2$:
#
# $PA_{b2} = c_{12}x_{1} + c_{52}x_5 = (c_{12} - c_{52})x_5 + c_{52}$
#
# Ожидаемый выигрыш A при реализации стратегии $b_3$:
#
# $PA_{b3} = c_{13}x_{1} + c_{53}x_5 = (c_{13} - c_{53})x_5 + c_{53}$
#
# Оптимальная стратегия A: $PA_{b2} = PA_{b3}$

# In[11]:


assert reduced_game.matrix.shape == (2, 2)

(a, b), (c, d) = reduced_game.matrix.tolist()
PA_b_first = lambda x: (a - c) * x + c
PA_b_second = lambda x: (b - d) * x + d

# Находим точку пересечения решая систему Ax = b.
PA_A = np.array([
    [-(a - c), 1],
    [-(b - d), 1],
])
PA_b = np.array([c, d])
(x_intersect, y_intersect) = np.linalg.solve(PA_A, PA_b)


# In[12]:


# Отрисовка графиков пересечения.
X = np.linspace(0, 1, 50)
PA_b_first_y = PA_b_first(X)
PA_b_second_y = PA_b_second(X)
max_PA_b_y = np.max(np.concatenate((PA_b_first_y, PA_b_second_y)))

plt.figure(figsize=(8, 6))
plt.title("Оптимальная стратегия игрока A")

PA_b_first_label, PA_b_second_label = reduced_game.player_b_strategy_labels
plt.plot(X, PA_b_first_y, label=f"PA_{PA_b_first_label} = {a - c}x + {c}", color="red")
plt.plot(X, PA_b_second_y, label=f"PA_{PA_b_second_label} = {b - d}x + {d}", color="blue")

```

```
# Точка пересечения.
plt.plot(x_intersect, y_intersect, "o", color="black", label="max-min")
# Проекции точки пересечения на оси.
plt.vlines(x_intersect, min(PA_b_first_y), y_intersect, color="black",
 linestyles='dashed')
plt.hlines(y_intersect, min(X), x_intersect, color="black", linestyles='dashed')
# Ограничение [0, 1] - вероятность.
plt.xlim(0, 1)
# Ограничение на нормализованные элементы матрицы.
plt.ylim(0, max_PA_b_y)
# Подписи осей.
plt.xlabel("Вероятность выбора стратегии игроком A")
plt.ylabel("Ожидаемый выигрыш игрока A")
# Сегменты под графиками.
plt.fill_between(X, PA_b_first_y, color='red', alpha=0.1)
plt.fill_between(X, PA_b_second_y, color='blue', alpha=0.1)
# Отображение значений координат точки пересечения на осях
plt.text(x_intersect, 0.05, f'{x_intersect:.2f}')
plt.text(-0.05, y_intersect, f'{y_intersect:.2f}')

plt.legend(loc=(1.04, 0.2))
plt.grid(True)


# In[13]:


# Смешанные стратегии игрока A и цена игры.
assert check_resulted_game_price(reduced_game, y_intersect)
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_a_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_a_strategy_labels,
        [x_intersect, 1 - x_intersect],
    )),
    player_name="A",
)
print(mixed_strategies)


# #### 2.1.2. Для игрока B
# Пусть
# - $y_2$ - вероятность выбора игроком B стратегии $b_2$.
# - $y_3 = 1 - y_2$ - вероятность выбора игроком B стратегии $b_3$.
#
# Ожидаемый проигрыш B при реализации стратегии $a_1$:
#
# $PB_{a1} = c_{12}y_2 + c_{13}y_3 = (c_{12} - c_{13})y_2 + c_{13}$
#
# Ожидаемый проигрыш B при реализации стратегии $a_5$:
#
# $PB_{a5} = c_{52}y_2 + c_{53}y_3 = (c_{52} - c_{53})y_2 + c_{53}$
```

```
#
# Оптимальная стратегия B: $PB_{a1} = PB_{a5}$

# In[14]:


assert reduced_game.matrix.shape == (2, 2)

(a, b), (c, d) = reduced_game.matrix.tolist()
PB_a_first = lambda x: (a - b) * x + b
PB_a_second = lambda x: (c - d) * x + d

# Находим точку пересечения решая систему Ax = b.
PB_A = np.array([
    [-(a - b), 1],
    [-(c - d), 1],
])
PB_b = np.array([b, d])
(x_intersect, y_intersect) = np.linalg.solve(PB_A, PB_b)


# In[15]:


# Отрисовка графиков пересечения.
X = np.linspace(0, 1, 50)
PB_a_first_y = PB_a_first(X)
PB_a_second_y = PB_a_second(X)
max_PB_a_y = np.max(np.concatenate((PB_a_first_y, PB_a_second_y)))

plt.figure(figsize=(8, 6))
plt.title("Оптимальная стратегия игрока B")

PB_a_first_label, PB_a_second_label = reduced_game.player_a_strategy_labels
plt.plot(X, PB_a_first_y, label=f"PB_{PB_a_first_label} = {a - c}x + {b}", color="red")
plt.plot(X, PB_a_second_y, label=f"PB_{PB_a_second_label} = {c - d}x + {d}", color="blue")

# Точка пересечения.
plt.plot(x_intersect, y_intersect, "o", color="black", label="min-max")
# Проекции точки пересечения на оси.
plt.vlines(x_intersect, min(PB_a_first_y), y_intersect, color="black", linestyles='dashed')
plt.hlines(y_intersect, min(X), x_intersect, color="black", linestyles='dashed')
# Ограничение [0, 1] - вероятность.
plt.xlim(0, 1)
# Ограничение на нормализованные элементы матрицы.
plt.ylim(0, max_PB_a_y)
# Подписи осей.
plt.xlabel("Вероятность выбора стратегии игроком B")
plt.ylabel("Ожидаемый проигрыш игрока B")
```

```
# Сегменты над графиками.
plt.fill_between(X, PB_a_first_y, max_PB_a_y, color='red', alpha=0.1)
plt.fill_between(X, PB_a_second_y, max_PB_a_y, color='blue', alpha=0.1)
# Отображение значений координат точки пересечения на осях.
plt.text(x_intersect, 0.05, f'{x_intersect:.2f}')
plt.text(-0.05, y_intersect, f'{y_intersect:.2f}')

plt.legend(loc=(1.04, 0.2))
plt.grid(True)


# In[16]:


# Смешанные стратегии игрока B и цена игры.
assert check_resulted_game_price(reduced_game, y_intersect)
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_b_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_b_strategy_labels,
        [x_intersect, 1 - x_intersect],
    )),
    player_name="B",
)
print(mixed_strategies)


# ### 2.2. Аналитический (матричный) метод

# #### 2.2.1. Обратная матрица для игрока A
#
# Для игрока $A$ ($h$ - цена игры; $y_1, ..., y_m$ - смешанные стратегии игрока $A$):

# ![analytical_A](./img/analytical_A.png)

# In[17]:


analytical_solver = AnalyticalSolver(reduced_game)
first_mixed_strategy, second_mixed_strategy, game_price_value = analytical_solver.player_a_solve()


# In[18]:


# Смешанные стратегии игрока A и цена игры.
assert check_resulted_game_price(
    game_matrix=reduced_game,
    game_price_value=game_price_value,
)

```

```
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_a_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_a_strategy_labels,
        (first_mixed_strategy, second_mixed_strategy),
    )),
    player_name="A",
)
print(mixed_strategies)


# #### 2.2.1. Прямая матрица для игрока B
# Для игрока $B$ ($g$ - цена игры; $x_1, ..., y_n$ - смешанные стратегии игрока $B$):

# ![analytical_B](./img/analytical_B.png)

# In[19]:


analytical_solver = AnalyticalSolver(reduced_game)
first_mixed_strategy, second_mixed_strategy, game_price_value = analytical_solver.player_b_solve()


# In[20]:


# Смешанные стратегии игрока B и цена игры.
assert check_resulted_game_price(
    game_matrix=reduced_game,
    game_price_value=game_price_value,
)

mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_b_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_b_strategy_labels,
        (first_mixed_strategy, second_mixed_strategy),
    )),
    player_name="B",
)
print(mixed_strategies)


# ### 2.3. Графический метод (задача ЛП)
#
# <div style="text-align:left;">
#     <img src="img/LP_task_02.png" alt="LP_task_02" width="600" height="300">
# </div>
# <div style="text-align:left;">
#     <img src="img/system_matrix_game.png" alt="system_matrix_game" width="300" height="330">
```

```
#     <img src="img/LP_task_01.png" alt="LP_task_01" width="300" height="330">
# </div>

# In[21]:


# Подготовка входных данных ЗЛП.
n_rows, n_cols = reduced_game.matrix.shape
input_data = {
    # F = x1 + x2
    "obj_func_coffs": [1] * n_cols,
    # A - матрица сериализуется в массив массивов JSON.
    "constraint_system_lhs": reduced_game.matrix.tolist(),
    # b - вектор-столбец ограничений.
    "constraint_system_rhs": [1] * n_rows,
    # Экстремум, направление оптимизации функции.
    "func_direction": "max"
}

input_path = Path('input_LPP.json')
_ = input_path.write_text(json.dumps(input_data, indent=2))


# #### 2.3.1. Двойственная ЗЛП для игрока A
# ![LP_problem_A](./img/LP_problem_A.png)

# In[22]:


assert reduced_game.matrix.shape == (2, 2)

(a, b), (c, d) = reduced_game.matrix.tolist()
# Прямая для 1-го ограничения.
first_constraint = lambda y_1: (1 / c) - (a / c) * y_1
# Прямая для 2-го ограничения.
second_constraint = lambda y_1: (1 / d) - (b / d) * y_1
# Точка пересечения.
(y_1_intersect, y_2_intersect) = np.linalg.solve(
    np.array([
        [a, c],
        [b, d],
    ]),
    np.array([1, 1]),
)
# Выражаем целевую функцию через y_2: y_2 = F - y_1, где F -> min.
player_a_target_function_value = y_1_intersect + y_2_intersect
F_func = lambda y_1: (y_1_intersect + y_2_intersect) - y_1


# In[23]:


```

```
# Отрисовка графиков пересечения.
X = np.linspace(0, 1, 100)
first_constraint_y2 = first_constraint(X)
second_constraint_y2 = second_constraint(X)
F_y2 = F_func(X)
max_y2 = np.max(np.concatenate((first_constraint_y2, second_constraint_y2))) + 0.15

plt.figure(figsize=(8, 6))
plt.title("Графическое решение ДЗ ЛП для игрока А")

plt.plot(X, first_constraint_y2, label=f"y_2 ≥ {1 / c:.2f} + {- a / c:.2f}y_1", color="red")
plt.plot(X, second_constraint_y2, label=f"y_2 ≥ {1 / d:.2f} + {- b / d:.2f}y_1", color="blue")
plt.plot(X, F_y2, "--", label=f"y_2 = {y_1_intersect + y_2_intersect:.2f} - y_1", color="purple")

# Точка пересечения.
plt.plot(y_1_intersect, y_2_intersect, "o", color="purple", label="min")
# Проекции точки пересечения на оси.
plt.vlines(y_1_intersect, min(second_constraint_y2), y_2_intersect, color="black", linestyles='dashed')
plt.hlines(y_2_intersect, min(X), y_1_intersect, color="black", linestyles='dashed')
# Ограничения на неотрицательные решения.
plt.xlim(0, 0.15)
plt.ylim(0, 0.175)
# Подписи осей.
plt.xlabel("y_1")
plt.ylabel("y_2")
# Сегменты над графиками.
plt.fill_between(X, first_constraint_y2, max_y2, color='red', alpha=0.1)
plt.fill_between(X, second_constraint_y2,max_y2, color='blue', alpha=0.1)

# Отображение значений координат точки пересечения на осях.
plt.text(y_1_intersect, 0.002, f'{y_1_intersect:.3f}')
plt.text(-0.01, y_2_intersect, f'{y_2_intersect:.3f}')

plt.legend(loc=(1.04, 0.2))
plt.grid(True)


# In[24]:


# Смешанные стратегии игрока А и цена игры.
game_price_value = 1 / player_a_target_function_value
assert check_resulted_game_price(
    game_matrix=reduced_game,
    game_price_value=game_price_value,
)

player_a_mixed_strategies = [
```

```
    var_value * game_price_value
    for var_value in [y_1_intersect, y_2_intersect]
]
player_a_mixed_strategies = player_a_mixed_strategies[:len(reduced_game.player_a_strategy_labels)]
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_a_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_a_strategy_labels,
        player_a_mixed_strategies,
    )),
    player_name="A",
)
print(mixed_strategies)


# #### 2.3.2. Прямая ЗЛП для игрока B
# ![LP_problem_B](./img/LP_problem_B.png)

# In[25]:


assert reduced_game.matrix.shape == (2, 2)

(a, b), (c, d) = reduced_game.matrix.tolist()
# Прямая для 1-го ограничения.
first_constraint = lambda x_1: (1 / b) - (a / b) * x_1
# Прямая для 2-го ограничения.
second_constraint = lambda x_1: (1 / d) - (c / d) * x_1
# Точка пересечения.
(x_1_intersect, x_2_intersect) = np.linalg.solve(
    np.array([
        [a, b],
        [c, d],
    ]),
    np.array([1, 1]),
)
# Выражаем целевую функцию через x_2: x_2 = F - x_1, где F -> max.
player_b_target_function_value = x_1_intersect + x_2_intersect
F_func = lambda x_1: (x_1_intersect + x_2_intersect) - x_1


# In[26]:


# Отрисовка графиков пересечения.
X = np.linspace(0, 1, 100)
first_constraint_x2 = first_constraint(X)
second_constraint_x2 = second_constraint(X)
F_x2 = F_func(X)

plt.figure(figsize=(8, 6))
```

```
plt.title("Графическое решение ПЗ ЛП для игрока B")

plt.plot(X, first_constraint_x2, label=f"x_2 ≤ {1 / b:.2f} + {- a / b:.2f}x_1", color="red")
plt.plot(X, second_constraint_x2, label=f"x_2 ≤ {1 / d:.2f} + {- c / d:.2f}x_1", color="blue")
plt.plot(X, F_x2, "--", label=f"x_2 = {x_1_intersect + x_2_intersect:.2f} - x_1", color="purple")

# Точка пересечения.
plt.plot(x_1_intersect, x_2_intersect, "o", color="purple", label="max")
# Проекции точки пересечения на оси.
plt.vlines(x_1_intersect, min(second_constraint_x2), x_2_intersect, color="black", linestyles='dashed')
plt.hlines(x_2_intersect, min(X), x_1_intersect, color="black", linestyles='dashed')
# Ограничения на неотрицательные решения.
plt.xlim(0, 0.15)
plt.ylim(0, 0.175)
# Подписи осей.
plt.xlabel("x_1")
plt.ylabel("x_2")
# Сегменты над графиками.
plt.fill_between(X, first_constraint_x2, color='red', alpha=0.1)
plt.fill_between(X, second_constraint_x2, color='blue', alpha=0.1)

# Отображение значений координат точки пересечения на осях
plt.text(x_1_intersect, 0.002, f'{x_1_intersect:.3f}')
plt.text(-0.01, x_2_intersect, f'{x_2_intersect:.3f}')

plt.legend(loc=(1.04, 0.2))
plt.grid(True)


# In[27]:


# Смешанные стратегии игрока B и цена игры.
game_price_value = 1 / player_b_target_function_value
assert check_resulted_game_price(
    game_matrix=reduced_game,
    game_price_value=game_price_value,
)

player_b_mixed_strategies = [var_value * game_price_value for var_value in [x_1_intersect, x_2_intersect]]
player_b_mixed_strategies = player_b_mixed_strategies[:len(reduced_game.player_b_strategy_labels)]
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_b_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_b_strategy_labels,
        player_b_mixed_strategies,
```

```
    )),
    player_name="B",
)
print(mixed_strategies)


# ### 2.4. Симплекс-метод (задача ЛП)
#
# <div style="text-align:left;">
#     <img src="img/LP_simplex.png" alt="LP_task_02" width="600" height="300">
# </div>
#

# #### 2.4.1. Двойственная задача ЛП для игрока A

# In[28]:


player_a_problem = DualProblem(input_path)


# In[29]:


player_a_solution = player_a_problem.solve()
player_a_var_values, player_a_target_function_value = player_a_solution


# In[30]:


# Смешанные стратегии игрока A и цена игры.
game_price_value = 1 / player_a_target_function_value
assert check_resulted_game_price(
    game_matrix=reduced_game,
    game_price_value=game_price_value,
)

player_a_mixed_strategies = [var_value * game_price_value for var_value in player_a_var_values]
player_a_mixed_strategies = player_a_mixed_strategies[:len(reduced_game.player_a_strategy_labels)]
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_a_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_a_strategy_labels,
        player_a_mixed_strategies,
    )),
    player_name="A",
)
print(mixed_strategies)

```

```python

# #### 2.4.2. Прямая задача ЛП для игрока B

# In[31]:


player_b_problem = SimplexProblem(input_path)
player_b_problem


# In[32]:


player_b_solution = player_b_problem.solve()
player_b_var_values, player_b_target_function_value = player_b_solution


# In[33]:


# Смешанные стратегии игрока B и цена игры.
game_price_value = 1 / player_b_target_function_value
assert check_resulted_game_price(
    game_matrix=reduced_game,
    game_price_value=game_price_value,
)

player_b_mixed_strategies = [var_value * game_price_value for var_value in player_b_var_values]
player_b_mixed_strategies = player_b_mixed_strategies[:len(reduced_game.player_b_strategy_labels)]
mixed_strategies = get_resulted_mixed_strategies(
    player_labels=game_matrix.player_b_strategy_labels,
    labels_to_probability=dict(zip(
        reduced_game.player_b_strategy_labels,
        player_b_mixed_strategies,
    )),
    player_name="B",
)
print(mixed_strategies)


# ### 3. Расчёт цены игры исходной матрицы
# Вспомним изначальную заданную матрицу игры, до нормализации и найдём её цену игры и смешанные стратегии игроков.

# In[34]:


# Вычитаем ранее добавленное слагаемое нормализации.
game_matrix.matrix -= normalizer
game_matrix
```

```python


# In[35]:


print(f"Нижняя цена игры:  {game_matrix.lowest_game_price[1]}\n"
      f"Верхняя цена игры: {game_matrix.highest_game_price[1]}")


# In[36]:


original_game_price: float = game_price_value - normalizer
print(f"Цена игры исходной матрицы: {original_game_price:.2f}")
```

ПРИЛОЖЕНИЕ Б

Ссылка на репозиторий с исходным кодом задачи

https://github.com/aaaaaaaalesha/10-game_theory