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

Рубежный контроль №1
«Монотонный итеративный алгоритм»

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 монотонный итеративный алгоритм решения антагонистической игры двух лиц в нормальной форме.

Задание

Для приведённой ниже матрицы стратегий найти цену игры и оптимальные стратегии обоих игроков монотонным итеративным методом.

$$\begin{pmatrix} 1 & 11 & 11 \\ 7 & 5 & 8 \\ 16 & 6 & 2 \end{pmatrix}$$

Ход работы

Для реализации решения рубежного контроля был использован язык программирования Python. К проверке представляется интерактивный блокнот Jupyter Notebook в файле monotonic_algorythm.ipynb, и импортированный файл *monotonic* (см. приложение А).

Инициализация матрицы представлена на рисунке 1. Нижняя и верхняя цены игры: 5 и 11. Седловой точки нет. Для однозначной идентификации назовём игроков А и В.

Рубежный контроль №1

"Монотонный итеративный алгоритм решения матричной игры"

Выполнил: Александров А. Н., ИУ8-104

Вариант: 1

Задание

Пусть с помощью матрицы А задана матричная антагонистическая игра двух игроков. По данной матрице необходимо найти решение игры для первого игрока (то есть, вектор оптимальных стратегий и цену игры), используя монотонный итеративный алгоритм.

Игра Г размера (3 x 3) задана матрицей:

1	11	11
7	5	8
16	6	2

5]:
```python
import logging

import numpy as np

from game_theory.utils.matrix_games.monotonic.monotonic import Monotonic

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

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

5]:
```python
# Входная матрица прямоугольной игры с нулевой суммой.
game_matrix = GameMatrix(
    matrix=np.array(
        [
            [1, 11, 11],
            [7, 5, 8],
            [16, 6, 2],
        ],
        dtype=int,
    )
)
game_matrix
```

5]:
```
+------------------------------------------------+
|          Таблица стратегий (игрока A)          |
+---------------+----+----+----+-----------------+
|   Стратегии   | b1 | b2 | b3 | MIN выигрыш A |
+---------------+----+----+----+-----------------+
|       a1      | 1  | 11 | 11 |        1        |
|       a2      | 7  | 5  | 8  |        5        |
|       a3      | 16 | 6  | 2  |        2        |
| MAX проигрыш B | 16 | 11 | 11 |                 |
+---------------+----+----+----+-----------------+
```

Рисунок 1 – Задание исходной матрицы игры с нулевой суммой

В данном случае доминируемых стратегий нет (см. рисунок 2), поэтому перейдём сразу к решению игры в смешанных стратегиях.

```
reduced_game: GameMatrix = game_matrix.reduce_dimension(method='dominant_absorption')
reduced_game
```

```
+-----------------------------------------------+
|           Таблица стратегий (игрока A)        |
+----------------+----+----+----+---------------+
|   Стратегии    | b1 | b2 | b3 | MIN выигрыш A |
+----------------+----+----+----+---------------+
|       a1       | 1  | 11 | 11 |       1       |
|       a2       | 7  | 5  | 8  |       5       |
|       a3       | 16 | 6  | 2  |       2       |
| MAX проигрыш B | 16 | 11 | 11 |               |
+----------------+----+----+----+---------------+
```

Рисунок 2 – Проверка отсутствия доминируемых стратегий

Далее приводится вывод программы для решения игры. Подыгры в методе решались сведением матричной игры к симплекс-методу. Промежуточные симплекс-таблицы опущены, чтобы не нагромождать листинг. Чистая стратегия на первой итерации выбирается случайно.

```
Решение игры относительно игрока A
Итерация 0:
	x^0 = [0 0 1]
	c^0 = [16.  6.  2.]
	v^0 = 2.0
	J^0 = [3]

Итерация 1:
Рассмотрим подыгру Г^1:
[[11.]
 [ 8.]
 [ 2.]]
Седловая точка найдена: (11.0, [1, 0, 0])
Оптимальная стратегия игрока:
	‾x_1 = [1 0 0]
	‾c_1 = [ 1. 11. 11.]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[16.  6.  2.]
 [ 1. 11. 11.]]
Удаление NBR-стратегий ['b2']
Матрица после уменьшения размерности:
[[16.  2.]
 [ 1. 11.]]
+----+--------+---------+---------+
|    |  Si0   |   x4    |   x3    |
+----+--------+---------+---------+
| x1 | 0.0575 |  0.0057 | -0.0632 |
| x2 | 0.0805 | -0.0920 |  0.0115 |
| F  | 0.1379 | -0.0862 | -0.0517 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x4 = x3 = 0, x1 = 0.057, x2 = 0.080
Целевая функция: F = 0.138

В результате получена оптимальная стратегия: (α_1, 1 - α_1) = [0.583 0.417]
```

```
      x^1 = [0.583 0.    0.417]
      c^1 = [7.251 8.917 7.25 ]
      v^1 = 7.25
      J^1 = [3]
```

Итерация 2:

```
Рассмотрим подыгру Г^2:
[[11.]
 [ 8.]
 [ 2.]]
Седловая точка найдена: (11.0, [1, 0, 0])
Оптимальная стратегия игрока:
      ¯x_2 = [1 0 0]
      ¯c_2 = [ 1. 11. 11.]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[ 7.251  8.917  7.25 ]
 [ 1.    11.    11.   ]]
Удаление NBR-стратегий ['b2']
Матрица после уменьшения размерности:
[[ 7.251  7.25 ]
 [ 1.    11.   ]]
+----+--------+---------+---------+
|    |  Si0   |   x4    |   x3    |
+----+--------+---------+---------+
| x1 | 0.1379 |  0.0138 | -0.1517 |
| x2 | 0.0000 | -0.1000 |  0.1000 |
| F  | 0.1379 | -0.0862 | -0.0517 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x4 = x3 = 0, x1 = 0.138, x2 = 0.000
Целевая функция: F = 0.138
В результате получена оптимальная стратегия: (α_2, 1 - α_2) = [0. 1.]
      x^2 = [0.583 0.    0.417]
      c^2 = [7.251 8.917 7.251]
      v^2 = 7.251
      J^2 = [1, 3]
```

Итерация 3:

```
Рассмотрим подыгру Г^3:
[[ 1. 11.]
 [ 7.  8.]
 [16.  2.]]
+----+--------+---------+---------+---------+
|    |  Si0   |   x4    |   x1    |   x5    |
+----+--------+---------+---------+---------+
| x3 | 0.0088 | -0.0702 | -0.6053 |  0.0614 |
| x2 | 0.1228 |  0.0175 |  1.5263 | -0.1404 |
| F  | 0.1316 | -0.0526 | -0.0789 | -0.0789 |
+----+--------+---------+---------+---------+
Оптимальное решение найдено!
x4 = x1 = x5 = 0, x3 = 0.009, x2 = 0.123
Целевая функция: F = 0.132
Оптимальная стратегия игрока:
      ¯x_3 = [0.    0.933 0.067]
      ¯c_3 = [7.6   5.067 7.6  ]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[7.251 8.917 7.251]
 [7.6   5.067 7.6  ]]
Удаление NBR-стратегий ['b3']
Матрица после уменьшения размерности:
[[7.251 8.917]
 [7.6   5.067]]
```

```
+----+--------+---------+---------+
|    |  Si0   |    x3   |    x4   |
+----+--------+---------+---------+
| x2 | 0.0537 | -0.2873 |  0.2336 |
| x1 | 0.0816 |  0.1633 | -0.2449 |
| F  | 0.1353 | -0.1241 | -0.0113 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x3 = x4 = 0, x2 = 0.054, x1 = 0.082
Целевая функция: F = 0.135
В результате получена оптимальная стратегия: (α_3, 1 - α_3) = [0.397 0.603]
      x^3 = [0.352 0.37  0.278]
      c^3 = [7.389 7.389 7.389]
      v^3 = 7.389
      J^3 = [1, 2, 3]
Так как в J^3 попали все номера столбцов, останавливаем алгоритм
```

Решение игры относительно игрока B

```
Итерация 0:
      x^0 = [1 0 0]
      c^0 = [ 1.   7. 16.]
      v^0 = 1.0
      J^0 = [1]
```

Итерация 1:

```
Рассмотрим подыгру Г^1:
[[ 1.]
 [11.]
 [11.]]
Седловая точка найдена: (11.0, [0, 1, 0])
Оптимальная стратегия игрока:
      ¯x_1 = [0 1 0]
      ¯c_1 = [11.  5.  6.]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[ 1.  7. 16.]
 [11.  5.  6.]]
Удаление NBR-стратегий ['b3']
Матрица после уменьшения размерности:
[[ 1.  7.]
 [11.  5.]]
+----+--------+---------+---------+
|    |  Si0   |    x3   |    x4   |
+----+--------+---------+---------+
| x2 | 0.0833 | -0.0972 |  0.0139 |
| x1 | 0.0833 |  0.0694 | -0.1528 |
| F  | 0.1667 | -0.0278 | -0.1389 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x3 = x4 = 0, x2 = 0.083, x1 = 0.083
Целевая функция: F = 0.167
В результате получена оптимальная стратегия: (α_1, 1 - α_1) = [0.5 0.5]
      x^1 = [0.5 0.5 0. ]
      c^1 = [ 6.  6. 11.]
      v^1 = 6.0
      J^1 = [1, 2]
```

Итерация 2:

```
Рассмотрим подыгру Г^2:
[[ 1.  7.]
 [11.  5.]
 [11.  8.]]
Седловая точка найдена: (8.0, [0, 0, 1])
Оптимальная стратегия игрока:
      ¯x_2 = [0 0 1]
      ¯c_2 = [11.  8.  2.]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[ 6.  6. 11.]
```

```
 [11.  8.  2.]]
Удаление NBR-стратегий ['b2']
Матрица после уменьшения размерности:
[[ 6. 11.]
 [11.  2.]]
+----+--------+---------+---------+
|    |  Si0   |    x3   |    x4   |
+----+--------+---------+---------+
| x2 | 0.0459 | -0.1009 |  0.0550 |
| x1 | 0.0826 |  0.0183 | -0.1009 |
| F  | 0.1284 | -0.0826 | -0.0459 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x3 = x4 = 0, x2 = 0.046, x1 = 0.083
Целевая функция: F = 0.128
В результате получена оптимальная стратегия: (α_2, 1 - α_2) = [0.357 0.643]
      x^2 = [0.321 0.321 0.357]
      c^2 = [7.786 6.714 7.786]
      v^2 = 6.714
      J^2 = [2]
```

Итерация 3:

```
Рассмотрим подыгру Г^3:
[[7.]
 [5.]
 [8.]]
Седловая точка найдена: (8.0, [0, 0, 1])
Оптимальная стратегия игрока:
      ¯x_3 = [0 0 1]
      ¯c_3 = [11.  8.  2.]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[ 7.786  6.714  7.786]
 [11.     8.     2.   ]]
Удаление NBR-стратегий ['b1']
Матрица после уменьшения размерности:
[[6.714 7.786]
 [8.    2.   ]]
+----+--------+---------+---------+
|    |  Si0   |    x3   |    x4   |
+----+--------+---------+---------+
| x2 | 0.0219 | -0.1594 |  0.1374 |
| x1 | 0.1228 |  0.0409 | -0.1637 |
| F  | 0.1447 | -0.1184 | -0.0263 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x3 = x4 = 0,
x2 = 0.022, x1 = 0.123
Целевая функция: F = 0.145
В результате получена оптимальная стратегия: (α_3, 1 - α_3) = [0.152 0.848]
      x^3 = [0.273 0.273 0.455]
      c^3 = [8.273 6.909 6.909]
      v^3 = 6.909
      J^3 = [2, 3]
```

Итерация 4:

```
Рассмотрим подыгру Г^4:
[[ 7. 16.]
 [ 5.  6.]
 [ 8.  2.]]
```

```
+----+--------+---------+---------+---------+
|    |  Si0   |   x4    |   x2    |   x5    |
+----+--------+---------+---------+---------+
| x3 | 0.0789 | -0.1404 |  0.3333 |  0.0614 |
| x1 | 0.0526 |  0.0175 |  0.3333 | -0.0702 |
| F  | 0.1316 | -0.1228 | -0.3333 | -0.0088 |
+----+--------+---------+---------+---------+
Оптимальное решение найдено!
x4 = x2 = x5 = 0, x3 = 0.079, x1 = 0.053
Целевая функция: F = 0.132
Оптимальная стратегия игрока:
      ̅x_4 = [0.4 0.  0.6]
      ̅c_4 = [7.  7.6 7.6]
Находим оптимальную стратегию игрока в подыгре из двух строк:
[[8.273 6.909 6.909]
 [7.    7.6   7.6  ]]
Удаление NBR-стратегий ['b3']
Матрица после уменьшения размерности:
[[8.273 6.909]
 [7.    7.6  ]]
+----+--------+---------+---------+
|    |  Si0   |   x4    |   x3    |
+----+--------+---------+---------+
| x1 | 0.0414 |  0.4825 | -0.5238 |
| x2 | 0.0940 | -0.5702 |  0.4762 |
| F  | 0.1353 | -0.0877 | -0.0476 |
+----+--------+---------+---------+
Оптимальное решение найдено!
x4 = x3 = 0, x1 = 0.041, x2 = 0.094
Целевая функция: F = 0.135
В результате получена оптимальная стратегия: (α_4, 1 - α_4) = [0.694 0.306]
      x^4 = [0.361 0.083 0.556]
      c^4 = [7.389 7.389 7.389]
      v^4 = 7.389
      J^4 = [1, 2, 3]
Так как в J^4 попали все номера столбцов, останавливаем алгоритм

Цена игры: 5 <= 7.389 <= 11
```

Смешанные стратегии игрока A		
a1	a2	a3
0.352	0.370	0.278

Смешанные стратегии игрока B		
b1	b2	b3
0.361	0.083	0.556

То есть решение, полученное для матричной игры монотонным итеративным алгоритмом:

$$7.389;\ S_A\colon (0.352, 0.370, 0.278);\ S_B\colon (0.361, 0.083, 0.556).$$

Вывод

В данной работе была исследована антагонистическая игра двух лиц с нулевой суммой, а также монотонный итеративный метод её решения. Для решения задачи был реализован алгоритм на Python, с помощью которого было найдено следующее решение игры:

$$7.389;\ S_A: (0.352, 0.370, 0.278);\ S_B: (0.361, 0.083, 0.556),$$

что совпадает с решением (с заданной точностью), найденным для этой же матрицы аналитическим методом. Также можно сравнить его с решением Брауна-Робинсон из лабораторной работы 1:

$$7.403;\ \widetilde{S_A}[239]: (0.393, 0.356, 0.251);\ \widetilde{S_B}[239]: (0.356, 0.092, 0.552).$$

Преимущество данного численного метода заключается в строгой монотонной сходимости оценки для цены игры. Однако для этого требуется относительно бóльшая вычислительная сложность, чем в методе Брауна-Робинсон за счёт решения подыгр меньшей, но не всегда малой размерностей.

ПРИЛОЖЕНИЕ А
Листинг исходного кода реализации алгоритма

monotonic_algorythm.ipynb

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

# # Рубежный контроль №1
# **"Монотонный итеративный алгоритм решения матричной игры"**
#
# **Выполнил: Александров А. Н., ИУ8-104**
#
# **Вариант: 1**
#
# ## Задание
# Пусть с помощью матрицы A задана матричная антагонистическая игра двух игроков. По данной матрице необходимо найти решение игры для первого игрока (то есть, вектор оптимальных стратегий и цену игры), используя монотонный итеративный алгоритм.
#
# Игра Г размера (3 x 3) задана матрицей:
#
# | 1  | 11 | 11 |
# |:--:|:--:|:--:|
# | 7  | 5  | 8  |
# | 16 | 6  | 2  |
#

# In[55]:


import logging

import numpy as np

from game_theory.utils.matrix_games.monotonic.monotonic import Monotonic

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

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


# In[56]:


# Входная матрица прямоугольной игры с нулевой суммой.
game_matrix = GameMatrix(
    matrix=np.array(
        [
            [1, 11, 11],
            [7, 5, 8],
            [16, 6, 2],
        ],
        dtype=int,
    )
)
game_matrix


# In[57]:


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


# In[58]:


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


# ### Монотонный итеративный алгоритм

# In[59]:


reduced_game
```

```
# In[60]:


monotonic_method = Monotonic(reduced_game)


# In[61]:


(
    (game_price_for_a, player_a_mixed_strategies),
    (game_price_for_b, player_b_mixed_strategies),
) = monotonic_method.solve()


# In[62]:


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

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,
        list(player_a_mixed_strategies),
    )),
    player_name="A",
)
print(mixed_strategies)


# In[63]:


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

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

monotonic.py

```
"""Монотонный итеративный алгоритм решения матричной игры (n x m)-игры с нулевой суммой."""
import logging
import random

import numpy as np
from numpy import ndarray

from game_theory.utils.matrix_games.game_matrix import GameMatrix
from game_theory.utils.matrix_games.types import IndexType, ValueType

_logger = logging.getLogger(__name__)


class Monotonic:
    """Класс инкапсулирует решение матричной игры монотонным итеративным методом."""

    def __init__(self, game_matrix: GameMatrix):
        self.game: GameMatrix = game_matrix

        self.iteration_number: int = 0
        self.strategy_index: IndexType
        self.strategy_x: np.ndarray[float]
        self.scores_c: np.ndarray[ValueType]
        self.price_v: ValueType
        self.indicator_mask_j: np.ndarray[bool]

    def solve(self):
```

```python
        # Решение за игрока A.
        _logger.info("Решение игры относительно игрока A")
        price_a, strategy_a = self._base_solve(np.float16(self.game.matrix))
        iterations_a_count = self.iteration_number
        # Решения за игрока B.
        _logger.info("Решение игры относительно игрока B")
        price_b, strategy_b = self._base_solve(np.float16(self.game.matrix.T))
        iterations_b_count = self.iteration_number

        _logger.info(f"Итераций игроков сделано {(iterations_a_count, iterations_b_count)}")
        return (price_a, strategy_a), (price_b, strategy_b)

    def _base_solve(self, matrix: np.ndarray[ValueType]):
        m, n = matrix.shape
        self.iteration_number = 0
        _logger.info("Итерация 0:")
        # Выбираем произвольную (x^0) чистую стратегию (выставляя 1 только в одну позицию).
        strategy_index: IndexType = random.randint(0, m - 1)
        self.strategy_x = np.array([0] * n)
        self.strategy_x[strategy_index] = 1
        # Выбираем вектор (c^0), соответствующий выбранной стратегии.
        self.scores_c: np.ndarray = matrix[strategy_index].copy()
        # Текущая цена игры.
        self.price_v = np.min(self.scores_c)
        # Вектор-индикатор, который показывает принадлежность к множеству.
        self.indicator_mask_j: np.ndarray[bool] = np.isclose(self.scores_c, self.price_v)
        self.__log_calculated_parameters()

        alpha_values = np.array((np.inf, np.inf))
        # Выполняем итерации без заданной точности, то есть пока α_N не станет 0.
        while not np.allclose(alpha_values, [0, 1]):
            optimal_strategy_x_, optimal_scores_c_, alpha_values = self.perform_iteration(matrix)

            alpha, _ = alpha_values
            if not np.isclose(alpha, 0):
                self.strategy_x = (1 - alpha) * self.strategy_x + alpha * optimal_strategy_x_
                self.scores_c = (1 - alpha) * self.scores_c + alpha * optimal_scores_c_
                self.price_v = np.min(self.scores_c)
                self.indicator_mask_j = np.isclose(self.scores_c, self.price_v)
                self.__log_calculated_parameters()
                # Если в J_i попали все столбцы, завершаем алгоритм.
                if np.all(self.indicator_mask_j):
                    _logger.info(
                        f"Так как в J^{self.iteration_number} попали все номера столбцов, "
                        f"останавливаем алгоритм"
                    )
                    return self.price_v, self.strategy_x.copy()

        _logger.info(f"Получили α_{self.iteration_number} = {alpha:.0f}, поэтому останавливаем
алгоритм")

        return self.price_v, self.strategy_x.copy()

    def perform_iteration(self, matrix: np.ndarray[ValueType], accuracy=3) -> tuple[ndarray, ndarray,
ndarray]:
        self.iteration_number += 1
        i = self.iteration_number
        _logger.info(f"\nИтерация {self.iteration_number}:")
        # Выбираем только столбцы, удовлетворяющие нашему индикатору.
        sub_game_matrix_a = GameMatrix(matrix[:, self.indicator_mask_j].copy())
        _logger.info(f"Рассмотрим подыгру Г^{i}: " f"\n{np.round(sub_game_matrix_a.matrix, accuracy)}")
        # Решаем подыгру и находим оптимальную стратегию x_.
        _, optimal_strategy_x_ = sub_game_matrix_a.solve()
        optimal_scores_c_: np.ndarray = self.__reduce_sum(matrix, np.array(optimal_strategy_x_))
        _logger.info(
            f"Оптимальная стратегия игрока: "
            f"\n\t x_{i} = {np.round(optimal_strategy_x_, accuracy)}"
            f"\n\t c_{i} = {np.round(optimal_scores_c_, accuracy)}"
        )
        # Находим оптимальную стратегию игрока в подыгре из двух строк.
        sub_game_gamma = GameMatrix(np.stack((self.scores_c, optimal_scores_c_)))
        _logger.info(
            f"Находим оптимальную стратегию игрока в подыгре из двух строк: "
            f"\n{np.round(sub_game_gamma.matrix, accuracy)}"
        )
        sub_game_gamma = sub_game_gamma.reduce_dimension(method="nbr_drop")
        _logger.info(
            f"Матрица после уменьшения размерности: "
            f"\n{np.round(sub_game_gamma.reduce_dimension().matrix, accuracy)}"
        )
        _, alpha_values = sub_game_gamma.solve()
        alpha_values = (alpha_values[1] if len(alpha_values) == 2 else 0, alpha_values[0])
        _logger.info(
            f"В результате получена оптимальная стратегия: "
            f"(α_{i}, 1 - α_{i}) = "
```

```python
            f"{np.round(alpha_values, accuracy)}"
        )

        return np.array(optimal_strategy_x_), optimal_scores_c_, np.array(alpha_values)

    @staticmethod
    def __reduce_sum(lhs: np.ndarray, rhs: np.ndarray) -> np.ndarray:
        return np.sum((rhs * lhs.T).T, 0)

    def __log_calculated_parameters(self, accuracy=3):
        j_indexes = [i + 1 for i in list(*np.where(self.indicator_mask_j))]
        _logger.info(
            "\n".join(
                [
                    f"\tx^{self.iteration_number} = {np.round(self.strategy_x, accuracy)}",
                    f"\tc^{self.iteration_number} = {np.round(self.scores_c, accuracy)}",
                    f"\tv^{self.iteration_number} = {round(self.price_v, accuracy)}",
                    f"\tJ^{self.iteration_number} = {j_indexes}",
                ]
            )
        )
```