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

Лабораторная работа №1
«Аналитический и численный (Брауна-Робинсон) методы решения антагонистической игры в смешанных стратегиях»

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 аналитический (обратной матрицы) и численный (Брауна-Робинсон) методы нахождения смешанных стратегий в антагонистической игре двух лиц в нормальной форме.

Задание

Для приведённой ниже матрицы стратегий найти цену игры и оптимальные стратегии обоих игроков методами обратной матрицы и Брауна-Робинсон. Сравнить полученные результаты.

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

Ход работы

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

В исполняемом «ноутбуке» импортируются реализованные модули *matrix_games* и *brown_robinson*, инкапсулирующие логику и алгоритмы для матричных игр и численного метода Брауна-Робинсон соответственно. Ознакомиться со всем исходным кодом данной работы можно, посетив репозиторий, ссылка на который представлена в приложении Б. Инициализация матрицы представлена на рисунке 1. Нижняя и верхняя цены игры: 5 и 11. Седловой точки нет. Для однозначной идентификации назовём игроков A и B.

```python
import logging
from pathlib import Path
from copy import deepcopy

import numpy as np

from game_theory.utils.matrix_games.brown_robinson.brown_robinson import BrownRobinson
from game_theory.utils.matrix_games.brown_robinson.labels import (
    MAXMIN_ESTIMATION_LABEL,
    MINMAX_ESTIMATION_LABEL,
    ACCURACY_LABEL,
)
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,
)

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

```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
```

```
+-----------------------------------------------+
|          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 |               |
+-----------------+----+----+----+---------------+
```

```python
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:  5
Верхняя цена игры: 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

1. 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:

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

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

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

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

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):

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

$$\dots$$

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

На рисунках 3 и 4 представлены решения матричной игры в смешанных стратегиях для игроков A и B соответственно.

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

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

```
Цена игры: 5 <= 7.389 <= 11
+-------------------------------+
| Смешанные стратегии игрока A |
+---------+---------+-----------+
|    a1   |    a2   |    a3     |
+---------+---------+-----------+
|  0.352  |  0.370  |  0.278    |
+---------+---------+-----------+
```

```
sum((first_mixed_strategy, second_mixed_strategy, third_mixed_strategy,))
```

```
1.0
```

Рисунок 3 – Аналитический метод поиска смешанных стратегий игрока A

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

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

```
Цена игры: 5 <= 7.389 <= 11
+-------------------------------+
| Смешанные стратегии игрока B |
+---------+---------+-----------+
|    b1   |    b2   |    b3     |
+---------+---------+-----------+
|  0.361  |  0.083  |  0.556    |
+---------+---------+-----------+
```

```
sum((first_mixed_strategy, second_mixed_strategy, third_mixed_strategy,))
```

```
1.0
```

Рисунок 4 – Аналитический метод поиска смешанных стратегий игрока B

Полученные цена игры и смешанные стратегии игроков:

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

2. Численный метод Брауна-Робинсон

Метод Брауна-Робинсон является приближённым численным методом решения произвольной игры Γ размера $(m \times n)$ для заранее заданного параметра ε, характеризующего оценку погрешности итерационного алгоритма. Тогда при $\varepsilon \to 0$, получаемая нами оценка для цены игры будет в точности совпадать с той, которую мы бы получили с использованием аналитического метода.

На практике нам может быть достаточно и результата, полученного с меньшей точностью. В данной работе рассмотрим решение матричной игры до уровня погрешности $\varepsilon \leq 0.1$. На рисунке 5 представлена таблица итеративного алгоритма Брауна-Робинсон для решения исходной матричной игры. Для решения задачи потребовалось 239 итераций алгоритма. Полная версия данной таблицы приведена в приложении В.

Уровень погрешности: $\varepsilon \leq 0,1$

```
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 |               |
+-----------------+----+----+----+---------------+
```

```
brown_robinson = BrownRobinson(game_matrix=reduced_game, accuracy=0.1)
df = brown_robinson.solve(out=Path("iterations_table.csv"), mode="previous")
df
```

	k	A	B	a1	a2	a3	b1	b2	b3	ВЦИ	НЦИ	ε
	1	a1	b3	11	8	2	1	11	11	11.000000	1.000000	10.000000
	2	a1	b1	12	15	18	2	22	22	9.000000	1.000000	8.000000
	3	a3	b1	13	22	34	18	28	24	11.333333	6.000000	3.000000
	4	a3	b1	14	29	50	34	34	26	12.500000	6.500000	2.500000
	5	a3	b3	25	37	52	50	40	28	10.400000	5.600000	2.500000
	...	...	...	...	...	...	...	...	...	...	...	...
	235	a1	b1	1775	1733	1692	1645	1775	1790	7.553191	7.000000	0.190691
	236	a1	b1	1776	1740	1708	1646	1786	1801	7.525424	6.974576	0.162924
	237	a1	b1	1777	1747	1724	1647	1797	1812	7.497890	6.949367	0.135390
	238	a1	b1	1778	1754	1740	1648	1808	1823	7.470588	6.924370	0.108088
	239	a1	b1	1779	1761	1756	1649	1819	1834	7.443515	6.899582	0.081015

239 rows × 12 columns

Рисунок 5 – Таблица итеративного алгоритма Брауна-Робинсон

Для расчёта приближённых смешанных стратегий игроков достаточно произвести подсчёт частот использования стратегий. На рисунке 6 продемонстрирован результат решения нормализованной матричной игры с нулевой суммой методом Брауна-Робинсон.

```
# Приближенная цена игры игроков.
game_price_value = brown_robinson.game_price_estimation
# Приближенные смешанные стратегии игроков.
(
    (strategy_a_1, strategy_a_2, strategy_a_3),
    (strategy_b_1, strategy_b_2, strategy_b_3),
) = brown_robinson.mixed_strategies
```

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

```
Цена игры: 5 <= 7.403 <= 11
```

```
mixed_strategies = get_resulted_mixed_strategies( •••
```

```
+-----------------------------+
| Смешанные стратегии игрока A |
+---------+---------+---------+
|    a1   |    a2   |    a3   |
+---------+---------+---------+
|  0.393  |  0.356  |  0.251  |
+---------+---------+---------+
```

```
sum((strategy_a_1, strategy_a_2, strategy_a_3))
```

```
1.0
```

```
mixed_strategies = get_resulted_mixed_strategies( •••
```

```
+-----------------------------+
| Смешанные стратегии игрока B |
+---------+---------+---------+
|    b1   |    b2   |    b3   |
+---------+---------+---------+
|  0.356  |  0.092  |  0.552  |
+---------+---------+---------+
```

```
sum((strategy_b_1, strategy_b_2, strategy_b_3))
```

```
1.0
```

Рисунок 6 – Найденные приближенное решение нормализованной матричной игры методом Брауна-Робинсон

Полученные оценки цены игры и смешанных стратегий игроков:

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

3. Сравнительная оценка погрешностей вычислений

Полученные результаты и их приведены в сводной таблице 1. Графики сходимости приближенных значений цен игры и оценки погрешности приведены на рисунках 7 и 8 соответственно. Таким образом полученное приближенное решение удовлетворяет заданной точности $\varepsilon \leq 0.1$.

Таблица 1 – Сводная таблица сравнительной оценки погрешностей

	Цена игры	Смешанные стратегии игрока А			Смешанные стратегии игрока В		
		a_1	a_2	a_3	b_1	b_2	b_3
Аналитический (матричный) метод	7,389	0,352	0,370	0,278	0,361	0,083	0,556
Численный метод Брауна-Робинсон	7,403	0,393	0,356	0,251	0,356	0,092	0,552
Абсолютная погрешность вычислений, Δ	0,014	0,041	0,014	0,027	0,005	0,009	0,004
Относительная погрешность вычислений, %	3,6	11,6	3,7	9,7	1,4	10,8	0,7

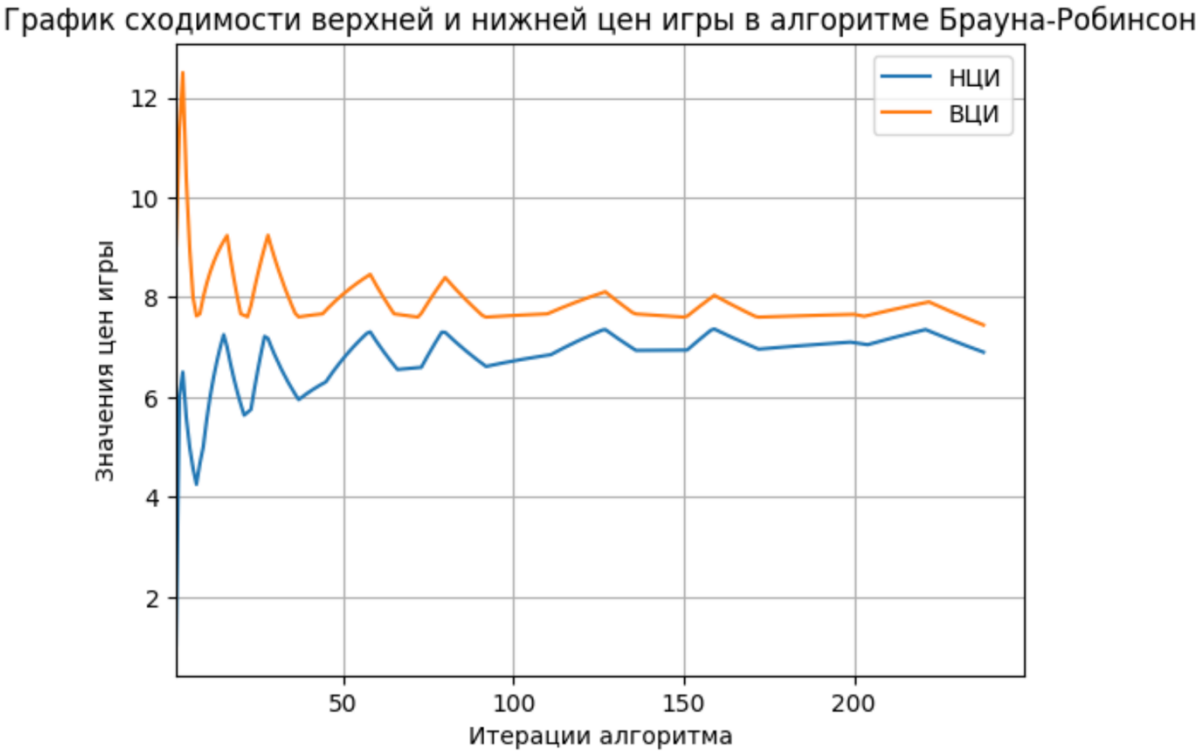

Рисунок 7 – График сходимости верхней и нижней цен игры в алгоритме Брауна–Робинсон

Рисунок 8 – График оценки погрешности алгоритма Брауна–Робинсон

Вывод

В данной работе была исследована антагонистическая игра двух лиц с нулевой суммой, а также аналитический (обратной матрицы) и численный (Брауна-Робинсон) методы её решения.

Сначала было получено точное эталонное решение с помощью аналитического метода:

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

К недостаткам данного метода можно отнести ограничения к условиям его применимости (матрица игры должна быть квадратной и невырожденной), а также относительно высокую вычислительную сложность: $O(n^3)$, где n – число стратегий произвольного игрока.

Численный итерационный метод Брауна-Робинсон нивелирует оба недостатка аналитического метода за счёт уменьшения точности вычислений. Задав погрешность $\varepsilon \leq 0.1$, было получено следующее приближённое решение матричной игры:

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

Сложность данного метода линейна по числу стратегий игроков: $O(m + n)$, где m и n – число стратегий игроков A и B соответственно. Недостатком данного алгоритма остаётся его немонотонность.

В пункте 3 была рассмотрена сравнительная оценка погрешностей, где была осуществлена повторная проверка, что полученное решение точно с учётом заданного значения ε. Можно сделать вывод, что метод Брауна–Робинсон хорошо подходит для вычисления приближенного решения произвольной матричной игры с нулевой суммой.

ПРИЛОЖЕНИЕ А
Листинг интерактивного блокнота brown-robinson.ipynb

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

# # Лабораторная работа №1.
# **"Аналитический и численный (Брауна-Робинсон) методы решения антагонистической игры в
смешанных стратегиях**
#
# **Выполнил: Александров А. Н., ИУ8-104**
#
# **Вариант: 1**
#
# ## Задание
# Найти цену игры и оптимальные стратегии обоих игроков методами обратной матрицы (аналитически)
и Брауна-Робинсон. Сравнить полученные результаты.
#
# Игра Г размера (3 x 3) задана матрицей:
#
# | 1  | 11 | 11 |
# |:--:|:--:|:--:|
# | 7  | 5  | 8  |
# | 16 | 6  | 2  |
#

# In[501]:


import logging
from pathlib import Path
from copy import deepcopy

import numpy as np

from game_theory.utils.matrix_games.brown_robinson.brown_robinson import BrownRobinson
from game_theory.utils.matrix_games.brown_robinson.labels import (
    MAXMIN_ESTIMATION_LABEL,
    MINMAX_ESTIMATION_LABEL,
    ACCURACY_LABEL,
)
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,
)

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


# In[502]:


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


# In[503]:


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


# In[504]:


game_matrix: GameMatrix = deepcopy(original_game_matrix)
normalizer: int = game_matrix.normalize_matrix()
```

```
game_matrix


# In[505]:


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


# In[506]:


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


# In[507]:


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


# ### 1. Аналитический (матричный) метод

# #### 1.1. 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[508]:


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


# In[509]:


# 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, third_mixed_strategy),
    )),
    player_name="A",
)
print(mixed_strategies)


# In[510]:


sum((first_mixed_strategy, second_mixed_strategy, third_mixed_strategy,))


# #### 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[511]:


analytical_solver = AnalyticalSolver(reduced_game)
(
    first_mixed_strategy,
    second_mixed_strategy,
```

```
    third_mixed_strategy,
    game_price_value,
) = analytical_solver.player_b_solve()


# In[512]:


# 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, third_mixed_strategy),
    )),
    player_name="B",
)
print(mixed_strategies)


# In[513]:


sum((first_mixed_strategy, second_mixed_strategy, third_mixed_strategy,))


# ### Возврат к цене игры для исходной матрицы

# In[514]:


assert check_resulted_game_price(
    game_matrix=original_game_matrix,
    # Уменьшаем ЦИ на слагаемое, используемое при нормализации матрицы.
    game_price_value=game_price_value - normalizer,
)


# ### 2. Численный метод Брауна-Робинсон
#
# Уровень погрешности: $\varepsilon \le 0,1$

# In[515]:


reduced_game


# In[516]:


brown_robinson = BrownRobinson(game_matrix=reduced_game, accuracy=0.1)
df = brown_robinson.solve(out=Path("iterations_table.csv"))
df


# In[517]:


# Приближенная цена игры игроков.
game_price_value = brown_robinson.game_price_estimation
# Приближенные смешанные стратегии игроков.
(
    (strategy_a_1, strategy_a_2, strategy_a_3),
    (strategy_b_1, strategy_b_2, strategy_b_3),
) = brown_robinson.mixed_strategies


# In[518]:


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

```
# In[519]:


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,
        (strategy_a_1, strategy_a_2, strategy_a_3),
    )),
    player_name="A",
)
print(mixed_strategies)


# In[520]:


sum((strategy_a_1, strategy_a_2, strategy_a_3))


# In[521]:


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,
        (strategy_b_1, strategy_b_2, strategy_b_3),
    )),
    player_name="B",
)
print(mixed_strategies)


# In[522]:


sum((strategy_b_1, strategy_b_2, strategy_b_3))


# In[523]:


df[[MAXMIN_ESTIMATION_LABEL, MINMAX_ESTIMATION_LABEL]].plot(
    title="График сходимости верхней и нижней цен игры в алгоритме Брауна-Робинсон",
    xlabel="Итерации алгоритма",
    ylabel="Значения цен игры",
    xlim=(1, None),
    grid=True,
)


# In[524]:


plt = df[ACCURACY_LABEL].plot(
    title="График оценки погрешности алгоритма Брауна-Робинсон",
    xlabel="Итерации алгоритма",
    ylabel="Значение погрешности ε",
    xlim=(1, None),
    ylim=(0, None),
    grid=True,
)


### Возврат к цене игры для исходной матрицы

# In[525]:


assert check_resulted_game_price(
    game_matrix=original_game_matrix,
    # Уменьшаем ЦИ на слагаемое, используемое при нормализации матрицы.
    game_price_value=game_price_value - normalizer,
)
```

brown_robinson.py:

```python
"""Итерационный численный метод Брауна-Робинсон решения (n x m)-игры с нулевой суммой."""
import logging
import random
from pathlib import Path

import numpy as np
import pandas as pd

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

from .labels import (
    ACCURACY_LABEL,
    CHOSEN_A_LABEL,
    CHOSEN_B_LABEL,
    ITERATION_LABEL,
    MAXMIN_ESTIMATION_LABEL,
    MINMAX_ESTIMATION_LABEL,
)

_logger = logging.getLogger(__name__)


class BrownRobinson:
    """Класс инкапсулирует решение матричной игры методом Брауна-Робинсон."""

    def __init__(self, game_matrix: GameMatrix, accuracy: float = 0.1):
        self.game_matrix: GameMatrix = game_matrix

        # Случайным образом производим выбор игроков A и B.
        self.player_a_strategy_index: IndexType = random.randint(0, game_matrix.shape[0] - 1)
        self.player_b_strategy_index: IndexType = random.randint(0, game_matrix.shape[1] - 1)

        # Накопленные значения выигрыша/проигрыша игроков A и B.
        self.player_a_accumulated_values: np.ndarray[ValueType] = self.player_a_strategy_values
        self.player_b_accumulated_values: np.ndarray[ValueType] = self.player_b_strategy_values

        # Текущая усреднённая верхняя и нижняя цены игры (ВЦИ и НЦИ).
        self.minmax_price_estimation: float = max(self.player_a_strategy_values)
        self.maxmin_price_estimation: float = min(self.player_b_strategy_values)

        # Заполняем строку первой итерации метода.
        self.solution_table = pd.DataFrame.from_records(
            [
                {
                    ITERATION_LABEL: 1,
                    CHOSEN_A_LABEL: self.player_a_strategy_labels[self.player_a_strategy_index],
                    CHOSEN_B_LABEL: self.player_b_strategy_labels[self.player_b_strategy_index],
                    **dict(zip(self.player_a_strategy_labels, self.player_a_accumulated_values)),
                    **dict(zip(self.player_b_strategy_labels, self.player_b_accumulated_values)),
                    MINMAX_ESTIMATION_LABEL: self.minmax_price_estimation,
                    MAXMIN_ESTIMATION_LABEL: self.maxmin_price_estimation,
                    ACCURACY_LABEL: self.minmax_price_estimation - self.maxmin_price_estimation,
                }
            ]
        )

        # Точность вычислений численного метода.
        self.accuracy: float = accuracy
        # Метка решённой задачи.
        self.is_solved = False

    @property
    def player_a_strategy_labels(self) -> list[LabelType]:
        """Возвращает список меток стратегий игрока A."""
        return self.game_matrix.player_a_strategy_labels

    @property
    def player_b_strategy_labels(self) -> list[LabelType]:
        """Возвращает список меток стратегий игрока B."""
        return self.game_matrix.player_b_strategy_labels

    @property
    def player_a_strategy_values(self) -> np.ndarray[ValueType]:
        """Значения выигрышей игрока A при текущей стратегии."""
        return self.game_matrix.matrix.T[self.player_b_strategy_index].copy()

    @property
```

```
    def player_b_strategy_values(self) -> np.ndarray[ValueType]:
        """Значения проигрышей игрока B при текущей стратегии."""
        return self.game_matrix.matrix[self.player_a_strategy_index].copy()

    @property
    def mixed_strategies(self) -> tuple[tuple[float, ...] | None, tuple[float, ...] | None]:
        """
        Возвращает кортеж смешанных стратегий игроков, если решение было найдено и
        (None, None), если задача ещё не была решена.
        """
        if not self.is_solved:
            return None, None

        iterations_count: int = len(self.solution_table)
        mixed_strategies_player_a: pd.Series = (
            self.solution_table[CHOSEN_A_LABEL].value_counts().sort_index() / iterations_count
        )
        mixed_strategies_player_b: pd.Series = (
            self.solution_table[CHOSEN_B_LABEL].value_counts().sort_index() / iterations_count
        )
        return tuple(mixed_strategies_player_a.values), tuple(mixed_strategies_player_b.values)

    @property
    def game_price_estimation(self) -> ValueType | None:
        """Оценка для цены игры на данной итерации алгоритма."""
        last_row: pd.Series = self.solution_table.iloc[-1]
        # В качестве оценки берём верхнюю цену игры.
        return last_row[MINMAX_ESTIMATION_LABEL]

    def solve(
        self,
        mode="random",
        out: Path | None = None,
    ) -> pd.DataFrame:
        """
        Производит решение матричной игры за обоих игроков методом Брауна-Робинсон.
        :param out: Опциональный путь до файла для записи таблицы итераций алгоритма.
        :param str mode: Регламентирует, как выбирать стратегии в случае коллизий.
            - "random" (default) - использует случайную из лучших стратегий;
            - "previous" - использует стратегию с прошлой итерации.
        :return:
        """
        if self.is_solved:
            return self.__write_result(out)

        # Берём предыдущую строку итерации алгоритма.
        last_row: pd.Series = self.solution_table.iloc[-1]
        # Проделываем итерации алгоритма, пока не достигнем величины, меньшей ε.
        while last_row[ACCURACY_LABEL] > self.accuracy:
            # Производим очередную итерацию алгоритма.
            last_row: pd.Series = self.__perform_iteration(last_row, mode=mode)
            # Добавляем очередную строку в таблицу.
            self.solution_table.loc[len(self.solution_table)] = last_row

        self.is_solved = True
        return self.__write_result(out)

    def __write_result(self, out_path: Path | None = None) -> pd.DataFrame:
        # Округляем до 3 значащих цифр после запятой.
        self.solution_table = self.solution_table.round(3)
        # Экспортируем результат в CSV-таблицу, если передан путь.
        if isinstance(out_path, Path):
            self.solution_table.to_csv(out_path, index=False)

        return self.solution_table

    def __perform_iteration(self, previous_row: pd.Series, mode="random") -> pd.Series:
        """
        Производит очередную итерацию алгоритма Брауна-Робинсон.
        :param pd.Series previous_row: Строка предыдущей итерации алгоритма.
        :param str mode: Регламентирует, как выбирать стратегии в случае коллизий.
            - "random" (default) - использует случайную из лучших стратегий;
            - "previous" - использует стратегию с прошлой итерации.
        :return: Вычисленное значение ошибки (точности) ε на данной итерации.
        """
        # Выбираем лучшие стратегии игроков в новой итерации.
        (self.player_a_strategy_index, self.player_b_strategy_index) =
self.__choose_strategies(mode=mode)
        # Добавляем выбранные стратегии к накопленным ранее.
        self.player_a_accumulated_values += self.player_a_strategy_values
        self.player_b_accumulated_values += self.player_b_strategy_values
```

```
        iteration_k: int = previous_row[ITERATION_LABEL] + 1
        # Храним минимальную верхнюю и максимальную нижнюю цены игры
        # (но в строке итерации выводим именно текущие оценки для ЦИ).
        high_price_estimate: float = max(self.player_a_accumulated_values) / iteration_k
        self.minmax_price_estimation = min(self.minmax_price_estimation, high_price_estimate)
        low_price_estimate: float = min(self.player_b_accumulated_values) / iteration_k
        self.maxmin_price_estimation = max(self.maxmin_price_estimation, low_price_estimate)

        # Добавляем строку новой итерации к таблице.
        return pd.Series(
            [
                iteration_k,
                self.player_a_strategy_labels[self.player_a_strategy_index],
                self.player_b_strategy_labels[self.player_b_strategy_index],
                *self.player_a_accumulated_values,
                *self.player_b_accumulated_values,
                high_price_estimate,
                low_price_estimate,
                self.minmax_price_estimation - self.maxmin_price_estimation,
            ],
            index=previous_row.index,
        )

    def __choose_strategies(self, mode="random") -> tuple[IndexType, IndexType]:
        """
        Выбирает индексы лучших стратегий игроков для следующей итерации алгоритма.
        :param str mode: Регламентирует, как выбирать стратегии в случае коллизий.
          - "random" (default) - использует случайную из лучших стратегий;
          - "previous" - использует стратегию с прошлой итерации.
        :return: Кортеж из двух индексов лучших стратегий для игроков A и B соответственно.
        """
        # Находим лучшие стратегии игрока A.
        max_strategy_indexes: np.ndarray[IndexType] = np.flatnonzero(
            self.player_a_accumulated_values == np.max(self.player_a_accumulated_values)
        )
        # Находим лучшие стратегии игрока B.
        min_strategy_indexes: np.ndarray[IndexType] = np.flatnonzero(
            self.player_b_accumulated_values == np.min(self.player_b_accumulated_values)
        )

        chosen_strategy_indexes = [self.player_a_strategy_index, self.player_b_strategy_index]
        for i, best_strategy_indexes in enumerate((max_strategy_indexes, min_strategy_indexes)):
            if len(best_strategy_indexes) == 1:
                chosen_strategy_indexes[i] = best_strategy_indexes[0]
                continue

            # Если происходит коллизия лучших стратегий, производим отбор
            match mode:
                case "previous":
                    # Ничего не делаем, оставляем предыдущую стратегию.
                    continue
                case "random":
                    # Если их несколько, рандомим между ними.
                    chosen_strategy_indexes[i] = random.choice(best_strategy_indexes)
                case _:
                    err_msg = f"Invalid mode {mode}"
                    raise MatrixGameException(err_msg)

        return tuple(chosen_strategy_indexes)
```

ПРИЛОЖЕНИЕ Б

Ссылка на репозиторий с исходным кодом задачи

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

ПРИЛОЖЕНИЕ В

Таблица итераций алгоритма Брауна–Робинсон

k	A	B	a1	a2	a3	b1	b2	b3	ВЦИ	НЦИ	ε
1	a1	b3	11	8	2	1	11	11	11.0	1.0	10.0
2	a1	b1	12	15	18	2	22	22	9.0	1.0	8.0
3	a3	b1	13	22	34	18	28	24	11.333	6.0	3.0
4	a3	b1	14	29	50	34	34	26	12.5	6.5	2.5
5	a3	b3	25	37	52	50	40	28	10.4	5.6	2.5
6	a3	b3	36	45	54	66	46	30	9.0	5.0	2.5
7	a3	b3	47	53	56	82	52	32	8.0	4.571	1.5
8	a3	b3	58	61	58	98	58	34	7.625	4.25	1.125
9	a2	b3	69	69	60	105	63	42	7.667	4.667	1.125
10	a2	b3	80	77	62	112	68	50	8.0	5.0	1.125
11	a1	b3	91	85	64	113	79	61	8.273	5.545	1.125
12	a1	b3	102	93	66	114	90	72	8.5	6.0	1.125
13	a1	b3	113	101	68	115	101	83	8.692	6.385	1.125
14	a1	b3	124	109	70	116	112	94	8.857	6.714	0.911
15	a1	b3	135	117	72	117	123	105	9.0	7.0	0.625
16	a1	b3	146	125	74	118	134	116	9.125	7.25	0.375
17	a1	b3	157	133	76	119	145	127	9.235	7.0	0.375
18	a1	b1	158	140	92	120	156	138	8.778	6.667	0.375
19	a1	b1	159	147	108	121	167	149	8.368	6.368	0.375
20	a1	b1	160	154	124	122	178	160	8.0	6.1	0.375
21	a1	b1	161	161	140	123	189	171	7.667	5.857	0.375
22	a1	b1	162	168	156	124	200	182	7.636	5.636	0.375
23	a2	b1	163	175	172	131	205	190	7.609	5.696	0.359
24	a2	b1	164	182	188	138	210	198	7.833	5.75	0.359
25	a3	b1	165	189	204	154	216	200	8.16	6.16	0.359
26	a3	b1	166	196	220	170	222	202	8.462	6.538	0.359
27	a3	b1	167	203	236	186	228	204	8.741	6.889	0.359
28	a3	b1	168	210	252	202	234	206	9.0	7.214	0.359
29	a3	b1	169	217	268	218	240	208	9.241	7.172	0.359
30	a3	b3	180	225	270	234	246	210	9.0	7.0	0.359
31	a3	b3	191	233	272	250	252	212	8.774	6.839	0.359
32	a3	b3	202	241	274	266	258	214	8.562	6.688	0.359
33	a3	b3	213	249	276	282	264	216	8.364	6.545	0.359
34	a3	b3	224	257	278	298	270	218	8.176	6.412	0.359
35	a3	b3	235	265	280	314	276	220	8.0	6.286	0.359
36	a3	b3	246	273	282	330	282	222	7.833	6.167	0.359
37	a3	b3	257	281	284	346	288	224	7.676	6.054	0.359
38	a3	b3	268	289	286	362	294	226	7.605	5.947	0.355
39	a2	b3	279	297	288	369	299	234	7.615	6.0	0.355
40	a2	b3	290	305	290	376	304	242	7.625	6.05	0.355
41	a2	b3	301	313	292	383	309	250	7.634	6.098	0.355
42	a2	b3	312	321	294	390	314	258	7.643	6.143	0.355
43	a2	b3	323	329	296	397	319	266	7.651	6.186	0.355
44	a2	b3	334	337	298	404	324	274	7.659	6.227	0.355
45	a2	b3	345	345	300	411	329	282	7.667	6.267	0.355
46	a2	b3	356	353	302	418	334	290	7.739	6.304	0.355
47	a1	b3	367	361	304	419	345	301	7.809	6.404	0.355
48	a1	b3	378	369	306	420	356	312	7.875	6.5	0.355
49	a1	b3	389	377	308	421	367	323	7.939	6.592	0.355
50	a1	b3	400	385	310	422	378	334	8.0	6.68	0.355
51	a1	b3	411	393	312	423	389	345	8.059	6.765	0.355
52	a1	b3	422	401	314	424	400	356	8.115	6.846	0.355
53	a1	b3	433	409	316	425	411	367	8.17	6.925	0.355
54	a1	b3	444	417	318	426	422	378	8.222	7.0	0.355
55	a1	b3	455	425	320	427	433	389	8.273	7.073	0.355
56	a1	b3	466	433	322	428	444	400	8.321	7.143	0.355
57	a1	b3	477	441	324	429	455	411	8.368	7.211	0.355
58	a1	b3	488	449	326	430	466	422	8.414	7.276	0.329
59	a1	b3	499	457	328	431	477	433	8.458	7.305	0.3
60	a1	b1	500	464	344	432	488	444	8.333	7.2	0.3
61	a1	b1	501	471	360	433	499	455	8.213	7.098	0.3
62	a1	b1	502	478	376	434	510	466	8.097	7.0	0.3
63	a1	b1	503	485	392	435	521	477	7.984	6.905	0.3
64	a1	b1	504	492	408	436	532	488	7.875	6.812	0.3
65	a1	b1	505	499	424	437	543	499	7.769	6.723	0.3
66	a1	b1	506	506	440	438	554	510	7.667	6.636	0.3
67	a1	b1	507	513	456	439	565	521	7.657	6.552	0.3
68	a2	b1	508	520	472	446	570	529	7.647	6.559	0.3
69	a2	b1	509	527	488	453	575	537	7.638	6.565	0.3

70	a2	b1	510	534	504	460	580	545	7.629	6.571	0.3
71	a2	b1	511	541	520	467	585	553	7.62	6.577	0.3
72	a2	b1	512	548	536	474	590	561	7.611	6.583	0.3
73	a2	b1	513	555	552	481	595	569	7.603	6.589	0.298
74	a2	b1	514	562	568	488	600	577	7.676	6.595	0.298
75	a3	b1	515	569	584	504	606	579	7.787	6.72	0.298
76	a3	b1	516	576	600	520	612	581	7.895	6.842	0.298
77	a3	b1	517	583	616	536	618	583	8.0	6.961	0.298
78	a3	b1	518	590	632	552	624	585	8.103	7.077	0.298
79	a3	b1	519	597	648	568	630	587	8.203	7.19	0.298
80	a3	b1	520	604	664	584	636	589	8.3	7.3	0.298
81	a3	b1	521	611	680	600	642	591	8.395	7.296	0.298
82	a3	b3	532	619	682	616	648	593	8.317	7.232	0.298
83	a3	b3	543	627	684	632	654	595	8.241	7.169	0.298
84	a3	b3	554	635	686	648	660	597	8.167	7.107	0.298
85	a3	b3	565	643	688	664	666	599	8.094	7.047	0.298
86	a3	b3	576	651	690	680	672	601	8.023	6.988	0.298
87	a3	b3	587	659	692	696	678	603	7.954	6.931	0.298
88	a3	b3	598	667	694	712	684	605	7.886	6.875	0.298
89	a3	b3	609	675	696	728	690	607	7.82	6.82	0.298
90	a3	b3	620	683	698	744	696	609	7.756	6.767	0.298
91	a3	b3	631	691	700	760	702	611	7.692	6.714	0.298
92	a3	b3	642	699	702	776	708	613	7.63	6.663	0.298
93	a3	b3	653	707	704	792	714	615	7.602	6.613	0.297
94	a2	b3	664	715	706	799	719	623	7.606	6.628	0.297
95	a2	b3	675	723	708	806	724	631	7.611	6.642	0.297
96	a2	b3	686	731	710	813	729	639	7.615	6.656	0.297
97	a2	b3	697	739	712	820	734	647	7.619	6.67	0.297
98	a2	b3	708	747	714	827	739	655	7.622	6.684	0.297
99	a2	b3	719	755	716	834	744	663	7.626	6.697	0.297
100	a2	b3	730	763	718	841	749	671	7.63	6.71	0.297
101	a2	b3	741	771	720	848	754	679	7.634	6.723	0.297
102	a2	b3	752	779	722	855	759	687	7.637	6.735	0.297
103	a2	b3	763	787	724	862	764	695	7.641	6.748	0.297
104	a2	b3	774	795	726	869	769	703	7.644	6.76	0.297
105	a2	b3	785	803	728	876	774	711	7.648	6.771	0.297
106	a2	b3	796	811	730	883	779	719	7.651	6.783	0.297
107	a2	b3	807	819	732	890	784	727	7.654	6.794	0.297
108	a2	b3	818	827	734	897	789	735	7.657	6.806	0.297
109	a2	b3	829	835	736	904	794	743	7.661	6.817	0.297
110	a2	b3	840	843	738	911	799	751	7.664	6.827	0.297
111	a2	b3	851	851	740	918	804	759	7.667	6.838	0.297
112	a2	b3	862	859	742	925	809	767	7.696	6.848	0.297
113	a1	b3	873	867	744	926	820	778	7.726	6.885	0.297
114	a1	b3	884	875	746	927	831	789	7.754	6.921	0.297
115	a1	b3	895	883	748	928	842	800	7.783	6.957	0.297
116	a1	b3	906	891	750	929	853	811	7.81	6.991	0.297
117	a1	b3	917	899	752	930	864	822	7.838	7.026	0.297
118	a1	b3	928	907	754	931	875	833	7.864	7.059	0.297
119	a1	b3	939	915	756	932	886	844	7.891	7.092	0.297
120	a1	b3	950	923	758	933	897	855	7.917	7.125	0.297
121	a1	b3	961	931	760	934	908	866	7.942	7.157	0.297
122	a1	b3	972	939	762	935	919	877	7.967	7.189	0.297
123	a1	b3	983	947	764	936	930	888	7.992	7.22	0.297
124	a1	b3	994	955	766	937	941	899	8.016	7.25	0.297
125	a1	b3	1005	963	768	938	952	910	8.04	7.28	0.297
126	a1	b3	1016	971	770	939	963	921	8.063	7.31	0.293
127	a1	b3	1027	979	772	940	974	932	8.087	7.339	0.264
128	a1	b3	1038	987	774	941	985	943	8.109	7.352	0.251
129	a1	b1	1039	994	790	942	996	954	8.054	7.302	0.251
130	a1	b1	1040	1001	806	943	1007	965	8.0	7.254	0.251
131	a1	b1	1041	1008	822	944	1018	976	7.947	7.206	0.251
132	a1	b1	1042	1015	838	945	1029	987	7.894	7.159	0.251
133	a1	b1	1043	1022	854	946	1040	998	7.842	7.113	0.251
134	a1	b1	1044	1029	870	947	1051	1009	7.791	7.067	0.251
135	a1	b1	1045	1036	886	948	1062	1020	7.741	7.022	0.251
136	a1	b1	1046	1043	902	949	1073	1031	7.691	6.978	0.251
137	a1	b1	1047	1050	918	950	1084	1042	7.664	6.934	0.251
138	a2	b1	1048	1057	934	957	1089	1050	7.659	6.935	0.251
139	a2	b1	1049	1064	950	964	1094	1058	7.655	6.935	0.251
140	a2	b1	1050	1071	966	971	1099	1066	7.65	6.936	0.251
141	a2	b1	1051	1078	982	978	1104	1074	7.645	6.936	0.251
142	a2	b1	1052	1085	998	985	1109	1082	7.641	6.937	0.251
143	a2	b1	1053	1092	1014	992	1114	1090	7.636	6.937	0.251
144	a2	b1	1054	1099	1030	999	1119	1098	7.632	6.938	0.251
145	a2	b1	1055	1106	1046	1006	1124	1106	7.628	6.938	0.251
146	a2	b1	1056	1113	1062	1013	1129	1114	7.623	6.938	0.251

147	a2	b1	1057	1120	1078	1020	1134	1122	7.619	6.939	0.251
148	a2	b1	1058	1127	1094	1027	1139	1130	7.615	6.939	0.251
149	a2	b1	1059	1134	1110	1034	1144	1138	7.611	6.94	0.251
150	a2	b1	1060	1141	1126	1041	1149	1146	7.607	6.94	0.251
151	a2	b1	1061	1148	1142	1048	1154	1154	7.603	6.94	0.251
152	a2	b1	1062	1155	1158	1055	1159	1162	7.618	6.941	0.251
153	a3	b1	1063	1162	1174	1071	1165	1164	7.673	7.0	0.251
154	a3	b1	1064	1169	1190	1087	1171	1166	7.727	7.058	0.251
155	a3	b1	1065	1176	1206	1103	1177	1168	7.781	7.116	0.251
156	a3	b1	1066	1183	1222	1119	1183	1170	7.833	7.173	0.251
157	a3	b1	1067	1190	1238	1135	1189	1172	7.885	7.229	0.251
158	a3	b1	1068	1197	1254	1151	1195	1174	7.937	7.285	0.251
159	a3	b1	1069	1204	1270	1167	1201	1176	7.987	7.34	0.251
160	a3	b1	1070	1211	1286	1183	1207	1178	8.038	7.362	0.24
161	a3	b3	1081	1219	1288	1199	1213	1180	8.0	7.329	0.24
162	a3	b3	1092	1227	1290	1215	1219	1182	7.963	7.296	0.24
163	a3	b3	1103	1235	1292	1231	1225	1184	7.926	7.264	0.24
164	a3	b3	1114	1243	1294	1247	1231	1186	7.89	7.232	0.24
165	a3	b3	1125	1251	1296	1263	1237	1188	7.855	7.2	0.24
166	a3	b3	1136	1259	1298	1279	1243	1190	7.819	7.169	0.24
167	a3	b3	1147	1267	1300	1295	1249	1192	7.784	7.138	0.24
168	a3	b3	1158	1275	1302	1311	1255	1194	7.75	7.107	0.24
169	a3	b3	1169	1283	1304	1327	1261	1196	7.716	7.077	0.24
170	a3	b3	1180	1291	1306	1343	1267	1198	7.682	7.047	0.24
171	a3	b3	1191	1299	1308	1359	1273	1200	7.649	7.018	0.24
172	a3	b3	1202	1307	1310	1375	1279	1202	7.616	6.988	0.24
173	a3	b3	1213	1315	1312	1391	1285	1204	7.601	6.96	0.239
174	a2	b3	1224	1323	1314	1398	1290	1212	7.603	6.966	0.239
175	a2	b3	1235	1331	1316	1405	1295	1220	7.606	6.971	0.239
176	a2	b3	1246	1339	1318	1412	1300	1228	7.608	6.977	0.239
177	a2	b3	1257	1347	1320	1419	1305	1236	7.61	6.983	0.239
178	a2	b3	1268	1355	1322	1426	1310	1244	7.612	6.989	0.239
179	a2	b3	1279	1363	1324	1433	1315	1252	7.615	6.994	0.239
180	a2	b3	1290	1371	1326	1440	1320	1260	7.617	7.0	0.239
181	a2	b3	1301	1379	1328	1447	1325	1268	7.619	7.006	0.239
182	a2	b3	1312	1387	1330	1454	1330	1276	7.621	7.011	0.239
183	a2	b3	1323	1395	1332	1461	1335	1284	7.623	7.016	0.239
184	a2	b3	1334	1403	1334	1468	1340	1292	7.625	7.022	0.239
185	a2	b3	1345	1411	1336	1475	1345	1300	7.627	7.027	0.239
186	a2	b3	1356	1419	1338	1482	1350	1308	7.629	7.032	0.239
187	a2	b3	1367	1427	1340	1489	1355	1316	7.631	7.037	0.239
188	a2	b3	1378	1435	1342	1496	1360	1324	7.633	7.043	0.239
189	a2	b3	1389	1443	1344	1503	1365	1332	7.635	7.048	0.239
190	a2	b3	1400	1451	1346	1510	1370	1340	7.637	7.053	0.239
191	a2	b3	1411	1459	1348	1517	1375	1348	7.639	7.058	0.239
192	a2	b3	1422	1467	1350	1524	1380	1356	7.641	7.062	0.239
193	a2	b3	1433	1475	1352	1531	1385	1364	7.642	7.067	0.239
194	a2	b3	1444	1483	1354	1538	1390	1372	7.644	7.072	0.239
195	a2	b3	1455	1491	1356	1545	1395	1380	7.646	7.077	0.239
196	a2	b3	1466	1499	1358	1552	1400	1388	7.648	7.082	0.239
197	a2	b3	1477	1507	1360	1559	1405	1396	7.65	7.086	0.239
198	a2	b3	1488	1515	1362	1566	1410	1404	7.652	7.091	0.239
199	a2	b3	1499	1523	1364	1573	1415	1412	7.653	7.095	0.239
200	a2	b3	1510	1531	1366	1580	1420	1420	7.655	7.1	0.239
201	a2	b3	1521	1539	1368	1587	1425	1428	7.657	7.09	0.239
202	a2	b2	1532	1544	1374	1594	1430	1436	7.644	7.079	0.239
203	a2	b2	1543	1549	1380	1601	1435	1444	7.631	7.069	0.239
204	a2	b2	1554	1554	1386	1608	1440	1452	7.618	7.059	0.239
205	a2	b2	1565	1559	1392	1615	1445	1460	7.634	7.049	0.239
206	a1	b2	1576	1564	1398	1616	1456	1471	7.65	7.068	0.239
207	a1	b2	1587	1569	1404	1617	1467	1482	7.667	7.087	0.239
208	a1	b2	1598	1574	1410	1618	1478	1493	7.683	7.106	0.239
209	a1	b2	1609	1579	1416	1619	1489	1504	7.699	7.124	0.239
210	a1	b2	1620	1584	1422	1620	1500	1515	7.714	7.143	0.239
211	a1	b2	1631	1589	1428	1621	1511	1526	7.73	7.161	0.239
212	a1	b2	1642	1594	1434	1622	1522	1537	7.745	7.179	0.239
213	a1	b2	1653	1599	1440	1623	1533	1548	7.761	7.197	0.239
214	a1	b2	1664	1604	1446	1624	1544	1559	7.776	7.215	0.239
215	a1	b2	1675	1609	1452	1625	1555	1570	7.791	7.233	0.239
216	a1	b2	1686	1614	1458	1626	1566	1581	7.806	7.25	0.239
217	a1	b2	1697	1619	1464	1627	1577	1592	7.82	7.267	0.239
218	a1	b2	1708	1624	1470	1628	1588	1603	7.835	7.284	0.239
219	a1	b2	1719	1629	1476	1629	1599	1614	7.849	7.301	0.239
220	a1	b2	1730	1634	1482	1630	1610	1625	7.864	7.318	0.239
221	a1	b2	1741	1639	1488	1631	1621	1636	7.878	7.335	0.239
222	a1	b2	1752	1644	1494	1632	1632	1647	7.892	7.351	0.239
223	a1	b2	1763	1649	1500	1633	1643	1658	7.906	7.323	0.239

224	a1	b1	1764	1656	1516	1634	1654	1669	7.875	7.295	0.239
225	a1	b1	1765	1663	1532	1635	1665	1680	7.844	7.267	0.239
226	a1	b1	1766	1670	1548	1636	1676	1691	7.814	7.239	0.239
227	a1	b1	1767	1677	1564	1637	1687	1702	7.784	7.211	0.239
228	a1	b1	1768	1684	1580	1638	1698	1713	7.754	7.184	0.239
229	a1	b1	1769	1691	1596	1639	1709	1724	7.725	7.157	0.239
230	a1	b1	1770	1698	1612	1640	1720	1735	7.696	7.13	0.239
231	a1	b1	1771	1705	1628	1641	1731	1746	7.667	7.104	0.239
232	a1	b1	1772	1712	1644	1642	1742	1757	7.638	7.078	0.239
233	a1	b1	1773	1719	1660	1643	1753	1768	7.609	7.052	0.239
234	a1	b1	1774	1726	1676	1644	1764	1779	7.581	7.026	0.219
235	a1	b1	1775	1733	1692	1645	1775	1790	7.553	7.0	0.191
236	a1	b1	1776	1740	1708	1646	1786	1801	7.525	6.975	0.163
237	a1	b1	1777	1747	1724	1647	1797	1812	7.498	6.949	0.135
238	a1	b1	1778	1754	1740	1648	1808	1823	7.471	6.924	0.108
239	a1	b1	1779	1761	1756	1649	1819	1834	7.444	6.9	0.081