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.

```
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')
```

```
# 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
```

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

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 2 (см. рисунок 2). Нижняя и верхняя цены игры для данной матрицы: 7 и 13. В данном случае доминируемых стратегий нет (см. рисунок 3), поэтому перейдём сразу к решению игры в смешанных стратегиях.

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

```
Прибавили ко всем элементам исходной матрицы 2
```

```
+-------------------------------------------------+
|           Таблица стратегий (игрока A)          |
+-----------------+----+----+----+-----------------+
|   Стратегии     | b1 | b2 | b3 | MIN выигрыш A |
+-----------------+----+----+----+-----------------+
|       a1        | 3  | 13 | 13 |        3        |
|       a2        | 9  | 7  | 10 |        7        |
|       a3        | 18 | 8  | 0  |        0        |
| MAX проигрыш B | 18 | 13 | 13 |                 |
+-----------------+----+----+----+-----------------+
```

```
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: 13
```

Рисунок 2 – Нормализация прямоугольной матрицы игры

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

```
+-------------------------------------------------+
|           Таблица стратегий (игрока A)          |
+-----------------+----+----+----+-----------------+
|   Стратегии     | b1 | b2 | b3 | MIN выигрыш A |
+-----------------+----+----+----+-----------------+
|       a1        | 3  | 13 | 13 |        3        |
|       a2        | 9  | 7  | 10 |        7        |
|       a3        | 18 | 8  | 0  |        0        |
| MAX проигрыш B | 18 | 13 | 13 |                 |
+-----------------+----+----+----+-----------------+
```

Рисунок 3 – Проверка отсутствия доминируемых стратегий

2. 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 4 и 5 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: 7 <= 8.962 <= 13
+-------------------------------+
| Смешанные стратегии игрока A |
+---------+---------+-----------+
|    a1   |    a2   |    a3     |
+---------+---------+-----------+
|  0.295  |  0.513  |  0.192    |
+---------+---------+-----------+
```

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

```
1.0
```

Рисунок 4 – 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: 7 <= 8.962 <= 13
+-------------------------------+
| Смешанные стратегии игрока B |
+---------+---------+-----------+
|    b1   |    b2   |    b3     |
+---------+---------+-----------+
|  0.404  |  0.212  |  0.385    |
+---------+---------+-----------+
```

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

```
1.0
```

Рисунок 5 – Аналитический метод поиска смешанных стратегий игрока B

Полученные цена игры и смешанные стратегии игроков:

$$8.962;\ S_A: (0.295, 0.513, 0.192);\ S_B: (0.404, 0.212, 0.385).$$

Возвращаясь к исходной матрице, получим итоговый ответ:

$$6.962;\ S_A: (0.295, 0.513, 0.192);\ S_B: (0.404, 0.212, 0.385).$$

3. Численный метод Брауна-Робинсон

Метод Брауна-Робинсон является приближённым численным методом решения произвольной игры Г 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 6 представлена таблица итеративного алгоритма Брауна-Робинсон для решения нормализованной матричной игры. Для решения задачи потребовалось 102 итерации 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       | 3  | 13 | 13 |       3        |
|       a2       | 9  | 7  | 10 |       7        |
|       a3       | 18 | 8  | 0  |       0        |
| MAX проигрыш B | 18 | 13 | 13 |                |
+----------------+----+----+----+----------------+
```

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

	k	A	B	a1	a2	a3	b1	b2	b3	ВЦИ	НЦИ	ε
0	1	a3	b1	3	9	18	18	8	0	18.000	0.000	18.000
1	2	a3	b3	16	19	18	36	16	0	9.500	0.000	9.500
2	3	a2	b3	29	29	18	45	23	10	9.667	3.333	6.167
3	4	a1	b3	42	39	18	48	36	23	10.500	5.750	3.750
4	5	a1	b3	55	49	18	51	49	36	11.000	7.200	2.300
...	...	...	...	...	...	...	...	...	...	...	...	...
97	98	a1	b1	904	880	834	825	893	926	9.224	8.418	0.331
98	99	a1	b1	907	889	852	828	906	939	9.162	8.364	0.273
99	100	a1	b1	910	898	870	831	919	952	9.100	8.310	0.211
100	101	a1	b1	913	907	888	834	932	965	9.040	8.257	0.151
101	102	a1	b1	916	916	906	837	945	978	8.980	8.206	0.092

102 rows × 12 columns

Рисунок 6 – Таблица итеративного алгоритма Брауна-Робинсон

Для расчёта приближённых смешанных стратегий игроков достаточно произвести подсчёт частот использования стратегий. На рисунке 7 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: 7 <= 8.980 <= 13
```

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

```
+-----------------------------+
| Смешанные стратегии игрока A |
+---------+---------+---------+
|    a1   |    a2   |    a3   |
+---------+---------+---------+
|  0.353  |  0.500  |  0.147  |
+---------+---------+---------+
```

```
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   |
+---------+---------+---------+
|  0.402  |  0.206  |  0.392  |
+---------+---------+---------+
```

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

```
1.0
```

Рисунок 7 – Найденные приближенное решение нормализованной матричной игры методом Брауна-Робинсон

Полученные оценки цены игры и смешанных стратегий игроков:

$$8.980; \widetilde{S_A}[102]: (0.353, 0.500, 0.147);\ \widetilde{S_B}[102]: (0.402, 0.206, 0.392).$$

Возвращаясь к исходной матрице, получим итоговый ответ:

$$6.980;\ \widetilde{S_A}[102]: (0.353, 0.500, 0.147);\ \widetilde{S_B}[102]: (0.402, 0.206, 0.392).$$

4. 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 8 и 9 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	6,962	0,295	0,513	0,192	0,404	0,212	0,385
Численный метод Брауна-Робинсон	6,980	0,353	0,500	0,147	0,402	0,206	0,392
Абсолютная погрешность вычислений, Δ	0,018	0,058	0,013	0,045	0,002	0,006	0,007
Относительная погрешность вычислений, %	0,3	19,7	2,5	23,4	0,5	2,8	1,8

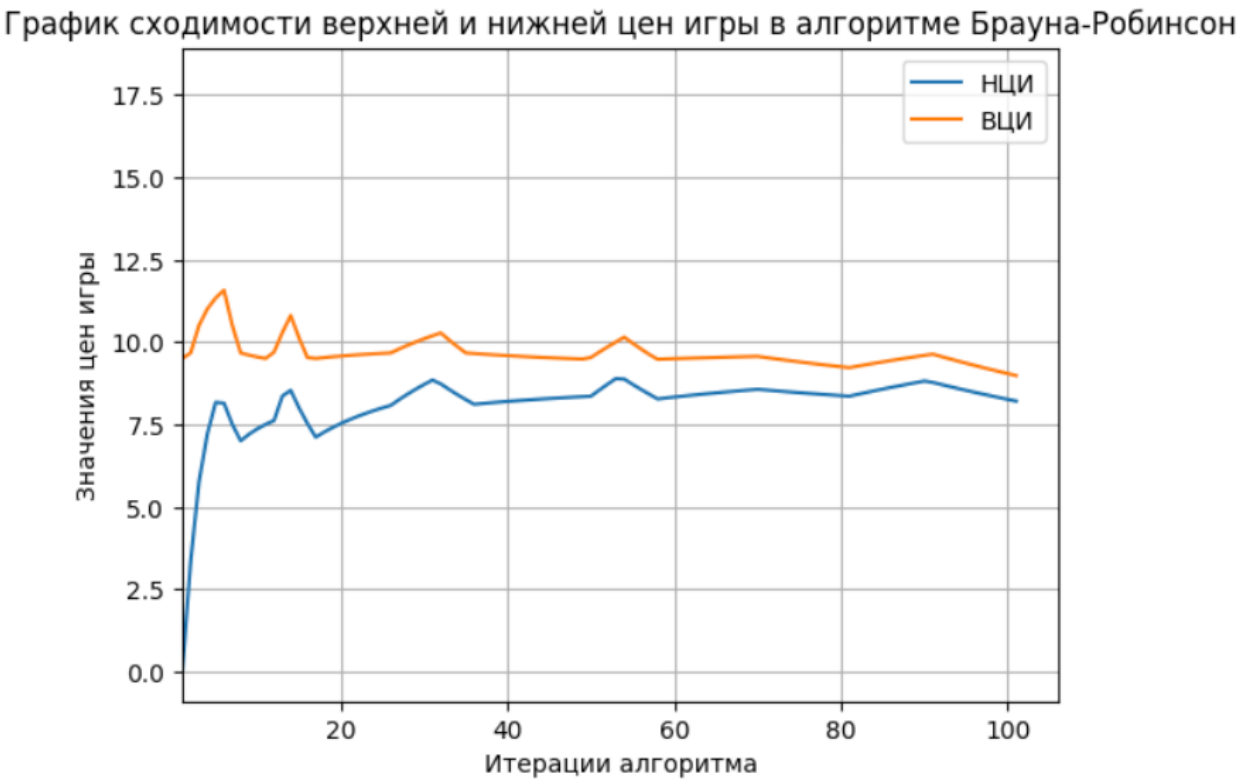

Рисунок 8 – График сходимости верхней и нижней цен игры в алгоритме Брауна–Робинсон

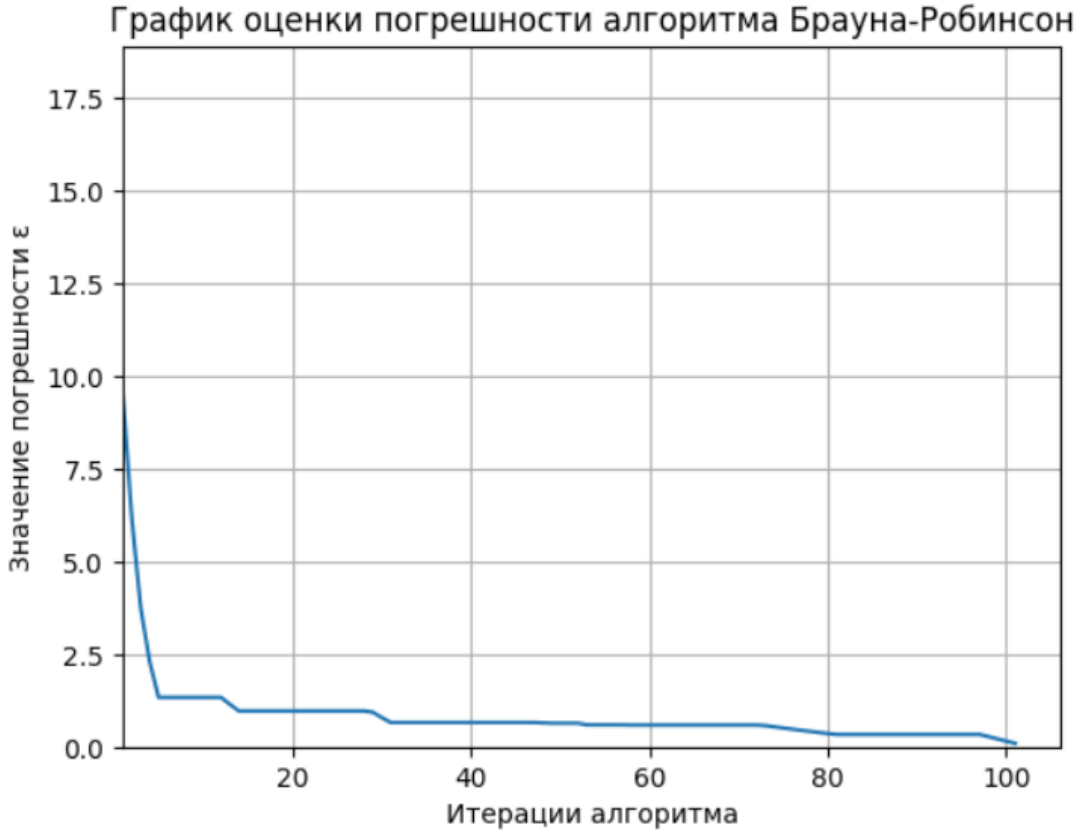

Рисунок 9 – График оценки погрешности алгоритма Брауна–Робинсон

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:

$$6.962;\ S_A: (0.295, 0.513, 0.192);\ S_B: (0.404, 0.212, 0.385).$$

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:

$$6.980;\ \widetilde{S_A}[102]: (0.353, 0.500, 0.147);\ \widetilde{S_B}[102]: (0.402, 0.206, 0.392).$$

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 4 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	a3	b1	3	9	18	18	8	0	18.0	0.0	18.0
2	a3	b3	16	19	18	36	16	0	9.5	0.0	9.5
3	a2	b3	29	29	18	45	23	10	9.667	3.333	6.167
4	a1	b3	42	39	18	48	36	23	10.5	5.75	3.75
5	a1	b3	55	49	18	51	49	36	11.0	7.2	2.3
6	a1	b3	68	59	18	54	62	49	11.333	8.167	1.333
7	a1	b3	81	69	18	57	75	62	11.571	8.143	1.333
8	a1	b1	84	78	36	60	88	75	10.5	7.5	1.333
9	a1	b1	87	87	54	63	101	88	9.667	7.0	1.333
10	a2	b1	90	96	72	72	108	98	9.6	7.2	1.333
11	a2	b1	93	105	90	81	115	108	9.545	7.364	1.333
12	a2	b1	96	114	108	90	122	118	9.5	7.5	1.333
13	a2	b1	99	123	126	99	129	128	9.692	7.615	1.333
14	a3	b1	102	132	144	117	137	128	10.286	8.357	1.143
15	a3	b1	105	141	162	135	145	128	10.8	8.533	0.967
16	a3	b3	118	151	162	153	153	128	10.125	8.0	0.967
17	a3	b3	131	161	162	171	161	128	9.529	7.529	0.967
18	a3	b3	144	171	162	189	169	128	9.5	7.111	0.967
19	a2	b3	157	181	162	198	176	138	9.526	7.263	0.967
20	a2	b3	170	191	162	207	183	148	9.55	7.4	0.967
21	a2	b3	183	201	162	216	190	158	9.571	7.524	0.967
22	a2	b3	196	211	162	225	197	168	9.591	7.636	0.967
23	a2	b3	209	221	162	234	204	178	9.609	7.739	0.967
24	a2	b3	222	231	162	243	211	188	9.625	7.833	0.967
25	a2	b3	235	241	162	252	218	198	9.64	7.92	0.967
26	a2	b3	248	251	162	261	225	208	9.654	8.0	0.967
27	a2	b3	261	261	162	270	232	218	9.667	8.074	0.967
28	a1	b3	274	271	162	273	245	231	9.786	8.25	0.967
29	a1	b3	287	281	162	276	258	244	9.897	8.414	0.967
30	a1	b3	300	291	162	279	271	257	10.0	8.567	0.933
31	a1	b3	313	301	162	282	284	270	10.097	8.71	0.79
32	a1	b3	326	311	162	285	297	283	10.188	8.844	0.656
33	a1	b3	339	321	162	288	310	296	10.273	8.727	0.656
34	a1	b1	342	330	180	291	323	309	10.059	8.559	0.656
35	a1	b1	345	339	198	294	336	322	9.857	8.4	0.656
36	a1	b1	348	348	216	297	349	335	9.667	8.25	0.656
37	a1	b1	351	357	234	300	362	348	9.649	8.108	0.656
38	a2	b1	354	366	252	309	369	358	9.632	8.132	0.656
39	a2	b1	357	375	270	318	376	368	9.615	8.154	0.656
40	a2	b1	360	384	288	327	383	378	9.6	8.175	0.656
41	a2	b1	363	393	306	336	390	388	9.585	8.195	0.656
42	a2	b1	366	402	324	345	397	398	9.571	8.214	0.656
43	a2	b1	369	411	342	354	404	408	9.558	8.233	0.656
44	a2	b1	372	420	360	363	411	418	9.545	8.25	0.656
45	a2	b1	375	429	378	372	418	428	9.533	8.267	0.656
46	a2	b1	378	438	396	381	425	438	9.522	8.283	0.656
47	a2	b1	381	447	414	390	432	448	9.511	8.298	0.656
48	a2	b1	384	456	432	399	439	458	9.5	8.312	0.656
49	a2	b1	387	465	450	408	446	468	9.49	8.327	0.646
50	a2	b1	390	474	468	417	453	478	9.48	8.34	0.636
51	a2	b1	393	483	486	426	460	488	9.529	8.353	0.636
52	a3	b1	396	492	504	444	468	488	9.692	8.538	0.636
53	a3	b1	399	501	522	462	476	488	9.849	8.717	0.636

54	a3	b1	402	510	540	480	484	488	10.0	8.889	0.591
55	a3	b1	405	519	558	498	492	488	10.145	8.873	0.591
56	a3	b3	418	529	558	516	500	488	9.964	8.714	0.591
57	a3	b3	431	539	558	534	508	488	9.789	8.561	0.591
58	a3	b3	444	549	558	552	516	488	9.621	8.414	0.591
59	a3	b3	457	559	558	570	524	488	9.475	8.271	0.586
60	a2	b3	470	569	558	579	531	498	9.483	8.3	0.586
61	a2	b3	483	579	558	588	538	508	9.492	8.328	0.586
62	a2	b3	496	589	558	597	545	518	9.5	8.355	0.586
63	a2	b3	509	599	558	606	552	528	9.508	8.381	0.586
64	a2	b3	522	609	558	615	559	538	9.516	8.406	0.586
65	a2	b3	535	619	558	624	566	548	9.523	8.431	0.586
66	a2	b3	548	629	558	633	573	558	9.53	8.455	0.586
67	a2	b3	561	639	558	642	580	568	9.537	8.478	0.586
68	a2	b3	574	649	558	651	587	578	9.544	8.5	0.586
69	a2	b3	587	659	558	660	594	588	9.551	8.522	0.586
70	a2	b3	600	669	558	669	601	598	9.557	8.543	0.586
71	a2	b3	613	679	558	678	608	608	9.563	8.563	0.586
72	a2	b2	626	686	566	687	615	618	9.528	8.542	0.586
73	a2	b2	639	693	574	696	622	628	9.493	8.521	0.586
74	a2	b2	652	700	582	705	629	638	9.459	8.5	0.571
75	a2	b2	665	707	590	714	636	648	9.427	8.48	0.538
76	a2	b2	678	714	598	723	643	658	9.395	8.461	0.506
77	a2	b2	691	721	606	732	650	668	9.364	8.442	0.475
78	a2	b2	704	728	614	741	657	678	9.333	8.423	0.444
79	a2	b2	717	735	622	750	664	688	9.304	8.405	0.415
80	a2	b2	730	742	630	759	671	698	9.275	8.388	0.386
81	a2	b2	743	749	638	768	678	708	9.247	8.37	0.358
82	a2	b2	756	756	646	777	685	718	9.22	8.354	0.331
83	a1	b2	769	763	654	780	698	731	9.265	8.41	0.331
84	a1	b2	782	770	662	783	711	744	9.31	8.464	0.331
85	a1	b2	795	777	670	786	724	757	9.353	8.518	0.331
86	a1	b2	808	784	678	789	737	770	9.395	8.57	0.331
87	a1	b2	821	791	686	792	750	783	9.437	8.621	0.331
88	a1	b2	834	798	694	795	763	796	9.477	8.67	0.331
89	a1	b2	847	805	702	798	776	809	9.517	8.719	0.331
90	a1	b2	860	812	710	801	789	822	9.556	8.767	0.331
91	a1	b2	873	819	718	804	802	835	9.593	8.813	0.331
92	a1	b2	886	826	726	807	815	848	9.63	8.772	0.331
93	a1	b1	889	835	744	810	828	861	9.559	8.71	0.331
94	a1	b1	892	844	762	813	841	874	9.489	8.649	0.331
95	a1	b1	895	853	780	816	854	887	9.421	8.589	0.331
96	a1	b1	898	862	798	819	867	900	9.354	8.531	0.331
97	a1	b1	901	871	816	822	880	913	9.289	8.474	0.331
98	a1	b1	904	880	834	825	893	926	9.224	8.418	0.331
99	a1	b1	907	889	852	828	906	939	9.162	8.364	0.273
100	a1	b1	910	898	870	831	919	952	9.1	8.31	0.211
101	a1	b1	913	907	888	834	932	965	9.04	8.257	0.151
102	a1	b1	916	916	906	837	945	978	8.98	8.206	0.092