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 №2
«Аналитический и численный методы решения непрерывной выпукло-вогнутой игры»

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

Найти оптимальные стратегии непрерывной выпукло-вогнутой антагонистической игры аналитическим и численным методами.

Задание

Пусть функция выигрыша (ядро) антагонистической игры, заданной на единичном квадрате, непрерывна:

$$H(x,y) \in C(\Pi), \Pi = [0,1] \times [0,1].$$

Тогда существуют нижняя и верхняя цены игры, и, кроме того:

$$h = \overline{h} \equiv \max_F \min_y E(F,y) = \min_G \max_x E(x,G) \equiv \underline{h},$$

а для среднего выигрыша игры имеют место равенства:

$$E(x,G) = \int_0^1 H(x,y)dG(y), \;\; E(F,y) = \int_0^1 H(x,y)dF(x),$$

где $F(x), G(y)$ – произвольные вероятностные меры выбора стратегий для обоих игроков, заданные на единичном интервале.

Теорема. *Пусть* $\Gamma = (X, Y, H)$, $X \subset R^m$, $Y \subset R^n$ *—вогнуто-выпуклая игра. Тогда значение игры* v *равно*

$$v = \min_y \max_x H(x,y) = \max_x \min_y H(x,y).$$

В игре Γ *всегда существует ситуация равновесия* (x^*, y^*), *где* $x^* \in X$, $y^* \in Y$ *— чистые стратегии игроков* 1 *и* 2, *на которых достигаются внешние экстремумы в* [illegible]. *Если при этом функция* $H(x,y)$ *строго вогнута (выпукла) по переменной* $x(y)$ *при любом фиксированном* $y \in Y$ $(x \in X)$, *то игрок* 1 (2) *имеет единственную оптимальную стратегию, которая является чистой.*

Функция ядра имеет вид:

$$H(x,y) = ax^2 + by^2 + cxy + dx + ey.$$

Параметры функции ядра по варианту, следующие:

a	b	c	d	e
-5	$5/12$	$10/3$	$-2/3$	$-4/3$

Ход работы

Для реализации решения лабораторной работы был использован язык программирования Python. К защите представляется интерактивный блокнот Jupyter Notebook в файле continuous_convex-concave.ipynb (см. приложение А).

В исполняемом «ноутбуке» импортируются реализованный модуль *convex_concave.numeric*, инкапсулирующий логику и алгоритм для численного метода решения игры с непрерывным ядром. Ознакомиться со всем исходным кодом данной работы можно, посетив репозиторий, ссылка на который представлена в приложении Б. Задание входных параметров представлено на рисунке 1.

Лабораторная работа №2

"Непрерывные выпукло-вогнутые игры"

Выполнил: Александров А. Н., ИУ8-104

Вариант: 1

Задание

Функция ядра имеет вид:

$$H(x,y) = ax^2 + by^2 + cxy + dx + ey,$$

где:

a	b	c	d	e
-5	5/12	10/3	-2/3	-4/3

Найти оптимальные стратегии непрерывной выпукло-вогнутой антагонистической игры аналитическим и численным методами.

```
import logging

import sympy
from sympy import N, Eq, abc
# To represent multiple expressions in output of single cell.
from IPython.display import display

from game_theory.utils.continuous_games.convex_concave.numeric import NumericMethod

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

```
ROUND_CONST: int = 3
A, B, C, D, E = (
    -5,
    5 / 12,
    10 / 3,
    -2 / 3,
    -4 / 3,
)
```

Рисунок 1 – Задание входных параметров игры с непрерывным ядром

1. Аналитический метод решения

Функция ядра имеет следующий вид:

$$H(x, y) = -5x^2 + 0{,}417y^2 + 3{,}33xy - 0{,}667x - 1{,}33y.$$

На рисунке 2 представлена проверка вторых частных производных для определения, является ли представленная игра выпукло-вогнутой.

1. Аналитическое решение

```
# Входные параметры: коэффициенты функции ядра.
a, b, c, d, e = sympy.var(("a", "b", "c", "d", "e"))
# Переменные.
x, y = sympy.symbols(("x", "y"))

# Задание функции ядра.
kernel_func = sympy.Lambda(
    (x, y),
    a * x ** 2 + b * y ** 2 + c * x * y + d * x + e * y,
)
kernel_func_subs = kernel_func.subs({a: A, b: B, c: C, d: D, e: E})

display(kernel_func)
N(kernel_func_subs, ROUND_CONST)
```

$$\left((x,\ y) \mapsto ax^2 + by^2 + cxy + dx + ey\right)$$

$$\left((x,\ y) \mapsto -5.0x^2 + 3.33xy - 0.667x + 0.417y^2 - 1.33y\right)$$

```
kernel_xx = sympy.diff(kernel_func(x, y), x, 2, evaluate=False)
kernel_xx_eval = kernel_xx.doit()
kernel_xx_subs = kernel_xx_eval.subs({a: A, b: B, c: C, d: D, e: E})

display(Eq(kernel_xx, kernel_xx_eval))
N(Eq(kernel_xx_eval, kernel_xx_subs), ROUND_CONST)
```

$$\frac{\partial^2}{\partial x^2}\left(ax^2 + by^2 + cxy + dx + ey\right) = 2a$$

$$2.0a = -10.0$$

```
kernel_yy = sympy.diff(kernel_func(x, y), y, 2, evaluate=False)
kernel_yy_eval = kernel_yy.doit()
kernel_yy_subs = kernel_yy_eval.subs({a: A, b: B, c: C, d: D, e: E})

display(Eq(kernel_yy, kernel_yy_eval))
N(Eq(kernel_yy_eval, kernel_yy_subs), ROUND_CONST)
```

$$\frac{\partial^2}{\partial y^2}\left(ax^2 + by^2 + cxy + dx + ey\right) = 2b$$

$$2.0b = 0.833$$

```
is_convex_concave: bool = kernel_xx_subs < 0 < kernel_yy_subs
assert is_convex_concave, (
    "Игра не является выпукло-вогнутой, т.к. для функции ядра одновременно не выполняется
    f"H_xx = {2 * a:.2f} < 0 и H_yy = {2 * b:.2f} > 0"
)
```

Рисунок 2 – Вычисление вторых частных производных для проверки выпукло-вогнутой игры

Находим первые частные производные. Приравняв их к нулю, выражаем соответствующие переменные (см. рисунок 3).

Для нахождения оптимальных стратегий найдем производные функции ядра по каждой переменной

```
# Производная по x.
kernel_x = sympy.diff(kernel_func(x, y), x, evaluate=False)
kernel_x_eval = kernel_x.doit()
kernel_x_subs = kernel_x_eval.subs({a: A, b: B, c: C, d: D, e: E})
# Производная по y.
kernel_y = sympy.diff(kernel_func(x, y), y, evaluate=False)
kernel_y_eval = kernel_y.doit()
kernel_y_subs = kernel_y_eval.subs({a: A, b: B, c: C, d: D, e: E})
```

```
display(Eq(kernel_x, kernel_x_eval))
N(Eq(kernel_x_eval, kernel_x_subs), ROUND_CONST)
```

$$\frac{\partial}{\partial x}\left(ax^2 + by^2 + cxy + dx + ey\right) = 2ax + cy + d$$

$$2.0ax + cy + d = -10.0x + 3.33y - 0.667$$

```
display(Eq(kernel_y, kernel_y_eval))
N(Eq(kernel_y_eval, kernel_y_subs), ROUND_CONST)
```

$$\frac{\partial}{\partial y}\left(ax^2 + by^2 + cxy + dx + ey\right) = 2by + cx + e$$

$$2.0by + cx + e = 3.33x + 0.833y - 1.33$$

После приравнивания производных к нулю получим

```
# Выражаем решение производной через x.
zero_kernel_x, = sympy.solve(Eq(kernel_x_eval, 0), x)
zero_kernel_x_subs = zero_kernel_x.subs({a: A, b: B, c: C, d: D, e: E})
# Выражаем решение производной через y.
zero_kernel_y, = sympy.solve(Eq(kernel_y_eval, 0), y)
zero_kernel_y_subs = zero_kernel_y.subs({a: A, b: B, c: C, d: D, e: E})
```

```
display(Eq(x, zero_kernel_x))
display(N(Eq(x, zero_kernel_x_subs), ROUND_CONST))
```

$$x = \frac{-cy - d}{2a}$$

$$x = 0.333y - 0.0667$$

```
display(Eq(y, zero_kernel_y))
display(N(Eq(y, zero_kernel_y_subs), ROUND_CONST))
```

$$y = \frac{-cx - e}{2b}$$

$$y = 1.6 - 4.0x$$

Рисунок 3 – Нахождение частных производных по переменным для определения оптимальных стратегий

Беря во внимание, что x и y принимают неотрицательные значения для оптимальных стратегий, задаём их кусочно. Находим решение игры, решая систему из двух кусочно-заданных выражений для x и y и подставляя полученные значения в функцию ядра H (см. рисунок 4).

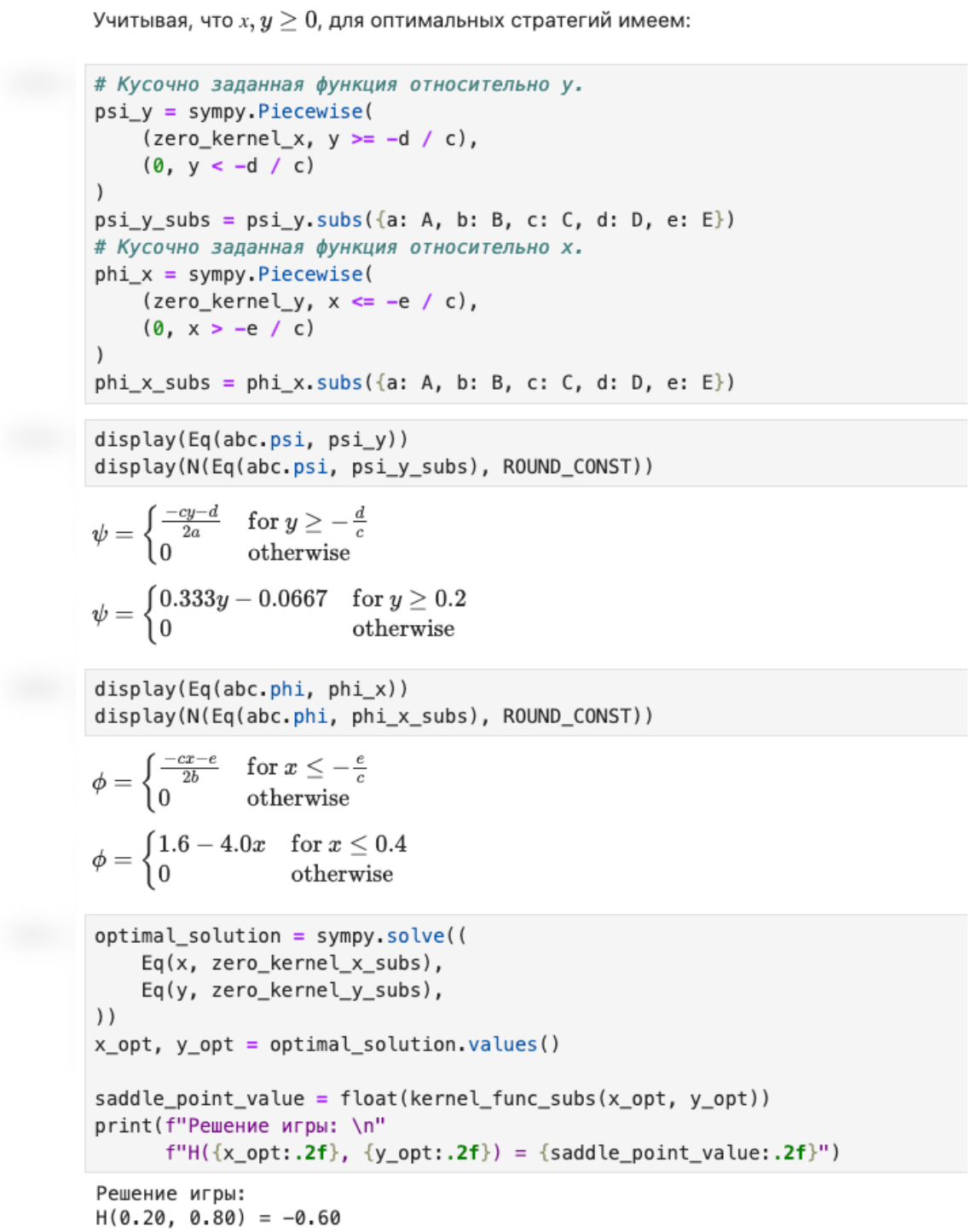

Рисунок 4 – Решение системы уравнений и нахождение решения непрерывной выпукло-вогнутой игры аналитическим методом

Таким образом получили точное решение игры: ***x = 0,2; y = 0,8; H = -0,6***.

2. Численный метод решения

Для решения игры с непрерывным ядром используем метод аппроксимации функции выигрышей на сетке. С каждой итерацией алгоритма будем увеличивать шаг сетки. Если в матрице игры, полученной на итерации *N*, не будет седловой точки, будем применять уже известный численный метод Брауна-Робинсон для отыскания численного решения квадратной матричной игры размерности *N+1*.

Остановку алгоритма будем производить исходя из суммарного изменения оценки цены игры за *k* последних итераций (по умолчанию: $k = 5$), которое должно быть не больше заданного значения точности ($\varepsilon = 0{,}1$). С ростом числа итераций, *k* необходимо увеличивать. Результат работы алгоритма представлен ниже:

N = 2 (шаг: 0.500)

Таблица стратегий (игрока A)				
Стратегии	b1	b2	b3	MIN выигрыш A
a1	0.000	-0.562	-0.917	-0.917
a2	-1.583	-1.312	-0.833	-1.583
a3	-5.667	-4.562	-3.250	-5.667
MAX проигрыш B	0.000	-0.562	-0.833	

Седловой точки нет. Решение методом Брауна-Робинсон:
x = 0.000; y = 0.500; H = -0.872

N = 3 (шаг: 0.333)

Таблица стратегий (игрока A)					
Стратегии	b1	b2	b3	b4	MIN выигрыш A
a1	0.000	-0.398	-0.704	-0.917	-0.917
a2	-0.778	-0.806	-0.741	-0.583	-0.806
a3	-2.667	-2.324	-1.889	-1.361	-2.667
a4	-5.667	-4.954	-4.148	-3.250	-5.667
MAX проигрыш B	0.000	-0.398	-0.704	-0.583	

Седловой точки нет. Решение методом Брауна-Робинсон:
x = 0.000; y = 0.000; H = -0.727

N = 4 (шаг: 0.250)

Таблица стратегий (игрока A)						
Стратегии	b1	b2	b3	b4	b5	MIN выигрыш A
a1	0.000	-0.307	-0.562	-0.766	-0.917	-0.917
a2	-0.479	-0.578	-0.625	-0.620	-0.562	-0.625
a3	-1.583	-1.474	-1.312	-1.099	-0.833	-1.583
a4	-3.312	-2.995	-2.625	-2.203	-1.729	-3.312
a5	-5.667	-5.141	-4.562	-3.932	-3.250	-5.667
MAX проигрыш B	0.000	-0.307	-0.562	-0.620	-0.562	

Седловой точки нет. Решение методом Брауна-Робинсон:
x = 0.000; y = 0.500; H = -0.591

N = 5 (шаг: 0.200)

```
+-----------------------------------------------------------------------------+
|                          Таблица стратегий (игрока A)                       |
+----------------+--------+--------+--------+--------+--------+--------+-----+
|   Стратегии    |   b1   |   b2   |   b3   |   b4   |   b5   |   b6   | MIN выигрыш A |
+----------------+--------+--------+--------+--------+--------+--------+-----+
|       a1       | 0.000  | -0.250 | -0.467 | -0.650 | -0.800 | -0.917 |     -0.917    |
|       a2       | -0.333 | -0.450 | -0.533 | -0.583 | -0.600 | -0.583 |     -0.600    |
|       a3       | -1.067 | -1.050 | -1.000 | -0.917 | -0.800 | -0.650 |     -1.067    |
|       a4       | -2.200 | -2.050 | -1.867 | -1.650 | -1.400 | -1.117 |     -2.200    |
|       a5       | -3.733 | -3.450 | -3.133 | -2.783 | -2.400 | -1.983 |     -3.733    |
|       a6       | -5.667 | -5.250 | -4.800 | -4.317 | -3.800 | -3.250 |     -5.667    |
| MAX проигрыш B | 0.000  | -0.250 | -0.467 | -0.583 | -0.600 | -0.583 |                |
+----------------+--------+--------+--------+--------+--------+--------+----------------+
Седловая точка найдена:
x = 0.200; y = 0.800; H = -0.600

N = 6 (шаг: 0.167)
+-----------------------------------------------------------------------------------------+
|                                Таблица стратегий (игрока A)                             |
+----------------+--------+--------+--------+--------+--------+--------+--------+-----+
|   Стратегии    |   b1   |   b2   |   b3   |   b4   |   b5   |   b6   |   b7   | MIN выигрыш A |
+----------------+--------+--------+--------+--------+--------+--------+--------+-----+
|       a1       | 0.000  | -0.211 | -0.398 | -0.562 | -0.704 | -0.822 | -0.917 |     -0.917    |
|       a2       | -0.250 | -0.368 | -0.463 | -0.535 | -0.583 | -0.609 | -0.611 |     -0.611    |
|       a3       | -0.778 | -0.803 | -0.806 | -0.785 | -0.741 | -0.674 | -0.583 |     -0.806    |
|       a4       | -1.583 | -1.516 | -1.426 | -1.312 | -1.176 | -1.016 | -0.833 |     -1.583    |
|       a5       | -2.667 | -2.507 | -2.324 | -2.118 | -1.889 | -1.637 | -1.361 |     -2.667    |
|       a6       | -4.028 | -3.775 | -3.500 | -3.201 | -2.880 | -2.535 | -2.167 |     -4.028    |
|       a7       | -5.667 | -5.322 | -4.954 | -4.562 | -4.148 | -3.711 | -3.250 |     -5.667    |
| MAX проигрыш B | 0.000  | -0.211 | -0.398 | -0.535 | -0.583 | -0.609 | -0.583 |                |
+----------------+--------+--------+--------+--------+--------+--------+--------+----------------+
Седловой точки нет. Решение методом Брауна-Робинсон:
x = 0.167; y = 0.667; H = -0.642

N = 7 (шаг: 0.143)
+-----------------------------------------------------------------------------------------------------+
|                                      Таблица стратегий (игрока A)                                   |
+----------------+--------+--------+--------+--------+--------+--------+--------+--------+-----+
|   Стратегии    |   b1   |   b2   |   b3   |   b4   |   b5   |   b6   |   b7   |   b8   | MIN выигрыш A |
+----------------+--------+--------+--------+--------+--------+--------+--------+--------+-----+
|       a1       | 0.000  | -0.182 | -0.347 | -0.495 | -0.626 | -0.740 | -0.837 | -0.917 |     -0.917    |
|       a2       | -0.197 | -0.311 | -0.408 | -0.488 | -0.551 | -0.597 | -0.626 | -0.638 |     -0.638    |
|       a3       | -0.599 | -0.645 | -0.673 | -0.685 | -0.680 | -0.658 | -0.619 | -0.563 |     -0.685    |
|       a4       | -1.204 | -1.182 | -1.143 | -1.087 | -1.014 | -0.923 | -0.816 | -0.692 |     -1.204    |
|       a5       | -2.014 | -1.923 | -1.816 | -1.692 | -1.551 | -1.393 | -1.218 | -1.026 |     -2.014    |
|       a6       | -3.027 | -2.869 | -2.694 | -2.502 | -2.293 | -2.066 | -1.823 | -1.563 |     -3.027    |
|       a7       | -4.245 | -4.019 | -3.776 | -3.515 | -3.238 | -2.944 | -2.633 | -2.304 |     -4.245    |
|       a8       | -5.667 | -5.372 | -5.061 | -4.733 | -4.388 | -4.026 | -3.646 | -3.250 |     -5.667    |
| MAX проигрыш B | 0.000  | -0.182 | -0.347 | -0.488 | -0.551 | -0.597 | -0.619 | -0.563 |                |
+----------------+--------+--------+--------+--------+--------+--------+--------+--------+----------------+
Седловой точки нет. Решение методом Брауна-Робинсон:
x = 0.286; y = 0.714; H = -0.645

Таким образом численно найдено решение задачи:
x ≈ 0.217; y ≈ 0.727; H ≈ -0.629
```

Численное решение игры: ***x ≈ 0,217; y ≈ 0,727; H ≈ -0,629.***

3. Сравнительная оценка погрешностей вычислений

Полученные результаты и их приведены в сводной таблице 1. Таким образом полученное приближенное решение удовлетворяет заданной точности $\varepsilon \leq 0.1$.

Таблица 1 – Сводная таблица сравнительной оценки погрешностей

	Цена игры	Оптимальные стратегии в непрерывной игре	
		x	y
Аналитический метод	-0,600	0,200	0,800
Численный метод	-0,629	0,217	0,727
Абсолютная погрешность вычислений, Δ	0,029	0,017	0,073
Относительная погрешность вычислений, %	4,6	8,5	9,1

Вывод

В данной работе была исследована непрерывная антагонистическая выпукло-вогнутая игра двух лиц, а также аналитический и численный методы её решения.

Сначала было получено точное эталонное решение с помощью аналитического метода:

$$x = 0,2;\ y = 0,8;\ H = -0,6.$$

Численный метод недостатки аналитического метода за счёт уменьшения точности вычислений. Задав погрешность $\varepsilon \leq 0.1$, было получено следующее приближённое решение матричной игры:

$$x \approx 0,217;\ y \approx 0,727;\ H \approx -0,629.$$

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 численный метод хорошо подходит для вычисления приближенного решения непрерывной выпукло-вогнутой игры.

ПРИЛОЖЕНИЕ А
Листинг реализации решения выпукло-вогнутой игры

continuous_convex-concave.ipynb

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

# # Лабораторная работа №2
# **"Непрерывные выпукло-вогнутые игры"**
#
# **Выполнил: Александров А. Н., ИУ8-104**
#
# **Вариант: 1**
#
# ## Задание
# Функция ядра имеет вид:
#
# $$H(x, y) = ax^2 + by^2 + cxy + dx + ey,$$
#
# где:
#
# |  a |   b  |   c  |   d  |   e  |
# |:--:|:----:|:----:|:----:|:----:|
# | -5 | 5/12 | 10/3 | -2/3 | -4/3 |
#
# Найти оптимальные стратегии непрерывной выпукло-вогнутой антагонистической игры аналитическим и численным методами.

# In[286]:


import logging

import sympy
from sympy import N, Eq, abc
# To represent multiple expressions in output of single cell.
from IPython.display import display

from game_theory.utils.continuous_games.convex_concave.numeric import NumericMethod

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


# In[287]:


ROUND_CONST: int = 3
A, B, C, D, E = (
    -5,
    5 / 12,
    10 / 3,
    -2 / 3,
    -4 / 3,
)


# ## 1. Аналитическое решение

# In[288]:


# Входные параметры: коэффициенты функции ядра.
a, b, c, d, e = sympy.var(("a", "b", "c", "d", "e"))
# Переменные.
x, y = sympy.symbols(("x", "y"))

# Задание функции ядра.
kernel_func = sympy.Lambda(
    (x, y),
    a * x ** 2 + b * y ** 2 + c * x * y + d * x + e * y,
)
kernel_func_subs = kernel_func.subs({a: A, b: B, c: C, d: D, e: E})

display(kernel_func)
N(kernel_func_subs, ROUND_CONST)


# In[289]:


kernel_xx = sympy.diff(kernel_func(x, y), x, 2, evaluate=False)
kernel_xx_eval = kernel_xx.doit()
```

```
kernel_xx_subs = kernel_xx_eval.subs({a: A, b: B, c: C, d: D, e: E})

display(Eq(kernel_xx, kernel_xx_eval))
N(Eq(kernel_xx_eval, kernel_xx_subs), ROUND_CONST)


# In[290]:


kernel_yy = sympy.diff(kernel_func(x, y), y, 2, evaluate=False)
kernel_yy_eval = kernel_yy.doit()
kernel_yy_subs = kernel_yy_eval.subs({a: A, b: B, c: C, d: D, e: E})

display(Eq(kernel_yy, kernel_yy_eval))
N(Eq(kernel_yy_eval, kernel_yy_subs), ROUND_CONST)


# In[291]:


is_convex_concave: bool = kernel_xx_subs < 0 < kernel_yy_subs
assert is_convex_concave, (
    "Игра не является выпукло-вогнутой, т.к. для функции ядра одновременно не выполняется оба условия:
\n"
    f"H_xx = {2 * a:.2f} < 0 и H_yy = {2 * b:.2f} > 0"
)


# Для нахождения оптимальных стратегий найдем производные функции ядра по каждой переменной

# In[292]:


# Производная по x.
kernel_x = sympy.diff(kernel_func(x, y), x, evaluate=False)
kernel_x_eval = kernel_x.doit()
kernel_x_subs = kernel_x_eval.subs({a: A, b: B, c: C, d: D, e: E})
# Производная по y.
kernel_y = sympy.diff(kernel_func(x, y), y, evaluate=False)
kernel_y_eval = kernel_y.doit()
kernel_y_subs = kernel_y_eval.subs({a: A, b: B, c: C, d: D, e: E})


# In[293]:


display(Eq(kernel_x, kernel_x_eval))
N(Eq(kernel_x_eval, kernel_x_subs), ROUND_CONST)


# In[294]:


display(Eq(kernel_y, kernel_y_eval))
N(Eq(kernel_y_eval, kernel_y_subs), ROUND_CONST)


# После приравнивания производных к нулю получим

# In[295]:


# Выражаем решение производной через x.
zero_kernel_x, = sympy.solve(Eq(kernel_x_eval, 0), x)
zero_kernel_x_subs = zero_kernel_x.subs({a: A, b: B, c: C, d: D, e: E})
# Выражаем решение производной через y.
zero_kernel_y, = sympy.solve(Eq(kernel_y_eval, 0), y)
zero_kernel_y_subs = zero_kernel_y.subs({a: A, b: B, c: C, d: D, e: E})


# In[296]:


display(Eq(x, zero_kernel_x))
display(N(Eq(x, zero_kernel_x_subs), ROUND_CONST))


# In[297]:


display(Eq(y, zero_kernel_y))
display(N(Eq(y, zero_kernel_y_subs), ROUND_CONST))


# Учитывая, что $x,y \ge 0$, для оптимальных стратегий имеем:
```

```
# In[298]:


# Кусочно заданная функция относительно y.
psi_y = sympy.Piecewise(
    (zero_kernel_x, y >= -d / c),
    (0, y < -d / c)
)
psi_y_subs = psi_y.subs({a: A, b: B, c: C, d: D, e: E})
# Кусочно заданная функция относительно x.
phi_x = sympy.Piecewise(
    (zero_kernel_y, x <= -e / c),
    (0, x > -e / c)
)
phi_x_subs = phi_x.subs({a: A, b: B, c: C, d: D, e: E})


# In[299]:


display(Eq(abc.psi, psi_y))
display(N(Eq(abc.psi, psi_y_subs), ROUND_CONST))


# In[300]:


display(Eq(abc.phi, phi_x))
display(N(Eq(abc.phi, phi_x_subs), ROUND_CONST))


# In[301]:


optimal_solution = sympy.solve((
    Eq(x, zero_kernel_x_subs),
    Eq(y, zero_kernel_y_subs),
))
x_opt, y_opt = optimal_solution.values()

saddle_point_value = float(kernel_func_subs(x_opt, y_opt))
print(f"Решение игры: \n"
      f"H({x_opt:.2f}, {y_opt:.2f}) = {saddle_point_value:.2f}")


# ## 2. Численное решение
# Для решения игры с непрерывным ядром используем метод аппроксимации функции выигрышей на сетке.

# In[302]:


numeric_method = NumericMethod(kernel_func_subs)


# In[303]:


x_opt, y_opt, game_price_estimate = numeric_method.solve()


# In[304]:


print(f"Решение игры: \n"
      f"x ≈ {x_opt:.2f}, y ≈ {y_opt:.2f}; H ≈ {game_price_estimate:.2f}")
```

game_theory/utils/continuous_games/convex_concave/numeric.py:

```
"""Численный метод решения выпукло-вогнутых игр с непрерывным ядром."""
import logging
from collections import deque

import numpy as np
import sympy
from sympy.core import function

from game_theory.utils.continuous_games.exceptions import ContinuousGameException
from game_theory.utils.continuous_games.types import SizeType, ValueType
from game_theory.utils.matrix_games.brown_robinson.brown_robinson import BrownRobinson
from game_theory.utils.matrix_games.game_matrix import GameMatrix

_logger = logging.getLogger(__name__)


class NumericMethod:
    """
    Численный метод решения выпукло-вогнутых игр с непрерывным ядром.
    """

    def __init__(self, kernel_func: function.Lambda, accuracy: float = 0.1, deltas_count: SizeType =
3):
        # Функция ядра непрерывной выпукло-вогнутой игры.
        self.kernel_func: function.Lambda = kernel_func
        # Точность численного метода.
        self.accuracy: float = accuracy
        # Разности между соседними вычисленными значениями цены игры на итерациях.
        max_len: SizeType = deltas_count
        self.__deltas: deque[float] = deque(maxlen=max_len)
        self.__estimates: deque[tuple[float, float, ValueType]] = deque(maxlen=max_len)

        # Проверяем, что игра является выпукло-вогнутой.
        x, y = sympy.symbols(("x", "y"))
        kernel_xx = sympy.diff(self.kernel_func(x, y), x, 2)
        kernel_yy = sympy.diff(self.kernel_func(x, y), y, 2)
        if not (kernel_xx < 0 < kernel_yy):
            err_msg = (
                "Игра не является выпукло-вогнутой, "
                "т.к. для функции ядра одновременно не выполняется оба условия: \n"
                f"H_xx = {kernel_xx:.2f} < 0 и H_yy = {kernel_yy:.2f} > 0"
            )
            raise ContinuousGameException(err_msg)

    def solve(self) -> tuple[float, float, ValueType]:
        n = 2
        prev_game_price_estimate, game_price_estimate = None, None
        while len(self.__deltas) == 0 or sum(self.__deltas) > self.accuracy:
            _logger.info(f"N = {n} (шаг: {1 / n:.3f})")
            grid_game_matrix = GameMatrix(self._generate_grid_approximation_matrix(n))
            _logger.info(grid_game_matrix)

            # Проверка седла.
            i, lgp_value = grid_game_matrix.lowest_game_price
            j, hgp_value = grid_game_matrix.highest_game_price
            if lgp_value == hgp_value:
                game_price_estimate = lgp_value
                x_estimate, y_estimate = i / n, j / n
                _logger.info("Седловая точка найдена:")
            else:
                br_method = BrownRobinson(grid_game_matrix, accuracy=self.accuracy)
                br_method.solve()
                game_price_estimate = br_method.game_price_estimation
                # _logger.info(f"Оценка для цены игры: {game_price_estimate}")
                x_mixed_strategies, y_mixed_strategies = br_method.mixed_strategies
                # _logger.info(f"Смешанные стратегии:\n X = {x_mixed_strategies}\n Y =
{y_mixed_strategies}")
                x_estimate, y_estimate = (np.argmax(x_mixed_strategies) / n,
np.argmax(y_mixed_strategies) / n)
                _logger.info("Седловой точки нет. Решение методом Брауна-Робинсон:")

            _logger.info(f"x = {x_estimate:.3f}; y = {y_estimate:.3f}; H =
{game_price_estimate:.3f}\n")
            if prev_game_price_estimate is not None:
                # Добавляем разность между оценками цен игры на текущей и предыдущей итерации в deque.
                self.__push_estimates(
                    delta=np.abs(game_price_estimate - prev_game_price_estimate),
                    x_est=x_estimate,
                    y_est=y_estimate,
                    game_price_est=game_price_estimate,
                )

            prev_game_price_estimate = game_price_estimate
```

```
            n += 1

        x_estimate, y_estimate, game_price_estimate = (
            np.mean([est_tuple[i] for est_tuple in self.__estimates]) for i in range(3)
        )
        _logger.info(
            "Таким образом численно найдено решение задачи:\n"
            f"x ≈ {x_estimate:.3f}; y ≈ {y_estimate:.3f}; H ≈ {game_price_estimate:.3f}"
        )

        return x_estimate, y_estimate, game_price_estimate

    def _generate_grid_approximation_matrix(self, iteration_number: int) -> np.ndarray:
        """
        Генерирует квадратную матрицу аппроксимации функции ядра (выигрышей) на сетке.
        Матрица имеет размерность `iteration_number` + 1
        Элемент матрицы a_ij = H(i / iteration_number; j / iteration_number), где H - функция ядра от
двух переменных.
        :param int iteration_number: Размерность сетки.
        :return: Матрица сетки.
        """
        return np.array(
            [
                [
                    float(self.kernel_func(i / iteration_number, j / iteration_number))
                    for j in range(iteration_number + 1)
                ]
                for i in range(iteration_number + 1)
            ]
        )

    def __push_estimates(self, delta: float, x_est: float, y_est: float, game_price_est: ValueType) ->
None:
        """Добавляем оценки результатов и разностей прироста цены игры в deque-контейнеры."""
        if len(self.__deltas) == self.__deltas.maxlen:
            self.__deltas.popleft()
            self.__estimates.popleft()

        self.__deltas.append(delta)
        self.__estimates.append((x_est, y_est, game_price_est))
```

ПРИЛОЖЕНИЕ Б

Ссылка на репозиторий с исходным кодом задачи

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