

Министерство науки и высшего образования Российской Федерации
Московский государственный технический университет имени
Н. Э. Баумана

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

Лабораторная работа №2
ПО ДИСЦИПЛИНЕ «ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ»
«Выпукло-вогнутые антагонистические игры»

Вариант 1

Студент: Анаян М. С., ИУ8-104
Преподаватель: Коннова Н. С.

Москва, 2020

Цель и задачи выполнения лабораторной работы

Цель работы – найти оптимальные стратегии непрерывной выпукло-вогнутой антагонистической игры аналитическим и численными методами.

Постановка задачи

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 цена игры, и, кроме того,

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

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.

Выпукло-вогнутая игра всегда разрешима в чистых стратегиях.

Выполнение лабораторной работы

Аналитическое решение

Функция ядра имеет вид:

$$H(x, y) = -5x^2 + \frac{5}{12}y^2 + \frac{10}{3}xy - \frac{2}{3}x - \frac{4}{3y}.$$

Условия принадлежности игры к классу выпукло-вогнутых выполняются:

$$H_{xx} = 2a = -10 < 0,$$

$$H_{yy} = 2b = \frac{5}{6} > 0;$$

Для нахождения оптимальных стратегий найдём производные функции ядра по каждой переменной:

$$H_x = 2ax + cy + d = -10 * x + \frac{10}{3} * y - \frac{2}{3},$$

$$H_y = 2by + cx + e = 10/3 * x + 5/6 * y - 4/3.$$

При $H_x = 0$ и $H_y = 0$ получим:

$$x = -\frac{cy + d}{2a} = \frac{1}{3}y - \frac{1}{15},$$

$$y = -\frac{cx + e}{2b} = -4x + \frac{8}{5}.$$

Поскольку $x \geq 0$ и $y \geq 0$, для максимальных стратегий имеем:

$$\psi(y) = \begin{cases} \frac{1}{3}y - \frac{1}{15}, & y \geq \frac{1}{5}, \\ 0, & y \leq \frac{1}{5}; \end{cases}$$

$$\phi(x) = \begin{cases} -4x + \frac{8}{5}, & x \geq \frac{2}{5}, \\ 0, & x \leq 2/5. \end{cases}$$

Решив систему для $\psi(y)$ и $\phi(x)$ относительно переменных x и y , получаем:

$$x^* = \frac{1}{5}, y^* = \frac{4}{5}.$$

При этом седловая точка игры $H(x^*, y^*) = -\frac{3}{5}$.

Вычисления произведены при помощи SageMath 8.9, с исходным кодом можно ознакомиться в репозитории по ссылке:

https://github.com/hms2010/GameTheory/blob/master/src/lab2/analytical_method.ipynb

Численное решение

Рассмотрим метод аппроксимации функции выигрышей на сетке. При помощи программы найдены решения при различном шаге сетки. В таблице ниже приведены этапы расчёта:

```

N = 1
  0.000 -0.917
-5.667 -3.250
Has saddle point: x = 0, y = 1, h = -11/12 = -0.917
N = 2
  0.000 -0.562 -0.917
-1.583 -1.312 -0.833
-5.667 -4.562 -3.250
Has no saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 0, y = 1, h = -
26213/29760 = -0.881
N = 3
  0.000 -0.398 -0.704 -0.917
-0.778 -0.806 -0.741 -0.583
-2.667 -2.324 -1.889 -1.361
-5.667 -4.954 -4.148 -3.250
Has no saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 1/3, y = 2/3, h = -
15589405/21517083 = -0.725
N = 4
  0.000 -0.307 -0.562 -0.766 -0.917
-0.479 -0.578 -0.625 -0.620 -0.562
-1.583 -1.474 -1.312 -1.099 -0.833
-3.312 -2.995 -2.625 -2.203 -1.729
-5.667 -5.141 -4.562 -3.932 -3.250
Has no saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 1/4, y = 1/2, h = -
1030987/1655040 = -0.623
N = 5
  0.000 -0.250 -0.467 -0.650 -0.800 -0.917
-0.333 -0.450 -0.533 -0.583 -0.600 -0.583
-1.067 -1.050 -1.000 -0.917 -0.800 -0.650
-2.200 -2.050 -1.867 -1.650 -1.400 -1.117
-3.733 -3.450 -3.133 -2.783 -2.400 -1.983
-5.667 -5.250 -4.800 -4.317 -3.800 -3.250
Has saddle point: x = 1/5, y = 4/5, h = -3/5 = -0.600
N = 6
  0.000 -0.211 -0.398 -0.562 -0.704 -0.822 -0.917
-0.250 -0.368 -0.463 -0.535 -0.583 -0.609 -0.611
-0.778 -0.803 -0.806 -0.785 -0.741 -0.674 -0.583
-1.583 -1.516 -1.426 -1.312 -1.176 -1.016 -0.833
-2.667 -2.507 -2.324 -2.118 -1.889 -1.637 -1.361
-4.028 -3.775 -3.500 -3.201 -2.880 -2.535 -2.167
-5.667 -5.322 -4.954 -4.562 -4.148 -3.711 -3.250
Has no saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 1/6, y = 1, h = -
100219/164160 = -0.610
N = 7
  0.000 -0.182 -0.347 -0.495 -0.626 -0.740 -0.837 -0.917
-0.197 -0.311 -0.408 -0.488 -0.551 -0.597 -0.626 -0.638
-0.599 -0.645 -0.673 -0.685 -0.680 -0.658 -0.619 -0.563
-1.204 -1.182 -1.143 -1.087 -1.014 -0.923 -0.816 -0.692
-2.014 -1.923 -1.816 -1.692 -1.551 -1.393 -1.218 -1.026
-3.027 -2.869 -2.694 -2.502 -2.293 -2.066 -1.823 -1.563
-4.245 -4.019 -3.776 -3.515 -3.238 -2.944 -2.633 -2.304
-5.667 -5.372 -5.061 -4.733 -4.388 -4.026 -3.646 -3.250
Has no saddle point
Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 1/7, y = 6/7, h = -
790504919/1269106272 = -0.623
N = 8
  0.000 -0.160 -0.307 -0.441 -0.562 -0.671 -0.766 -0.848 -0.917
-0.161 -0.270 -0.365 -0.447 -0.516 -0.572 -0.615 -0.645 -0.661
-0.479 -0.535 -0.578 -0.608 -0.625 -0.629 -0.620 -0.598 -0.562
-0.953 -0.957 -0.948 -0.926 -0.891 -0.842 -0.781 -0.707 -0.620
-1.583 -1.535 -1.474 -1.400 -1.312 -1.212 -1.099 -0.973 -0.833
-2.370 -2.270 -2.156 -2.030 -1.891 -1.738 -1.573 -1.395 -1.203
-3.312 -3.160 -2.995 -2.816 -2.625 -2.421 -2.203 -1.973 -1.729
-4.411 -4.207 -3.990 -3.759 -3.516 -3.259 -2.990 -2.707 -2.411
-5.667 -5.410 -5.141 -4.858 -4.562 -4.254 -3.932 -3.598 -3.250
Has no saddle point

```

Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 1/4, y = 3/4, h = -180774509/292806144 = -0.617									
N = 9									
0.000	-0.143	-0.276	-0.398	-0.510	-0.612	-0.704	-0.785	-0.856	-0.917
-0.136	-0.238	-0.329	-0.410	-0.481	-0.542	-0.593	-0.633	-0.663	-0.682
-0.395	-0.456	-0.506	-0.546	-0.576	-0.596	-0.605	-0.604	-0.593	-0.571
-0.778	-0.797	-0.807	-0.806	-0.794	-0.773	-0.741	-0.699	-0.646	-0.583
-1.284	-1.262	-1.230	-1.188	-1.136	-1.073	-1.000	-0.917	-0.823	-0.719
-1.914	-1.851	-1.778	-1.694	-1.601	-1.497	-1.383	-1.258	-1.123	-0.978
-2.667	-2.563	-2.449	-2.324	-2.189	-2.044	-1.889	-1.723	-1.547	-1.361
-3.543	-3.398	-3.243	-3.077	-2.901	-2.715	-2.519	-2.312	-2.095	-1.867
-4.543	-4.357	-4.160	-3.954	-3.737	-3.509	-3.272	-3.024	-2.765	-2.497
-5.667	-5.439	-5.202	-4.954	-4.695	-4.427	-4.148	-3.859	-3.560	-3.250
Has no saddle point									
Calculated with Brown-Robinson method with accuracy eps = 0.001 solution: x = 2/9, y = 2/3, h = -672667/1112940 = -0.604									
N = 10									
0.000	-0.129	-0.250	-0.362	-0.467	-0.562	-0.650	-0.729	-0.800	-0.863
-0.117	-0.212	-0.300	-0.379	-0.450	-0.512	-0.567	-0.613	-0.650	-0.679
-0.333	-0.396	-0.450	-0.496	-0.533	-0.562	-0.583	-0.596	-0.600	-0.596
-0.650	-0.679	-0.700	-0.713	-0.717	-0.713	-0.700	-0.679	-0.650	-0.613
-1.067	-1.062	-1.050	-1.029	-1.000	-0.963	-0.917	-0.863	-0.800	-0.729
-1.583	-1.546	-1.500	-1.446	-1.383	-1.312	-1.233	-1.146	-1.050	-0.946
-2.200	-2.129	-2.050	-1.962	-1.867	-1.762	-1.650	-1.529	-1.400	-1.262
-2.917	-2.812	-2.700	-2.579	-2.450	-2.312	-2.167	-2.013	-1.850	-1.679
-3.733	-3.596	-3.450	-3.296	-3.133	-2.962	-2.783	-2.596	-2.400	-2.196
-4.650	-4.479	-4.300	-4.112	-3.917	-3.712	-3.500	-3.279	-3.050	-2.812
-5.667	-5.463	-5.250	-5.029	-4.800	-4.562	-4.317	-4.062	-3.800	-3.529
Has saddle point: x = 1/5, y = 4/5, h = -3/5 = -0.600									

Итоговое численное решение:

$$x^* = 0.22, \quad y^* = 0.78, \quad H(x^*, y^*) = -0.603.$$

С исходным кодом на языке Python версии 3.*, реализующим метод, можно ознакомиться в приложении А либо в репозитории hms2010/GameTheory на github.com по ссылке:

<https://github.com/hms2010/GameTheory/blob/master/src/lab2/numerical.py>

Погрешность между аналитическим и приближенным решением методом аппроксимации на сетке составляет 0.33%.

Выводы

В результате выполнения лабораторной работы получены следующие результаты:

– изучен и реализован аналитический метод нахождения оптимальных стратегий в непрерывной выпукло-вогнутой антагонистической игре двух лиц;

– изучен и реализован численный метод аппроксимации на сетке нахождения оптимальных стратегий в непрерывной выпукло-вогнутой антагонистической игре двух лиц;

– найдена оптимальная стратегия обоих игроков: $x^* = 0.2, y^* = 0.8$; седловая точка при этом, $H(x^*, y^*) = -0.6$;

– погрешность приближенным решением методом аппроксимации на сетке относительно аналитического решения составляет 0.33%.

Исходные коды программ представлены по ссылке:

<https://github.com/hms2010/GameTheory/tree/master/lab2>.

Приложение А

```
import math
from fractions import Fraction
import random

def get_rand_max_index(arr):
    max_i = []
    max_el = max(arr)
    i = arr.index(max_el)
    max_i.append(i)
    while i < len(arr):
        try:
            i = arr.index(max_el, i + 1)
            max_i.append(i)
        except ValueError:
            break
    return random.choice(max_i)

def get_rand_min_index(arr):
    min_i = []
    min_el = min(arr)
    i = arr.index(min_el)
    min_i.append(i)
    while i < len(arr):
        try:
            i = arr.index(min_el, i + 1)
            min_i.append(i)
        except ValueError:
            break
    return random.choice(min_i)

def get_row_by_index(matrix, index):
    return matrix[index]

def get_column_by_index(matrix, index):
    return [matrix[i][index] for i in range(len(matrix))]

def get_max_index(arr):
    return arr.index(max(arr))

def get_min_index(arr):
    return arr.index(min(arr))

def vector_addition(a, b):
    return [i + j for i, j in zip(a, b)]

def brown_robinson_method(C, eps):
    m = len(C)    # A player strategies: strategy row consists of win of A
```

```

n = len(C[0]) # B player strategies: strategy column consist of loss of B

x = m * [0]
y = n * [0]

curr_strategy_a = 0
curr_strategy_b = 0

win_a = m * [0]
loss_b = n * [0]
curr_eps = math.inf
k = 0

lower_bounds = []
upper_bounds = []

while (curr_eps > eps):
    k += 1
    win_a = vector_addition(win_a, get_column_by_index(C, curr_strategy_b))
    loss_b = vector_addition(loss_b, get_row_by_index(C, curr_strategy_a))
    x[curr_strategy_a] += 1
    y[curr_strategy_b] += 1

    lower_bound = Fraction(min(loss_b), k)
    upper_bound = Fraction(max(win_a), k)
    lower_bounds.append(lower_bound)
    upper_bounds.append(upper_bound)

    curr_eps = min(upper_bounds) - max(lower_bounds)

    curr_strategy_a = get_rand_max_index(win_a)
    curr_strategy_b = get_rand_min_index(loss_b)

cost = max(lower_bounds) + Fraction(curr_eps, 2)

x = [Fraction(i, k) for i in x]
y = [Fraction(i, k) for i in y]

return x, y, cost

def H(x, y):
    ''' The coefficients from example presented below
    a, b = Fraction(-3, 1), Fraction(3, 2)
    c = Fraction(18/5)
    d, e = Fraction(-18, 50), Fraction(-72, 25)
    ...
    a, b = Fraction(-5, 1), Fraction(5, 12)
    c = Fraction(10, 3)
    d, e = Fraction(-2, 3), Fraction(-4, 3)
    return a * x**2 + b * y**2 + c * x * y + d * x + e * y

```



```

def find_saddle_point(C):
    max_min = None
    min_max = None

    m = len(C)
    n = len(C[0])

    max_loss = []
    for i in range(n):
        max_loss.append(max(get_column_by_index(C, i)))
    y = get_min_index(max_loss)
    min_max = max_loss[y]

    min_win = []
    for i in range(n):
        min_win.append(min(get_row_by_index(C, i)))
    x = get_max_index(min_win)
    max_min = min_win[x]
    return max_min == min_max, x, y

```

```

def limit(a, eps):
    ff = False
    N = -1
    for i in range(1, len(a)):
        if abs(a[i] - a[i - 1]) < eps:
            N = i
            ff = True
        elif ff:
            N = -1
            ff = False
    if not ff:
        return math.inf
    return a[N]

```

```

def calc_grid_element(H, i, j, N):
    return H(Fraction(i, N), Fraction(j, N))

```

```

def generate_grid_approximation(H, n):
    cur_H = []
    for i in range(n + 1):
        cur_H.append([0] * (n + 1))
    # fill matrix with calculated values
    for i in range(n + 1):
        for j in range(n + 1):
            cur_H[i][j] = calc_grid_element(H, i, j, n)
    return cur_H

```

```

def grid_approximation_method(H, eps):

```

```

cost_array = []
n = 1
while(True):
    # create matrix for this iteration
    cur_H = generate_grid_approximation(H, n)
    print("N = {:d}".format(n))
    for i in cur_H:
        print(*["{:8.3f}".format(float(j)) for j in i])
    has_saddle_point, x, y = find_saddle_point(cur_H)
    if has_saddle_point:
        h = cur_H[x][y]
        x = Fraction(x, n)
        y = Fraction(y, n)
        print("Has saddle point: x = {:}, y = {:}, h = {:} = {:.3f}".format(x, y, h, float(h)))
    else:
        print("Has no saddle point")
        x, y, h = brown_robinson_method(cur_H, eps)
        x = Fraction(get_max_index(x), n)
        y = Fraction(get_max_index(y), n)
        print("Calculated with Brown-
Robinson method with accuracy eps = {:.3f} solution: x = {:}, y = {:}, h = {:} = {:.3f}".format(e
ps, x, y, h, float(h)))
        cost_array.append(h)
        lim = limit(cost_array, eps)
        if lim != math.inf:
            return x, y, lim
        n += 1

def main():
    x, y, c = grid_approximation_method(H, 0.001)
    print("Found solution is: x = {:}, y = {:}, c = {:} = {:.3f}".format(x, y, c, float(c)))

if __name__ == "__main__":
    main()

```