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 №5

ПО ДИСЦИПЛИНЕ «ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ»

«Монотонный алгоритм»

Вариант 1

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

Москва, 2020

Цель и задачи выполнения лабораторной работы

Цель работы – изучить аналитический (обратной матрицы) и численный (монотонный алгоритм) методы нахождения смешанных стратегий в антагонистической игре двух лиц в нормальной форме.

Постановка задачи – найти цену игры и оптимальную стратегию обоих игрока А методами обратной матрицы и монотонного алгоритма, затем сравнить полученные результаты.

Выполнение лабораторной работы

(3 × 3)-игра Г задана платёжной матрицей:

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

Для расчёта методом обратной матрицы применяются следующие формулы:

$$x^* = \frac{uC^{-1}}{uC^{-1}u^T}, y^* = \frac{C^{-1}u^T}{uC^{-1}u^T}, v = \frac{1}{uC^{-1}u^T},$$

где $u = (1, 1, \dots, 1) \in \mathbb{R}^m$ для (m × n)-игры Г.

В соответствии с расчётом по заданным формулам для метода обратной матрицы получены следующие значения:

– стоимость игры $v = \frac{133}{18} \approx 7.39$,

– оптимальная смешанная стратегия игрока А $x^* = \left(\frac{19}{54}, \frac{10}{27}, \frac{5}{18}\right) = (0.35, 0.37, 0.28)$.

Вычисления произведены при помощи SageMath 8.9, с исходным кодом можно ознакомиться в репозитории по ссылке: https://github.com/hms2010/GameTheory/blob/master/lab1/lab1-analytical.ipynb

В таблице 1 приведён Вывод результатов работы монотонного алгоритма с $\varepsilon \leq 10^{-6}$:

Таблица 1 – Вывод результатов работы монотонного алгоритма

```
Enter current accuracy: 6
Cost matrix C:
```

```
[1, 11, 11]
[7, 5, 8]
[16, 6, 2]
---------------------------------------------------------------------------------------------------------
*********************************************************************************************************
Game solving with monotonous method...
Current iteration: 1
    Current strategy: [1, 0, 0]
    Current cost:     [1, 11, 11]
    J: [0]
    Current optimal strategy: [1.0994932919678094e-08, 1.944470646334113e-08, 0.9999999695603606]
    Current optimal cost:     [15.999999660073648, 6.000000035529958, 2.0000002156226353]
    Aplha =  0.4166666765702822
---------------------------------------------------------------------------------------------------------
Current iteration: 2
    Current strategy: [0.58333332801094, 8.101961218965034e-09, 0.41666666388709883]
    Current cost:     [7.250000006918251, 8.916666631952738, 7.2500000007102265]
    J: [0, 2]
    Current optimal strategy: [1.7448702481586535e-08, 0.9333333040635483, 0.06666667848774918]
    Current optimal cost:     [7.600000001697527, 5.066666783179964, 7.599999981419612]
    Aplha =  0.3968254052787554
---------------------------------------------------------------------------------------------------------
Current iteration: 3
    Current strategy: [0.35185185063448193, 0.3703703715320746, 0.27777777783344343]
    Current cost:     [7.388888896694099, 7.3888888816403355, 7.388888884902785]
Game solution was found.
*********************************************************************************************************
Player A strategy x = [0.351852, 0.37037, 0.277778]
Game cost =  7.388889
```

Получены следующие результаты:

– смешанная стратегия игрока A $x = (0.35, 0.37, 0.28)$,

– стоимость игры $v_m \approx 7.39$.

С исходным кодом на языке Python версии 3.*, реализующим монотонный метод, можно ознакомиться в приложении А либо в репозитории hms2010/GameTheory на github.com по ссылке: https://github.com/hms2010/GameTheory/blob/master/lab5/src.py.

Погрешность стоимости игры, полученной монотонного метода, относительно полученной методом обратной матрицы стоимости составила:

$$\delta_v = \frac{(v - v_m)}{v} \cdot 100\% = 0\%.$$

Выводы

В результате выполнения лабораторной работы получены следующие результаты:

– изучен и реализован аналитический (обратной матрицы) метод нахождения смешанных стратегий в антагонистической игре двух лиц в нормальной форме;

– изучен и реализован численный метод (монотонный алгоритм) нахождения смешанных стратегий в антагонистической игре двух лиц в нормальной форме;

– при заданной погрешности вычислений $\varepsilon \leq 10^{-6}$ для монотонного алгоритма оценка стоимости игры относительно стоимости игры, полученной методом обратной матрицы, имеет относительную погрешность 0%.

Исходные коды программ представлены по ссылке: https://github.com/hms2010/GameTheory/tree/master/lab5.

Приложение А

Исходный код в репозитории по ссылке:

https://github.com/hms2010/GameTheory/tree/master/lab5.

```
from scipy.optimize import linprog
from c import *

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 transpose_matrix(matrix):
    return [get_column_by_index(matrix, i) for i in range(len(matrix[0]))]

def is_equal_with_eps(a, b, eps = 10**-6):
    return (abs(a - b) < eps)

def get_min_indexes_list(arr, eps = 10**-6):
    min_i = []
    min_el = min(arr)
    for i in range(len(arr)):
        if is_equal_with_eps(arr[i], min_el, eps):
            min_i.append(i)
    return min_i

def simplex_method(C):
    A = transpose_matrix(C)
    for i in range(len(A)):
        for j in range(len(A[0])):
            A[i][j] *= -1
    b = [-1] * len(A)
    c = [1] * len(A[0])
    res = linprog(c, A_ub=A, b_ub=b, bounds=[(0, None) for i in range(len(c))])
    cost = 1 / res.fun
    strats = [i * cost for i in res.x]
    return strats

def get_cur_opt_cost(A, x):
    res = [0] * len(A[0])
    for i in range(len(A[0])):
        for j in range(len(A)):
            res[i] += x[j] * A[j][i]
    return res

def get_cur_strat_opt(alpha, prev, cur_opt):
    res = [0] * len(prev)
    for i in range(len(prev)):
        res[i] += (1 - alpha)*prev[i] + alpha*cur_opt[i]
    return res

def check_all_cost_equal(cur_cost, eps):
    for i in range(len(cur_cost)):
        for j in range(i, len(cur_cost)):
```

```
            if not is_equal_with_eps(cur_cost[i], cur_cost[j], eps):
                return False
        return True

def monotonous_method(C, eps = 10**-6):
    print("*" * 128)
    print("Game solving with monotonous method...")
    cur_strat, cur_cost = [1] + [0] * (len(C) - 1), get_row_by_index(C, 0)
    N = 0
    while True:
        N += 1
        print("Current iteration: {:d}".format(N))
        print("    Current strategy:", cur_strat)
        print("    Current cost:    ", cur_cost)
        if check_all_cost_equal(cur_cost, eps):
            print("Game solution was found.")
            print("*" * 128)
            return cur_strat, min(cur_cost)
        J = sorted(get_min_indexes_list(cur_cost, eps))
        print("    J:", J)
        subgame = []
        for i in range(len(C)):
            line = []
            for j in J:
                line.append(C[i][j])
            subgame.append(line)
        cur_opt_strat = simplex_method(subgame)
        print("    Current optimal strategy:", cur_opt_strat)
        cur_opt_cost = get_cur_opt_cost(C, cur_opt_strat)
        print("    Current optimal cost:    ", cur_opt_cost)
        subgame = [cur_cost, cur_opt_cost]
        subgame_cost = simplex_method(subgame)
        alpha = subgame_cost[1]
        print("    Aplha = ", alpha)
        print("-" * 128)
        if is_equal_with_eps(0, alpha, eps):
            print("Game solution was found.")
            print("*" * 128)
            return cur_strat, min(cur_cost)

        cur_strat = get_cur_strat_opt(alpha, cur_strat, cur_opt_strat)
        cur_cost = get_cur_strat_opt(alpha, cur_cost, cur_opt_cost)

def main():
    accuracy = int(input("Enter current accuracy: "))
    print("Cost matrix C:")
    for line in C:
        print(line)
    print("-" * 128)
    x, cost = monotonous_method(C, 10**(-accuracy))
    print("Player A strategy x =", [round(i, accuracy) for i in x])
    print("Game cost = ", round(cost, accuracy))

if __name__ == '__main__':
    main()
```