|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | |
| Федеральное государственное автономное образовательное учреждение высшего образования | |
| **«Дальневосточный федеральный университет»** (ДВФУ) | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | |
| **Департамент математического и компьютерного моделирования** | |
| **ЛАБОРАТОРНАЯ РАБОТА №4** | |
| По основной образовательной программе подготовки бакалавров  направлению 01.03.02 Прикладная математика и информатика  профиль «Системное программирование» | |
|  | Студент группы Б9122-01.03.02сп4  Салдаев Владимир Сергеевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_» января 2025 г. |
|  | Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (должность, ученое звание)  кандидат физико-математических наук  \_\_\_\_Яковлев Анатолий Александрович  (подпись) (ФИО)  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. |
| г. Владивосток  2025 | |

**Постановка задачи**

Дана матричная игра, заданная матрицей А размерности 6х8. Необходимо найти верхнюю и нижнюю цену игры и равновесное решение в смешанных стратегиях.

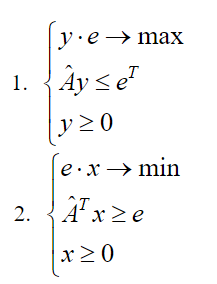
Нижняя цена игры:

Верхняя цена игры:

Поиск равновесного решения будет проходить с помощью симплекс-метода. Для решения матрица А должна быть неотрицательной. Чтобы это осуществить, добавим модуль минимального элемента матрицы А к каждому элементу матрицы.

Получается неотрицательная матрица :

Необходимо решить следующие задачи:



В этом случае оптимальная стратегия первого игрока будет найдена по формуле:

А оптимальная стратегия второго игрока будет найдена по формуле

Цена игры будет равна:

где – значение целевой функции, полученной в результате решения задач линейной оптимизации*.*

–находится прямой задачей;

–находится двойственной задачей;

**Решение прямой задачи симплекс-методом:**

Начальный шаг

Индекс разрешающей строки = 2

Индекс разрешающего столбца = 1

Разрешающий элемент = 93

Шаг 2

Индекс разрешающей строки = 6

Индекс разрешающего столбца = 8

Разрешающий элемент = 166.65

Шаг 3

Индекс разрешающей строки = 5

Индекс разрешающего столбца = 6

Разрешающий элемент = 95.81

Шаг 4

Индекс разрешающей строки = 4

Индекс разрешающего столбца = 7

Разрешающий элемент = 141.51

Оптимальное решение: [0.00242093 0. 0. 0. 0. 0.00461049 0.00333056 0.00131655]

Значение целевой функции: 0.011678521446400204

Результат:

y = [0.00242093 0. 0. 0. 0. 0.00461049 0.00333056 0.00131655]  
||y|| = 0.006320079135274677  
q = y / ||y|| = [0.38305342 0. 0. 0. 0. 0.72949808 0.52698084 0.20831184]   
Значение целевой функции = 0.011678521446400204

**Решение двойственной задачи:**

Двойственная задача приводится к каноническому виду, далее составляется симплекс-таблица.

Начальный шаг

Индекс разрешающей строки = 2

Индекс разрешающего столбца = 1

Разрешающий элемент = -93

Шаг 2

Индекс разрешающей строки = 9

Индекс разрешающего столбца = 5

Разрешающий элемент = -166.65

Шаг 3

Индекс разрешающей строки = 7

Индекс разрешающего столбца = 4

Разрешающий элемент = -95.81

Шаг 4

Индекс разрешающей строки = 8

Индекс разрешающего столбца = 3

Разрешающий элемент = -141.51

Оптимальное решение: [0.00159939 0. 0.00013278 0.00478981 0.00515654 0.]

Результат:

x = [0.00159939 0. 0.00013278 0.00478981 0.00515654 0.]  
||x|| = 0.007218577304516995  
p = x / ||x|| = [0.22156564 0. 0.01839458 0.66353984 0.71434249 0.]   
Значение целевой функции = 0.011678521446400204

Цена игры:

**Ответ:**

Нижняя цена игры =

Верхняя цена игры =

Оптимальная стратегия первого игрока:   
[0.22156564 0. 0.01839458 0.66353984 0.71434249 0.]  
Оптимальная стратегия второго игрока:   
[0.38305342 0. 0. 0. 0. 0.72949808 0.52698084 0.20831184]  
Цена игры:   
𝜙 = 1

**Приложения**

Код программы на языке Python:

import numpy as np

from tools import latex, print\_matrix

M, N = 6, 8

beta = None

def lower\_price(A):

    lower\_in\_lines = [np.min(A[i]) for i in range(A.shape[0])]

    A\_lower = np.max(lower\_in\_lines)

    return A\_lower

def higher\_price(A):

    higher\_in\_cols = [np.max(A[i]) for i in range(A.shape[0])]

    A\_higher = np.min(higher\_in\_cols)

    return A\_higher

def straight(A, b, c):

    table = np.block([

        [-c, np.zeros(M), 0],

        [A, np.eye(M), b]

    ])

    rows, cols = table.shape

    while True:

        r\_col = -1

        min\_val = float('inf')

        for col in range(N):

            if table[0, col] < 0 and table[0, col] < min\_val:

                r\_col = col

                min\_val = table[0, col]

        if r\_col == -1:

            break

        min\_ratio = float('inf')

        r\_row = -1

        for row in range(1, rows):

            ratio = table[row, cols - 1] / table[row, r\_col]

            if ratio > 0 and ratio < min\_ratio:

                min\_ratio = ratio

                r\_row = row

        r\_el = table[r\_row, r\_col]

        print(f'разрешающий элемент [строка, столбец] = [{r\_row}, {r\_col}] = {r\_el}')

        print(f'симплекс-таблица: {table}')

        print()

        table[r\_row] = table[r\_row] / r\_el

        for row in range(rows):

            if row != r\_row:

                table[row] = table[row] - table[row, r\_col] \* table[r\_row]

    y = np.zeros(N)

    for col in range(N):

        if np.count\_nonzero(table[1:, col]) == 1 and np.any(table[1:, col] == 1):

            r = np.where(table[1:, col] == 1)[0][0] + 1

            y[col] = table[r, cols - 1]

    function\_val = table[0, cols - 1]

    y\_norm = np.linalg.norm(y)

    q = y / y\_norm

    print(f'y: {y}, y\_norm: {y\_norm}')

    print(f"прямая задача \n \* q: {q} \n \* значение функции: {function\_val} \n")

def duality(A, b, c):

    global beta

    table = np.block([

        [b.reshape(1, -1), np.zeros(N), 0],

        [-A.T, np.eye(N), -c.reshape(-1, 1)]

    ])

    rows, cols = table.shape

    while True:

        min\_val = float('inf')

        r\_row = -1

        for row in range(1, rows):

            if table[row, cols - 1] < min\_val:

                min\_val = table[row, cols - 1]

                r\_row = row

        if r\_row == -1 or min\_val >= 0:

            break

        min\_ratio = float('inf')

        r\_col = -1

        for col in range(cols - 1):

            if table[r\_row, col] < 0:

                ratio = np.abs(table[0, col] / table[r\_row, col])

                if ratio < min\_ratio:

                    min\_ratio = ratio

                    r\_col = col

        r\_el = table[r\_row, r\_col]

        print(f'разрешающий элемент [строка, столбец] = [{r\_row}, {r\_col}] = {r\_el}')

        print(f'симплекс-таблица: {table}')

        print()

        table[r\_row] = table[r\_row] / r\_el

        for row in range(rows):

            if row != r\_row:

                table[row] = table[row] - table[row, r\_col] \* table[r\_row]

    x = np.zeros(M)

    for col in range(M):

        if np.count\_nonzero(table[1:, col]) == 1 and np.any(table[1:, col] == 1):

            r = np.where(table[1:, col] == 1)[0][0] + 1

            x[col] = table[r, cols - 1]

    function\_val = -table[0, cols - 1]

    x\_norm = np.linalg.norm(x)

    print(f'x: {x} x\_norm: {x\_norm}')

    p = x / x\_norm

    print(f"двойственная задача \n \* p: {p} \n \* значение функции: {function\_val} \n")

    game\_price = 1 / round(function\_val, 2) - np.abs(beta)

    print(f'цена игры: {game\_price}\n')

def main():

    global beta

    dt = np.float64

    A = np.array([

        [-6, 65, -24, -14, 4, -19, 15, -79],

        [-75, -46, 6, 77, 16, 43, -95, 52],

        [-69, -98, -15, 2, -34, 31, -27, -32],

        [-19, -8, 99, -47, -86, 73, -95, -99],

        [-9, 80, 78, 93, 69, -93, 54, 87],

        [-67, 76, -10, -85, -82, 66, -93, -47]

    ])

    beta = np.min(A)

    print(f'beta: {beta}')

    print(f'нижняя цена игры: {lower\_price(A)}, верхняя цена игры: {higher\_price(A)}')

    for r in range(A.shape[0]):

        for c in range(A.shape[1]):

            A[r, c] = A[r, c] + np.abs(beta)

    b = np.array([1, 1, 1, 1, 1, 1]).reshape(-1, 1)

    c = np.array([1, 1, 1, 1, 1, 1, 1, 1])

    straight(np.copy(A.astype(dt)), np.copy(b.astype(dt)), np.copy(c.astype(dt)))

    duality(np.copy(A.astype(dt)), np.copy(b.astype(dt)), np.copy(c.astype(dt)))

if \_\_name\_\_ == '\_\_main\_\_':

    main()