|  |  |
| --- | --- |
| 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 02.03.01 Математика и компьютерные науки  профиль «Сквозные цифровые технологии» | |
|  | Студент группы  Б9120-02.03.01сцт - Пограничный Кирилл  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |
|  | Преподаватель: Яковлев Анатолий Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. |
| г. Владивосток  2023 | |
|  | |

**Постановка задачи:**

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

А=

Нижняя цена игры:

 -83

Верхняя цена игры:

 66

Искать равновесное решение в смешанных стратегиях будем с помощью симплекс-метода. Для этого необходимо сделать матрицу А неотрицательной, поэтому к каждому элементу матрицы А добавим модуль минимального элемента  матрицы А.

 -98

Получается неотрицательная матрица .

 =

Необходимо решить следующие задачи:

1. 
2. 

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.

**Приложение**

import numpy as np  
from tabulate import tabulate  
  
  
class Solution:  
 \_\_m = 6  
 \_\_n = 8  
 \_\_A = np.array([])  
 \_\_b = np.array([])  
 \_\_x = np.array([])  
 \_\_c = np.array([])  
 \_\_support\_matrix = 0  
  
 def \_\_init\_\_(self):  
 np.random.seed(17)  
 self.\_\_c = np.array(np.random.randint(100, size=self.\_\_n))  
 self.\_\_b = np.array(np.random.randint(100, size=self.\_\_m))  
 self.\_\_A = np.array(np.random.randint(-100, 100, size=(self.\_\_m, self.\_\_n)))  
  
 print("c:\n", self.\_\_c)  
 print("b:\n", self.\_\_b)  
 print("A:\n", self.\_\_A, "\n")  
  
 def \_\_fun\_for\_print(self, mat, index\_l, index\_c, t):  
 print(f"Шаг алгоритма #{t}")  
 print(tabulate(mat, tablefmt="latex", floatfmt=".2f"))  
 print(f"Индекс разрешающей строки = {index\_l}")  
 print(f"Индекс разрещающего столбца = {index\_c}")  
 print(f"Разрешающий элемент = {mat[index\_l][index\_c]}")  
 answer = []  
 for i in range(14):  
 if sum(mat[0:, i]) == 1 and mat[0, i] == 0:  
 index\_line = np.where(mat[1:, i] == 1)[0][0] + 1  
 answer.append(mat[index\_line][mat[0].size - 1])  
 else:  
 answer.append(0)  
 print("Промежуточное решение:\n", np.array(answer))  
  
 def \_\_create\_support\_mat(self, mat):  
 main\_mat = np.zeros((9, 23))  
 main\_mat[1:, :6] = mat  
 main\_mat[1:, 6:14] = -np.eye(8)  
 main\_mat[1:, 14:22] = np.eye(8)  
 main\_mat[1:, 22] = 1  
 main\_mat[0, 14:22] = 1  
  
 temp = 0  
 for i in range(1, main\_mat.shape[0]):  
 temp += main\_mat[i]  
 main\_mat[0] -= temp  
  
 return main\_mat  
  
 def \_\_simplex\_method(self, main\_mat, m, n):  
 t = 0  
 index\_min\_line = 0  
 index\_min\_col = 0  
 while np.any(main\_mat[0, :n - 1] < 0):  
 t += 1  
 index\_min\_col = np.where(main\_mat == main\_mat[0, :(n - 1)].min())[1][0]  
 div\_last = []  
 for i in range(1, m):  
 div\_last.append(main\_mat[i][-1] / main\_mat[i][index\_min\_col])  
  
 index\_min\_line = np.where(div\_last == min(filter(lambda x: x > 0, div\_last)))[0][0] + 1  
  
 self.\_\_fun\_for\_print(main\_mat, index\_min\_line, index\_min\_col, t)  
 main\_mat[index\_min\_line] /= main\_mat[index\_min\_line][index\_min\_col]  
 cur\_col = main\_mat[:, index\_min\_col]  
  
 for i in range(len(cur\_col)):  
 if cur\_col[i] != 1:  
 main\_mat[i] -= main\_mat[index\_min\_line] \* cur\_col[i]  
  
 print('END OF SIMPLEX!!!: \n')  
 t += 1  
 self.\_\_fun\_for\_print(main\_mat, index\_min\_line, index\_min\_col, t)  
 return main\_mat  
  
 def \_\_create\_main(self, c=0, b=0):  
 if c == 0 and b == 0:  
 main\_mat = np.zeros((7, 15))  
 main\_mat[0][:8] = -1  
 main\_mat[1:, :8] = self.\_\_A  
 main\_mat[1:, 8: 14] = np.eye(6)  
 main\_mat[1:, 14] = 1  
 return main\_mat  
  
 main\_mat = np.zeros((9, 15))  
 main\_mat[0][:c.shape[0]] = -c  
 main\_mat[1:, :6] = self.\_\_A  
 main\_mat[1:, 6: 14] = np.eye(8)  
 main\_mat[1:, 14] = b  
 return main\_mat  
  
 def \_\_to\_ready\_double\_problem(self, mat, b=0):  
 if b == 0:  
 mat[0, :6] = 1  
 mat[0, 14] = 0  
 for i in range(15):  
 if sum(mat[1:, i]) == 1 and mat[0, i] != 0:  
 index\_line = np.where(mat[1:, i] == 1)[0][0] + 1  
 mat[0, :15] -= mat[index\_line] \* mat[0][i]  
 mat[0][i] = 0  
  
 print('TO READY DOUBLE PROBLEM: ', tabulate(mat, tablefmt="latex", floatfmt=".2f"))  
 return mat  
  
 mat[0, :8] = b  
 mat[0, 14] = 0  
 for i in range(15):  
 if sum(mat[1:, i]) == 1 and mat[0, i] != 0:  
 index\_line = np.where(mat[1:, i] == 1)[0][0] + 1  
 mat[0, :15] -= mat[index\_line] \* mat[0][i]  
 mat[0][i] = 0  
  
 return mat  
  
 def execute(self):  
 tmp = []  
 for i in range(self.\_\_A[0].size):  
 tmp.append(max(self.\_\_A[0:, i]))  
 print("Верхняя цена игры:", min(tmp))  
  
 tmp.clear()  
  
 for i in range(self.\_\_A[0, :6].size):  
 tmp.append(min(self.\_\_A[i, 0:]))  
 print("Нижняя цена игры:", max(tmp))  
  
 beta = min(tmp)  
 print("β =", beta)  
 self.\_\_A[0:] += abs(beta)  
 print("\nНеотрицательная A:\n", self.\_\_A)  
  
 print("---Прямая---")  
 print(tabulate(self.\_\_simplex\_method(self.\_\_create\_main(), m=7, n=15),  
 tablefmt="latex", floatfmt=".2f"))  
  
 q = [0.00066279, 0.00086283, 0.00227322, 0., 0.00175046, 0.0038538, 0.00412645, 0.]  
 print("\nx =", q)  
 print("||y|| =", np.linalg.norm(q))  
 q /= np.linalg.norm(q)  
 print("q =", q)  
 print("Значение целевой функции =", 0.01, "\n\n")  
  
 support\_mat = self.\_\_simplex\_method(self.\_\_create\_support\_mat(self.\_\_A.T), m=9, n=23)  
  
 double\_mat = np.column\_stack([support\_mat[0:, :14], support\_mat[0:, 22]])  
 print("---DOUBLE MATRIX---\n")  
 print(tabulate(double\_mat, floatfmt=".2f", tablefmt="latex"))  
  
 double\_mat = self.\_\_to\_ready\_double\_problem(double\_mat)  
  
 print("---Двойственная---\n")  
 print("---Возвращаемое значение---\n", self.\_\_simplex\_method(double\_mat, m=7, n=15))  
  
 p = [0.00316193, 0.00234683, 0.00223194, 0.00312174, 0.00148573, 0.00118138]  
 print("\nx =", p)  
 print("||x|| =", np.linalg.norm(p))  
 p /= np.linalg.norm(p)  
 print("p = ", p)  
 print("Значение целевой функции =", 0.01)  
 fi = 1 / 0.01 - abs(beta)  
 print("fi = ", fi)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 Solution().execute()