Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Санкт-Петербургский государственный архитектурно-строительный университет»**

|  |  |
| --- | --- |
| Факультет: | Инженерной экологии и городского хозяйства |
| Кафедра: | Информационных систем и технологий |

|  |
| --- |
| ОТЧЕТ |
| Решение в смешанных стратегиях |
|  |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | Мельниченко Дмитрий Сергеевич | | |
| Направление подготовки | | 01.03.02 – Прикладная математика и информатика | |
|  | |  | |
| Группа: | ПМИб-4 | | |
|  | |  | |
| Руководитель СПбГАСУ:  Яркова Ольга Николаевна | | |  |
|  | | |  |

Санкт-Петербург

2023 г

Модели и методы анализа конфликтных ситуаций

Разработать программу для вычисления смешанных стратегий в стратегической игре. Входная информация: количество стратегий игрока А, количество стратегий игрока B, матрица выигрышей игрока А. Выходная информация: матрица игры после применения принципов доминирования; верхняя цена игры, нижняя цена игры, вывод о наличии/отсутствии решения в чистых стратегиях, при наличии седловой точки - решение игры и оптимальные стратегии; при отсутствии седловой точки - решение игры в смешанных стратегиях

Этапы метода:

1. Провести доминирование по строкам
2. Если второй игрок – не природа, провести доминирование по столбцам
3. В оставшейся матрице найти максимальные значения по столбцам и минимальные значения по строкам (метод максимина)
4. Найти минимум по максимумам и максимум по минимумам.
5. Оценить нижнюю и верхнюю цены игры.
6. Проанализировать наличии/отсутствии решения в чистых стратегиях, наличие седловой точки и оптимальных стратегий.
7. В случае отсутствия решения в чистых стратегиях найти решения в смешанных

Нахождение решения в смешанных стратегиях состоит в следующем:

В общем случае V\* ≠ V\* - седловой точки не существует. Оптимальное решение в чистых стратегиях также не существует. Однако, если расширить понятие чистой стратегии введением понятия смешанной стратегии, то удаётся реализовать алгоритм нахождения оптимального решения не вполне определённой игровой задачи. В такой ситуации предлагается использование статистического (вероятностного) подхода к нахождению оптимального решения антагонистической игры. Для каждого игрока, наряду с данным набором возможных для него стратегий, вводится неизвестный вектор вероятностей (относительных частот), с которыми следует применять ту или иную стратегию.

Обозначим вектор вероятностей (относительных частот) выбора заданных стратегий игрока A следующим образом:

Изображение выглядит как текст, Шрифт, белый, алгебра

Автоматически созданное описание

Аналогично для игрока B вводится неизвестный вектор вероятностей (относительных частот) имеет вид:

Изображение выглядит как текст, Шрифт, белый, типография

Автоматически созданное описание

Совокупность (комбинация) чистых стратегий A1, A2, …Am и B1, B2, …Bn в сочетании с векторами вероятностей выбора каждой из них называются смешанными стратегиями.

Основной теоремой в теории конечных антагонистических игр является Теорема фон Неймана: каждая конечная матричная игра имеет, по крайней мере, одно оптимальное решение, возможно, среди смешанных стратегий.

Из этой теоремы следует, что не вполне определённая игра имеет хотя бы одно оптимальное решение в смешанных стратегиях. В таких играх решением будет пара оптимальных смешанных стратегий P\* и Q\*, таких, что если один из игроков придерживается своей оптимальной стратегии, то и другому игроку не выгодно отклоняться от своей оптимальной стратегии.

Также стоит добавить следующее определение - Если вероятность (относительная частота) применения стратегии отлична от нуля, то такая стратегия называется активной.

Стратегии P\*, Q\* называются оптимальными смешанными стратегиями, если:

MA(P, Q\*) ≤ MA(P\*, Q\*) ≤ MA(P\*, Q)

В этом случае MA(P\*, Q\*) называется ценой игры и обозначается через V (V\* ≤ V ≤ V\*). Первое из неравенств (1)означает, что отклонение игрока A от своей оптимальной смешанной стратегии при условии, что игрок B придерживается своей оптимальной смешанной стратегии, приводит к уменьшению среднего выигрыша игрока A. Второе из неравенств означает, что отклонение игрока B от своей оптимальной смешанной стратегии при условии, что игрок A придерживается своей оптимальной смешанной стратегии, приводит к увеличению среднего проигрыша игрока B.

**Задание 2 – написание программного кода:**

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 1 Тестовые данные

Далее произведем доминирование матриц для поиска решения в чистых стратегиях.

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Рисунок 2 поиск седловой точки

Получившаяся матрица после доминирования:



Рисунок 3 матрица после доминирования

Построим систему уравнений для поиска вероятностей принятия той или иной стратегии:

Изображение выглядит как Шрифт, рукописный текст, текст, типография

Автоматически созданное описание =0

Изображение выглядит как Шрифт, текст, рукописный текст, снимок экрана

Автоматически созданное описание =0

Рисунок 4 система уравнений для поиска вероятностей

Где y это цена игры.

Далее найдем решение с помощью встроенных в питон методов решения СЛАУ:





Рисунок 5 использование метода питона для решения СЛАУ

В результате мы получим следующие вероятности принятия стратегий игроками:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание для игрока А, и

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание для игрока B

А также итоговая цена игры, как мат ожидание = 

Рисунок 6 итоговое решение

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 7 Результат работы с исходной матрицей.

**ПРИЛОЖЕНИЕ (код)**

from sympy import \*  
  
Inp\_type:int = int(input("Введите 0 для задания своей матрицы или 1 для тестовой(по варианту) или 2 для тестовой смешанных: "))  
size\_A =0  
size\_B =0  
  
if Inp\_type == 0:  
 size\_A:int = int(input("Введите кол-во стратегий для А: "))  
 size\_B:int = int(input("Введите кол-во стратегий для B: "))  
 matrix\_a =[]  
 for i in range(0,size\_A):  
 matrix\_a.append([])  
 for j in range(0,size\_B):  
 print("Введите элемент A(" + str(i+1) + "," + str(j+1) + ")")  
 matrix\_a[i].append(int(input("= ")))  
if Inp\_type == 1:  
 size\_A:int = 5  
 size\_B:int = 5  
 matrix\_a =[[-3,0,0,5,1],[2,1,7,6,3],[2,-5,4,7,8],[5,0,-3,1,4],[0,1,6,7,-1]]  
if Inp\_type == 2:  
 size\_A:int = 3  
 size\_B:int = 3  
 matrix\_a =[[4,7,2],[7,3,2],[2,1,8]]  
  
min\_A = []\*size\_A  
max\_B = []\*size\_B  
  
#Find down\_line  
for i in range(0,size\_A):  
 min\_val = matrix\_a[i][0]  
 for j in range(0,size\_B):  
 if matrix\_a[i][j] < min\_val:  
 min\_val = matrix\_a[i][j]  
 min\_A.append(min\_val)  
  
print("Max string A = ",min\_A)  
print("Нижняя граница игры = ",max(min\_A))  
  
# Find up\_line  
for i in range(0, size\_B):  
 max\_val = matrix\_a[0][i]  
 for j in range(0, size\_A):  
 if matrix\_a[j][i] > max\_val:  
 max\_val = matrix\_a[j][i]  
 max\_B.append(max\_val)  
  
print("Min string B = ",max\_B)  
print("Верхняя границы игры = ", min(max\_B))  
  
if min(max\_B) == max(min\_A) :  
 index\_a = 0  
 for i in range(0,len(min\_A)):  
 if max(min\_A) == min\_A[i]:  
 index\_a = i  
 break  
 index\_b = 0  
 for i in range(0, len(max\_B)):  
 if min(max\_B) == max\_B[i]:  
 index\_b = i  
 break  
 print("Седловая точка существует в A(",index\_a+1,") B(",index\_b+1,")")  
else:  
 print("Седловая точка не существует")  
  
print("Вывод матрицы после доминирования: ")  
  
a\_ok = 1  
b\_ok = 1  
new\_size\_a = size\_A  
new\_size\_b = size\_B  
  
new\_matrix = matrix\_a  
print("Start matrix =",new\_matrix)  
while a\_ok or b\_ok:  
 print("Start Dom")  
 #print("Size A =",new\_size\_a)  
 #print("Size B =",new\_size\_b)  
  
 a\_ok = 0  
 b\_ok = 0  
 index\_a\_for\_delete = set()  
  
  
 for start\_row in range(0, new\_size\_a-1):  
 for index\_a in range(start\_row+1,new\_size\_a):  
 index\_buf = index\_a  
 for index\_b in range(0,new\_size\_b):  
 if new\_matrix[index\_a][index\_b] > new\_matrix[start\_row][index\_b]:  
 index\_buf = -1  
 break  
 if index\_buf != -1:  
 a\_ok = 1  
 index\_a\_for\_delete.add(index\_buf)  
  
 for start\_row in range(new\_size\_a-1, 0,-1):  
 for index\_a in range(start\_row-1,-1,-1):  
 index\_buf = index\_a  
 for index\_b in range(0,new\_size\_b):  
 if new\_matrix[index\_a][index\_b] > new\_matrix[start\_row][index\_b]:  
 index\_buf = -1  
 break  
 if index\_buf != -1:  
 a\_ok =1  
 index\_a\_for\_delete.add(index\_buf)  
  
 new\_matrix\_buf = []\*len(index\_a\_for\_delete)  
 new\_size\_a\_buf = 0  
 for i in range(0,new\_size\_a):  
 if not (i in index\_a\_for\_delete):  
 new\_size\_a\_buf += 1  
 new\_matrix\_buf.append(new\_matrix[i])  
 new\_size\_a = new\_size\_a\_buf  
 new\_matrix = new\_matrix\_buf  
 index\_b\_for\_delete = set()  
  
 for start\_column in range(0, new\_size\_b - 1):  
 for index\_b in range(start\_column + 1, new\_size\_b):  
 index\_buf = index\_b  
 for index\_a in range(0, new\_size\_a):  
 if new\_matrix[index\_a][index\_b] < new\_matrix[index\_a][start\_column]:  
 index\_buf = -1  
 break  
 if index\_buf != -1:  
 b\_ok = 1  
 index\_b\_for\_delete.add(index\_buf)  
 for start\_column in range(new\_size\_b - 1, 0, -1):  
 for index\_b in range(start\_column - 1, -1, -1):  
 index\_buf = index\_b  
 for index\_a in range(0, new\_size\_a):  
 if new\_matrix[index\_a][index\_b] < new\_matrix[index\_a][start\_column]:  
 index\_buf = -1  
 break  
 if index\_buf != -1:  
 b\_ok = 1  
 index\_b\_for\_delete.add(index\_buf)  
  
  
 new\_matrix\_buf = []  
 new\_size\_b\_buf = 0  
 for index\_a in range(0,new\_size\_a):  
 new\_matrix\_buf.append([])  
 for index\_b in range(0, new\_size\_b):  
 if not (index\_b in index\_b\_for\_delete):  
 new\_size\_b\_buf += 1  
 new\_matrix\_buf[index\_a].append(new\_matrix[index\_a][index\_b])  
  
 new\_matrix = new\_matrix\_buf  
 new\_size\_b = int(new\_size\_b\_buf/new\_size\_a)  
 print("Iteration matrix = ",new\_matrix)  
  
  
  
  
A\_size = 0  
B\_Size = 0  
for row in new\_matrix:  
 A\_size+=1  
 buf = ""  
 B\_Size = 0  
 for val in row:  
 B\_Size+=1  
 buf+= str(val) + " "  
 print(buf)  
  
print("Result = = ",new\_matrix)  
print("New A size =",A\_size)  
print("New\_B\_Size =",B\_Size)  
P = []  
Q = []  
for i in range(0,A\_size):  
 P.append(Symbol('p' + str(i)))  
  
for i in range(0,B\_Size):  
 Q.append(Symbol('q' + str(i)))  
  
print(P)  
print(Q)  
  
Yy = Symbol('y')  
P\_system = [0]\*A\_size  
Q\_system = [0]\*B\_Size  
  
for i in range(0,A\_size):  
 for j in range(0, B\_Size):  
 Q\_system[i] += new\_matrix[i][j]\*P[j]  
 Q\_system[i] -= Yy  
 #print("Q[",i,"]",Q\_system[i])  
  
  
  
for i in range(0,B\_Size):  
 for j in range(0, A\_size):  
 P\_system[i] += new\_matrix[j][i]\*Q[j]  
 P\_system[i] -= Yy  
 #print("P[",i,"]",P\_system[i])  
  
Q\_system.append(-1)  
P\_system.append(-1)  
  
  
  
  
  
Q.append(Yy)  
P.append(Yy)  
P\_system,Q\_system = Q\_system,P\_system  
for j in range(0, B\_Size):  
 Q\_system[len(Q\_system) - 1] +=Q[j]  
for j in range(0, A\_size):  
 P\_system[len(P\_system) - 1] +=P[j]  
  
for i in range(0,A\_size+1):  
 print("Q[",i,"]",Q\_system[i])  
for i in range(0,B\_Size+1):  
 print("P[",i,"]",P\_system[i])  
print("Q\_c =",Q)  
print("P\_c =",P)  
print("Q\_s =",Q\_system)  
print("P\_s =",P\_system)  
  
print("\nResults:\n")  
  
for solution\_1 in linsolve(Q\_system, Q):  
 Result = solution\_1  
print("P = ",Result)  
  
check\_sym = 0  
for j in range(0,A\_size):  
 print(P[j]," =",float(Result[j]))  
 check\_sym += float(Result[j])  
  
print("Chek sym =",check\_sym)  
for solution\_2 in linsolve(P\_system, P):  
 Result = solution\_2  
print("Q=",Result)  
  
check\_sym = 0  
for j in range(0,A\_size):  
 print(Q[j]," =",float(Result[j]))  
 check\_sym+=float(Result[j])  
  
print("Chek sym =",check\_sym)  
print("y =",float(Result[len(Result)-1]))