纳什均衡求解器

201300096 杜兴豪 人工智能学院

模型描述

我的模型分为两个大的部分，有文件IO及数据预处理和纳什均衡求解器部分，在纳什均衡求解器中，我分别对两种不同的NE设计了算法，具体描述如下

文件IO和数据预处理

该部分需要处理输入文件，并将其转化为代码可以理解的形式。观察输入文件的组织（以example0为例）：

`['NFG 1 R "some\n', 'lines\n', 'of\n', 'comments" { "Player 1" "Player 2" } { 3 2 }\n', '\n', '1 1 0 2 0 2 1 1 0 3 2 0 \n']`

需要处理的有玩家、玩家操作数、收益矩阵部分。容易发现玩家操作数的个数就可以代表玩家的个数，因此我们只需要读取输入文件的倒数第三个部分（玩家操作数）和最后一个部分（收益矩阵）即可。

关于收益矩阵，文件中的排列为按照玩家顺序，依次列举同一个方块内的所有元素，这里采用的存储方法是为每个玩家独立分配一块空间存放自己的收益矩阵，有利于之后的线性规划求解，因此我们首先要将属于同一个玩家的收益提出，并变形为和博弈任务一致的形状。按照上述处理方法得到的输入输出对应关系如下：

```
raw input:
['NFG 1 R "some\n', 'lines\n', 'of\n', 'comments" { "Player 1" "Player 2" } { 3 2 }\n', '\n', '1 1 0 2 0 2 1 1 0 3 2 0
player actions:  [3, 2]
payoff matrix: [array([[1, 1],
       [0, 0],
       [0, 2]]), array([[1, 1],
       [2, 3],
       [2, 0]])]
```

根据得到的数据，构造纳什均衡求解器类（NESolver）并定义类中两个NE的求解算法，求解器类的成员如下：

```python
class NESolver:
    def __init__(self, player_actions, payoff_matrix):
        ''' 初始化成员变量 '''
        self.player_actions = player_actions
        self.payoff_matrix = payoff_matrix
        self.num_players = len(player_actions)
        self.PNE = []
        self.MNE = []

    def SolvePNE(self):
        ''' 求解纯策略纳什均衡 '''
        ...

    def SolveMNE(self):
        ''' 求解混合策略纳什均衡 '''
        ...

    def eliminate_dominate(self):
        ''' 识别收益矩阵中的占优策略，并删除弱势策略 '''
```

```
        '''

    ''' 以及对应的将求得的NE映射到合法输出的成员方法等 '''
```

读取文件的方法为`load_data`，将纳什均衡写入文件的方法为`write_nash`。

纯策略纳什均衡求解

PNE的计算可以利用每个玩家对每个位置只可能记录一次均衡点的特性做到。算法流程如下：

- 针对每个玩家，找出其有可能成为均衡点的所有点的坐标
- 当获得所有可能的候选点后，对每个玩家的每个候选点，查看其是否出现在所有玩家的候选集中
  - 如果确实存在于所有玩家的候选集，则该点为均衡点，加入PNE列表
  - 存在某个玩家，其候选集中不含有该点，则该点不是均衡点，则不加入PNE
- 获得所有PNE的坐标后，利用玩家操作数成员将坐标转化为输出格式的字符串

混合策略纳什均衡求解

由于题目仅需要考虑二人博弈下的MNE求解问题，因此无需考虑多人情况下的问题，这里分别对两个玩家的MNE进行求解。

考虑这样一个问题：当Player1的策略分布为MNE下的分布时，Player2在该分布下的任意概率分布下，应当都无法获得更优的收益，即其所有合理的可行解的收益应当相同，否则Player2可以选择让Player1收益更低的那个选择，此时无法均衡。

按照上述逻辑，可以为Player2构造线性规划问题（此时解出的是Player1的MNE）：

- 优化目标（最大化）：以Player1的各策略概率为变量，Player2的某个选择下的收益
- 等式约束：
  1. 优化目标和Player2的所有选择下的收益都应该相等
  2. Player1的所有策略概率和为1
- 不等式约束：Player1的任意选择概率大于等于0

以上述约束构建线性规划问题，利用`scipy.linprog`函数即可解出Player1的MNE

对于Player2的MNE求解和上述过程相同。

改进策略

针对特定问题（example0），其收益矩阵定义如下（Player1的选择为CDE）：

	A	B
C	1,1	1,1
D	0,2	0,3
E	0,2	2,0

观察到Player1选择D时，无论Player2的策略如何变化，其收益恒为0，此时若令Player2的策略为p_1, p_2，则可以列出线性方程组：

$$\begin{cases} p_1 + p_2 = v \\ 0 = v \\ 2p_2 = v \end{cases} \Rightarrow p_1 = p_2 = 0$$

此时线性规划就无解了（不满足概率和为0的等式约束）。但是我们知道$p_1 = p_2 = 0.5$就是一个合法的解，因此需要针对这个问题进行修改。观察到Player1的D选项其实是一个弱势策略，弱势策略的加入会导致这样的等式约束出现：

$$ap_1 + bp_2 = v = map_1 + nbp_2 \Rightarrow v = 0$$

当p_1, p_2都可变时，优化目标很容易就变成了0，进而导致所有策略都为0为合法解，此时又不满足等式约束，导致求解失败。

因此需要找到并删除弱势策略，即solver类中的`eliminate_dominate()`方法，其算法原理为：

- 针对每个Player，比较第i列和第j列
- 若有一列整列都大于另一列，则删除那个较小的列（同时玩家操作数-1）

将该方法应用于求解MNE方法的头部，即可避免出现由弱势策略导致的求解失败问题。（此处对收益矩阵进行了删除操作，并修改了玩家操作数，会导致生成满足输出要求的PNE方法生成错误的输出。因此该方法应该在PNE完全操作完毕后再使用）

实验结果展示

针对有答案的example文件，我们对比算法输出和实际结果如下：

```
H:\Anaconda3\envs\torch\python.exe E:\NJU\JuniorII\博弈论\hw\博弈论期中大作业\main.py
reference: ['1,0,0,1,0\n', '1,0,0,1/2,1/2\n']
solution : ['1,0,0,1,0\n', '1,0,0,0.5,0.5\n']
reference: ['1,0,0,1,0\n', '1,0,0,0.5,0.5\n']
solution : ['1,0,0,1,0\n', '1,0,0,0.5,0.5\n']
reference: ['0,1/3,1/3,0,0,1/3,1/3,0,0,1/3,1/3,0\n', '0,1/3,1/3,0,0,1/3,0,1/3,1/3,0,0,1/3\n', '1/3,0,0,1/3,1/3,0,1/3,0,0,1/3,1/3,0
solution : ['0,0.333,0.333,0,0,0.333,0,0.333,0.333,0,0,0.333\n']
reference: ['[0,1,0,1]\n']
solution : ['0,1,0,1\n']
reference: ['[1,0,0,1,0,0,1,0,0]\n', '[0,1,0,0,1,0,0,1,0]\n', '[0,0,1,0,0,1,0,0,1]']
solution : ['1,0,0,1,0,0,1,0,0\n', '0,1,0,0,1,0,0,1,0\n', '0,0,1,0,0,1,0,0,1\n']
reference: ['0,1,0,0,0,0,0,0,0,0,0,1,0,0,0\n', '0,439/1560,0,0,19/120,0,0,437/780,0,0,0,0,575/1363,86/1363,702/1363,0\n', '0,87/
solution : ['0,1,0,0,0,0,0,0,0,0,0,1,0,0,0\n', '0,0,0,1,0,0,0,0,0,0,0,0,1,0,0\n']
```

其中reference为读取示例中的结果，solution为对应的算法结果。由于`scipy.linprog`函数只能求出一组MNE解，因此很多具有多组MNE解的问题没能完全回答，但大体上是一致的，证明实现结果正确。

并且由于我的实现中有大量剪枝操作，并且没有调用过多复杂函数，因此我的实现的计算效率极高，计算5个example中的NE并写入文件总共只需要0.15s左右，因此理论上在问题足够多，并且保证正确率的情况下，我的实现得分很高。