博弈论期中作业实验报告

191300087 左之睿

代码使用方法

直接运行即可，但是要求nfg文件内部必须有以下格式：
Players: 4
Actions: 4 4 4 4
（必须有这个换行，否则无法识别收益）
3 2 ...（收益，最后以换行符结尾）

此外，收益数字之间只能有一个空格(split设置的是按一个空格分割)且以换行符结尾，否则会将数字和换行符读成一个元素导致失败
(我设置的读取规则是从Players:开头的行读取玩家数目，从Actions:开头的行读取每个玩家的策略个数，并且将nfg内的空行视为收益的前一行)
对于input文件夹里的所有输入均可正常运行，但对于example里的部分输入会加载失败，人工修改输入格式后除了格式上有差异(如分数和小数，1和1.0)，实际上结果与给出的output相同

使用的算法

对于二人博弈找所有的混合策略纳什均衡，使用Labeled Polytopes Algorithm
对于多人博弈找纯策略纳什均衡，则是通过计算最优反应来找NE

大体思路

二人

使用Labeled Polytopes算法，
求解半空间中的极点：

$$P = \{x \in R^M | x \geq \mathbf{0},\ B^T x \leq \mathbf{1}\}$$
$$Q = \{y \in R^N | Ay \leq \mathbf{1},\ y \geq \mathbf{0}\}$$

注意算法要求收益矩阵元素非负，故需要遍历收益矩阵，只要遇到负元素就可以停止，直接加上最小的负元素的绝对值即可，这样并不会影响求出的混合策略。

此外，python求解线性优化问题只能有一个不等式，因此需要将$x \geq \mathbf{0}$这一条件融入$B^T x \leq \mathbf{1}$，即在B后补充一个负的单位矩阵，并在$\mathbf{1}$下补充相应数目的0

论文中结论为$x \in P - \{\mathbf{0}\}, y \in Q - \{\mathbf{0}\},\ if\ (x, y)\ is\ completely\ labeled,\ ne\ is\ (x/1^T x, y/1^T y)$

结合该结论，首先求解极点，再求解满足条件的下标即可。

多人

对每个玩家i计算其他玩家动作的笛卡尔积(这里使用了标准库itertool中的product函数)。
对于每个笛卡尔积，玩家i有几种策略，那他就有几种收益，我们从中选取最大的那个就是当前笛卡尔积(对手行为)下的最优反应

得到所有玩家的最优反应后，从中选取在每个玩家的最优反应都出现了的反应即为PNE，详情见代码注释

其他模块代码

加载数据

这一部分较为朴素，读取输入文件，然后根据行首字符来判断是否是所需要的数据

```python
def load_data(filename):
    with open(filename, 'r') as f:
        flag = 0
        payoffs = []
        tmp_payoff = []
        for line in f:
            if line[0:8] == "Players:":
                n_players = line[9]
            if line[0:8] == "Actions:":
                actions = list(line[9:-1].split(" "))
            if line.count('\n') == len(line):  # 意味着这一行是空行，后面全是收益
                flag = 1
                continue
            if flag == 1:
                tmp_payoff.append(line.split(" "))
    tmp_payoff[0].pop()
    for i in range(len(tmp_payoff[0])):
        tmp_payoff[0][i] = int(tmp_payoff[0][i])
    tmp_payoff = list(np.ravel(tmp_payoff))  # 把所有收益列成了一个长列表
    '''str转int'''
    n_players = int(n_players)
    for i in range(len(actions)):
        actions[i] = int(actions[i])
    for i in range(n_players):  # 一个玩家一个ndarray,n个玩家一组受益
        p = [tmp_payoff[j] for j in range(len(tmp_payoff)) if j % n_players == i]
        tmp = np.array([p])
        tmp = np.reshape(tmp, actions, order='F')
        payoffs.append(tmp)

    return n_players, actions, payoffs
```

输出文件

将计算出来的解写入到输出文件中，当混合策略的概率值太小时，将其舍入为0

```python
def write_ne(result, out_path):
    with open(out_path, 'w') as f:
        for res in result:  # res是一个解,result是解得集合
            to_str = [str(i) if i > 1e-15 else '0' for i in res]
            tmp_str = to_str[0]
            for i in range(1, len(to_str) - 1):
                tmp_str = tmp_str + ',' + to_str[i]
            tmp_str = tmp_str + ',' + to_str[-1] + '\n'
            f.writelines(tmp_str)
```

结果分析

打印运行时间，即：

```
for f in os.listdir('input'):
    if f.endswith('.nfg'):
        t1 = time.time()
        nash('input/' + f, 'output/' + f.replace('nfg', 'ne'))
        t2 = time.time()
        print("computing " + f + " time costs: ", t2 - t1)
```

最终结果分析如下：

1、求PNE的耗时都较短

2、求二人MNE时，策略空间较大时耗时会比较长，这是因为极点数量很多，比如13,14,15.nfg，需要花费5-15秒时间

3、通常来说，求解pne或者mne都只需要零点几秒甚至更短的时间