博弈论及其应用
期中大作业: 纳什均衡求解器
实验报告

191250026, 丁云翔, 191250026@smail.nju.edu.cn

2021 年 5 月 28 日

实验报告中只简要介绍使用的算法和实现过程，完整实现见”main.py”。

核心算法位于nash函数中，以及我添加的一个readPayoff函数（均位于”main.py”）。在nash函数中，首先对输入文件进行读取和解析，从中提取出玩家数、每个玩家的的动作数以及所有的收益，实现如下图：

```
def nash(in_path, out_path):
    # load file
    global index
    n_player = 0
    actionNum = []
    file = open(in_path, "r")
    allLines = file.readlines()
    for line in allLines:
        if '{' in line:
            line1 = line.split('{')
            line2 = line1[1].split('}')[0].split("\"")
            while '' in line2:
                line2.remove('')
            while ' ' in line2:
                line2.remove(' ')
            n_player = len(line2)
            line3 = line1[2].split('}')[0].split(" ")
            while '' in line3:
                line3.remove('')
            for i in line3:
                actionNum.append(int(i))
    payoff = allLines[-1].split(" ")
    while '' in payoff:
        payoff.remove('')
    index = 0
    payoffMaxtix = readPayoff(payoff, n_player, n_player, actionNum)
```

然后调用readPayoff函数，解析收益序列，利用numpy提供的高维数组存储所有玩家所有可能的动作组合所对应的收益。这样存储以后可以依次以所有玩家选取的动作序号和玩家序号为下标，从高维数组中读出在这一所有玩家的动作组合中某个玩家的收益，实现如下图：

```
index = 0

def readPayoff(payoff, remain, n_player, actionNum):
    if remain == 0:
        point = np.zeros(n_player, dtype=np.int)
        global index
        for i in range(0, n_player):
            point[i] = payoff[index]
            index += 1
        return point
    else:
        newlist = readPayoff(payoff, remain - 1, n_player, actionNum)
        for i in range(0, actionNum[remain - 1]):
            if i == 0:
                newlist = np.expand_dims(newlist, 0).repeat(actionNum[remain - 1], axis=0)
            else:
                newlist[i] = readPayoff(payoff, remain - 1, n_player, actionNum)
    return newlist
```

读取完成后回到nash函数，首先求纯策略纳什均衡PNE，求解思路是对所有动作组合进行判断，如果在这一组合下，任何玩家单独改变策略，其自身的收益都不增加，则这一动作组合为一个纳什均衡。故只需遍历所有动作组合，并依次遍历所有玩家的其他动作，判断收益是否增加即可，实现如下图：

```
# get NE
result = []
mixResult = []
nowAction = np.zeros(n_player, dtype=np.int)
notquit = 1
while notquit == 1:
    flag = 1
    now = payoffMaxtix
    for n in range(0, n_player):
        now = now[nowAction[n]]
    for i in range(0, n_player):
        for j in range(0, actionNum[-(i+1)]):
            temp = payoffMaxtix
            for k in range(0, i):
                temp = temp[nowAction[k]]
            temp = temp[j]
            for k in range(i + 1, n_player):
                temp = temp[nowAction[k]]
            if now[-(i+1)] < temp[-(i+1)]:
                flag = 0
    if flag == 1:
        result.append(list(nowAction))
    nowAction[-1] += 1
    for i in range(1, n_player + 1):
        if nowAction[-i] >= actionNum[i - 1]:
            nowAction[-i] = 0
            if i == n_player:
                notquit = 0
                break
            else:
                nowAction[-(i + 1)] += 1
```

然后再判断玩家数是否为2，如果是则继续求解混合策略纳什均衡MNE，求解思路是线性规划（参考了Labeled polytopes算法）。以玩家1为例，假设玩家2有n个动作，其选择每个动作的概率依次为$p_1, p_2, ..., p_{n-1}, 1-p_1-p_2-...-p_{n-1}$（每个概率都应属于$[0,1]$），则可以根据玩家2选择动作的概率计算玩家1选择每个动作时玩家1自身收益的期望，则需最小化优化目标u，且u应大于等于玩家1选择每个动作时玩家1自身收益的期望，所得到的n个概率的取值即为混合策略纳什均衡中玩家2的各个动作的概率取值，同理能得到混合策略纳什均衡中玩家1的各个动作的概率取值，由此可以求解出一个混合策略纳什均衡。其中线性规划的求解使用scipy.optimize中的linprog函数，求解后需检验是否与纯策略纳什均衡相同，若相同则舍弃以避免重复，实现如下图：

```
if n_player == 2:
    c = [1]
    A = []
    b = []
    another = [0]
    u_bounds = (None, None)
    all_bounds = [u_bounds]
    for i in range(0, actionNum[0] - 1):
        c.append(0)
        another.append(1)
        all_bounds.append((0, 1))
    for i in range(0, actionNum[1]):
        temp = [-1]
        for j in range(0, actionNum[0] - 1):
            temp.append(payoffMaxtix[i][j][1] - payoffMaxtix[i][-1][1])
        A.append(temp)
        b.append(-payoffMaxtix[i][-1][1])
    A.append(another)
    b.append(1)
    c = np.array(c)
    A = np.array(A)
    b = np.array(b)
    res1 = linprog(c, A_ub=A, b_ub=b, bounds=tuple(all_bounds))
    c = [1]
    A = []
    b = []
```

```
    another = [0]
    u_bounds = (None, None)
    all_bounds = [u_bounds]
    for i in range(0, actionNum[1] - 1):
        c.append(0)
        another.append(1)
        all_bounds.append((0, 1))
    for i in range(0, actionNum[0]):
        temp = [-1]
        for j in range(0, actionNum[1] - 1):
            temp.append(payoffMaxtix[j][i][0] - payoffMaxtix[-1][i][0])
        A.append(temp)
        b.append(-payoffMaxtix[-1][i][0])
    A.append(another)
    b.append(1)
    c = np.array(c)
    A = np.array(A)
    b = np.array(b)
    res2 = linprog(c, A_ub=A, b_ub=b, bounds=tuple(all_bounds))
    mixResult.append([res1.x, res2.x])
```

由于linprog函数只能求得一个最优解，当有无穷个混合策略纳什均衡时，linprog函数求得的最优解为其中的一个，而求解极点难以实现，故算法最后的结果中只有纯策略纳什均衡和无穷个混合策略纳什均衡中的一个。

最后将结果按输出格式写入输出文件，实现如下图：

```
# write file
file = open(out_path, "w")
for i in result:
    resultstr = "("
    for n in range(0, n_player):
        if n != 0:
            resultstr += ','
        resultstr += '('
        for j in range(0, actionNum[n]):
            if j != 0:
                resultstr += ','
            if j == i[n_player - 1 - n]:
                resultstr += '1'
            else:
                resultstr += '0'
        resultstr += ')'
    resultstr += ')'
    file.write(resultstr + '\n')
if n_player == 2:
    for i in mixResult:
        flag = 1
        resultstr = "("
        resultstr += '('
        p = 1.0
        for j in range(0, actionNum[0] - 1):
            if abs(1.0 - i[0][j + 1]) < 10e-5:
                flag = 2
                resultstr += '1'
                p -= 1
            elif abs(i[0][j + 1]) < 10e-5:
                resultstr += '0'
            else:
                resultstr += str(i[0][j + 1])
                p -= i[0][j + 1]
            resultstr += ','
        if abs(1 - p) < 10e-5:
            flag = 2
            resultstr += '1'
        elif abs(p) < 10e-5:
            resultstr += '0'
        else:
            resultstr += str(p)
        resultstr += ')'
        resultstr += ','
        resultstr += '('
        p = 1.0
        for j in range(0, actionNum[1] - 1):
            if flag == 2 and abs(1.0 - i[1][j + 1]) < 10e-5:
                flag = 0
                resultstr += '1'
                p -= 1
            elif abs(i[1][j + 1]) < 10e-5:
```

```
                resultstr += '0'
            else:
                resultstr += str(i[1][j + 1])
                p -= i[1][j + 1]
            resultstr += ','
        if flag == 2 and abs(1 - p) < 10e-5:
            flag = 0
            resultstr += '1'
        elif abs(p) < 10e-5:
            resultstr += '0'
        else:
            resultstr += str(p)
        resultstr += ')'
        resultstr += ')'
        if flag != 0:
            file.write(resultstr + '\n')
file.close()
```

在examples文件夹下对应的测试文件上进行测试，对比标准答案发现纯策略纳什均衡均求解正确，求得的一个混合策略纳什均衡也均位于解空间的区间内。