作业

纳什均衡求解器

庄镇华 502022370071

A Game Theory Homework Assignment

2023 年 4 月 21 日

1 实验目的

- 掌握求解纳什均衡的相关算法
- 锻炼数学基础能力及编程求解问题的能力

2 实验内容

本次实验要求使用 Python 语言在给定代码框架下编程求解纳什均衡 (Nash Equilibrium, NE), 包括纯策略 NE 与混合策略 NE, 并提交相应源码、输出文件以及实验报告。

3 算法原理

Labeled polytopes 算法

Labeled polytopes 算法即为标签多面体算法。简单来讲，当混合策略 NE 有无穷多个时，可表示为若干凸集的并集，给解空间的极点设置一些标签 k，这些混合策略 NE 凸集极点的标签需要被完全标记，因而可以据此找到所有的混合策略 NE。

3.1 最优反应多面体

混合策略的求解过程即为最优反应多面体（best response polyhedron）的求解过程。设 x 和 y 分别表示玩家 1 和玩家 2 的混合策略，A 和 B 分别表示玩家 1 和玩家 2 的收益矩阵，u 和 v 分别表示玩家 1 和玩家 2 期望收益的上包络（upper envelope），M 和 N 表示玩家 1 和玩家 2 的纯策略集合，$M = \{1, \dots, m\}, N = \{m+1, \dots, m+n\}$，玩家 1 和玩家 2 的最优反应多面体 $\overline{P}$ 和 $\overline{Q}$ 分别被定义为

$$\overline{P} = \{(x, v) \in \mathbb{R}^M \times \mathbb{R} \mid x \geq \mathbf{0}, \mathbf{1}^\top x = 1, B^\top x \leq \mathbf{1}v\},$$
$$\overline{Q} = \{(y, u) \in \mathbb{R}^N \times \mathbb{R} \mid Ay \leq \mathbf{1}u, y \geq \mathbf{0}, \mathbf{1}^\top y = 1\}.$$

3.2 完全标记

关于标签的概念，具体来说，当点 (y, u) 使得 $\overline{Q}$ 中的第 k 个不等式取等号时，我们认为该点有标签 $k \in M \cup N$。并且当 $k = i \in M$ 时，$\sum_{j \in N} a_{ij} y_j = u$ 成立；当 $k = j \in N$ 时，$y_j = 0$ 成立。同理，可以定义 $\overline{P}$ 中点的标签，即如果 $x_i = 0$，则有标签 $i \in M$；如果 $\sum_{i \in M} b_{ij} x_i = v$，则有标签 $j \in N$。

关于完全标记的概念，对于 $((x, v), (y, u)) \in \bar{P} \times \bar{Q}$，当它是完全标记时，$(x, y)$ 为纳什均衡点，完全标记意味着对于任意标记 $k \in M \cup N$，k 都会出现在 (x, v) 或 (y, u) 中。

3.3 形式化简

为了方便求解，我们可以通过平移收益矩阵的方式使得 A 和 $B^\top$ 非负，这样期望收益的上包络 u 和 v 均为非零向量，可以同时除以来化简掉 u 和 v，得到新的最优反应多面体

$$P = \{x \in \mathbb{R}^M \mid x \geq \mathbf{0}, B^\top x \leq \mathbf{1}\},$$
$$Q = \{y \in \mathbb{R}^N \mid Ay \leq \mathbf{1}, \quad y \geq \mathbf{0}\}.$$

3.4 求解结果

求解出 P 和 Q 之后，可以通过遍历两个集合的极点来找到均衡点，对于 $x \in P - \{\mathbf{0}\}$ 和 $y \in Q - \{\mathbf{0}\}$，如果 (x, y) 是完全标记的，那么 $\left(x/(1^\top x), y/(1^\top y)\right)$ 就是混合策略纳什均衡。

4 实现过程

我们将整个过程分为 4 个部分，即文件读取、纯策略求解、混合策略求解和结果输出。

4.1 文件读取

实验以 examples 文件夹下的.nfg 文件为样例输入，输入格式参考 Gambit format，最主要的信息包括玩家数目、玩家动作数目、玩家收益矩阵等。通过观察 Gambit 格式发现，玩家收益矩阵在最后一行，而玩家动作在倒数第三行，因此通过 readlines 方式按行读取数据，通过正则匹配函数来定位'{ }' 里的玩家动作数目，该数组的长度即为玩家数目。关于玩家收益，需要按玩家来整合每个玩家的收益矩阵，使用 numpy 的 reshape 函数按照 Fortran 的方式进行高维优先整合。

```
# read the strategic game (.nfg) file in Gambit format
def read_info(in_path):
    with open(in_path) as f:
        lines = f.readlines()
        payoffs, players = lines[-1].strip().split(), lines[-3]
    all_payoffs, action_shapes = [], [int(_) for _ in re.search('{ (\d+ )*}',
    players).group()[1:-1].split()]
    for player in range(len(action_shapes)):
        payoff = np.array([int(_) for i, _ in enumerate(payoffs) if i % len(
    action_shapes) == player]).reshape(action_shapes, order='F')
        all_payoffs.append(payoff)
    return np.array(all_payoffs), action_shapes
```

4.2 纯策略求解

当有 3 个即以上玩家时，仅需要求解所有纯策略纳什均衡，每个玩家都有自己的收益矩阵，并且收益矩阵中的第 i 维对应于第 i 个玩家的策略。对于玩家 i，要计算在其余玩家选定策略后，它选择哪个策略可以达到最大收益，只需计算玩家 i 的收益矩阵中第 i 维（numpy 数组定义的维度顺序）的最大值。使用 np.where 和 np.max 函数得到所有玩家的坐标集合，然后求交集即为最终的所有纯策略纳什均衡。

```
# more than two players, solve all pure strategy equilibrium
def calc_pne(all_payoffs):
    player_num, action_shapes, pnes = len(all_payoffs), all_payoffs[0].shape, set
    ()
    for i, payoff in enumerate(all_payoffs):
        cur_pnes = set()
```

```
        max_pos = list(np.where(payoff == np.max(payoff, axis=i, keepdims=True)))
        cur_pnes_num = len(max_pos[0])
        for j in range(cur_pnes_num):
            temp = []
            for k in range(player_num):
                temp.append(max_pos[k][j])
            cur_pnes.add(tuple(temp))
        pnes = cur_pnes if i == 0 else pnes.intersection(cur_pnes)

    nes = []
    for pne in pnes:
        ne = []
        for _ in range(player_num):
            cur = [0] * action_shapes[_]
            cur[pne[_]] = 1
            ne += cur
        nes.append(ne)
    return nes
```

4.3 混合策略求解

Labeled polytopes 算法原理在上一部分已经介绍过，下面简单介绍代码实现。化简后的最优多面体表示为

$$P = \{x \in \mathbb{R}^M \mid x \geq \mathbf{0}, B^\top x \leq \mathbf{1}\},$$
$$Q = \{y \in \mathbb{R}^N \mid Ay \leq \mathbf{1}, \quad y \geq \mathbf{0}\}.$$

我们使用 scipy.spatial 库中的 HalfspaceIntersection 函数计算 P 和 Q 的极点。将原始不等式转化为函数需要的形式

$$P = \left\{x \in \mathbb{R}^M \mid -x \leq 0, B^\top x \leq 1\right\}$$
$$Q = \left\{y \in \mathbb{R}^N \mid Ay \leq 1, \quad -y \leq \mathbb{0}\right\}$$

该函数包含两个参数，分别为 halfspaces 和 interior_point，前者代表约束不等式形成的 halfspaces，将 $Mx + b \leq 0$ 不等式以 $[M; b]$ 形式表示，后者表示在 halfspaces 区域内部的点，可以通过如下线性规划求得

$$\begin{aligned} &\max && y \\ &s.t. && Mx + y\|M_i\| \leq -b \end{aligned}$$

得到约束空间的极点后，接下来计算非零极点的标记，对于 halfspaces 为 $[M, b]$ 的非零极点 z，$Mz - b$ 向量中值为零的元素所在位置就是 z 的标记。计算出 P 和 Q 中极点及其标记后，继续遍历两个集合，找到完全标记的组合即为最终结果。

```
# two players, solve all pure and mixed strategy equilibrium via labeled polytopes
def calc_mne(all_payoffs):
    res = []
    A, B = all_payoffs
    # make sure A and B are nonnegative to eliminate the u and v
    if np.min(A) < 0:
        A = A + abs(np.min(A))
```

```
    if np.min(B) < 0:
        B = B + abs(np.min(B))

    m, n = A.shape
    P_halfspace = calc_halfspace(B.transpose(), 'P')
    Q_halfspace = calc_halfspace(A, 'Q')
    P_vertices = calc_vertices_and_labels(P_halfspace)
    Q_vertices = calc_vertices_and_labels(Q_halfspace)

    for x_vertex, x_label in P_vertices:
        labels = np.zeros(m + n, dtype=int)
        labels[x_label] = 1
        for y_vertex, y_label in Q_vertices:
            labels[y_label] += 1
            if np.all(labels):
                res.append(np.append(x_vertex / sum(x_vertex), y_vertex / sum(
    y_vertex)))
            labels[y_label] -= 1
    return res
```

4.4 结果输出

按照规定格式输出结果，即输出文件每一行对应一个 NE, 每一行中的元素依次表示从第一个玩家到最后一个玩家采取每个动作的概率，使用 Fraction().limit_denominator() 函数以分数形式输出，方便测试对比。

```
def write_res(nes, out_path):
    with open(out_path, 'w') as f:
        for ne in nes:
            line = ''
            for i, v in enumerate(ne):
                v = str(Fraction(v).limit_denominator())
                line += v if i == 0 else ',' + v
            f.write(line + '\n')
```

5 实验分析

以 examples 文件夹下的.nfg 文件为样例输入，.ne 文件为样例输出，对代码进行了测试。测试结果显示，程序的输出结果和样例输出完全一致，这说明了代码的正确性，并且代码结果以分数的形式输出，方便对比测试。

相较于其他算法，实验采用的 Labeled polytopes 算法实现较复杂，但效率较高，能得出所有结果；其他比如 Support enumeration 算法实现简单，效率低，能得出大部分结果，Lemke-Howson 效率高但只保证一个 NE；因而最终选择实现 Labeled polytopes 算法。