1 Summary

After learning about online learning and a proof of the minimax theorem and support we started exploring computing Nash equilibria in two player games. In this lecture we learn how to find the Nash equilibrium for both players using Linear Programming if we know the support of the Nash for both players. We look at the brute-force approach and then a smarter approach made by Carlton E. Lemke and Joseph T. Howson [1]. It is said to be one of the best combinatorial algorithms for finding a Nash equilibrium but is exponential in the worst case [2]. We then analyze the algorithm.

2 Linear Program

If given the support for the Nash in a two player game we can compute it using Linear Programming. Let $R, C \in \mathbb{R}^{n \times m}$ be the payoff matrices for the row and column players respectively. Any support $S, T (x,y)$ must satisfy:

- $x_i \geq 0, \forall i \in [n]$  
- $x_i = 0, \forall i \notin S$  
- $\sum_{i \in S} x_i = 1$  
- $(Ry)_i \geq (Ry)_j, \forall i \in S, j \in [n]$  
- $y_i \geq 0, \forall i \in [m]$  
- $y_i = 0, \forall i \notin T$  
- $\sum_{i \in T} y_i = 1$  
- $(C^\top x)_i \geq (C^\top x)_j, \forall i \in T, j \in [m]$  

3 The Brute-force algorithm

The Brute-force algorithm to solve the linear program simply checks for feasibility of $LP(S,T)$ for index sets $S, T$. If any feasible solution tuple $(x,y)$ is found it is a Nash.
\[
LP(S,T)
\]

\[\begin{align*}
(C^\top x)_i & \geq (C^\top x)_j \forall i \in T, j \in [m]. \\
(Ry)_i & \geq (Ry)_j \forall i \in S, j \in [n]. \\
\sum_{i \in S} x_i & = 1. \\
\sum_{i \in T} y_i & = 1. \\
x_i & = 0 \forall i \notin S. \\
y_i & = 0 \forall i \notin T. \\
x_i & \geq 0 \forall i \in [n]. \\
y_i & \geq 0 \forall i \in [m].
\end{align*}\]

The worst case complexity of the brute-force algorithm is exponential in the size of the strategies of the row and column player. In particular it equals, \(O(2^{n+m}) = 2^{n+m} \cdot \text{poly}(n,m)\).

### 4 The Lemke-Howson algorithm

The Lemke-Howson algorithm assumes that the payoff matrices for the row and column player are non-negative i.e. all values in the matrices are \(\geq 0\).

The basic approach of the Lemke-Howson algorithm is to maintain a single guess of the supports and modify it slightly every iteration.

\[P_1 = \{ x \in \mathbb{R}^n : \forall i \in [n] \ x_i \geq 0, \forall j \in [m] \ (x^\top C)_j \leq 1 \}.\]
\[P_2 = \{ y \in \mathbb{R}^m : \forall i \in [m] \ y_i \geq 0, \forall j \in [n] \ (Ry)_j \leq 1 \}.\]

\[\text{nrml}(x) = \left( \sum_{i \in [n]} x_i \right)^{-1} \cdot x \quad \text{nrml}(y) = \left( \sum_{i \in [m]} y_i \right)^{-1} \cdot y\]

**Definition 4.1 (Label)** \(x\) has label \(i\) if \(x_i = 0\) or \((x^\top C)_i = 1\). \(y\) has label \(j\) if \(y_j = 0\) or \((Ry)_j = 1\).

**Lemma 4.1** Let \(x^* \in P_1, y^* \in P_2, x^*, y^*\) have all labels and assume \(x^*, y^*\) are non-zero vectors. It holds that \((\text{nrml}(x^*),\text{nrml}(y^*))\) is a Nash equilibrium.

**Proof:** \(\forall i \in [n],\) either \(x_i^* = 0\) or \((Ry^*)_i = 1\) i.e. \(i\) is the best response of the row player to \(\text{nrml}(y^*)\).

\(\forall j \in [m],\) either \(y_j^* = 0\) or \((x^* C)_j = 1\) i.e. \(j\) is the best response of the column player to \(\text{nrml}(x^*)\).
We can conclude that,

\[ x^*_i > 0 \Rightarrow (Ry^*)_i \geq (Ry^*)_j \ \forall j \in [n]. \]
\[ y^*_i > 0 \Rightarrow (x^* C)_i \geq (x^* C)_j \ \forall j \in [m]. \]

\[ \therefore \text{the same is true for nrml}(x^*),\text{nrml}(y^*). \]

They satisfy the linear program \( LP(supp(x^*), supp(y^*)) \) and thus the inverse is true as well.

**Definition 4.2 (Vertex)** A vertex player of polytope \( P_1 \) is given by \( n \) linearly independent equalities (the test constraints of \( P_1 \) are strict inequalities). A vertex for \( P_2 \) is given by \( m \) linearly independent equalities (the rest constraints of \( P_1 \) are strict inequalities).

For \( P_1 \cup P_2 \) is \( n + m \). This is the non-degenerate case.

We define the Lemke-Howson algorithm as follows then:

```
Algorithm 1 Lemke-Howson algorithm
\[ x = 0 \text{ and } y = 0. \]
\[ k = k_0 = 1. \]
\[ \text{loop} \]
\[ \quad \text{In } P_1 \text{ find the neighbor vertex } x' \text{ of } x \text{ with label } k' \text{ instead of } k. \]
\[ \quad \text{Remove label } k \text{ and add label } k'. \]
\[ \quad \text{Set } x = x'. \]
\[ \quad \text{if } k' = 1 \text{ then STOP.} \]
\[ \quad \text{end if} \]
\[ \quad \text{In } P_2 \text{ find the neighbor vertex } y' \text{ of } y \text{ with label } k'' \text{ instead of } k'. \]
\[ \quad \text{Remove label } k' \text{ and add label } k''. \]
\[ \quad \text{Set } y = y'. \]
\[ \quad \text{if } k'' = 1 \text{ then STOP.} \]
\[ \quad \text{end if} \]
\[ \quad \text{Set } k = k''. \]
\[ \text{end loop} \]
```

**Theorem 4.2** The Lemke-Howson algorithm outputs a Nash equilibrium.

**Proof:** Define a graph with vertices in \( P_1 \cup P_2 \). Each vertex \((x, y)\) has:

- One duplicate label. This vertex is adjacent to exactly two other vertices, since we can remove the duplicate label from \( x \) and pivot in \( P_1 \), or remove the duplicate label from \( y \) and pivot in \( P_2 \).
• They have all labels exactly once. This vertex has only one neighbor (remove label 1 from whichever vector has it).

Since each vertex in the graph has degree 1 or 2, the graph is a union of cycles and paths.

1. Lemke-Howson algorithm begins at the configuration (0, 0).
2. (0, 0) has all labels and is therefore an endpoint of a path component.
3. The algorithm will terminate at the other endpoint of the path.
4. The other point is not (0, 0) and cannot be (x, 0) or (0, y).

From Lemma 4.1 this new point must be a Nash equilibrium.

5 Extra

Corollary 5.1 (Odd Number) For non-degenerate games, the number of Nash equilibria is odd.

Proof: In a graph, the number of vertices with degree odd is even since,

$$\sum_v d_v = 2E.$$  

Hence we have an even number of odd vertices. But, (0, 0) is an odd vertex and not a Nash equilibrium.

Theorem 5.2 The Lemke-Howson algorithm runs in exponential time in worst-case [3].

6 Approximating Nash equilibria

Definition 6.1 (k-uniform) A strategy $x$ is called k-uniform when every coordinate $x_i$ is a multiple of $\frac{1}{k}$.

A k-uniform strategy has support size at most k.

Theorem 6.1 (Approximate Nash with small support) Let $\epsilon > 0$. For any two player game, there always exists a k-uniform $\epsilon$-approximate Nash equilibrium for $k = \frac{12\log n}{\epsilon^2}$ [4].

Rubinstein recently proved that this bound is tight [5].

7 Related

Even though Nash’s proof guarantees existence of Nash equilibria the reduction of the existence of a mixed equilibrium to the invariant and then the use of Brouwer’s fixpoint theorem means that the proof is not very constructive. Finding the Brouwer fixpoint is a hard problem [6]. The actual complexity of finding these Nash was discovered in a seminal paper published by Daskalakis et al in 2009 [7].
8 Implementation

Here is an example of Python implementation of Lemke-Howson Algorithm for two-player game [8].

```python
def lemke_howson_tbl(tableaux, bases, init_pivot, max_iter=10**6):
    
    Main body of the Lemke-Howson algorithm implementation.

    Parameters
    ----------
    tableaux : tuple(ndarray(float, ndim=2))
        Tuple of two arrays containing the tableaux, of shape (n, m+n+1)
        and (m, m+n+1), respectively. Modified in place.

    bases : tuple(ndarray(int, ndim=1))
        Tuple of two arrays containing the bases, of shape (n,) and
        (m,), respectively. Modified in place.

    init_pivot : scalar(int)
        Initial pivot, an integer k such that 0 <= k < m+n, where
        integers 0, ..., m-1 and m, ..., m+n-1 correspond to the actions
        of players 0 and 1, respectively.

    max_iter : scalar(int), optional(default=10**6)
        Maximum number of pivoting steps.

    Returns
    -------
    converged : bool
        Whether the pivoting terminated before `max_iter` was reached.

    init_player = int((bases[0]==init_pivot).any())
    players = [init_player, 1 - init_player]

    pivot = init_pivot
    num_iter = 0

    converged = False
```

Here is an example of Python implementation of Lemke-Howson Algorithm for two-player game [8].
while True:
    for i in players:
        # Determine the leaving variable
        row_min = min_ratio_test(tableaux[i], pivot)

        # Pivoting step: modify tableau in place
        pivoting(tableaux[i], pivot, row_min)

        # Update the basic variables and the pivot
        bases[i][row_min], pivot = pivot, bases[i][row_min]

        num_iter += 1

        if pivot == init_pivot:
            converged = True
            break
        if num_iter >= max_iter:
            return converged
else:
    continue
break

return converged
References


