The Lemke-Howson Algorithm

In this note we introduce an algorithm that finds Nash equilibria of 2-player, finite, strategic games. In doing so, we will prove that the algorithm is correct: it terminates, and it finds a Nash equilibrium. We hence prove the special case of Nash's theorem where there are only 2 players.

The algorithm originally appeared in the paper [1] of Lemke and Howson, in 1964. The Lemke-Howson algorithm resembles the simplex algorithm (from linear programming), especially since the algorithm consists of iterated *pivoting*. On a related note, we have already seen that the entire set of Nash equilibria in *zero-sum* 2-player games can be found (and proved to exist) by using linear programming methods. Here is one similarity and one difference between the two methods.

- The simplex algorithm can take an exponential number of iterations [Klee and Minty 1969] and so can the Lemke-Howson algorithm [Savani and von Stengel 2004].
- Other techniques to solve linear programs are known that run in polynomial time (e.g., the ellipsoid and interior point methods) but no polynomial time technique is known for finding Nash equilibria (even for 2-player games).

In the first part, we describe the Lemke-Howson algorithm at a high level. In the second part, we show how the algorithm can be executed using tableaux, using an example. In the third part, we discuss degeneracy issues.

Our notation follows [2], which we repeat here. Let player 1 have m actions labeled $M = \{1, \ldots, m\}$ and player 2 have n actions labeled $N = \{m+1, \ldots, m+n\}$. We represent the payoffs for our two-player game with $m \times n$ matrices. The matrix A represents the payoffs for player 1, and the matrix B represents the payoffs for player 2. We think of player 1 picking rows and player 2 picking columns; so a mixed strategy for player 1 is an m-element row vector that is stochastic (the entries are nonnegative and sum to 1) and similarly a mixed strategy for player 2 is an n-element stochastic column vector. With these notations, the payoff to player 1 (resp. 2) under mixed action profile $(x^\top, y)$ is $x^\top Ay$ (resp. $x^\top By$). The support $\text{supp}(\cdot)$ of a vector is the set of indices where the vector is nonzero. We use X to denote the set of all mixed strategies of player 1 (i.e., X is the set of all stochastic m-element row vectors) and Y is defined similarly. The symbol $\mathbf{0}$ represents the all-zero vector, whose size and orientation should be determined from context.

Assumption 1. We assume that all entries of A and B are nonnegative, that A has no all-zero columns, and B has no all-zero rows.

This assumption is no loss of generality, since adding the same large positive number to every entry of A (or B) clearly does not change the structure of the Nash equilibria.

1. The Basic Idea

As we observed last class, one brute-force algorithm (noted in [Dickhaut and Kaplan 1991] and probably earlier) for finding all Nash equilibria relies on guessing the *support* of the equilibrium, and then solving a linear program to determine what values the nonzero variables can take on. The Lemke-Howson algorithm uses a similar idea; we maintain a single guess as to what the supports should be, and in each iteration we change the guess only a little bit.

The easiest description of the algorithm, and the easiest proof of Nash's theorem for 2-player games, relies on two polytopes which we now define. A polytope is the same as the feasible region for an LP: a system of linear equalities and inequalities. Let B_j denote the

column of B corresponding to action j and let A^i denote the row of A corresponding to action i. Here are the two polytopes:

$$P_1 = \{x \in \mathbb{R}^M \mid (\forall i \in M : x_i \geq 0) \ \& \ (\forall j \in N : x^\top B_j \leq 1)\}$$
$$P_2 = \{y \in \mathbb{R}^N \mid (\forall i \in N : y_i \geq 0) \ \& \ (\forall i \in M : A^i y \leq 1)\}$$

Note that we don't restrict x and y to be stochastic here, only nonnegative. For a nonzero nonnegative x, we can *normalize* it to a stochastic vector $\text{nrml}(x)$ as follows,

$$\text{nrml}(x) := \left(\sum_i x_i\right)^{-1} x.$$

The inequalities that define P_1 have the following meaning:

- if $x \in P_1$ meets $x_i \geq 0$ with equality then i is not in the support of x
- if $x \in P_1$ meets $x^\top B_j \leq 1$ with equality then j is a best response to $\text{nrml}(x)$

So the polyhedra P_i somehow encode information about best responses and the support. We now make this encoding explicit.

Let us say that $x \in P_1$ has *label* k, where $k \in M \cup N = \{1, \ldots, m+n\}$, if either $k \in M$ and $x_k = 0$, or $k \in N$ and $x^\top B_k = 1$. Similarly $y \in P_2$ has *label* k if either $k \in N$ and $y_k = 0$, or $k \in M$ and $A^k y = 1$. As a consequence of the Support Characterization, we have the following.

Theorem 1. Suppose that $x \in P_1$ and $y \in P_2$, and neither x nor y is the all-zero vector. Then x and y together have *all* labels from 1 to k if and only if $(\text{nrml}(x), \text{nrml}(y))$ is a Nash equilibrium. All Nash equilibria arise in this way.

Proof. For each $i \in M$, we know that either $x_i = 0$ or i is a best response by player 1 to the normalized strategy corresponding to y. Thus (by the S. C.) player 1 is using a best mixed response. Similarly player 2 is using a best mixed response, so we have a NE.

On the other hand, given any Nash equilibrium (x', y') let λ_1 be the value that player 1 obtains and λ_2 be the value that player 2 obtains. It is easy to check that $x'/\lambda_2 \in P_1$ and $y'/\lambda_1 \in P_2$, and the S. C. again quickly proves that x' and y' together have all labels. □

Next we employ a common sort of assumption from the mathematical programming literature. Assumption 1 ensures that the polyhedron P_1 is bounded and m-dimensional, and P_2 is bounded and n-dimensional. A bounded polyhedron is called a *polytope*. A d-dimensional polytope is *simple* if every vertex (i.e., every extreme point) meets exactly d of the defining inequalities with equality.

Assumption 2. The polytopes P_1 and P_2 are simple.

A game that does not satisfy Assumption 2 is called *degenerate*. Intuitively, degeneracy involves a special relationship amongst the payoffs, and so "most" games are non-degenerate. (More specifically, in the $2mn$-dimensional space of all games, the degenerate ones occupy regions of dimension at most $2mn - 1$.) There is a standard way to deal with degenerate cases called *perturbation*, where we change all the payoff values by a little amount (either numerically with random numbers, or symbolically). We will explain later in concrete terms how to apply the Lemke-Howson algorithm to games that are degenerate.

For a point x in a polyhedron P, a defining inequality $\aleph$ is *binding* if x meets $\aleph$ with equality. The *binding subsystem* $\beta(x)$ corresponding to x is the set of all defining inequalities that are binding. The following facts are basic elements of polyhedral theory; intuitively, it is easy to see that they hold in dimension at most 3. (The first item is just the definition of "simple").

Theorem 2. In a simple d-dimensional polytope,

(a) every vertex v has $|\beta(v)| = d$, i.e., is incident on exactly d faces;

(b) for two distinct vertices v, v' we have $\beta(v) \neq \beta(v')$;

(c) (pivoting) every vertex is incident on exactly d edges; in particular, for each $\aleph \in \beta(v)$, there is a unique neighbour v' of v with $\beta(v') \cap \beta(v) = \beta(v) \backslash \{\aleph\}$

Notice that in P_1 and P_2, binding inequalities correspond to the *labels* we defined earlier. E.g., x has label $i \in M$ iff $x_i \geq 0$ is binding for x, and x has label $j \in N$ iff $x^\top B_j \leq 1$ is binding for x. Because of this, applying Theorem 2(c) to P_1 or P_2 will be called *removing the label k from v and obtaining new vertex v'*, where k is the label corresponding to inequality $\aleph$. Similarly, since v' in Theorem 2(c) has exactly one new label that v didn't have, namely the label k' corresponding to the unique inequality in $\beta'(v) \backslash \beta(v)$, we will say that *the label k' was added*.

We are now able to state the Lemke-Howson algorithm and prove its correctness (in the nondegenerate case). Throughout the algorithm, x is a vertex of P_1 and y is a vertex of P_2.

Algorithm 3–0.1 The Lemke-Howson algorithm.

Let x (resp. y) be the all-zero vector $\mathbf{0}$ of length m (resp. n)
Let k_0 be any label of x
Let $k = k_0$
loop
    In P_1, remove the label k from x; let x' be the new vertex and k' the label added
    Let $x = x'$
    If $k' = k_0$, stop looping
    In P_2, remove the label k' from y; let y' be the new vertex and k'' the label added
    Let $y = y'$
    If $k'' = k_0$, stop looping
    Let $k = k''$
end loop
Output $(\text{nrml}(x), \text{nrml}(y))$

(It should be easy to see how to modify the algorithm so that the first label is removed from y instead of from x.)

Claim 1. The Lemke-Howson algorithm outputs a Nash equilibrium.

Proof. Each time we apply Theorem 2(c) and update the point x or y it will be called a *pivot*. Let us define a *configuration* to be any pair (x, y) such that x is a vertex of P_1, y is a vertex of P_2, and every label in $M \cup N - k_0$ is had by either x or y (or both). The key point is that, at every point during the algorithm, x and y together form a configuration. (To see this, recall that x has exactly m labels, y has exactly n labels, there are $m + n$ labels in total, and use induction on the number of pivots.)

We say that configurations (x, y) and (x', y') are *adjacent* if either

(a) $x = x'$ and an edge of P_2 connects y to y', or

(b) $y = y'$ and an edge of P_1 connects x to x'.

First, notice that each pivot of the algorithm moves us from one configuration to an adjacent configuration. Next, consider the following two kinds of configurations:

x **and** y **together have all labels:** this configuration is adjacent to exactly one other configuration, since exactly one of x or y has label k_0, and we need to remove that label from whichever one has it.

x **and** y **share a duplicate label:** this configuration is adjacent to exactly two other configurations, 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.

Consider a graph whose nodes are all the configurations, and whose edges are all pairs of adjacent configurations. The above analysis shows that every node has degree 1 or 2; hence every connected component is a path or a cycle. Viewed in this graph, the Lemke-Howson algorithm begins at the configuration $(\mathbf{0}, \mathbf{0})$; this configuration has all labels and is therefore an endpoint of a path component. The algorithm walks along this path until it finds another degree-1 node. From the above analysis we see that such a node has all labels; furthermore, it cannot be that the final configuration is $(\mathbf{0}, \mathbf{0})$. (It still might be possible to end at a configuration of the form $(x, \mathbf{0})$ or $(\mathbf{0}, y)$, but we leave it as an exercise to show that this is not possible.) Hence, using Theorem 1 we end at a configuration corresponding to a Nash equilibrium. □

In the proof it is also clear that the "configuration graph" has an even number of degree-1 nodes, and all of them except $(\mathbf{0}, \mathbf{0})$ are Nash equilibria. With a little more work we can hence prove the following:

Corollary 1. A nondegenerate two-player strategic game has a finite, odd number of Nash equilibria.

In particular, of course, the game cannot have 0 equilibria.

2. Tableau Method and Example

To apply the tableau method to find Nash equilibria using the Lemke-Howson algorithm, we use the following four steps.

(a) Preprocessing.

(b) Initialization of tableaux.

(c) Repeated pivoting.

(d) Recover Nash equilibrium from final tableaux.

In the tableau method, we introduce *slack variables*, and use the terminology *basic and non-basic variables*. For our purposes the *basic variables* and *set of labels* have **opposite meanings** since labels imply a tight inequality and basic variables are not tight. Hence, "enters the basis" means the same as "label is removed" and "leaves the basis" means that "label is added."

Step 1. Preprocessing

Recall that iterated elimination of strictly dominated strategies preserves all Nash equilibria. Elimination reduces the size of the game, and therefore will reduce the amount of work involved with the pivoting later on. Hence, one should apply this elimination before beginning. *Strict domination by mixed strategies also applies here!*

Next, to ensure that the game satisfies the conditions of Theorem 1, add a suitably large constant to the entries of each payoff matrix.

Step 2. Initialization of Tableaux

For the purposes of solving the game we need two tableaux, one for each player. Let r_i be the slack in the constraint $A^i y \leq 1$ and let s_j be the slack in the constraint $x^\top B_j \leq 1$. We then obtain the system

$$Ay + r = \mathbf{1}, B^\top x + s = \mathbf{1}, \text{ and } x, y, r, s \text{ are nonnegative.}$$

In the initial tableaux, the basis is $\{r_i \mid i \in M\} \cup \{s_j \mid j \in N\}$ and so we rewrite the equations so as to solve for them.

As an example we will use the following game.

$p1 \backslash p2$	4	5	6
1	1,2	3,1	0,0
2	0,1	0,3	2,1
3	2,0	1,0	1,3

Notice that the entries are positive, no strict domination occurs, and furthermore that there are no pure Nash equilibria. The game satisfies Assumption 1 and it happens to also satisfy Assumption 2.

The initial tableaux are $r = \mathbf{1} - Ay$,

$$\begin{array}{llll} r_1 = 1 & -y_4 & -3y_5 & \\ r_2 = 1 & & & -2y_6 \\ r_3 = 1 & -2y_4 & -y_5 & -y_6 \end{array} \quad \begin{array}{l} \text{[A1]} \\ \text{[A2]} \\ \text{[A3]} \end{array}$$

and $s = \mathbf{1} - B^\top x$,

$$\begin{array}{llll} s_4 = 1 & -2x_1 & -x_2 & \\ s_5 = 1 & -x_1 & -3x_2 & \\ s_6 = 1 & & -x_2 & -3x_3 \end{array} \quad \begin{array}{l} \text{[B1]} \\ \text{[B2]} \\ \text{[B3]} \end{array}$$

Step 3. Pivoting

We need to arbitrarily choose some x or y variable to bring in to the basis, corresponding to the arbitrary choice k_0 of label that we remove. Let's bring x_1 in. By considering the min-ratio rule (i.e., looking at the coefficients of x_1 in the [B] tableau) it is s_4 that must leave the basis. Therefore we solve [B1] for x_1, obtaining a new equation [B'1], and we substitute the new equation into [B2] and [B3] obtaining

$$\begin{array}{llll} x_1 = 1/2 & -1/2 s_4 & -1/2 x_2 & \\ s_5 = 1/2 & +1/2 s_4 & -5/2 x_2 & \\ s_6 = 1 & & -x_2 & -3x_3 \end{array} \quad \begin{array}{l} \text{[B'1]} \\ \text{[B'2]} \\ \text{[B'3]} \end{array}$$

The main feature of the Lemke-Howson algorithm, as we discussed in the first section, is that the variable which just left the basis determines the variable to enter the basis next. There are $m + n$ *complementary pairs* of variables: $\{r_i, x_i\}$ for $i \in M$ and $\{s_j, y_j\}$ for $i \in N$. Each pair corresponds (in an inverse sense) to the labels we mentioned earlier, e.g., x_i is basic iff x does not have label i and s_j is basic iff x does not have label j.

The $m + n$ *complementarity conditions*

$$r_i x_i = 0, i \in M \qquad s_j y_j = 0, j \in N.$$

tell us when to stop. Initially, all complementarity conditions are satisfied. We keep performing pivots until the complementarity conditions are again satisfied. Equivalently, we pivot until,

between the two tableaux, in each complementary pair of variables, exactly one is basic and exactly one is non-basic.

In this case, since s_4 just left the basis, y_4 must be brought in. Examining the [A] tableau we see that r_3 is the winner of the min-ratio rule, and is therefore leaves the basis. We obtain the following.

$$\begin{array}{llll} r_1 = 1/2 & +1/2r_3 & -5/2y_5 & +1/2y_6 \\ r_2 = 1 & & & -2y_6 \\ y_4 = 1/2 & -1/2r_3 & -1/2y_5 & -1/2y_6 \end{array}$$

Since r_3 left, now x_3 enters the other tableau, and by the min-ratio rule s_6 leaves.

$$\begin{array}{llll} x_1 = 1/2 & -1/2s_4 & -1/2x_2 & \\ s_5 = 1/2 & +1/2s_4 & -5/2x_2 & \\ x_3 = 1/3 & & -1/3x_2 & -1/3s_6 \end{array}$$

Since s_6 left, now y_6 enters, and by the min-ratio rule r_2 leaves.

$$\begin{array}{llll} r_1 = 3/4 & +1/2r_3 & -5/2y_5 & -1/4r_2 \\ y_6 = 1/2 & & & -1/2r_2 \\ y_4 = 1/4 & -1/2r_3 & -1/2y_5 & +1/4r_2 \end{array}$$

Since r_2 left, now x_2 enters, and by the min-ratio rule s_5 leaves.

$$\begin{array}{llll} x_1 = 2/5 & -3/5s_4 & +1/5s_5 & \\ x_2 = 1/5 & +1/5s_4 & -2/5s_5 & \\ x_3 = 4/15 & -1/15s_4 & +2/15s_5 & -1/3s_6 \end{array}$$

Since s_5 left, now y_5 enters, and by the min-ratio rule r_1 leaves.

$$\begin{array}{llll} y_5 = 3/10 & +1/5r_3 & -2/5r_1 & -1/10r_2 \\ y_6 = 1/2 & & & -1/2r_2 \\ y_4 = 1/10 & -3/5r_3 & +1/5r_1 & +3/10r_2 \end{array}$$

Step 4. Output

Since x_1 was the initial variable to enter the basis, and r_1 just left, the complementarity conditions are now satisfied. (More generally, if x_i was the initial variable to enter, we stop when x_i or its complement leaves.) In a tableau, we obtain values for the basic variables by setting the non-basic variables to zero. Hence the variables' values are

$$r = (0, 0, 0), s = (0, 0, 0), x = (2/5, 1/5, 4/15), y = (1/10, 3/10, 1/2).$$

Therefore, the Nash equilibrium we just found is

$$(\text{nrml}(x), \text{nrml}(y)) = ((6/13, 3/13, 4/13), (1/9, 3/9, 5/9)).$$

3. Degeneracy

The effect of nondegeneracy on the tableau method is the following:

Proposition 1. When running the Lemke-Howson algorithm in tableau form on a nondegenerate game, in each iteration there is a unique variable that wins the min-ratio test.

However, degenerate games occur frequently in practice. In a general game, we can still use the tableau method, but we will be faced with the problem of breaking ties in some manner. Furthermore, just as in the simplex algorithm, if we have a "bad" tie-breaking rule, then our program can enter a loop and run forever.

By using infinitesimal perturbations, we can obtain a "good" tie-breaking rule that can be performed in polynomial time; see the section of [2] on the *lexicographic method*. However, this rule is impractical to perform by hand, so we will not describe it here, and no simpler one appears to be known in the literature.

One consequence of the lexicographic method is that the following pivot rule will work for *any* game: when faced with a tie to break, make the choice arbitrarily; if you later come back to the same basis, then break the tie in a different way. *This is the approach we expect you to take when solving games by hand, but we will try to set things up in such a way that you quickly reach an equilibrium no matter how you break the ties.*

For further edification, we state the following characterization of nondegenerate games, which is not too hard to prove.

Proposition 2. A 2-player finite strategic game is nondegenerate if and only if, for any mixed strategy α of a player, the number of pure best responses by their opponent does not exceed $|\text{supp}(\alpha)|$.

References

[1] C. E. Lemke and J. J. T. Howson. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics*, 12(2):413–423, 1964.

[2] B. von Stengel. Computing equilibria for two-person games. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory, Vol. 3*, pages 1723–1759, 2002.