Static Equilibria

Empirical IO: Bergen, 2021

Anders Munk-Nielsen

December 2nd, 2021

Introduction

Introduction

- **Goal:** Cover solution algorithms

Discrete Games

Discrete Games

Discrete Game: $(\{A_i\}_{i\in\{1,2\}}, \{u_i(a_1, a_2)\}_{i\in\{1,2\}})$

- Discrete action sets, $A_i = \{a_1, ..., a_N\}$
- Utility matrices, U_i $(N_1 \times N_2)$

Discrete Games

Discrete Game: $(\{A_i\}_{i\in\{1,2\}}, \{u_i(a_1, a_2)\}_{i\in\{1,2\}})$

- Discrete action sets, $A_i = \{a_1, ..., a_N\}$
- Utility matrices, U_i $(N_1 \times N_2)$

- **Pure strategy:** an element in A_i for each player
- **Best response:** $BR_i(a_j) = \arg\max_{a_i} u_i(a_i, a_j)$.

I want to play a game

Fra sidst

		Player 2				
		Run	Hide	Play d.	Fight	Freeze
Player 1	Run	8,2	9,7	5,5	4,6	8,3
	Hide	9,4	1,0	7,6	5,8	4,1
	Play dead	5,4	7,9	2,8	9,6	7,0
	Fight	3,4	9,8	2,2	0,1	6,3
	Freeze	7,4	5,4	3,9	0,4	2,2

Nash Equilibrium by Enumeration

Brute Force Algorithm (Enumeration)

For each player i do:

- For each opponent action a_j:
  - Construct $BR_i(a_j) \subset A_i$

$NE = \{(a_1, a_2) | a_1 \in BR_1(a_2) \wedge a_2 \in BR_2(a_1)\}$

Nash Equilibrium by Enumeration

Brute Force Algorithm (Enumeration)

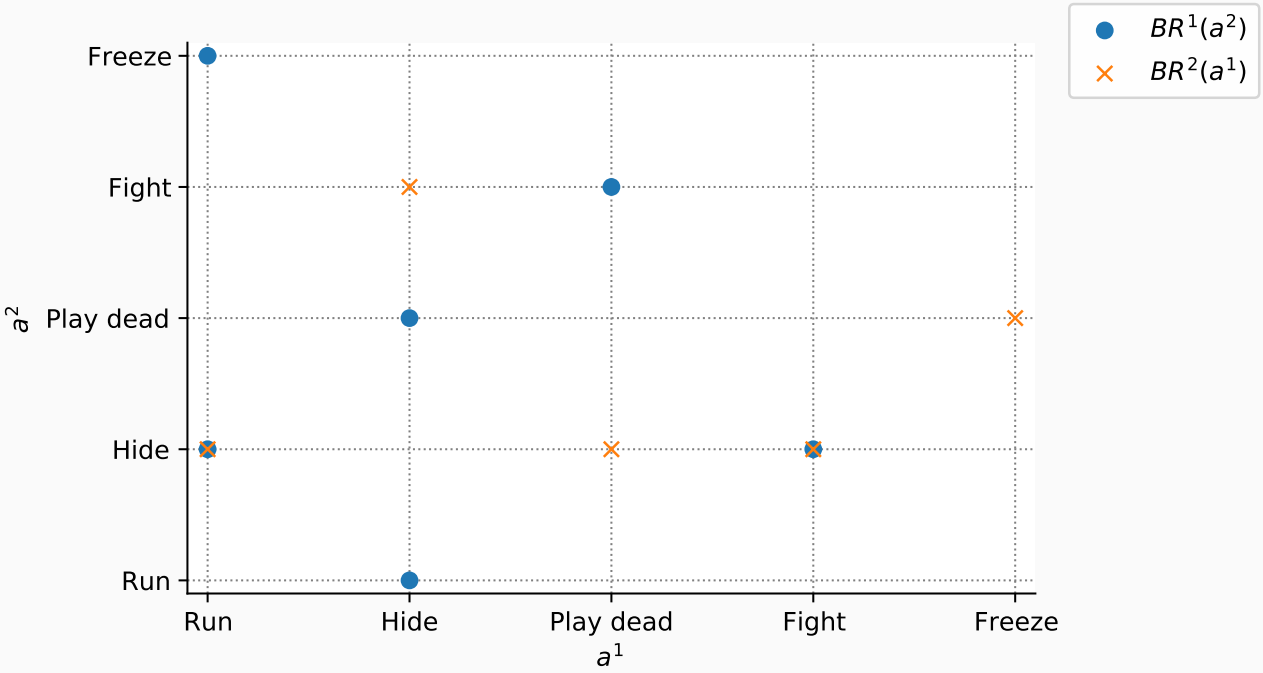
For each player i do:

- For each opponent action a_j:
  - Construct $BR_i(a_j) \subset A_i$

$NE = \{(a_1, a_2) | a_1 \in BR_1(a_2) \wedge a_2 \in BR_2(a_1)\}$

- **Full solution:** Finds all NE
- **Complexity:** Maximal, evaluates all payoffs.
- **Improvements:** For player $i = 2$: only try out $a_2 \in "BR_1^{-1}(a_1)"$

Graphically



NE Occur at *overlapping points*.

IESDS

Definition: Strict Dominance

if $u_i(a_i, a) > u_i(a_i', a)$ regardless of opponent action $a \in A_j$, a_i is strictly dominated by a_i'.

- **Remark:**

IESDS

Definition: Strict Dominance

if $u_i(a_i, a) > u_i(a_i', a)$ regardless of opponent action $a \in A_j$, a_i is strictly dominated by a_i'.

- **Remark:**

Iterated Elimination of Strictly Dominated Strategies

Keep doing:

- For each player i
  - For each action $a_i \in A_i$
    - If a_i is *strictly* dominated by any $a_i' \in A_i$, remove it from A_i.
- If loop passed all players without detecting dominance, exit.

Mixed Strategies

Fictitious Play

Algorithm: Fictitious Play

Initialize strategies, e.g. as uniform random: $\alpha_i^0 = (\frac{1}{\#}, ..., \frac{1}{\#})$. For step $s = 1, ..., S$

- For all players i: set a_i^* as the best response to the empirical frequencies α_j^s observed for the opponent thus far.
  - $a_i^* \in \arg\max_{a \in A_i} \sum_{j=1}^{N} \alpha_j^{s-1} u_i(a, a_j)$.
  - Store a_i^* in the history and recompute α_i^s.

Fictitious Play

Algorithm: Fictitious Play

Initialize strategies, e.g. as uniform random: $\alpha_i^0 = (\frac{1}{\#}, ..., \frac{1}{\#})$. For step $s = 1, ..., S$

- For all players i: set a_i^* as the best response to the empirical frequencies α_j^s observed for the opponent thus far.
  - $a_i^* \in \arg\max_{a \in A_i} \sum_{j=1}^{N} \alpha_j^{s-1} u_i(a, a_j)$.
  - Store a_i^* in the history and recompute α_i^s.

- Convergence properties?
  - Finite zero-sum games
  - Games solvable by IESDS
- Cycles can occur!

Enumeration Algorithms

- **Problem split in two**
  1. Find NE for a given support (set of strategies with positive probability)
  2. Find all NE (loop over all possible supports)

Enumeration Algorithms

- **Problem split in two**
  1. Find NE for a given support (set of strategies with positive probability)
  2. Find all NE (loop over all possible supports)
- **1** boils down to finding strategy pairs such that players are willing to mix
  - I.e. so that the expected payoff from all pure strategies in the support are identical

Enumeration Algorithms

- **Problem split in two**
  1. Find NE for a given support (set of strategies with positive probability)
  2. Find all NE (loop over all possible supports)
- **1** boils down to finding strategy pairs such that players are willing to mix
  - I.e. so that the expected payoff from all pure strategies in the support are identical
- **Algebraically:** Find $\alpha_1, \alpha_2 \in \Delta$ s.t.

$$\alpha_1' U_2 = v\mathbf{1}_{|A_2|} \quad \text{for some } v \in \mathbb{R},$$
$$U_2 \alpha_1 = w\mathbf{1}_{|A_1|} \quad \text{for some } w \in \mathbb{R}.$$

(subject to v and w being rational, i.e. maximizing utility cf. other pure strategies)

Enumeration Algorithms

- **Problem split in two**
  1. Find NE for a given support (set of strategies with positive probability)
  2. Find all NE (loop over all possible supports)
- **1** boils down to finding strategy pairs such that players are willing to mix
  - I.e. so that the expected payoff from all pure strategies in the support are identical
- **Algebraically:** Find $\alpha_1, \alpha_2 \in \Delta$ s.t.

$$\alpha_1' U_2 = v\mathbf{1}_{|A_2|} \quad \text{for some } v \in \mathbb{R},$$
$$U_2\alpha_1 = w\mathbf{1}_{|A_1|} \quad \text{for some } w \in \mathbb{R}.$$

  (subject to v and w being rational, i.e. maximizing utility cf. other pure strategies)
- Utility levels are indeterminate, so equivalently, we equate expected payoffs

$$\alpha_1' U_2[:, k] = \alpha_1' U_2[:, j] \quad \forall k, j \in A_2$$
$$U_1[k, :]\alpha_2 = U_1[j, :]\alpha_2 \quad \forall k, j \in A_1$$

  - Subject to rationality

Lemke-Hawson Algorithm

- `nashpy.Game(...).lemke_howson_enumeration()`
- Efficient enumeration of the simplex
- Returns an *iterator* object: i.e. computation by demand
- Starts by dropping a "label":
  - With multiplicity: this determines which equilibrium the algorithm terminates at.
  - $\Rightarrow$ can find at most $N_1 + N_2$ equilibria

Homotopy Methods

Homotopy Method (intuitively)

- Perturb the payoffs to a game that *can* be solved.
- Solve the game iteratively, letting the pertubation go to zero.

Homotopy Methods

Homotopy Method (intuitively)

- Perturb the payoffs to a game that *can* be solved.
- Solve the game iteratively, letting the pertubation go to zero.

Logit Correspondance

Suppose agents instead maximize perturbed utilities,

$$\tilde{u}_i(a) = u(a) + \sigma\varepsilon(a), \quad \varepsilon(a) \sim \text{IID Extreme Value T1}.$$

- **Choices** then become probabilistic with

$$\Pr(a_i|a_{-i}) = \frac{\exp\left[u_i(a_i, a_{-i})/\sigma\right]}{\sum_{a\in A_i} \exp\left[u_i(a, a_{-i})/\sigma\right]}.$$

- **Limit cases** as expected:
  - $\sigma \to \infty$: $\Pr(a_i|a_{-i}) = \frac{1}{|A_i|}$, uniform random choices.
  - $\sigma \to 0$: $\Pr(a_i|a_{-i}) = \mathbf{1}\{a_i = \arg\max_{a\in A_i} u_i(a, a_{-i})\}$: collapses to best response.

Example: Football

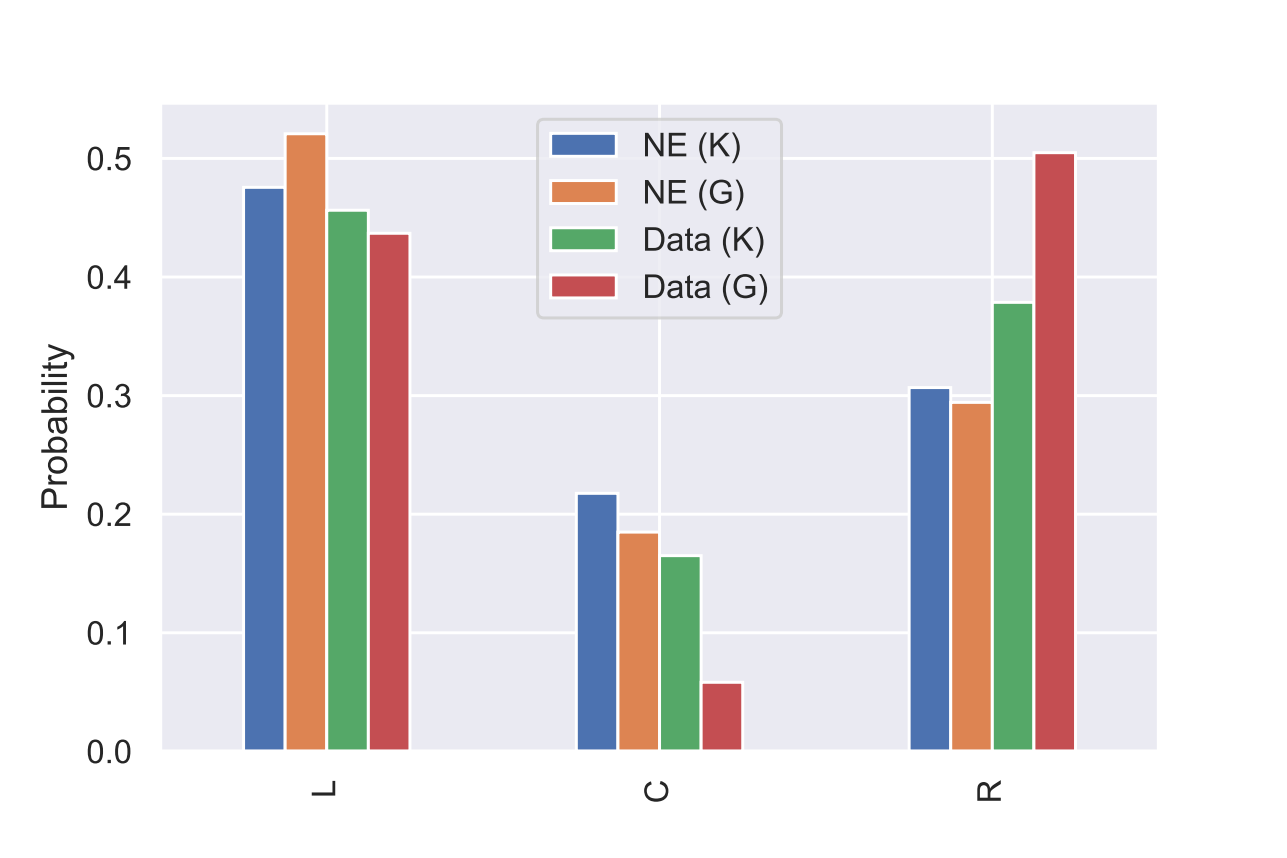
*"I **never** talk about penalty kicks. It would be stupid to give anything away."* – Kasper Schmeichel (Danish goal keeper)

Example: Football Penalty Kicks

- **Data:** 103 penalty kicks from premier league, 2016/17 (source: Kaggle user mauryashubham)
- **Goal probabilities:**

		Goalie		
		L	C	R
Kicker	L	65%	100%	88%
	C	100%	0%	89%
	R	83%	100%	56%

Example Football Penalty Kicks



In equilibrium, $\mathbb{E}(u^{\text{kicker}}) = \alpha^{\text{kicker}\prime} U = U\alpha^{\text{goalie}} = 78.2\%$.

Zero-sum Games

Zero-sum Games

Def.: Strictly Competitive Game

A 2-player game $(\{A_1, A_2\}, \{u_1, u_2\})$ is strictly competitive if

$$u_1(a) \geq u_1(b) \Leftrightarrow u_2(a) \leq u_2(b).$$

Zero-sum Games

Def.: Strictly Competitive Game

A 2-player game $(\{A_1, A_2\}, \{u_1, u_2\})$ is strictly competitive if

$$u_1(a) \geq u_1(b) \Leftrightarrow u_2(a) \leq u_2(b).$$

Definition: Zero-sum Game

A 2-player game $(\{A_1, A_2\}, \{u_1, u_2\})$ is zero sum if

$$u_1(a_1, a_2) + u_2(a_1, a_2) = k$$

for some constant $k \in \mathbb{R}$. (Also called "constant sum")

In Python:

```
U1 = np.array([...])
U2 = -U1
```

Minimax

- **Heuristic, "maximin":** choosing actions according to

$$\max_{a_i} \min_{a_j} u_i(a_i, a_j).$$

  - Choosing the best worst-case outcome.

- **Property:** In a *zero sum game* Nash equilibrium, players are playing maximin/minimax strategies.

Minimax Theorem

For any payoff-matrix U, there exist mixed strategies, $\alpha^{1\star} \in \Delta^1, \alpha^{2\star} \in \Delta^2$, s.t.

$$\max_{\alpha^1} \min_{\alpha^2} \alpha^{1\prime} U \alpha^2 = \max_{\alpha^1} \min_{\alpha^2} \alpha^{1\prime} U \alpha^2 = \alpha^{1\star\prime} U \alpha^{\star 2}.$$

- **Note**: If the game is *symmetric*, $\alpha^{1\star\prime} U \alpha^{\star 2} = \mathbf{0}$.

The Two Linear Programs

Player 1 solves

$$\begin{aligned} \overline{v} \max_{\alpha^1} \; & v \\ \text{u.b.b.} \;\; & v \leq \alpha^1 U_{(:,a)} \; \forall a \\ & \alpha^1 \geq 0 \\ & \mathbf{1}' \alpha^1 = 1 \end{aligned}$$

Player 2 solves

$$\begin{aligned} \underline{v} \min_{\alpha^2} \; & v \\ \text{u.b.b.} \;\; & v \geq U_{(a,:)} \alpha^2 \; \forall a \\ & \alpha^2 \geq 0 \\ & \mathbf{1}' \alpha^2 = 1 \end{aligned}$$

- **Minimax Thm.** says that there exist $\alpha^{1\star}, \alpha^{2\star}$ solving both with common value

$$\underline{v} = \overline{v} = \alpha^{1\star\prime} U \alpha^{2\star}.$$

The Two Linear Programs

Player 1 solves

$$\begin{aligned} \overline{v} \max_{\alpha^1} \; & v \\ \text{u.b.b.} \;\; & v \leq \alpha^1 U_{(:,a)} \; \forall a \\ & \alpha^1 \geq 0 \\ & \mathbf{1}' \alpha^1 = 1 \end{aligned}$$

Player 2 solves

$$\begin{aligned} \underline{v} \min_{\alpha^2} \; & v \\ \text{u.b.b.} \;\; & v \geq U_{(a,:)} \alpha^2 \; \forall a \\ & \alpha^2 \geq 0 \\ & \mathbf{1}' \alpha^2 = 1 \end{aligned}$$

- **Minimax Thm.** says that there exist$\alpha^{1\star}, \alpha^{2\star}$ solving both with common value
$$\underline{v} = \overline{v} = \alpha^{1\star\prime} U \alpha^{2\star}.$$
- **Implication:** $(\alpha^{1\star}, \alpha^{2\star})$ is the Nash Equilibrium!

The Problem on LP Form

- **Python:** `scipy.optimize.linprog` takes problems

$$\begin{aligned} &\max_x c'x \\ \text{s.t. } &Ax \leq b \\ &\tilde{A}x = \tilde{b} \end{aligned}$$

The Problem on LP Form

- **Python:** `scipy.optimize.linprog` takes problems

$$\begin{aligned} &\max_x c'x \\ \text{s.t. } &Ax \leq b \\ &\tilde{A}x = \tilde{b} \end{aligned}$$

- Let us think of $x = (v, \alpha_1^1, ..., \alpha_K^1)' \in \mathbb{R}^{K+1}$ and write

$$c = \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}_{K+1\times 1}, \quad A = \begin{pmatrix} 1 & -U_{11} & \cdots & -U_{1K} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & -U_{K1} & \cdots & -U_{KK} \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \end{pmatrix}_{2K\times K+1}, \quad b = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}_{2K\times 1}$$

$$\tilde{A} = (0, 1, \cdots, 1)_{1\times k+1}, \quad \tilde{b} = 1.$$

Translating to LP Form

- $Ax \leq b$ now has two parts
  - Top: $v - U\alpha \leq \mathbf{0}$
  - Bottom: $0 \cdot v - \mathbf{I}\alpha \leq \mathbf{0} \Leftrightarrow \alpha \geq \mathbf{0}$

Translating to LP Form

- $Ax \leq b$ now has two parts
  - Top: $v - U\alpha \leq \mathbf{0}$
  - Bottom: $0 \cdot v - \mathbf{I}\alpha \leq \mathbf{0} \Leftrightarrow \alpha \geq \mathbf{0}$
- $\tilde{A}x = \tilde{b}$ is a scalar condition:
  - $0 \cdot v + \mathbf{1}'\alpha = 1 \Leftrightarrow \sum_{\ell=1}^{K} \alpha_\ell = 1$ ensuring that probabilities sum to 1

Translating to LP Form

- $Ax \leq b$ now has two parts
  - Top: $v - U\alpha \leq \mathbf{0}$
  - Bottom: $0 \cdot v - \mathbf{I}\alpha \leq \mathbf{0} \Leftrightarrow \alpha \geq \mathbf{0}$
- $\tilde{A}x = \tilde{b}$ is a scalar condition:
  - $0 \cdot v + \mathbf{1}'\alpha = 1 \Leftrightarrow \sum_{\ell=1}^{K} \alpha_\ell = 1$ ensuring that probabilities sum to 1
- **Conclusion:** We give this to `linprog`

$$
\begin{aligned}
&\max_{\alpha^1} v \\
\text{s.t. } & v \leq \textstyle\sum_{\ell=1}^{K} \alpha_\ell^1 U_{\ell j} \quad j = 1, ..., K \\
& \alpha_j \geq 0 \quad j = 1, ..., k_1 \\
& \textstyle\sum_{\ell=1}^{k} \alpha_\ell = 1.
\end{aligned}
$$

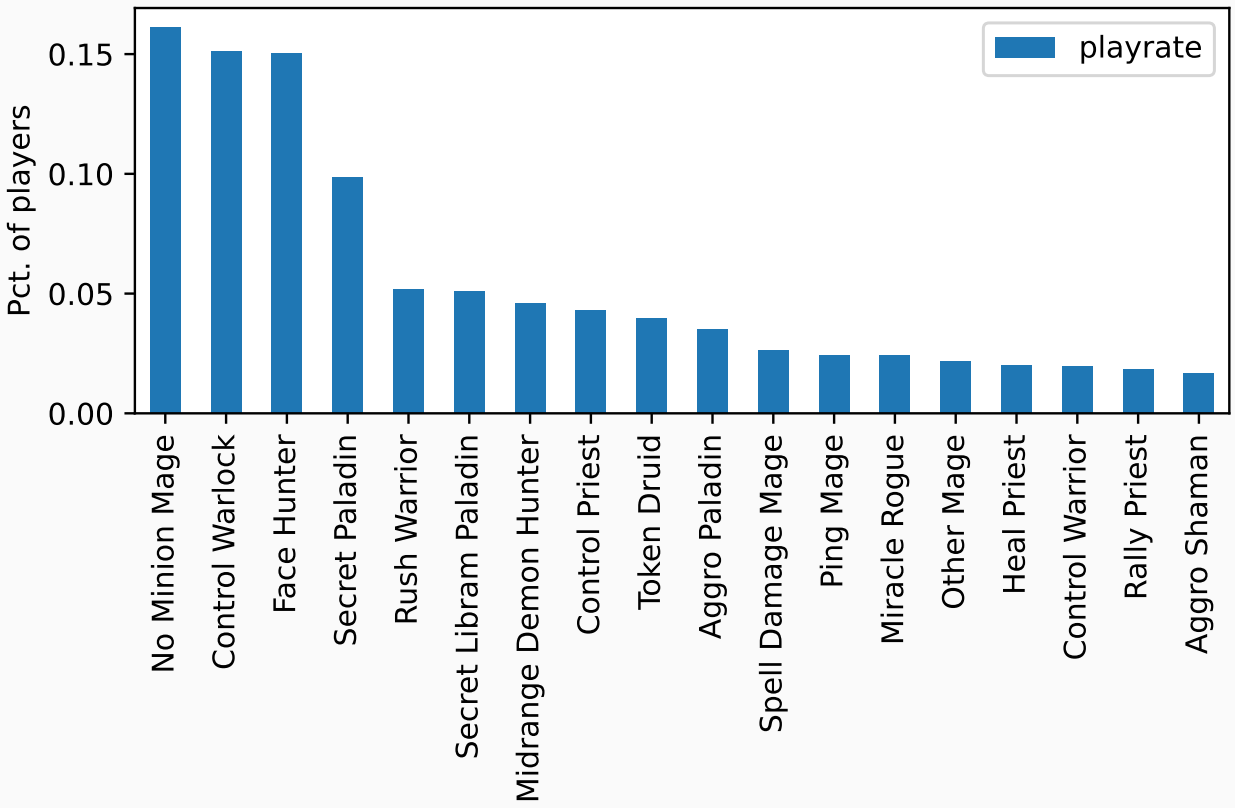
Python-code

```python
from scipy.optimize import linprog
def: solve_by_LP(U):
        num_vars, num_cols = U.shape
        assert num_vars == num_cols # U must be square
        oo = np.zeros((1,num_vars))
        ii = np.ones((1,num_vars))
        # objective: c = [-1, 0, 0, ..., 0]
        c = np.insert(oo, 0, -1.0)
        # inequality constraints: A*x <= b
        top  = np.hstack( (ii.T, -1*U.T) )
        bot  = np.hstack( (oo.T, -1*np.eye(num_vars)) )
        A_ub = np.vstack((top, bot))
        b_ub = np.zeros((1, 2*num_vars))
        b_ub = np.matrix(b_ub)
        # equality contraints: Ax = b
        A_eq = np.matrix(np.hstack((0, np.ones((num_vars,)))))
        b_eq = 1.0 # just one condition so scalar
        # call solver
        sol = linprog(c, A_ub, b_ub, A_eq, b_eq)
        return sol.x[1:] # first entry is just a dummy!
```

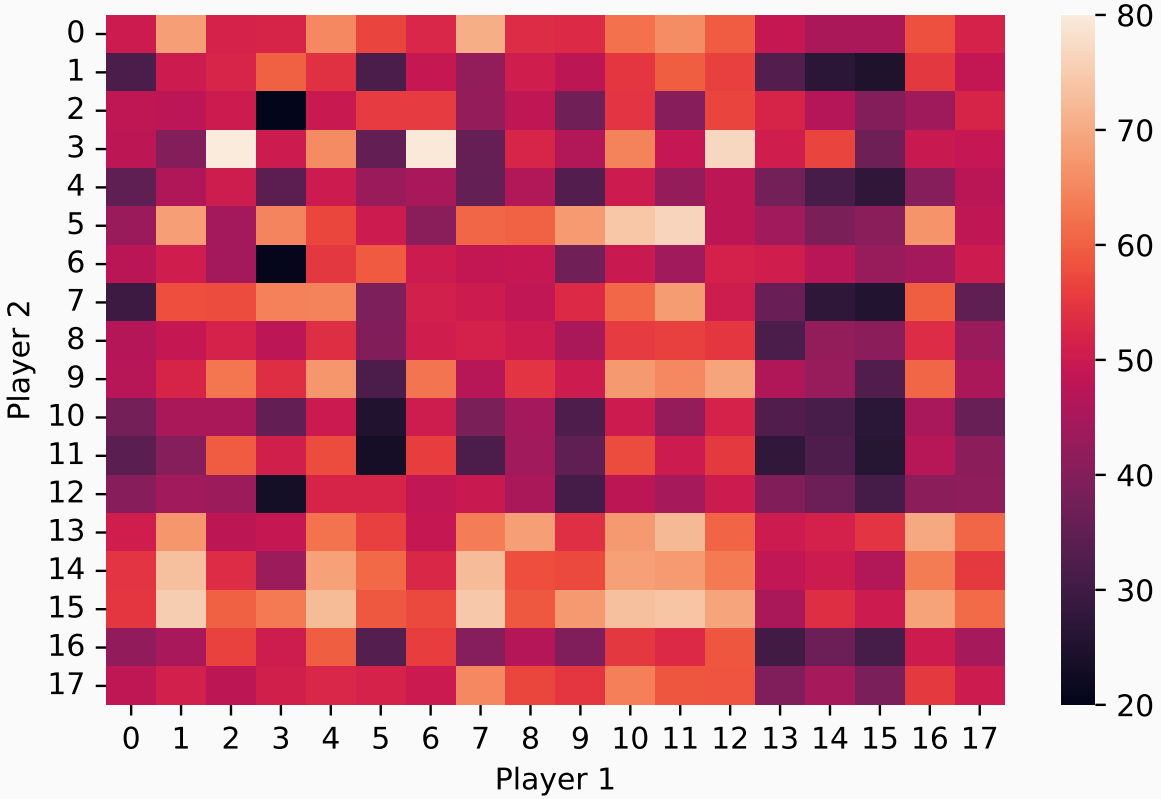
LP vs. Enumeration

- 3×3 game: `support enumeration` can be faster
- 6×6 game: 4.58ms vs. 1.09sec.
- LP is only **polynomial** in time
  - ... enumeration is **exponential**.

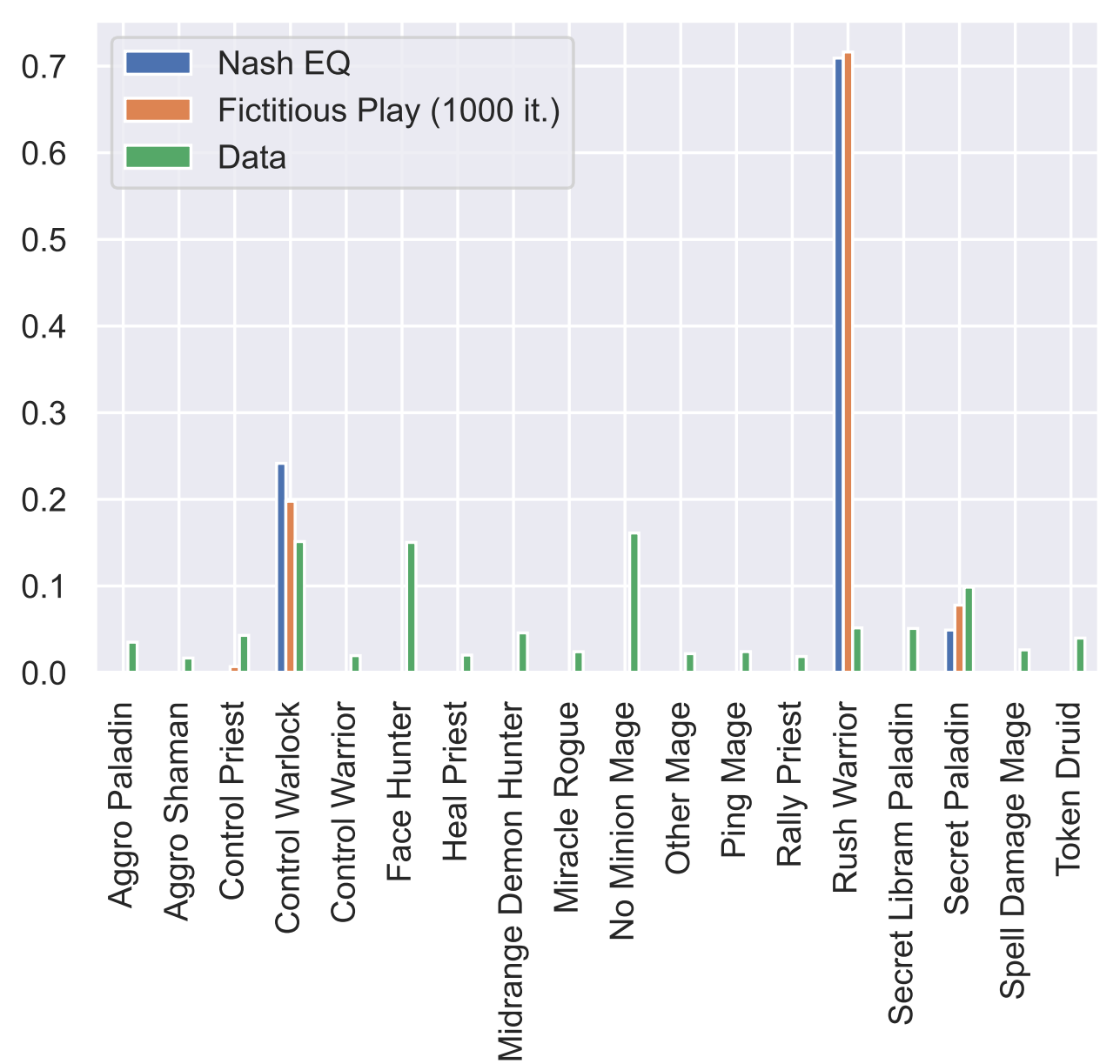
Example: Hearthstone



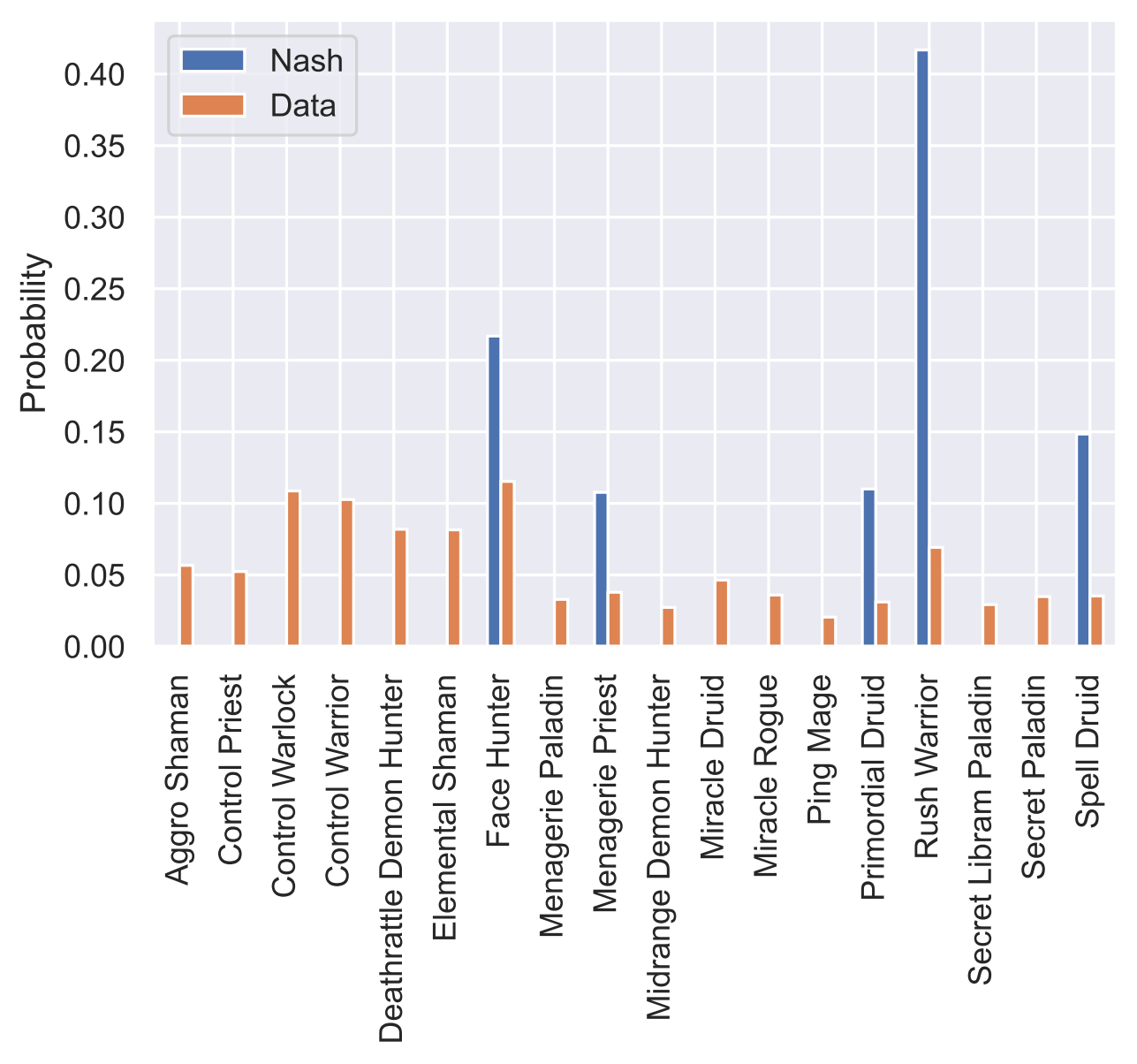
Win rates


Player 2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Player 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
80
70
60
50
40
30
20

Model vs. Data



A Later Snapshot



Continuous Games

Existence: Pure strategies

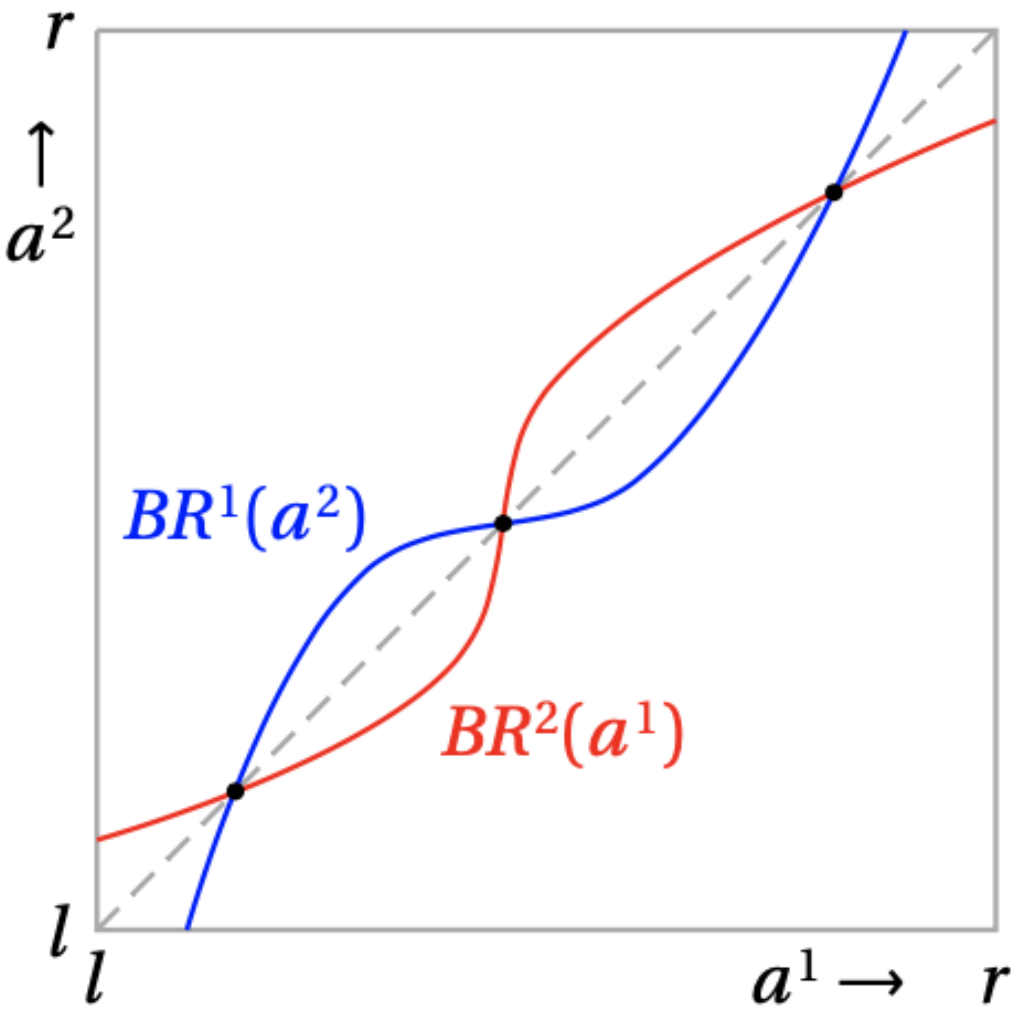
Theorem: Existence of Pure Strategy Equilibria

If

- action spaces are compact,
- best response functions are continuous,
- the game is symmetric,

Then a pure strategy Nash equilibrium exists.

Existence Graphically



Iterated Best Response

Algorithm: Iterated Best Response

Choose some initial actions, (a_1^0, a_2^0).

- For each player, set $a_i^{[it]} = BR_i(a_j^{[it-1]})$.

Stop if $\|a^{[it]} - a^{[it-1]}\| < \epsilon^{tol}$.

Iterated Best Response

Algorithm: Iterated Best Response

Choose some initial actions, (a_1^0, a_2^0).

- For each player, set $a_i^{[it]} = BR_i(a_j^{[it-1]})$.

Stop if $\|a^{[it]} - a^{[it-1]}\| < \epsilon^{tol}$.

- **Pro:** Simple to code up + fast,
- **Con:** Many equilibria are not locally stable.
  - Can result in cycles: e.g. in Prisonner's Dilemma-type situations, or with undercutting.

Newton's Method

Algorithm: Newton's Method

Just give

$$\mathbf{f}(\mathbf{a}) \equiv \begin{pmatrix} a_1 - BR_1(a_2) \\ a_2 - BR_2(a_1) \end{pmatrix},$$

to `scipy.optimize.fsolve` and pray for success.

Newton's Method

Algorithm: Newton's Method

Just give

$$\mathbf{f}(\mathbf{a}) \equiv \begin{pmatrix} a_1 - BR_1(a_2) \\ a_2 - BR_2(a_1) \end{pmatrix},$$

to `scipy.optimize.fsolve` and pray for success.

- **Pro:** Faster convergence properties,
- **Cons:** Not all equilibria are locally stable

Example: Logit Demand

Model: Logit Demand

$$s_i(p_i, p_j) = \frac{\exp(-\gamma p_i)}{\exp(\delta) + \exp(-\gamma p_i) + \exp(-\gamma p_j)}.$$

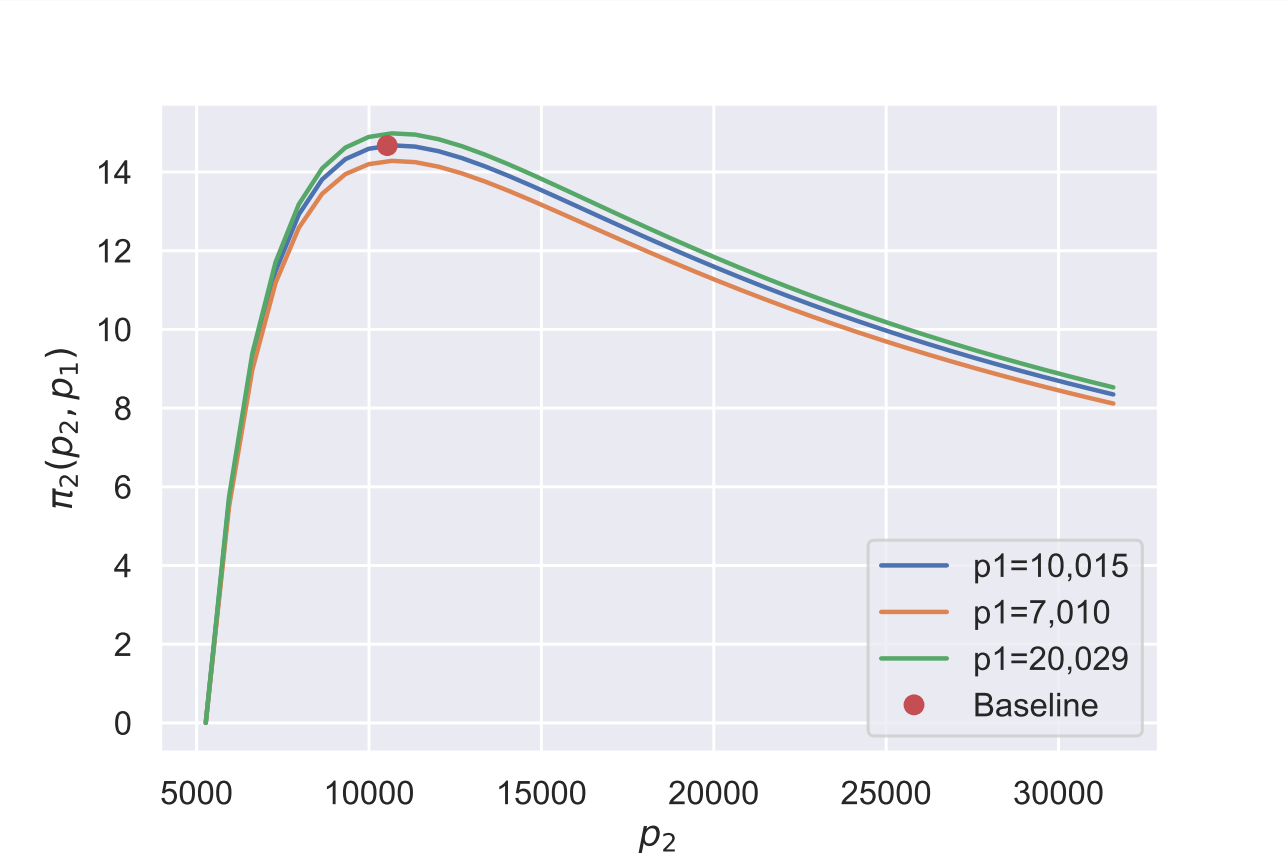
Example: Logit Demand

Model: Logit Demand

$$s_i(p_i, p_j) = \frac{\exp(-\gamma p_i)}{\exp(\delta) + \exp(-\gamma p_i) + \exp(-\gamma p_j)}.$$

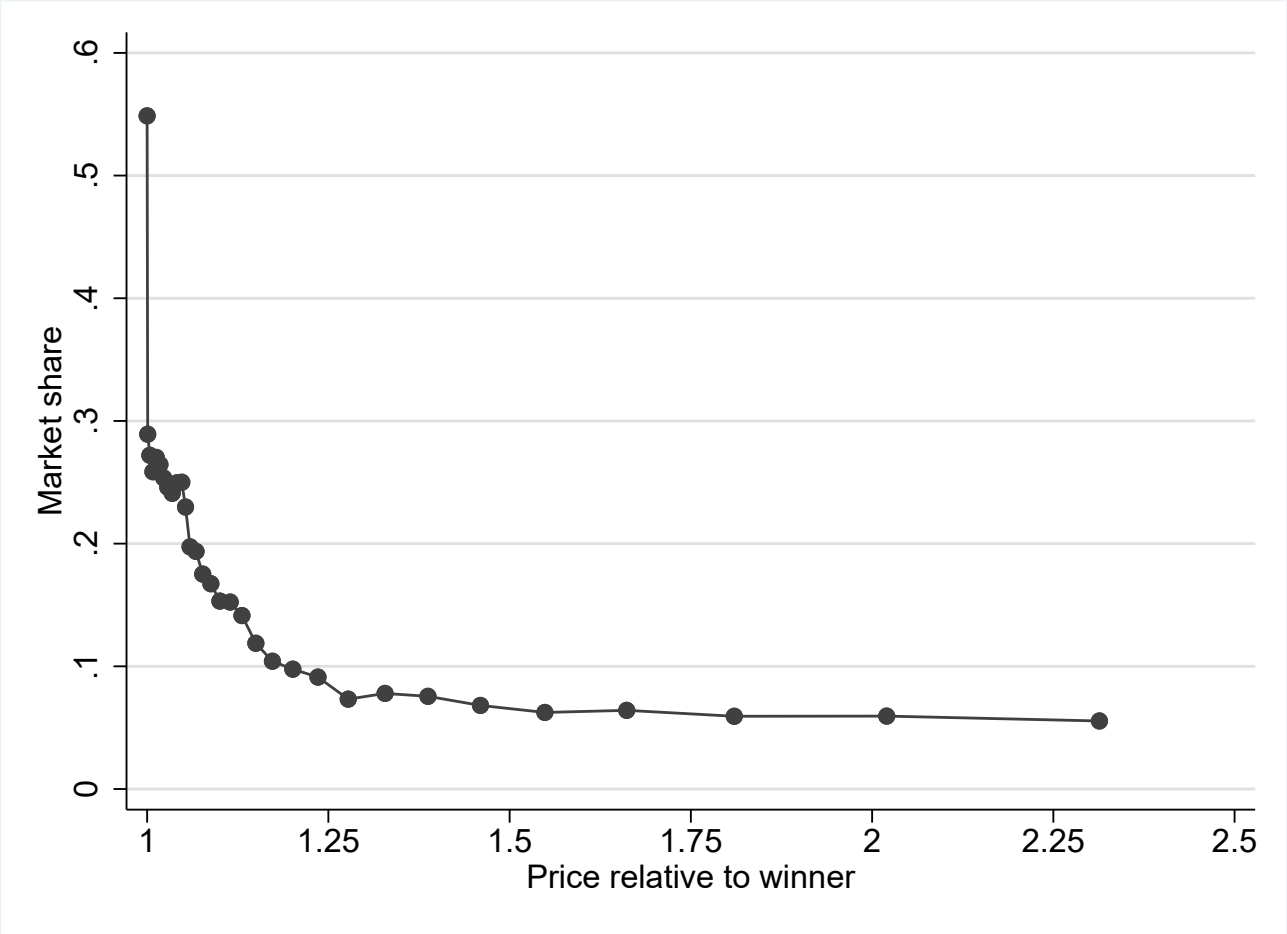
- Here, $\gamma > 0$ determines the strength of interaction between the two BR curves.
- $\delta \in \mathbb{R}$ determines the outside option attractiveness.
- Note that even the monopolist (e.g. with $p_j = \infty$) will choose some $p_i < \infty$
  - But without an outside option (e.g. $\delta = -\infty$) it is optimal to let $p_i \to \infty$.

Estimated Application



Weak cross-price elasticities $\Rightarrow$ simple NE to find!

Example: Undercutting Game



$x = \frac{p_j}{\min_k p_k}$, $y =$ market share.

Undercutting Game

Demand with Attention-shifter

$$s_i(p_i, p_j) = \frac{\exp(-\gamma p_i + \alpha \mathbf{1}\{p_i \leq p_j\})}{\exp(\delta) + \exp(-\gamma p_i + \alpha \mathbf{1}\{p_i \leq p_j\}) + \exp(-\gamma p_j + \alpha \mathbf{1}\{p_j \leq p_i\})}.$$

Undercutting Game

Demand with Attention-shifter

$$s_i(p_i, p_j) = \frac{\exp(-\gamma p_i + \alpha \mathbf{1}\{p_i \leq p_j\})}{\exp(\delta) + \exp(-\gamma p_i + \alpha \mathbf{1}\{p_i \leq p_j\}) + \exp(-\gamma p_j + \alpha \mathbf{1}\{p_j \leq p_i\})}.$$

- **Cheapest product:** Recommended to consumers
  Technically, we add to all utilities (rel. to outside) if there are ties... this may matter if we discretize the grid.

Varian 1980

Varian 1980

Consumers consist of Shoppers, S, and Loyals, L, with $S + L = 1$.

$$D = \begin{cases} S & \text{if } p_i > p_j \\ S + L & \text{if } p_i < p_j \\ 0 & \text{if } p_i > \overline{p} \end{cases}$$

Firms solve

$$\max_{p_i} (p_i - c) D(p_i, p_j).$$

Varian 1980

Varian 1980

Consumers consist of Shoppers, S, and Loyals, L, with $S + L = 1$.

$$D = \begin{cases} S & \text{if } p_i > p_j \\ S + L & \text{if } p_i < p_j \\ 0 & \text{if } p_i > \overline{p} \end{cases}$$

Firms solve

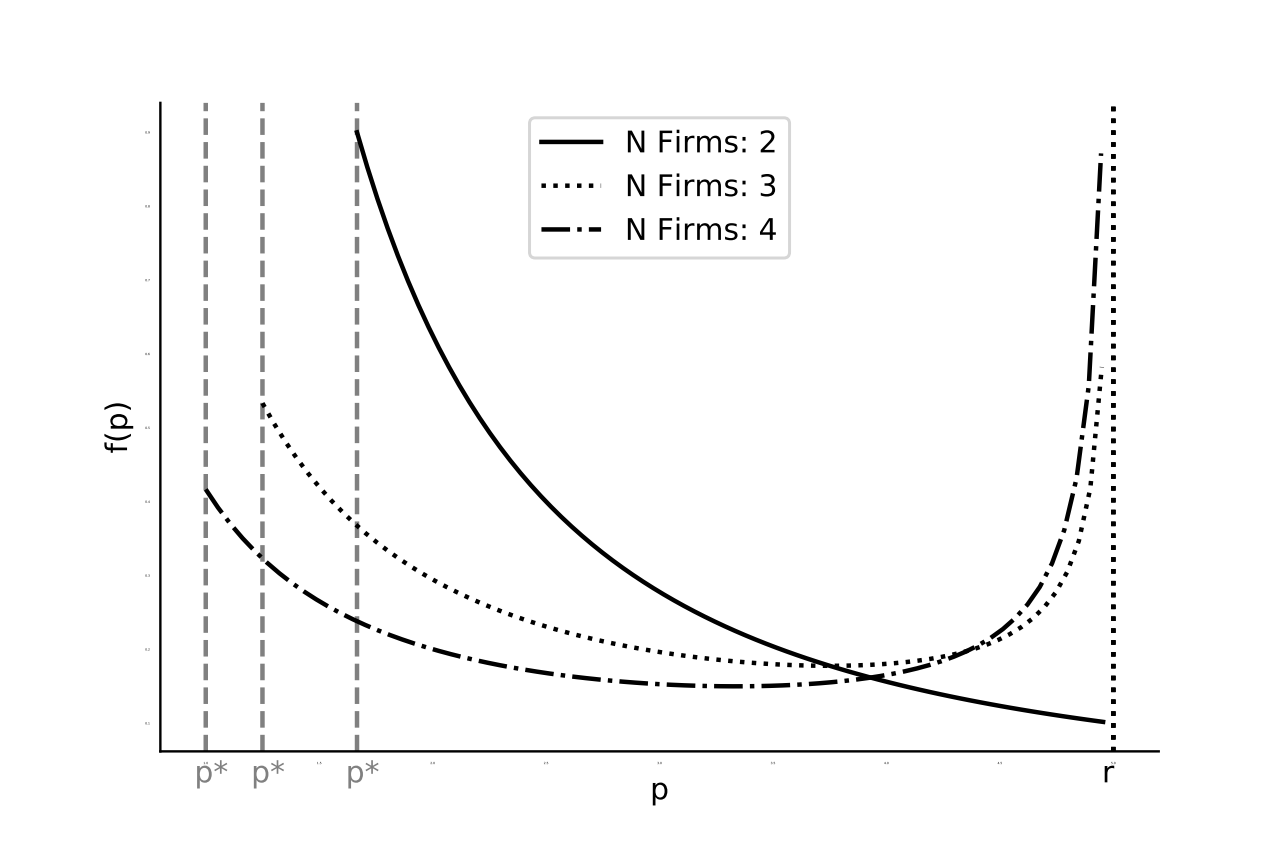
$$\max_{p_i} (p_i - c) D(p_i, p_j).$$

Results:

- No pure strategy equilibrium can exist.
- MSNE has support $[\underline{p}; \overline{p}]$ with $\underline{p} > c$.
- MSNE has no mass points.
- MSNE takes the form

$$F(p) = 1 - \left(\frac{\pi_{\text{lose}}(p)}{\pi_{\text{lose}}(p) - \pi_{\text{win}}(p)} \right)^{\frac{1}{n-1}}.$$

Varian: MSNE



PDF for the mixed strategy Nash equilibrium in Varian (1980), $r =$ choke-off price, $p^* = \inf \sup(p)$.

That's all