Dynamics of Games Final Project

Dan Leonte

February 29, 2024

Abstract

We present two algorithms for Conditonal and Unconditional Regret Matching, together with proof of Hannan Consistency. Specialising to two person zero-sum games, we prove such algorithms can be used to determine ϵ-Nash equilbria as product of empirical marginal distributions of past play. In the end, we discuss a practical implemention of regret matching algorithms for Colonel Blotto game from section 2.6 of [1].

Contents

1 Introduction

The purpose of this project is to describe how regret matching algorithms can be used to determine ϵ Nash equilibria in zero-sum two person games. These algorithms are not restrcied to the class of zero-sum two person games, hence we present them more generally and only restrict our attention at the end.

In subsections 1.1 and 1.2 we formalize the notions of game theory and present the algorithms. In section 2 we present Blackwell's approachability theorem, which is later used in section 3 to prove converge of conditional and unconditional regret matching algorithms to CE and CCE sets respectively. Section 4 then specializes to the connection between the CCE (no regret) set to Nash equilibria set in zero-sum two person games. Finally, section 4 is dedicated to practical observations for the unconditional regret matching algorithm.

1.1 Elementary notions in game theory

This subsection is adapted from [1] and [2].

Definition 1.1. *A game $\mathcal{G}$ is a tuple (N, S, u) where:*

- *N is the number of people playing the game*
- *$S_1, \dots, S_N$ are the pure actions (also called pure strategies) each player can take*
- *$u\colon S = S_1 \times \dots \times S_N \to \mathbb{R}^N$ represents the payoff function, fully determined by the player's actions.*

Let $\Delta_n = \{x \in \mathbb{R}^n \colon 0 \leq x_i \leq 1, x_1 + \dots + x_n = 1\}$ and $\Delta(S) = \Delta_{|S_1|} \times \dots \times \Delta_{|S_N|}$. We can extend the payoff function to (expected) payoffs of mixed strategies, i.e. for any

$$u\colon \Delta(S) \to \mathbb{R}^N, \quad q \to \mathbb{E}_q[u].$$

Definition 1.2. *A probability distribution $\hat{X} = (\hat{x}_1, \dots, \hat{x}_N) \in \Delta(S)$ is a Nash equilibrium if:*

$$\hat{x}_i = \operatorname*{argmax}_{x^i \in \Delta_{|S_i|}} u^i(\hat{x}_1, \dots, \hat{x}_{i-1}, x_i, \hat{x}_{i+1}, \dots, \hat{x}_N)$$

for any $1 \leq i \leq N$, where u^i denotes the i^{th} component of vector u. In other words, $\hat{X}$ is a Nash equilibrium if none of the players can increase their payoff by an unilateral change.

If $s_t = (s_t^1, \dots, s_t^N)$ are the actions undertaken at time t, we use the shorthand s^i for the action of player i and s^{-i} for the actions of all other players.

Definition 1.3. *A probability distribution ψ on $\Delta(S)$ is an $\epsilon-$correlated equilibrium (CE) for some $\epsilon \geq 0$ if:*

$$\sum_{s \in S:\ s^i = j} \psi(s) \cdot [u^i(k, s^{-i}) - u^i(s)] \leq \epsilon.$$

for any $1 \leq i \leq N$ and for any $j, k \in S^i$. If $\epsilon = 0$, this is a correlated equilibrium.

Definition 1.4. *A probability distribution ψ on $\Delta(S)$ is an $\epsilon-$coarse correlated equilibrium (CCE) for some $\epsilon \geq 0$ if:*

$$\sum_{s \in S} \psi(s) \cdot [u^i(k, s^{-i}) - u^i(s)] \leq \epsilon.$$

for any $k \in S_i$. If $\epsilon = 0$, this is a coarse correlated equilibrium.

Lemma 1. *The following inclusion holds:*

$$NE \subset CE \subset CCE.$$

By Nash's Existence Theorem and by definitions 1.3 and 1.4, it follows that the sets of CE and CCE are nonempty and convex.

1.2 Conditional and Unconditional Regret Matching Algorithms

Algorithms presented in the remaining of Section 1 are taken from [3].

Let $h_t = (s_1, \ldots, s_t)$ be the history up to time t.

We consider the case where players repeat the same game over and over again. We assume players have knowledge of their payoffs and of their opponent's choices. More precisely:

1. player i has access to $u^i(s)$ for any possible play $s \in S$
2. player i has access to the history h_t once players chose actions for the t^{th} game.

The first algorithm we study is:

Unconditional Regret Matching

For player i at time t, with observed history h_t, define for each strategy $k \in S_i$:

$$
\begin{aligned}
v_t^i(k) =& u^i(k, s_t^{-1}) - u^i(s_t^i, s_t^{-i}) && (1)\\
D_t^i(k) =& \sum_{\tau=1}^{t} v_\tau^i(k), \quad R_t^i(k) = [D_t^i(k)]^+ && (2)\\
p_{t+1}^i(k) =& \frac{[D_t^i(k)]^+}{\sum_{j\in S^i}[D_t^i(j)]^+} = \frac{R_t^i(k)}{\sum_{j\in S^i} R_t^i(k)} && (3)
\end{aligned}
$$

where $[x]^+ = \max(x, 0)$.

Player i then chooses an action according to the probability distribution $(p_{t+1}^i(k))_{k\in S^i}$.

Conditional Regret Matching

We present Hart and Mas-Colell's regret matching procedure and then a variation of this algorithm. We only prove convergence for the latter, as the analysis for the former is much longer.

For player i, at time t, and for each strategy $k \in S_i$, define:

$$
W_t^i(j,k) = \begin{cases} u_t^i(k, s_t^{-i}) - u_y^i(s_t), & \text{if } s_t^i = j \\ 0, & \text{if } s_t^i \neq j \end{cases} \tag{4}
$$

and

$$
D_t^i(j,k) = \frac{1}{t}\sum_{\tau=1}^{t} W_t^i(j,k). \tag{5}
$$

The variable $D_t^i(j,k)$ tells the difference between the utility of player i if they were to replace every play of action j with action k and the observed utilities of past play when action j was played. The regrets are then defined as:

$$
R_t^i(j,k) = \left[D_\tau^i(j,k)\right]^+ \text{ for any } j,k \in S^i \tag{6}
$$

Assume player i's last action is given by $s_t^i = j$. As per Hart and Mas-Colell's regret matching algorithm, the probabilities $p_{t+1}^i(k)$ of undertaking action k at time $t+1$ are:

$$
p_{t+1}^i(k) = \begin{cases} R_t^i(j,k)/\mu_i & \text{if } j \neq k \\ 1 - (\sum_{k'\neq j} R_t^i(j,k'))/\mu_i & \text{if } j = k \end{cases} \tag{7}
$$

The coefficient μ_i above can be different for each player i and must be large enough (e.g. $\mu_i > 2 \cdot M_i \cdot |S_i|$ where M_i is an upper bound for $u(\cdot,\cdot)$) so that definition 7 gives indeed a probability distribution.

The procedure from 7 has the advantage that probabilities are proportional to regrets, which is intuitively clear, but its analysis is long. Instead we, consider the following similar procedure.

At time t, player i chooses an action according to the stationary distribution of the matrix $(p_t^i(k))_{1 \leq i,k \leq |S_i|}$. Such a distribution always exists for a stochastic matrix (theorem 7).

Although this procedure seems less natural, as probabilities are no longer proportional to regrets, we use the stationary distribution and have more 'machinery' at our disposal.

We can consider further refinements of the algorithms for games with missing information.

1.3 Extensions

As indicated at the begining of subsection 1.2, the algorithms presented above require knowledge of the payoffs and of opponents' choice. If we only have knowledge of the payoffs, we can adjust definitions (4) and (5) on a 'frequency basis'. Define:

$$C_t^i(j,k) = \frac{1}{t}\left[\sum_{\tau \leq t:\ s_\tau^i = k} \frac{p_\tau^i(j)}{p_\tau^i(k)} u^i(s_\tau) - \sum_{\tau \leq t:\ s_\tau^i = j} u^i(s_\tau)\right]$$

and define probabilities of actions as in one of the methods above.

2 Blackwell's theorem

For ease of notation we state Blackwell's theorem in the two player game setting, but the same argument remains valid in an n person game.

Definition 2.1. *Let $\mathcal{X} = \{1, \ldots, m\}, \mathcal{Y} = \{1, \ldots, n\}$ be the sets of pure strategies and let the payoff function $u(\cdot, \cdot) \colon \Delta_m \times \Delta_n \to \mathbb{R}^d$ be a bilinear map. We say that $S \subset \mathbb{R}^d$ is:*

- *satisfiable iff $\exists p \in \mathcal{X}\ \forall q \in \mathcal{Y} \colon u(p, q) \in S$*
- *half-plane satisfiable iff for any half-space $H \supset S, H$ is satisfiable.*

For a nonempty, compact convex set S and for any vector $\lambda \in \mathbb{R}^d$, define the support function

$$w_S(\lambda) = \sup\{\lambda \cdot s \colon s \in S\}.$$

Then S is half-plane satisfiable iff for any λ, there exists a mixed strategy p depending on λ with:

$$\lambda \cdot u(p, q) \leq w_S(\lambda),\ \forall q.$$

Definition 2.2. *A set $S \subset \mathbb{R}^d$ is approachable if for any time t and history $h_t = (x_s, y_s)_{s \leq t}$, there exists a (mixed) strategy such that the average payoff converges to S a.s.*

Formally, define:

$$d_t = d(\frac{1}{t}\sum_{\tau=1}^{t} u(x_\tau, y_\tau), S) \tag{8}$$

Then S is approachable if there exist a strategy (depending only on the history) $\mathcal{A}(h_t) = p_{t+1}$ with

$$\lim_{t \to \infty} d_t = 0 \ a.s. \tag{9}$$

Theorem 2 (Blackwell)**.** *Let $S \subset \mathbb{R}^d$ be nonempty,closed and convex. Then S is approachable if it is half-plane satisfiable.*

A proof can be found in [4].

Remark. *Let $\pi(a) \in S$ be s.t. $d(a, \pi(a))$ is minimal. Let $\lambda(a) = a - \pi(a)$ and $\mathcal{H} = \{z \colon \lambda \cdot z \leq \lambda \cdot a\}$. Assume S is half-plane satisfiable. There must exist a payoff vector p such that:*

$$\lambda \cdot u(p, s^{-i}) \leq \lambda \cdot a \tag{10}$$

for any pure action s^{-i}. We will not use Blackwell's theorem in exactly the form stated in Theorem 2. Instead, by the proof of Blackwell's theorem, it is enough to show that if

$$a_t = \frac{1}{t} \sum_{\tau=1}^{t} u(x_\tau, y_\tau)$$

is the average payoff up to time t, *and if equation 10) holds for* $\lambda = \lambda(a_t)$, *then Blackwell's theorem conclusion is still valid.*

We will use this observation when applying Blackwell's theorem in section 3 and we will refer to it as the Blackwell's condition.

3 Hannan Consistency

3.1 Convergence of Unconditional Regret Matching to CCE

Assume that player i plays according to unconditional regret matching algorithm. We prove this procedure leads to no regret. As defined in subsection 1.2, the action probabilities at time t for unconditional regret matching are given by:

$$\begin{aligned} v_t^i(k) =& u^i(k, s_t^{-1}) - u^i(s_t^i, s_t^{-i}) \\ D_t^i(k) =& \sum_{\tau=1}^{t} v_\tau^i(k), \quad R_t^i(k) = [D_t^i(k)]^+ \\ p_{t+1}^i(k) =& \frac{R_t^i(k)}{\sum_{j \in S^i} R_t^i(k)} \end{aligned}$$

where $[x]^+ = \max(x, 0)$ and $k \in S_i$. Additionally, let $S_i = \{e_1, \ldots, e_n\}$ where $n := |S_i|$.

Proof. We aim to prove the set $S := \mathbb{R}_-^L$ is approachable by the $|S_i|$ dimensional vector of payoffs:

$$D_t = D_t^i(e_j)_{1 \le j \le |D_j|}$$

It is enough to show Blackwell's condition holds, i.e. for $\lambda = \lambda(D_t) \in \mathbb{R}^L$ and for any play s^{-i}, the following holds:

$$[\lambda_1, \ldots, \lambda_n] \cdot \begin{bmatrix} v^i(e_1) \\ v^i(e_2) \\ \cdots \\ v^i(e_n)) \end{bmatrix} = [\lambda_1, \ldots, \lambda_n] \cdot \begin{bmatrix} u(p^i, s^{-i}) - u(e_1, s^{-i}) \\ u(p^i, s^{-i}) - u(e_2, s^{-i}) \\ \cdots \\ u(p^i, s^{-i}) - u(e_n, s^{-i}) \end{bmatrix} \le w_S(\lambda) = \sup\{\lambda \cdot s \colon s \in S\}$$

If λ has a strictly-negative component, then $w_S(\lambda) = \infty$ and there is nothing to prove. Assume $\lambda \in \mathbb{R}_+^L$. In this case, $w_S(\lambda) = 0$. Note also that $\lambda(D)(e_j) = [D]^+(e_j) = R(e_j)$. We need to show:

$$\sum_{j=1}^{n} R^i(e_j) \left[\sum_{k=1}^{n} \frac{R^i(e_k)}{R^i(e_1) + \ldots R^i(e_n)} \cdot (A(e_k, q) - A(e_j, q)) \right] \le 0. \tag{11}$$

Note that the LHS cancels out after expanding the brackets, and so Blackwell's approachability conditions is satisfied and the joint distribution converges a.s. to the no regret set. $\square$

3.2 Convergence of Conditional Regret Matching to CE

Assume that player i plays according to conditional regret matching algorithm. We aim to prove that the empirical joint distribution of players' past actions, as defined in 4).5) and 6), converges a.s. to the set of CE.

As a remainder, we follow notation:

$$W_t^i(j,k) = \begin{cases} u^i(k, s_t^{-i}) - u^i(s_t), & \text{if } s_t^i = j \\ 0, & \text{if } s_t^i \neq j, \end{cases}$$

$$D_t^i(j,k) = \frac{1}{t} \sum_{\tau=1}^{t} W_t^i(j,k)$$

and

$$R_t^i(j,k) = \left[D_\tau^i(j,k)\right]^+ \text{ for any } j,k \in S^i.$$

Consider the stationary distribution $q_t^i = (q_t^i(j))_{j \in S_i}$ of the matrix of probabilities $(p_t^i(k))_{1 \leq i,k \leq |S_i|}$ and denote for simplicity $R_t^i(j,j) = 0$ for any $j \in S_i$. Then, at time t for player i we have

$$q_t^i(j) = \sum_{k \in S_i \colon k \neq j} \frac{1}{\mu_i} q_t^i(k) R_t^i(k,j) + q_t^i(j) \left[1 - \sum_{k \in S_i \colon k \neq j} \frac{1}{\mu_i} R_t^i(j,k)\right]$$

for every $j \in S_i$. Hence

$$\sum_{k \in S_i} q_t^i(k) R_t^i(k,j) = q_t^i(j) \sum_{k \in S_i} R_t^i(j,k) \tag{12}$$

We will use this equation to prove the half-plane satisfiability condition in Blackwell's theorem.

Proof. Let $L = \{(j,k) \in S_i \times S_i \colon j \neq k\}$. We aim to prove that the set $S \colon = \mathbb{R}_-^L$ is approachable by the $|L|$ dimensional payoff vector

$$D_t = D_t^i(j,k)_{(j,k) \in L}.$$

Hence, it is enough to show Blackwell's condition holds, i.e. for $\lambda = \lambda(D_t) \in \mathbb{R}^L$ and for any play s^{-i}, the following holds:

$$\lambda \cdot [\sum_{s^i \in S_i} q^i(s^i)\ W^i(s^i,k)] \leq w_S(\lambda) = \sup\{\lambda \cdot s \colon s \in S\}$$

If λ has a strictly-negative component, then $w_S(\lambda) = \infty$ and there is nothing to prove. Assume $\lambda \in \mathbb{R}_+^L$. In this case, $w_S(\lambda) = 0$ and we need to show:

$$\sum_{(j,k)\in L} \lambda(j,k) \left[\sum_{s^i\in S_i} q^i(s^i)\ W^i(j,k) \right] \le 0.$$

As $W^i(j,k) = 0$ unless the chosen action is $j = s^i$, we have the equivalent inequality:

$$\sum_{(j,k)\in L} \lambda(j,k)\ q^i(j)\ W^i(j,k) \le 0 \tag{13}$$

With $j = s^i$ in mind, we have:

$$\begin{aligned}
&\sum_{(j,k)\in L} \lambda(j,k)\ q^i(j)\ W^i(j,k) = \sum_{(j,k)\in L} \lambda(j,k)\ q^i(j) \left[u^i(k,s^{-i}) - u^i(s)\right] = \\
&= \sum_{(j,k)\in L} \lambda(k,j)\ q^i(k)\ u^i(j,s^{-i}) - \sum_{(j,k)\in L} \lambda(j,k)\ q^i(j)\ u^i(s) = \\
&= \sum_{(j,k)\in L} \left[\lambda(k,j)\ q^i(k) - \lambda(j,k)\ q^i(j)\right] u^i(s) = \\
&= \sum_{j\in S_i} u^i(s) \left[\sum_{k\in S_i:\ k\neq j} \lambda(k,j)\ q^i(k) - q^i(j) \sum_{k\in S_i:\ k\neq j} \lambda(j,k) \right] \\
&= \sum_{j\in S_i} u^i(s)\ \alpha(j)
\end{aligned}$$

where for a given j we define

$$\alpha(j) = \sum_{k\in S_i:\ k\neq j} \lambda(k,j)\ q^i(k) - q^i(j) \sum_{k\in S_i:\ k\neq j} \lambda(j,k). \tag{14}$$

Thus, inequality 13 is equivalent to:

$$\sum_{j\in S_i} u^i(s)\ \alpha(j) \le 0 \tag{15}$$

and holds with equality. Indeed, note that $\lambda(D_t) = [D_t]^+$ and $[D_t]^+(j,k) = R_t^i(j,k)$, hence $\lambda(D_t)(j,k) = R_t^i(j,k)$. Thus, inequality 15) holds with equality and Blackwell's condition from equation 10) is satisfied. By using the same reasoning as in Blackwell's theorem, we obtain the conclusion. □

Note that by lemma 1, both regret matching algorithms lead to no regret.

4 No regret set in zero-sum two person games and connection to Nash equilibria

From now on, we restrict to zero-sum two person games. The goal of this section is to prove theorem 3, adapted from [5], [6] and [7], and discuss empirical observations.

4.1 Theoretical arguments

Theorem 3. *Assume that both players play according to Hannan consistent strategies. Let p_n and q_n be the empirical marginal distributions of past play for player 1 and player 2 respectively:*

$$p_t(i) = \frac{1}{t}\sum_{\tau=1}^{t} I[\text{action } i] \quad \text{and} \quad q_t(j) = \frac{1}{t}\sum_{\tau=1}^{t} I[\text{action } j] \tag{16}$$

where $i \in S_1, j \in S_2$. If p^*, q^* are limit points of p_t and q_t respectively, then the product distribution $p^* \times q^*$ is a NE.

Note that theorem 3 doesn't guarantee the empirical joint distribution converges to a NE. In general, even in two person zero-sum games there exist CE (and implicitly CCE) that are not NE. One example from [6] is the game with payoff matrix:

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

and joint probability distribution :

$$\begin{bmatrix} 1/3 & 1/3 & 0 \\ 1/3 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We saw in section 3 that both conditional and unconditional regret matching are Hannan consistent. Then we can simply compute the empirical marginal distribution (i.e. the time average for each action) and obtain $\epsilon-$NE. To prove theorem 3, we need the following lemma, proved in [2]:

Lemma 4. *A pair (x^*, y^*) is a NE iff:*

$$u(x^*, y^*) = \max_x \min_y u(x, y) = \min_y \max_x u(x, y)$$

Call $u(x^*, y^*)$ the value of the game. By replacing the utility u with the loss $l := -u$, the lemma is further equivalent to

Lemma 5. *A pair* (x^*, y^*) *is a NE iff:*

$$l(x^*, y^*) = \max_y \min_x l(x, y) = \min_x \max_y l(x, y)$$

Call $V := l(x^*, y^*)$ the loss of the game.

Proof. Let $h_t = (I_\tau, J_\tau)_{\tau \leq t}$. As both players follow Hannan consistent strategies, there exists T such that all regrets are less than ϵ for all $t > T$. By the fact that p^* is a limit point of p_t, there exist $t_0 > T$ with:

$$\max_j l(p^*, j) \leq \max_j l(p_{t_0}, j) + \epsilon$$

□

Note that

$$\max_j l(p_{t_0}, j) + \epsilon = \max_j \frac{1}{t_0} \sum_{\tau=1}^{t_0} l(I_\tau, j) + \epsilon \leq \max_j \frac{1}{t_0} \sum_{\tau=1}^{t_0} l(I_\tau, J_t) + 2\epsilon \leq 3\epsilon$$

where the last two inequality are due to second and respectively first player having regrets $\leq \epsilon$. Taking $\epsilon \to 0$, we have:

$$\max_j l(p^*, j) \leq V.$$

Applying the same argument to the other player and applying Lemma 5), we have the desired result.

4.2 Empirical observations

As proved in the previous section, we expect the empirical marginal distributions of players' past play to converge to the set of NE. Paper [1] claims that the average strategy[1] also converges to the set of NE. We now discuss observations from the implementation of unconditional regret matching.

Figures 1,2a,2b illustrate slightly different values for the empirical marginal distributions of pure actions up to iterations 500000, 3000000 and 10000000 (500k, 3M, 10M).

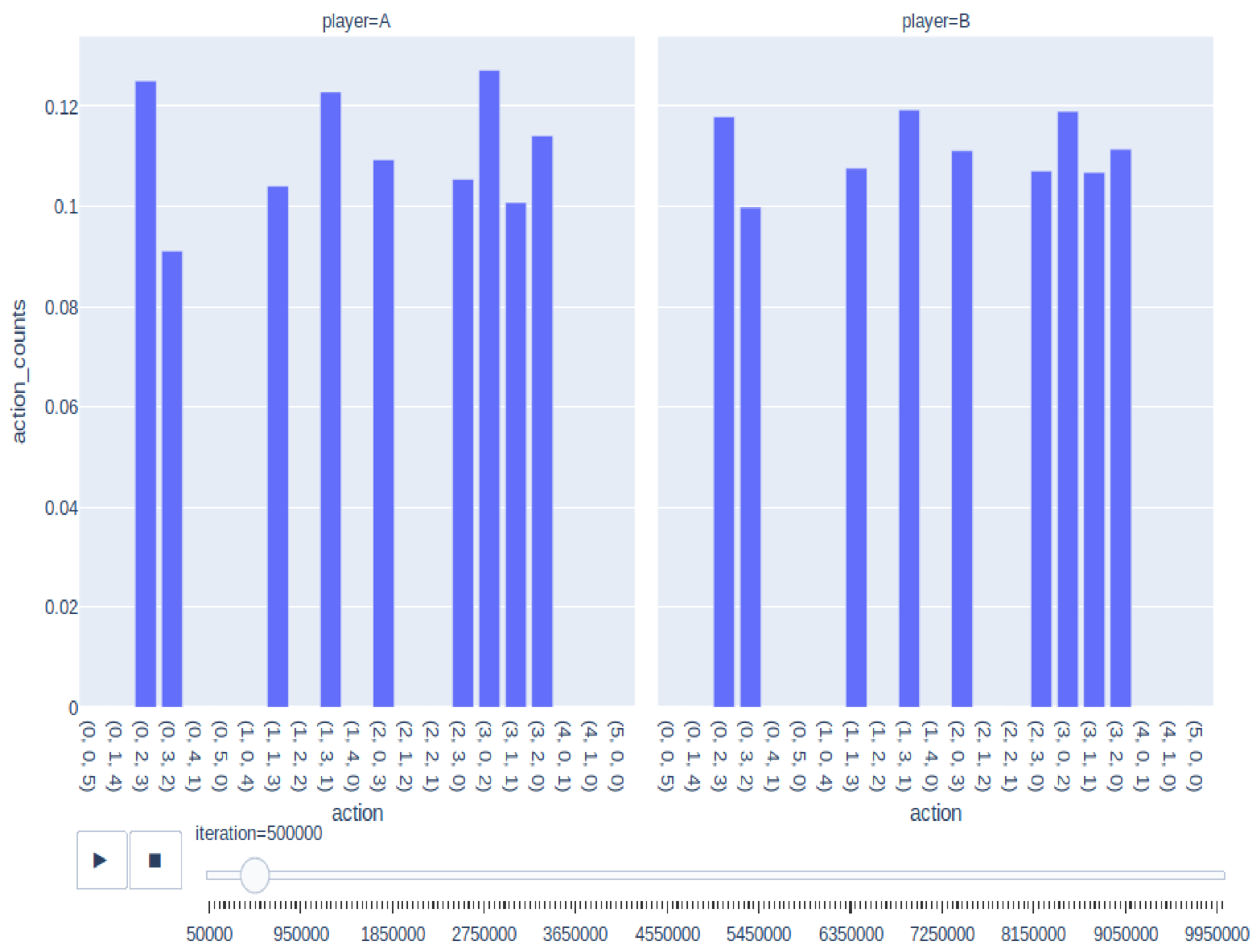
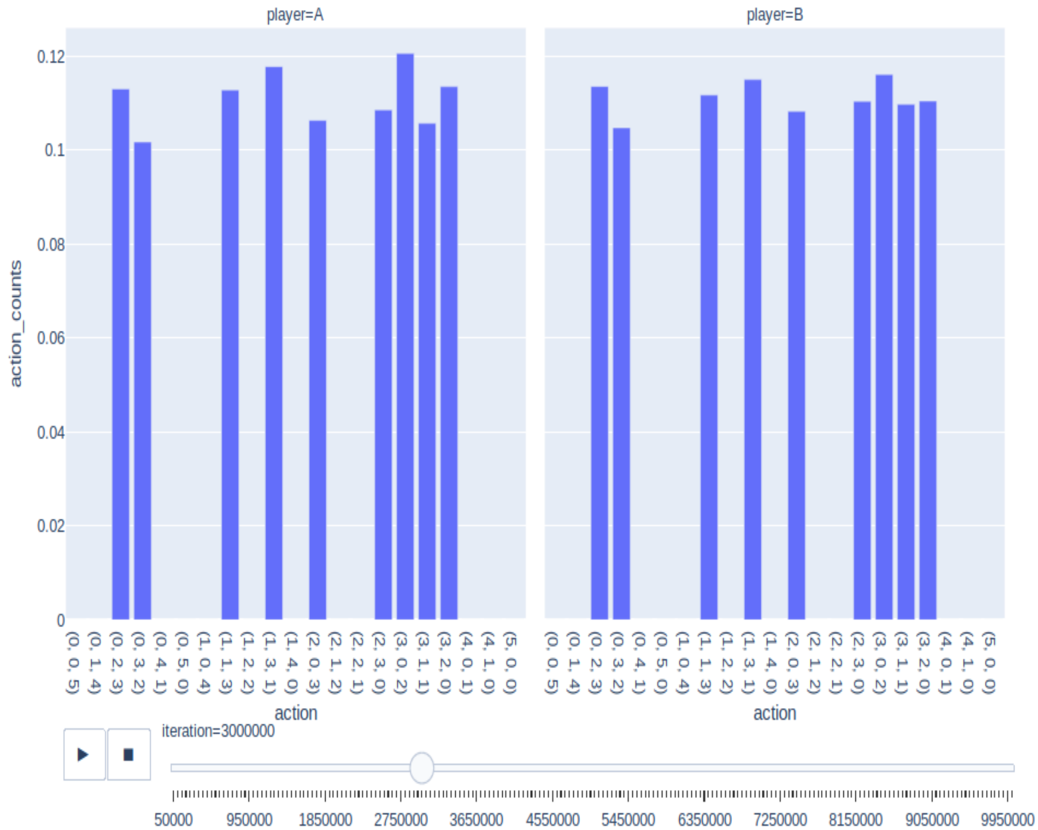
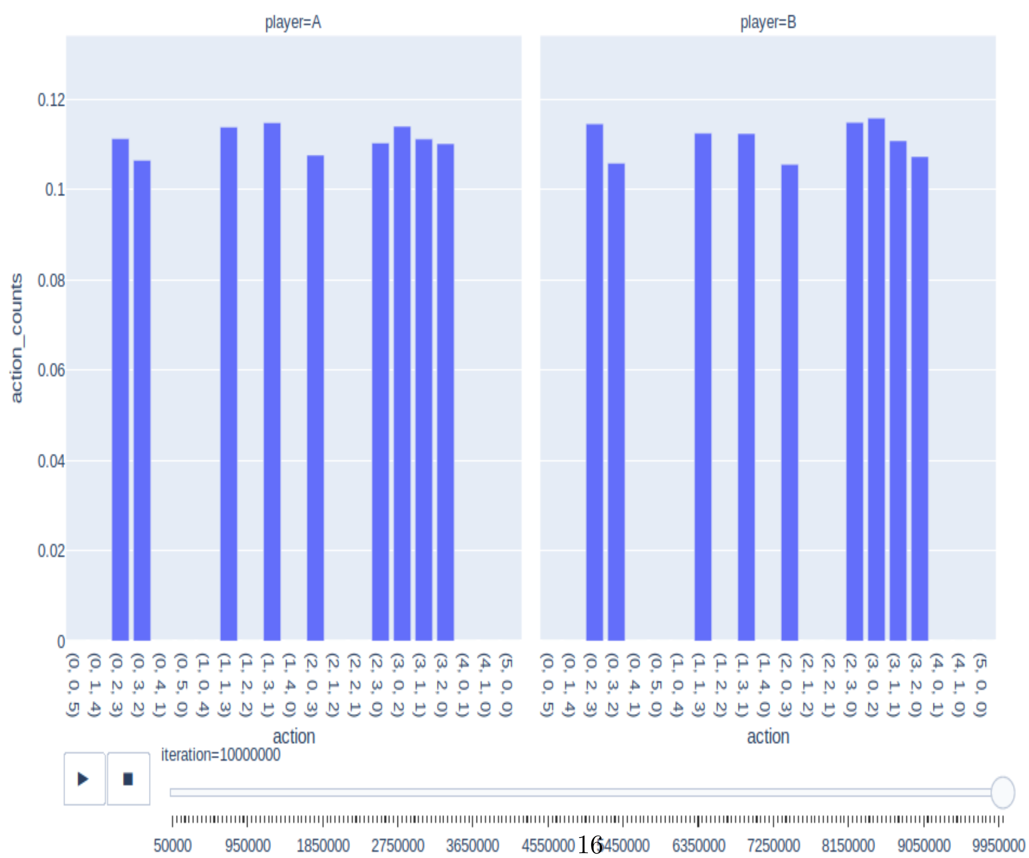


Figure 1: Empirical pure action distribution up to iteration 500000

[1] instead of the empirical marginal probabilities of past play, defined in equation 16), take the average strategy to be the time-average vector of probabilities of choosing actions



(a) Empirical pure action distribution up to iteration 3000000



(b) Empirical pure action distribution up to iteration 10000000

We observe similar fluctuations for the average strategy in Figures 3 and 4.

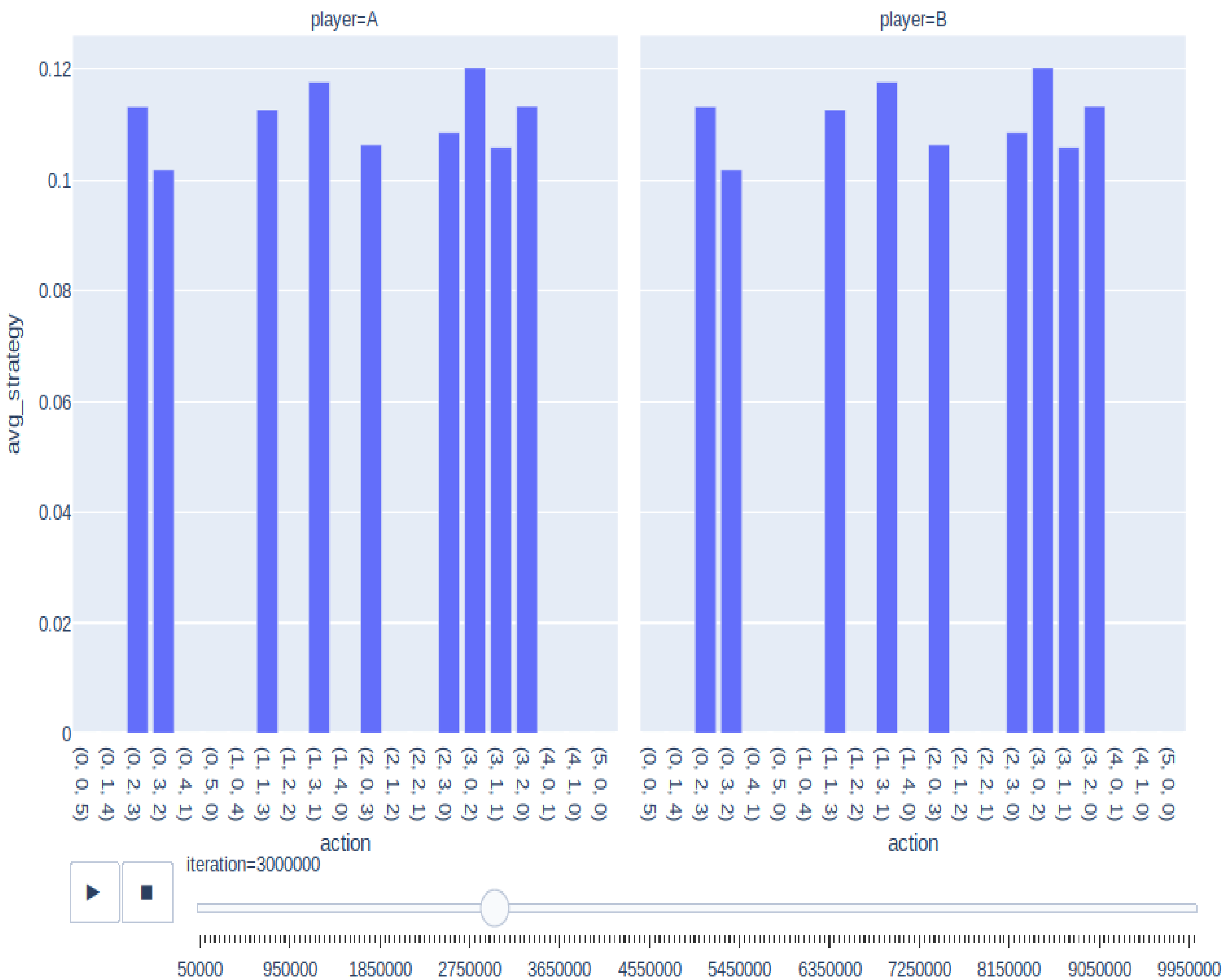


Figure 3: Average strategy at iteration 30000000

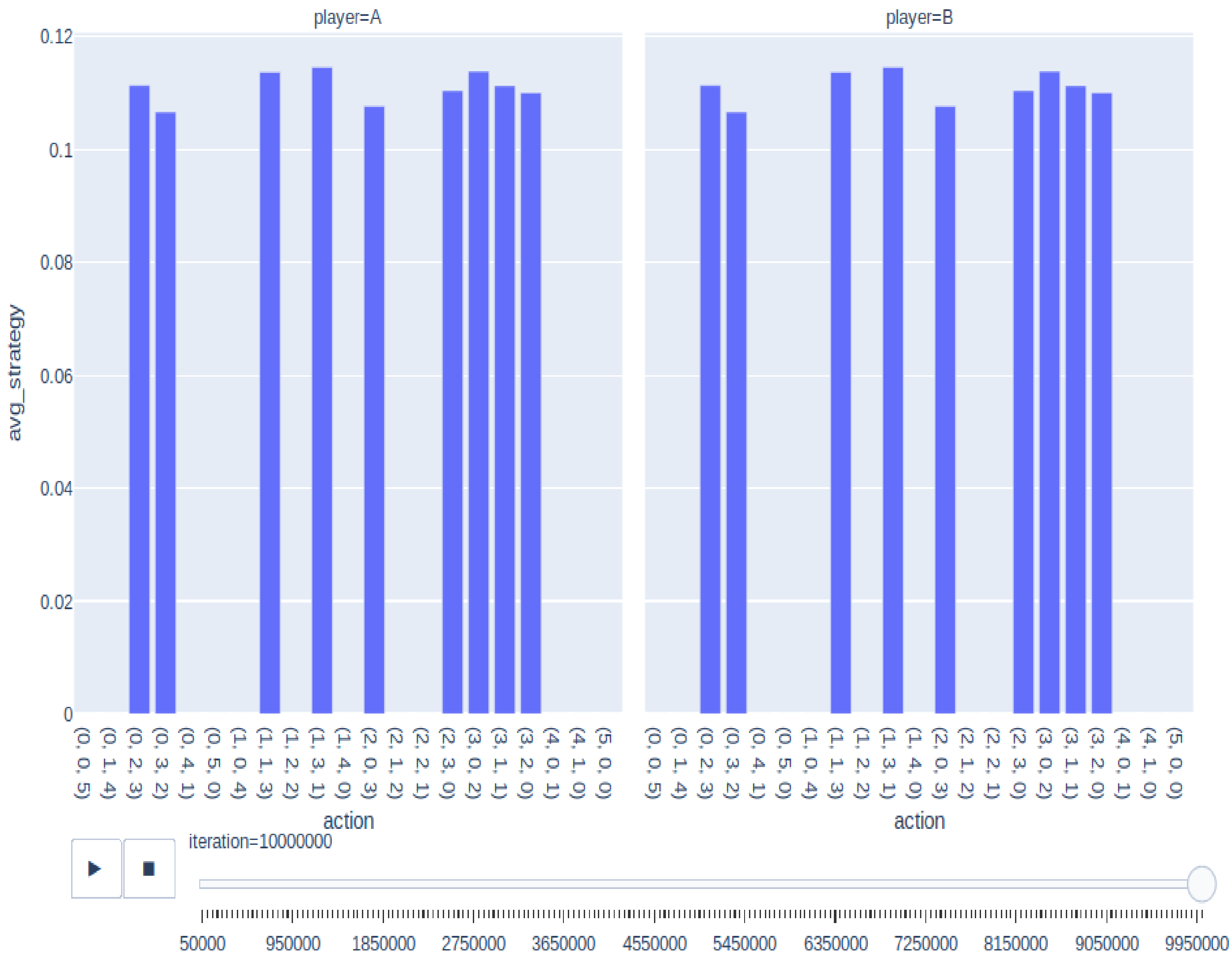
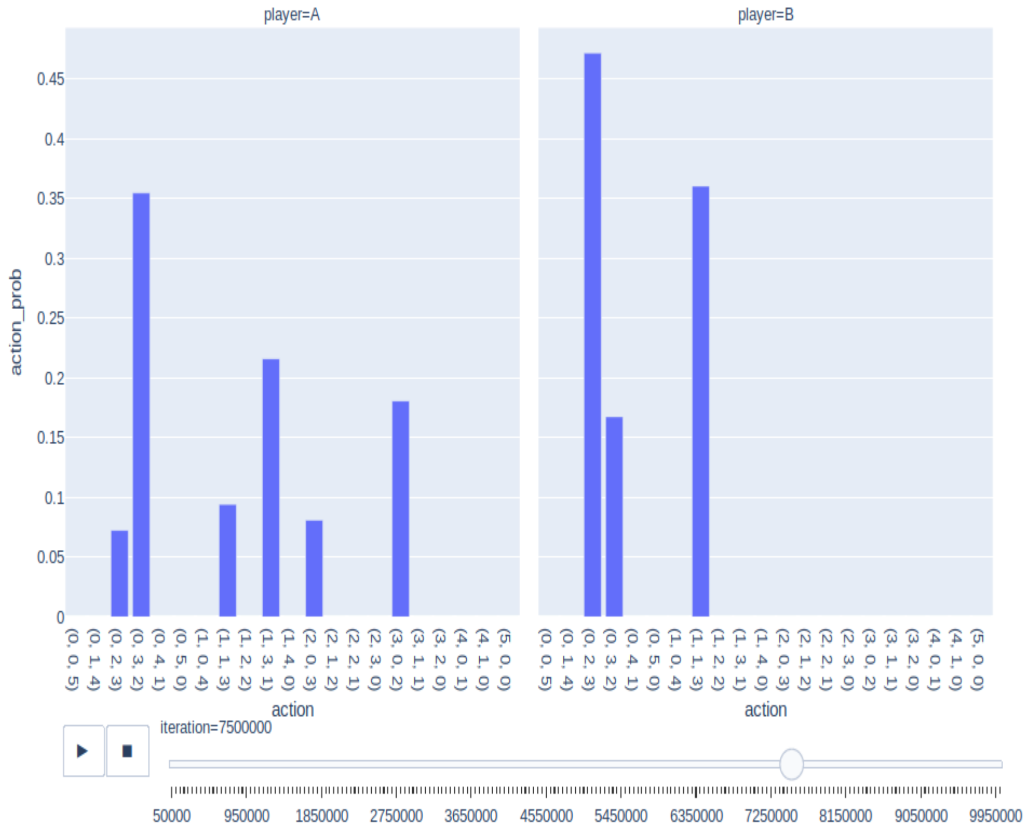
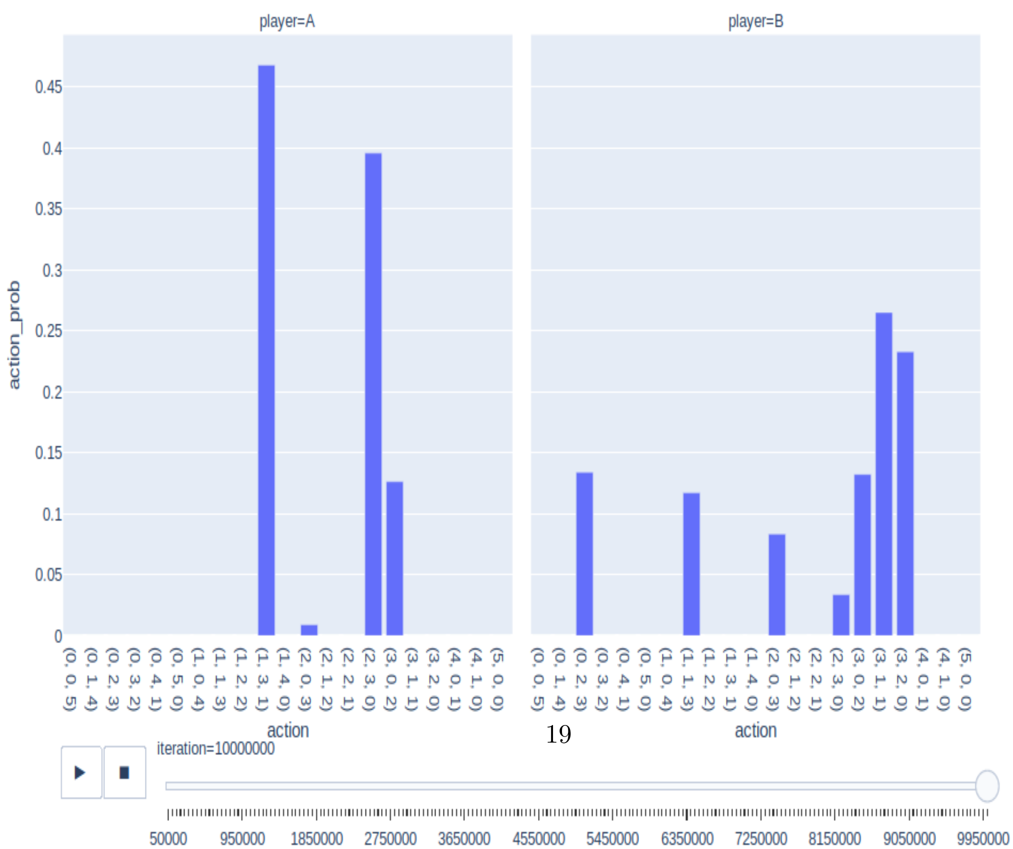


Figure 4: Average strategy at iteration 100000000

As compared to the empirical marginals, the probability of undertaking actions by the two players vary a lot (figures 5a and 5b). We notice such extreme jumps even at time intervals of 5000 iterations.



(a) Probability of undertaking actions at iteration 7500000

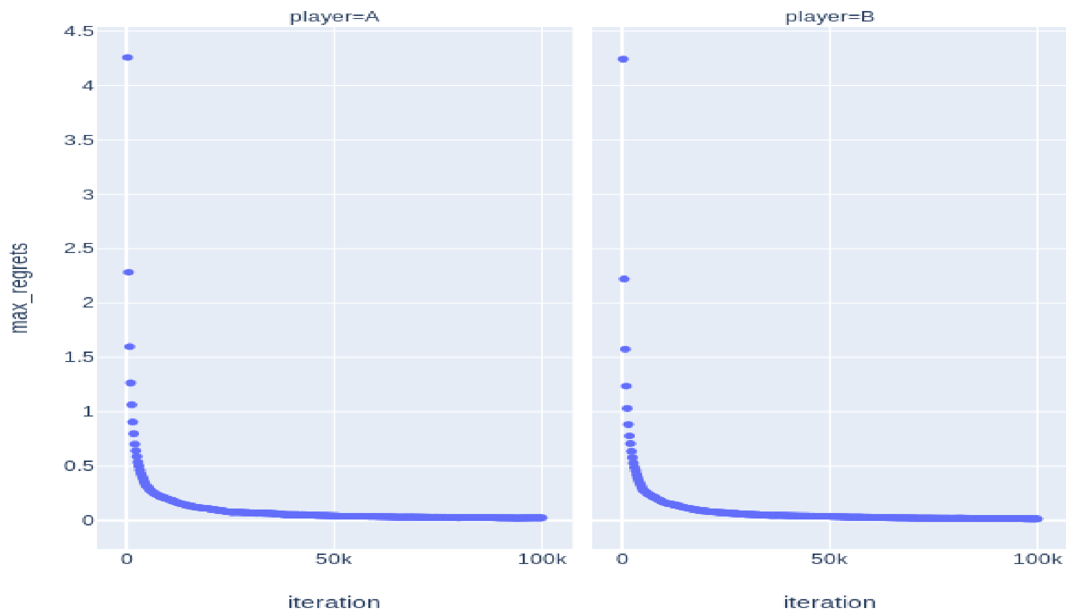


(b) Probability of undertaking actions at iteration 10000000

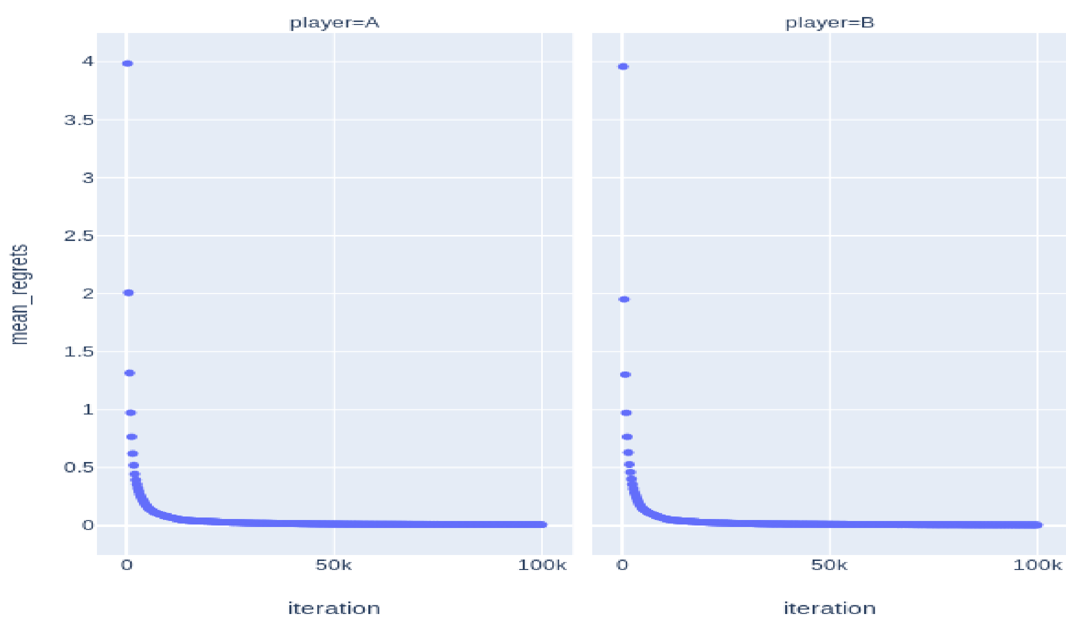
As expected, (unconditional) regret converges to 0. Figures 6a,6b show mean and max regret and figures 7a,7b show the same plot in log-log space.

After running multiple simulations, we empirically note that mean and max regrets converge to 0 like $\frac{1}{x^{\alpha}}$, with α usually between 1 and 2. This can be better visualised in figures 7a and 7b, where the regression lines have slope in the interval $(-1, 2)$.

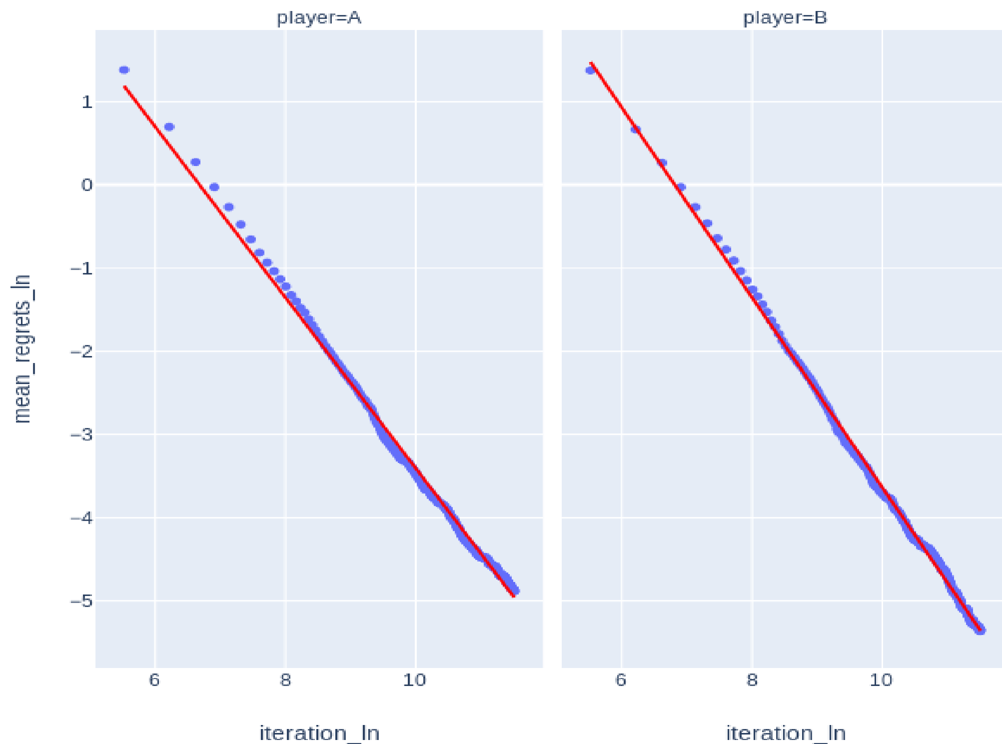
We have two possible cases. It could be that the empirical marginal distributions & average strategy do converge to a point, but haven't converged after 10M iterations. More likely, the empirical marginal & average strategy will never converge to a point and will 'cycle' ϵ close to the set of NE. In this case, it means that the NE set is not discrete (i.e. a finite set of points). It is not clear to me if any other conclusions on the shape of NE set can be made from this analysis.



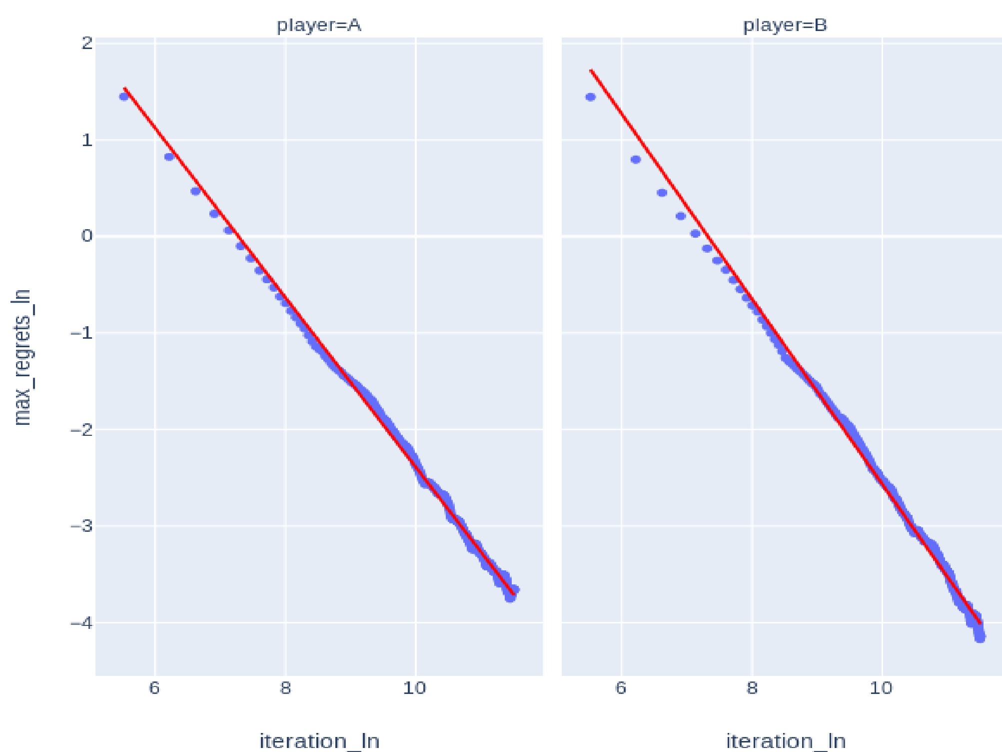
(a) Mean regret vs. iteration



(b) Max regret vs. iteration



(a) Mean regrets vs.iteration (log-log plot)



(b) Max regret vs. iteration (log-log plot)

References

[1] Neller TW, Lanctot M. An introduction to counterfactual regret minimization. In: Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013); 2013. .

[2] Clark T. *Lecture notes in Dynamics of Games and Learning Theory*. Department of Mathematics, Imperial College London; 2019.

[3] Hart S, Mas-Colell A. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*. 2000;68(5):1127–1150.

[4] Kleinberg R. *CS 683 — Learning, Games, and Electronic Markets*. Cornell University; 2007.

[5] Farina G, Kroer C, Sandholm T. Regret minimization in behaviorally-constrained zero-sum games. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org; 2017. p. 1107–1116.

[6] Cesa-Bianchi N, Lugosi G. *Prediction, learning, and games*. Cambridge university press; 2006.

[7] Hajek B. *Lecture notes in Game Theory*. Department of Electrical and Computer Engineering University of Illinois at Urbana-Champaign; 2019.

Appendices

A Theorems

Theorem 6 (von Neumann)**.** *There exists a pair* (x^*, y^*) *such that:*

$$u(x^*, y^*) = \max_x \min_y u(x, y) = \min_y \max_x u(x, y)$$

Theorem 7. *Every stochastic matrix has a stationary distribution.*

B Algorithm Implementation in Python

Auxiliary code to geenrate all possible strategies and determine the stationary distribution of a stochastic matrix

```
import numpy as np

def enumerate_actions(n,k):
    """ list all length k tuples with non-negative entries which sum up to n
    """
    result = []

    def helper(partial_sol,n,k):
        if k == 1:
            result.append(partial_sol + [n,])
        elif k > 1:
            for i in range(n+1):
                helper(partial_sol+[i],n-i,k-1)
    helper([],n,k)

    return np.array(result)

def evec_with_eval1(A):
    """ A is a stochastic matrix (np.array) which has a stationary distribution (left evector with evalue 1)
    """
    e_val,e_vec = np.linalg.eig(np.transpose(A))
    index_eval_1 = np.argmin(np.abs(e_val-1))
    return e_vec[:,index_eval_1]
```

Regret Matching Algorithms

```
import numpy as np
import pandas as pd
from auxiliary import enumerate_actions


class Game:

    def __init__(self,S,N,nr_steps,multiple):
        assert S > N
        self.actions    = enumerate_actions(S,N)
```

```
        self.nr_actions = len(self.actions)
        self.utilities = self.precompute_utilities()
        self.nr_steps = nr_steps
        self.multiple = multiple
        self.history = None


    def get_utility(self,action1_index, action2_index):

        soldiers = self.actions[action1_index] - self.actions[action2_index]
        positions_won  = (soldiers >0).sum()
        positions_lost = (soldiers <0).sum()

        return np.sign(positions_won - positions_lost)


    def precompute_utilities(self):
        utilities = np.empty([self.nr_actions,self.nr_actions])

        for i in range(self.nr_actions):
            for j in range(self.nr_actions):
                utilities[i,j] = self.get_utility(i,j)

        return utilities


    def train_unconditional_regret(self):
        p1 = Player(self.nr_actions)
        p2 = Player(self.nr_actions)

        p1.last_regret = [0] * self.nr_actions
        p2.last_regret = [0] * self.nr_actions

        history_regrets      = []
        history_prob         = []
        history_actions      = []
        history_avg_strategy = []
        history_max_regrets  = []
        history_mean_regrets = []


        for i in range(0,self.nr_steps+1):

            [action1_index,proportions_1] = p1.choose_strategy_unconditional()
            [action2_index,proportions_2] = p2.choose_strategy_unconditional()

            utility1 = self.utilities[action1_index,action2_index]
            utility2 = self.utilities[action2_index,action1_index]

            regret1 = self.utilities[:,action2_index] - utility1
            regret2 = self.utilities[:,action1_index] - utility2

            p1.update_unconditional(regret1,action1_index,proportions_1)
            p2.update_unconditional(regret2,action2_index,proportions_1)

            ###### add the history ####


            if i% self.multiple == 0 and i > 0 :

                #add regrets and probabilities, computing them from regrets
                history_regrets.append(list(p1.last_regret)+['A',i])
                history_regrets.append(list(p2.last_regret)+['B',i])

```

```
            history_max_regrets.append([p1.last_regret.max() / (i+1),'A',i])
            history_max_regrets.append([p2.last_regret.max() / (i+1),'B',i])

            history_mean_regrets.append([np.array([i if i>0 else 0 for i in p1.last_regret]).mean() / (i+1),'A',i])
            history_mean_regrets.append([np.array([i if i>0 else 0 for i in p2.last_regret]).mean() / (i+1),'B',i])

            history_prob.append(list(proportions_1)+['A',i])
            history_prob.append(list(proportions_2)+['B',i])

            #add action counts
            history_actions.append(list(p1.action_count / (i+1))+['A',i])
            history_actions.append(list(p2.action_count / (i+1))+['B',i])

            history_avg_strategy.append(list(p1.strategy /(i+1)) + ['A',i])
            history_avg_strategy.append(list(p2.strategy /(i+1)) + ['B',i])



        #save a history of probabilities, action played counts in a list
        columns = [str(tuple(i)) for i in self.actions] + ['player','iteration']
        df_regrets     = pd.DataFrame(history_regrets     ,columns = columns)
        df_prop        = pd.DataFrame(history_prob        ,columns = columns)
        df_count       = pd.DataFrame(history_actions     ,columns = columns)
        df_max_regrets = pd.DataFrame(history_max_regrets, columns = ['max_regrets','player','iteration'] )
        df_mean_regrets= pd.DataFrame(history_mean_regrets,columns = ['mean_regrets','player','iteration'])
        df_avg_strategy= pd.DataFrame(history_avg_strategy,columns = [str(tuple(i)) for i in self.actions] + ['player','iteration'])



        self.history = [df_regrets,df_prop,df_count,df_max_regrets,df_mean_regrets,df_avg_strategy]


    def train_conditional_regret(self,mu1,mu2,type_):
        p1 = Player(self.nr_actions)
        p2 = Player(self.nr_actions)
        p1.mu,p2.mu = mu1,mu2


        p1.last_regret = np.zeros([self.nr_actions,self.nr_actions])
        p2.last_regret = np.zeros([self.nr_actions,self.nr_actions])

        history_regrets = []
        history_prob    = []
        history_actions = []
        history_swaps   = [0,0]

        for i in range(0,self.nr_steps+1):

            history_regrets.append(list(p1.last_regret)+['A',i])
            history_regrets.append(list(p2.last_regret)+['B',i])

```

```python
            [action1_index, proportions_1] = p1.choose_strategy_conditional(type_)
            [action2_index, proportions_2] = p2.choose_strategy_conditional(type_)

            if action1_index != p1.last_action : history_swaps[0] += 1
            if action2_index != p2.last_action : history_swaps[1] += 1

            utility1 = self.utilities[action1_index, action2_index]
            utility2 = self.utilities[action2_index, action1_index]

            regret1 = self.utilities[:, action2_index] - utility1
            regret2 = self.utilities[:, action1_index] - utility2

            p1.update_conditional(regret1, action1_index, i+1)
            p2.update_conditional(regret2, action2_index, i+1)

            if i % self.multiple == 0:

                #print(p1.last_regret.max() / (i+1),p1.last_regret.max() / (i+1))


                history_prob.append(list(proportions_1 / proportions_1.sum())+['A',i])
                history_prob.append(list(proportions_2 / proportions_2.sum())+['B',i])

                #add action counts
                history_actions.append(list(p1.action_count / (i+1))+['A',i])
                history_actions.append(list(p2.action_count / (i+1))+['B',i])



        #save a history of probabilities, action played counts in a list
        columns = [str(tuple(i)) for i in self.actions] + ['player','iteration']
        df_regrets = pd.DataFrame(history_regrets    ,columns = columns)
        df_prop    = pd.DataFrame(history_prob     ,columns = columns)
        df_count   = pd.DataFrame(history_actions  ,columns = columns)

        self.history = [df_regrets, df_prop, df_count, history_swaps]


class Player:

    def __init__(self, nr_actions):
        self.action_count  = np.zeros(nr_actions)
        self.nr_actions = nr_actions

        #this will be sett to  vector [0] *nr_actions or  matrix [0] *nr_actions depending on which training method we choose
        self.last_regret   = None
        #needed only for collel conditional training
        self.last_action = None
        self.mu = None
        self.strategy = np.zeros(nr_actions)


    def choose_strategy_unconditional(self):
        proportions = np.array([i if i > 0 else 0 for i in self.last_regret])
        if proportions.sum() > 0:
            proportions = proportions / proportions.sum()
        elif proportions.sum() == 0:
            proportions = [1/self.nr_actions] * self.nr_actions
```

```
        strategy_index = np.random.choice(self.nr_actions, p=proportions)
        return  [strategy_index,np.array(proportions)]


    def update_unconditional(self,regret,action_index,proportions):
        self.last_regret  += regret
        self.action_count[action_index]  += 1
        self.strategy += proportions



    def choose_strategy_conditional(self,type_):
        if self.action_count.sum() == 0:
            proportions = [1/self.nr_actions] * self.nr_actions
        else:
            if  type_  == 'stationary':
                A = np.array([[i if i>0 else 0 for i in arr] for arr in self.last_regret]) / self.mu()
                for i in self.nr_actions:
                    A[i,i] = 0
                    A[i,i] = 1 - A[i,:].sum()

                # A is a stochastic matrix (row sums =1 )
                proportions = evec_with_eval1(A)

            elif type_ == 'collel':
                proportions = np.array([i if i >0 else 0 for i in self.last_regret[self.last_action,:]]) / self.mu
                proportions[self.last_action] = 0
                proportions[self.last_action] = 1 - proportions.sum()


        strategy_index = np.argmax(np.random.multinomial(1,proportions) > 0)
        return  [strategy_index,np.array(proportions)]   #update the last action in update method



    def update_conditional(self,regret,action_index,iteration):
        self.last_regret[action_index,:]  = (self.last_regret[action_index,:] * iteration + regret) / (iteration + 1)
        self.action_count[action_index]  += 1

       #update the last action with the new action played
        self.last_action = action_index
```

The algorithms are implementations of procedures described in [3] and [1]. The animated plots designed with Plotly will be presented in a jupyter notebook during the oral exam.