First-Order Algorithm with $\mathcal{O}(\ln(1/\epsilon))$ Convergence for ϵ-Equilibrium in Two-Person Zero-Sum Games*

Andrew Gilpin
Computer Science Dept.
Carnegie Mellon Univ.
Pittsburgh, PA, USA

Javier Peña
Tepper School of Business
Carnegie Mellon Univ.
Pittsburgh, PA, USA

Tuomas Sandholm
Computer Science Dept.
Carnegie Mellon Univ.
Pittsburgh, PA, USA

November 14, 2010

Abstract

We propose an iterated version of Nesterov's first-order smoothing method for the two-person zero-sum game equilibrium problem

$$\min_{x \in Q_1} \max_{y \in Q_2} x^{\mathrm{T}} A y = \max_{y \in Q_2} \min_{x \in Q_1} x^{\mathrm{T}} A y.$$

This formulation applies to matrix games as well as sequential games. Our new algorithmic scheme computes an ϵ-equilibrium to this min-max problem in $\mathcal{O}\left(\frac{\|A\|}{\delta(A)} \ln(1/\epsilon)\right)$ first-order iterations, where $\delta(A)$ is a certain condition measure of the matrix A. This improves upon the previous first-order methods which required $\mathcal{O}(1/\epsilon)$ iterations, and it matches the iteration complexity bound of interior-point methods in terms of the algorithm's dependence on ϵ. Unlike interior-point methods that are inapplicable to large games due to their memory requirements, our algorithm retains the small memory requirements of prior first-order methods. Our scheme supplements Nesterov's method with an outer loop that lowers the target ϵ between iterations (this target affects the amount of smoothing in the inner loop). Computational experiments both in matrix games and sequential games show that a significant speed improvement is obtained in practice as well, and the relative speed improvement increases with the desired accuracy (as suggested by the complexity bounds).

1 Introduction

Game-theoretic solution concepts provide an appealing normative basis for designing agents for multi-agent settings. The concepts are particularly robust in two-person zero-sum games. Equilibrium-finding algorithms for computing approximately optimal strategies have recently been successfully applied to games as large as two-person Texas Hold'em poker [3, 21, 22].

*A short early version of this paper appeared at the National Conference on Artificial Intelligence (AAAI), 2008.

The Nash equilibrium problem for a two-person zero-sum game can be formulated as a saddle-point problem (we will describe this in detail later). The latter can in turn be cast as a linear program (LP). However, for many interesting instances of games, such as those that arise in real poker, these LPs are enormous and unsolvable via standard algorithms such as the simplex or interior-point methods.

To address this computional challenge, some alternative algorithms have been developed and have been shown to be effective in finding ϵ-equilibria, where neither player can benefit more than ϵ by deviating. These include an algorithm based on regret minimization [22] (which has iteration-complexity $\mathcal{O}\left(1/\epsilon^2\right)$) and iterative bundle-based methods [9, 21].

Another recent approach [6] is based on Nesterov's [12, 13] first-order smoothing techniques. The main strength is simplicity and low computational cost of each iteration. That algorithm finds an ϵ-equilibrium within $\mathcal{O}(1/\epsilon)$ iterations. In contrast, interior-point methods find an ϵ-equilibrium within $\mathcal{O}(\ln(1/\epsilon))$ iterations [19], but do not scale to large games due to memory requirements.

In this paper we propose an iterated version of Nesterov's smoothing technique for nonsmooth convex optimization [13] that runs in $\mathcal{O}\left(\frac{\|A\|}{\delta(A)}\ln(1/\epsilon)\right)$ iterations. In terms of ϵ, the iteration complexity is thus the same as that of interior-point methods and exponentially better than that of prior first-order methods. The complexity also depends on a certain condition measure, $\delta(A)$, of the payoff matrix A. Unlike interior-point methods, we inherit the manageable memory usage of prior first-order methods. So, our algorithm scales to large games and small ϵ.

Our algorithm can be applied to general linear programs via the classical reformulation of a linear program as the Nash equilibrium problem of a two-person zero-sum game (see, e.g., [2]). It is known that the set of Nash equilibria of this reformulation is in one-to-one correspondence with the set of primal-dual optimal solutions to the linear program provided that both the primal and dual problems are strictly feasible [2]. For this class of linear programs, we obtain an infeasible primal-dual algorithm similar in spirit to the homogeneous self-dual interior-point methods for linear programming discussed in [19, 20]. From our main complexity result, it readily follows that for a linear program of the form

$$\begin{aligned} \max \quad & c^{\mathrm{T}}x \\ & Ax \leq b \\ & x \geq 0, \end{aligned}$$

our algorithm finds a primal-dual pair within ϵ of both feasibility and optimality in $\mathcal{O}(\mathsf{cond}(A, b, c)\ln(1/\epsilon))$ first-order iterations. Here $\mathsf{cond}(A, b, c)$ is a combination of the condition measure $\delta(\cdot)$ applied to the reformulation of the linear program as a Nash equilibrium problem, and a measure of strict primal and dual feasibility.

2 First-Order Methods

Assume $Q \subseteq \mathbb{R}^n$ is a compact convex set and $f : Q \to \mathbb{R}$ is convex. Consider the convex optimization problem

$$\min\{f(x) : x \in Q\}. \tag{1}$$

This paper is concerned with *first-order methods* for solving a particular form of problem (1). The defining feature of these methods is that the search direction at each main iteration is obtained using only first-order information, such as the gradient or subgradient of the function f. This feature makes their computational overhead per iteration extremely low, and hence makes them attractive for large-scale problems.

The complexity of first-order methods for finding an approximate solution to (1) depends on the properties of f and Q. For the setting where f is continuously differentiable with Lipschitz gradient, Nesterov [11] proposed a gradient-based algorithm with convergence rate $\mathcal{O}(1/\sqrt{\epsilon})$. In other words, within $\mathcal{O}(1/\sqrt{\epsilon})$ iterations, the algorithm outputs a value $x \in Q$ such that $f(x) \leq f(x') + \epsilon$ for all $x' \in Q$, including the optimal one. We refer to this algorithm as *Nesterov's optimal gradient algorithm* since it can be shown that for that smooth class of problems, no gradient-based algorithm has faster convergence. A variant by Lan, Liu, and Monteiro [8] also features $\mathcal{O}(1/\sqrt{\epsilon})$ convergence and outperformed the original in experiments.

For the setting where f is non-differentiable, *subgradient* algorithms are often used. They have complexity $\Theta(1/\epsilon^2)$ [4]. However, this pessimistic result is based on treating f as a black box, whose value and subgradient are available through an oracle. For a function f with a suitable structure, Nesterov [12, 13] devised a first-order method with convergence rate $\mathcal{O}(1/\epsilon)$. The method is based on a *smoothing* technique. The idea is that the structure of f can be used to construct a smooth function with Lipschitz gradient that resembles f. Then, a gradient algorithm (for example, Nesterov's optimal gradient algorithm) applied to the smooth function yields an approximate minimizer for f. This latter technique in particular applies to equilibrium problems arising in two-person zero-sum games, as explained below.

We note that first-order algorithms have also proven to be effective for finding approximate solutions to large-scale LPs [1] and to large-scale nonlinear convex programs [17]. These approaches use $\mathcal{O}(1/\epsilon^2)$ iterations on non-smooth problems. For a special class of continuously differentiable minimization problems (which is very different from our non-differentiable setting) the first-order algorithm presented by Smola *et al.* [17] runs in $\mathcal{O}(\ln(1/\epsilon))$ iterations.

2.1 Smoothing Scheme for Matrix Games

In this subsection we describe a smoothing method for the min-max matrix game problem

$$\min_{x \in \Delta_m} \max_{y \in \Delta_n} x^{\mathrm{T}} A y = \max_{y \in \Delta_n} \min_{x \in \Delta_m} x^{\mathrm{T}} A y \tag{2}$$

where $\Delta_m := \{x \in \mathbb{R}^m : \sum_{i=1}^m x_i = 1, x \geq 0\}$ is the set of mixed strategies for a player with m pure strategies. The game interpretation is that if player 1 plays $x \in \Delta_m$ and player 2 plays $y \in \Delta_n$, then 1 receives payoff $-x^{\mathrm{T}}Ay$ and 2 receives payoff $x^{\mathrm{T}}Ay$.

Nesterov [13] formulated a first-order smoothing technique for solving for each agent's strategy in a matrix game separately. We present that idea here, but applied to a formulation where we solve for both players' strategies at once.

Problem (2) can be rewritten as the primal-dual pair of nonsmooth optimization problems

$$\min\{f(x) : x \in \Delta_m\} = \max\{\phi(y) : y \in \Delta_n\}$$

where

$$\begin{aligned} f(x) &:= \max\left\{x^{\mathrm{T}}Av : v \in \Delta_n\right\}, \\ \phi(y) &:= \min\left\{u^{\mathrm{T}}Ay : u \in \Delta_m\right\}. \end{aligned}$$

For our purposes it will be convenient to cast this as the primal-dual nonsmooth convex minimization problem

$$\min\{F(x,y) : (x,y) \in \Delta_m \times \Delta_n\}, \tag{3}$$

where

$$F(x,y) = \max\left\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay : (u,v) \in \Delta_m \times \Delta_n\right\}. \tag{4}$$

Observe that $F(x,y) = f(x) - \phi(y)$ is convex and $\min\{F(x,y) : (x,y) \in \Delta_m \times \Delta_n\} = 0$. Also, a point $(x,y) \in \Delta_m \times \Delta_n$ is an ϵ-solution to (2) if and only if $F(x,y) \leq \epsilon$.

Since the objective function $F(x,y)$ in (3) is nonsmooth, a subgradient algorithm would be appropriate. Thus, without making any attempt to exploit the structure of our problem, we would be faced with a worst-case bound on a subgradient-based algorithm of $\mathcal{O}(1/\epsilon^2)$. However, we can get a much better bound by exploiting the structure of our problem as we now show.

The following objects associated to Equation (3) play a central role in the sequel. Let

$$\mathsf{Opt} := \mathrm{Argmin}\{F(x,y) : (x,y) \in \Delta_m \times \Delta_n\}$$

be the set of all optimal solutions and let $\mathsf{dist} : \Delta_m \times \Delta_n \to \mathbb{R}$ be the distance function to the set Opt, *i.e.*,

$$\mathsf{dist}(x,y) := \min\{\|(x,y) - (u,v)\| : (u,v) \in \mathsf{Opt}\}.$$

Let $(\bar{u},\bar{v}) \in \Delta_m \times \Delta_n$ and $\mu > 0$. Consider the following smoothed version of F:

$$F_\mu(x,y) = \max\left\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay - \frac{\mu}{2}\|(u,v) - (\bar{u},\bar{v})\|^2 : (u,v) \in \Delta_m \times \Delta_n\right\}, \tag{5}$$

where $\|\cdot\|$ denotes the Euclidean norm.

Let $(u(x,y), v(x,y)) \in \Delta_m \times \Delta_n$ denote the maximizer in (5). This maximizer is unique since the function

$$x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay - \frac{\mu}{2}\|(u,v) - (\bar{u},\bar{v})\|^2$$

is strictly concave in u and v [13]. It follows from [13, Theorem 1] that F_μ is smooth with gradient

$$\nabla F_\mu(x,y) = \begin{bmatrix} 0 & A \\ -A^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix},$$

and ∇F_μ is Lipschitz with constant $\frac{\|A\|^2}{\mu}$, where $\|A\|$ is the Euclidean matrix norm of A. Let

$$D := \max\left\{ \frac{\|(u,v) - (\bar{u},\bar{v})\|^2}{2} : (u,v) \in \Delta_m \times \Delta_n \right\}.$$

Nesterov's optimal gradient algorithm applied to the problem

$$\min\{F_\mu(x,y) : (x,y) \in \Delta_m \times \Delta_n\} \tag{6}$$

yields the following algorithm. Assume $(x_0, y_0) \in \Delta_m \times \Delta_n$ and $\epsilon > 0$ are given.

smoothing(A, x_0, y_0, ϵ)

1. Let $\mu = \frac{\epsilon}{2D}$ and $(w_0, z_0) = (x_0, y_0)$
2. For $k = 0, 1, \ldots$
   (a) $(u_k, v_k) = \frac{2}{k+2}(w_k, z_k) + \frac{k}{k+2}(x_k, y_k)$
   (b) $(x_{k+1}, y_{k+1}) =$

$$\operatorname{argmin}\left\{ \nabla F_\mu(u_k,v_k)^{\mathrm{T}}((x,y) - (u_k,v_k)) + \frac{\|A\|^2}{2\mu}\|(x,y) - (u_k,v_k)\|^2 : (x,y) \in \Delta_m \times \Delta_n \right\}$$

   (c) If $F(x_{k+1}, y_{k+1}) < \epsilon$ Return
   (d) $(w_{k+1}, z_{k+1}) =$

$$\operatorname{argmin}\left\{ \sum_{i=0}^{k} \frac{i+1}{2} \nabla F_\mu(u_i,v_i)^{\mathrm{T}}((w,z) - (u_i,v_i)) + \frac{\|A\|^2}{2\mu}\|(w,z) - (x_0,y_0)\|^2 : (w,z) \in \Delta_m \times \Delta_n \right\}$$

We note that the Algorithm **smoothing** is essentially identical to Algorithm (3.11) in [13] with a shift in the indices of the sequences (x_k, y_k) and (u_k, v_k). This re-statement of Nesterov's algorithm was proposed by Lan, Lu, and Monteiro [8].

Proposition 1 *Algorithm* **smoothing** *finishes in at most*

$$k = \left\lceil \frac{2\sqrt{2} \cdot \|A\| \cdot \sqrt{D} \cdot \mathsf{dist}(x_0, y_0)}{\epsilon} \right\rceil \tag{7}$$

first-order iterations.

PROOF. This result is similar to [8, Theorem 9]. Since this is a key step for our main complexity result, we next present a detailed proof. Recall the following property of Nesterov's smoothing algorithm (see [13, Theorem 2]).

$$\frac{(k+1)(k+2)}{4} F_\mu(x_k, y_k) \leq$$
$$\min_{(x,y)\in\Delta_m\times\Delta_n} \left\{ \frac{\|A\|^2}{2\mu} \|(x,y) - (x_0,y_0)\|^2 + \sum_{i=1}^{k} \frac{i+1}{2} [F_\mu(x_i,y_i) + \nabla F_\mu(x_i,y_i)^{\mathrm{T}}((x,y) - (x_i,y_i))] \right\}.$$

In particular, since F_μ is convex, it follows that for all $(x,y) \in \Delta_m \times \Delta_n$

$$F_\mu(x_k, y_k) - F_\mu(x,y) \leq \frac{2\|A\|^2 \cdot \|(x,y) - (x_0,y_0)\|^2}{\mu(k+1)(k+2)}.$$

On the other hand, by the construction (5) it follows that the smooth function F_μ satisfies

$$0 \leq F(x,y) - F_\mu(x,y) \leq \mu \cdot D \text{ for all } (x,y) \in \Delta_m \times \Delta_n.$$

Hence for all $(x,y) \in \Delta_m \times \Delta_n$

$$F(x_k,y_k) - F(x,y) \leq \mu \cdot D + F_\mu(x_k,y_k) - F_\mu(x,y) \leq \mu \cdot D + \frac{2 \cdot \|A\|^2 \cdot \|(x,y) - (x_0,y_0)\|^2}{\mu(k+1)(k+2)}.$$

Therefore, taking the minimum over $(x,y) \in \mathsf{Opt}$, we obtain

$$F(x_k,y_k) \leq \mu \cdot D + \frac{2 \cdot \|A\|^2 \cdot \mathsf{dist}(x_0,y_0)^2}{\mu(k+1)(k+2)} < \frac{\epsilon}{2} + \frac{4 \cdot \|A\|^2 \cdot D \cdot \mathsf{dist}(x_0,y_0)^2}{\epsilon \cdot k^2}. \tag{8}$$

In the last step we used $\mu = \frac{\epsilon}{2D}$. The bound (7) now readily follows from (8) □

Note that the vectors $\bar{u}$, $\bar{v}$ can be *any* vectors in Δ_m and Δ_n. In our implementation, we take these vectors to be those corresponding to a uniformly random strategy.

2.2 Iterated Smoothing Scheme for Matrix Games

We are now ready to present our main contribution. The new algorithm is a simple extension of Algorithm **smoothing**. At each iteration we call the basic smoothing subroutine with a target

accuracy. Between the iterations, we reduce the target accuracy by $\gamma > 1$. Consider the following iterated first-order method for minimizing $F(x, y)$.

iterated$(A, x_0, y_0, \gamma, \epsilon)$

1. Let $\epsilon_0 = F(x_0, y_0)$
2. For $i = 0, 1, \ldots$
   - $\epsilon_{i+1} = \frac{\epsilon_i}{\gamma}$
   - $(x_{i+1}, y_{i+1}) =$ **smoothing**$(A, x_i, y_i, \epsilon_{i+1})$
   - If $F(x_{i+1}, y_{i+1}) < \epsilon$ halt

While the modification to the algorithm is simple, it yields an exponential speedup with respect to reaching the target accuracy ϵ:

Theorem 2 *Each call to* **smoothing** *in Algorithm* **iterated** *halts in at most*

$$\frac{2\sqrt{2} \cdot \gamma \cdot \|A\| \cdot \sqrt{D}}{\delta(A)} \tag{9}$$

first-order iterations, where $\delta(A)$ is a positive condition measure of the matrix A.

Algorithm **iterated** *halts in at most*

$$\frac{\ln(2\|A\|/\epsilon)}{\ln(\gamma)}$$

outer iterations, that is, in at most

$$\frac{2\sqrt{2} \cdot \gamma \cdot \|A\| \cdot \ln(2\|A\|/\epsilon) \cdot \sqrt{D}}{\ln(\gamma) \cdot \delta(A)} \tag{10}$$

first-order iterations.

Proof. See Section 2.4 below. □

By setting $\gamma = e \approx 2.718...$ the bound (10) becomes

$$\frac{2\sqrt{2} \cdot e \cdot \|A\| \cdot \ln(2\|A\|/\epsilon) \cdot \sqrt{D}}{\delta(A)}.$$

It can be shown that this is the optimal setting of γ for the overall complexity bound in Theorem 2.

It is natural to ask whether the complexity of Algorithm **iterated** could be much worse than that of Algorithm **smoothing** when the condition measure $\delta(A)$ is close to zero. As Proposition 3 below states, Algorithm **iterated** converges in at most $\mathcal{O}(1/\epsilon)$ iterations regardless of $\delta(A)$. The dependence on ϵ in the same as that of Algorithm **smoothing**. Indeed, Proposition 3 is an easy consequence of Proposition 1. The specific complexity bound (12) therein is similar to (7).

Proposition 3 *For $i = 0, 1, \ldots,$ the i-th call to* **smoothing** *in Algorithm* **iterated** *halts in at most*

$$\frac{8 \cdot \|A\| \cdot D \cdot \gamma^{i+1}}{F(x_0, y_0)} \tag{11}$$

first-order iterations. Consequently, Algorithm **iterated** *halts in at most*

$$\frac{8 \cdot \gamma^2 \cdot \|A\| \cdot D}{(\gamma - 1) \cdot \epsilon} \tag{12}$$

first-order iterations.

PROOF. From Proposition 1, it follows that the i-th call to **smoothing** in Algorithm **iterated** halts in at most

$$\frac{2\sqrt{2} \cdot \|A\| \cdot \sqrt{D} \cdot \mathsf{dist}(x_i, y_i)}{\epsilon_{i+1}}$$

first-order iterations. Since $\epsilon_{i+1} = \epsilon_0/\gamma^{i+1} = F(x_0, y_0)/\gamma^{i+1}$ and $\mathsf{dist}(x_i, y_i) \leq 2\sqrt{2D}$, the bound (11) readily follows. Next, observe that Algorithm **iterated** will halt after at most N outer iterations, where N is such that

$$\epsilon_0/\gamma^N = \epsilon_N \leq \epsilon < \epsilon_{N-1} = \epsilon_0/\gamma^{N-1}.$$

Hence from (11), it follows that Algorithm **iterated** halts in at most

$$\sum_{i=0}^{N-1} \frac{8 \cdot \|A\| \cdot D}{\epsilon_0/\gamma^{i+1}} = \frac{8 \cdot \|A\| \cdot D}{\epsilon_0} \cdot \frac{\gamma(\gamma^N - 1)}{\gamma - 1} \leq \frac{8 \cdot \gamma^2 \cdot \|A\| \cdot D}{(\gamma - 1) \cdot \epsilon_0/\gamma^{N-1}} \leq \frac{8 \cdot \gamma^2 \cdot \|A\| \cdot D}{(\gamma - 1) \cdot \epsilon},$$

first-order iterations. □

2.3 The Condition Measure $\delta(A)$

We define the condition measure of a matrix A as

$$\delta(A) = \sup_{\delta} \left\{ \delta : \mathsf{dist}(x, y) \leq \frac{F(x, y)}{\delta} \quad \forall (x, y) \in \Delta_m \times \Delta_n \right\}.$$

Notice that $\delta(A)$ can be geometrically visualized as a measure of "steepness" of the function $F(x, y)$. The following technical lemma shows that $\delta(A) > 0$ for all A. In other words, the function F has a *sharp minimum.*

Lemma 4 *Assume $A \in \mathbb{R}^{m \times n}$ and F is as in* (4)*. There exists $\delta > 0$ such that*

$$\mathsf{dist}(x, y) \leq \frac{F(x, y)}{\delta} \text{ for all } (x, y) \in \Delta_m \times \Delta_n. \tag{13}$$

PROOF. Since the function $F : \Delta_m \times \Delta_n \to \mathbb{R}$ is polyhedral and its domain is a bounded polytope, its epigraph $\mathsf{epi}(F) = \{(x, y, t) : t \geq F(x, y),\ (x, y) \in \Delta_m \times \Delta_n\}$ is polyhedral. It thus follows that

$$\mathsf{epi}(F) = \mathsf{conv}\{(x_i, y_i, t_i) : i = 1, \ldots, M\} + \{0\} \times \{0\} \times [0, \infty)$$

for a finite set of points $(x_i, y_i, t_i) \in \Delta_m \times \Delta_n \times \mathbb{R}_+,\ i = 1, \ldots, M$. Therefore F can be expressed as

$$F(x, y) = \min \left\{ \sum_{i=1}^{M} t_i \lambda_i : \sum_{i=1}^{M} (x_i, y_i)\lambda_i = (x, y),\ \lambda \in \Delta_M \right\}. \tag{14}$$

Since $\min\{F(x, y) : (x, y) \in \Delta_m \times \Delta_n\} = 0$, we have $\min\{t_i, i = 1, \ldots, M\} = 0$. Without loss of generality assume $t_1 \geq t_2 \geq \cdots \geq t_N > 0 = t_{N+1} = \cdots = t_M$. We assume $N \geq 1$ as otherwise $\mathsf{Opt} = \Delta_m \times \Delta_n$ and (13) readily holds for any $\delta > 0$. Thus $\mathsf{Opt} = \mathsf{conv}\{(x_i, y_i) : i = N+1, \ldots, M\}$. Let

$$\begin{aligned} \delta &:= \frac{t_N}{\max\{\|(x_i, y_i) - (x, y)\| : i = 1, \ldots, N, (x, y) \in \mathsf{Opt}\}} \\ &= \frac{t_N}{\max\{\|(x_i, y_i) - (x_j, y_j)\| : i = 1, \ldots, N, j = N + 1, \ldots, M\}} \end{aligned}$$

We claim that δ satisfies (13). To prove this claim, let $(x, y) \in \Delta_m \times \Delta_n$ be any arbitrary point. We need to show that $\mathsf{dist}(x, y) \leq F(x, y)/\delta$. Assume $F(x, y) > 0$ as otherwise there is nothing to show. From (14) it follows that

$$(x, y) = \sum_{i=1}^{M} (x_i, y_i)\lambda_i,\ F(x, y) = \sum_{i=1}^{M} t_i \lambda_i = \sum_{i=1}^{N} t_i \lambda_i$$

for some $\lambda \in \Delta_M$. Let $\mu := \sum_{i=1}^{N} \lambda_i > 0$, and let $\tilde{\lambda} \in \Delta_N$ be the vector defined by putting $\tilde{\lambda}_i := \lambda_i/\mu,\ i = 1, \ldots, N$. In addition, let $(\hat{x}, \hat{y}) = \sum_{i=1}^{N} (x_i, y_i)\tilde{\lambda}_i = \sum_{i=1}^{N} (x_i, y_i)\lambda_i/\mu \in \Delta_m \times \Delta_n$, and $(\tilde{x}, \tilde{y}) \in \mathsf{Opt}$ be as follows

$$(\tilde{x}, \tilde{y}) := \begin{cases} \displaystyle\sum_{i=N+1}^{M} (x_i, y_i)\lambda_i/(1 - \mu) & \text{if} \quad \mu < 1 \\ (x_M, y_M) & \text{if} \quad \mu = 1 \end{cases}$$

Then $(x,y) = \mu(\hat{x},\hat{y}) + (1-\mu)(\tilde{x},\tilde{y})$ and consequently

$$\begin{aligned}
\|(x,y)-(\tilde{x},\tilde{y})\| &= \mu\|(\hat{x},\hat{y})-(\tilde{x},\tilde{y})\| \\
&= \mu\left\|\sum_{i=1}^{N}\tilde{\lambda}_i((x_i,y_i)-(\tilde{x},\tilde{y}))\right\| \\
&\leq \mu\sum_{i=1}^{N}\tilde{\lambda}_i\|(x_i,y_i)-(\tilde{x},\tilde{y})\| \\
&\leq \mu\max\{\|(x_i,y_i)-(x,y)\| : i=1,\ldots,N,\ (x,y)\in \mathsf{Opt}\} \\
&= \tfrac{\mu t_N}{\delta}.
\end{aligned}$$

To finish, observe that

$$F(x,y) = \sum_{i=1}^{N} t_i\lambda_i = \mu\sum_{i=1}^{N} t_i\tilde{\lambda}_i \geq \mu t_N.$$

Therefore,

$$\mathsf{dist}(x,y) \leq \|(x,y)-(\tilde{x},\tilde{y})\| \leq \mu t_N/\delta \leq F(x,y)/\delta.$$

□

2.4 Proof of Theorem 2

By construction, for each $i = 0, 1, \ldots$ we have

$$\mathsf{dist}(x_i,y_i) \leq \frac{\epsilon_i}{\delta(A)} = \frac{\gamma\cdot\epsilon_{i+1}}{\delta(A)}.$$

The iteration bound (9) then follows from Proposition 1.

After N outer iterations Algorithm **iterated** yields $(x_N,y_N) \in \Delta_m \times \Delta_n$ with

$$F(x_N,y_N) < \epsilon_N = \frac{F(x_0,y_0)}{\gamma^N} \leq \frac{2\|A\|}{\gamma^N}.$$

Thus, $F(x_N,y_N) < \epsilon$ for $N = \frac{\ln(2\|A\|/\epsilon)}{\ln(\gamma)}$ and (10) follows from (9). □

2.5 The Subroutine *smoothing* for Matrix Games

Algorithm **smoothing** involves fairly straightforward operations except for the solution of a subproblem of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|(u,v)\|^2 - (g,h)^{\mathrm{T}}(u,v) : (u,v)\in\Delta_m\times\Delta_n\right\}.$$

This problem in turn separates into two subproblems of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in \Delta_m\right\}. \tag{15}$$

Problem (15) can easily be solved via its Karush-Kuhn-Tucker optimality conditions:

$$u - g = \lambda \mathbf{1} + \mu,\ \lambda \in \mathbb{R},\ \mu \in \mathbb{R}^m_+,\ u \in \Delta_m,\ u^{\mathrm{T}}\mu = 0.$$

From these conditions it follows that the solution to (15) is given by

$$u_i = \max\{0, g_i - \lambda\},\ i = 1, \ldots, m,$$

where $\lambda \in \mathbb{R}$ is such that $\sum_{i=1}^m \max\{0, (g_i - \lambda)\} = 1$. This value of λ can be computed in $\mathcal{O}(m \ln(m))$ steps via a binary search in the sorted components of the vector g.

3 Smoothing Scheme for Sequential Games

Algorithm **iterated** and its complexity bound can be extended to sequential games. The Nash equilibrium problem of a two-player zero-sum sequential game with imperfect information can be formulated using the sequence form representation as the following saddle-point problem [7, 15, 18]:

$$\min_{x \in Q_1} \max_{y \in Q_2} x^{\mathrm{T}} A y = \max_{y \in Q_2} \min_{x \in Q_1} x^{\mathrm{T}} A y. \tag{16}$$

In this formulation, the vectors x and y represent the strategies of players 1 and 2 respectively. The strategy spaces $Q_i \subseteq \mathbb{R}^{S_i},\ i = 1, 2$ are the sets of realization plans of players 1 and 2 respectively, where S_i is the set of sequences of moves of player i.

The approach we presented for equilibrium finding in matrix games extends to sequential games in the natural way: recast (16) as a nonsmooth convex minimization problem

$$\min\{F(x, y) : (x, y) \in Q_1 \times Q_2\}, \tag{17}$$

for

$$F(x, y) = \max\{x^{\mathrm{T}} A v - u^{\mathrm{T}} A y : (u, v) \in Q_1 \times Q_2\}. \tag{18}$$

Algorithms **smoothing** and **iterated** extend to this context by replacing Δ_m and Δ_n with Q_1 and Q_2, respectively. Proposition 1 and Theorem 2 also extend in the same fashion. However, the critical subproblem in the subroutine **smoothing** becomes more challenging, as described next.

3.1 The Subroutine *smoothing* for Sequential Games

Here we describe how to solve each of the two argmin subproblems of **smoothing** in the sequential game case. Each of those two subproblems decomposes into two subproblems of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in Q\right\}, \tag{19}$$

where Q is a set of realization plans.

Our algorithm for this is a generalization of the solution approach described above for the case $Q = \Delta_k$. In order to describe it, we use some features of the sets of realization plans in the sequence form representation of sequential games. A detailed discussion of the sequence form can be found in [18]. Recall that an extensive form sequential game is given by a tree, payoffs at the leaves, chance moves, and information sets [14]. Each node in the tree determines a unique *sequence* of choices from the root to that node for each one of the players. Under the assumption of perfect recall, all nodes in an information set u of a player define the same sequence σ_u of choices.

Assume U is the set of information sets of a particular player. For each $u \in U$ let C_u denote the set of choices for that player. Then the set of sequences S of the player can be written as

$$S = \{\emptyset\} \cup \{\sigma_u c : u \in U,\, c \in C_u\}$$

where the notation $\sigma_u c$ denotes the sequence of moves σ_u followed by the move c. A *realization plan* for this player is a non-negative mapping $x : S \to \mathbb{R}$ that satisfies $x(\emptyset) = 1$, and

$$-x(\sigma_u) + \sum_{c \in C_u} x(\sigma_u c) = 0$$

for all $u \in U$. A realization plan x can be seen as an $|S|$-dimensional vector whose entries are indexed by the set of sequences S. Under this interpretation, $x(\sigma)$ is the component of x indexed by an element $\sigma \in S$.

It is immediate that the set of realization plans of the player as above, seen as $|S|$-dimensional vectors, can be written in the form

$$\{x \geq 0 : Ex = e\}$$

for some $(1 + |U|) \times |S|$ matrix E with entries $\{0, 1, -1\}$ and the $(1 + |U|)$-dimensional vector $e = (1, 0, \ldots, 0)^{\mathrm{T}}$. It also follows that sets of realization plans are *treeplexes.* A treeplex is a generalization of a simplex, and can be recursively defined as follows:

(C1) The empty set $\emptyset$ is a treeplex.

(C2) Assume $Q_j \subseteq \mathbb{R}^{d_j}$ for $j = 1, \ldots, k$ are treeplexes. Then the following set is a treeplex

$$\left\{ \left(u^0, u^1, \ldots, u^k\right) \in \mathbb{R}^{k+d_1+\cdots+d_k} : u^0 \in \Delta_k,\ u^j \in u^0_j \cdot Q_j,\ j = 1, \ldots, k \right\}.$$

(The operation $u^0_j \cdot Q_j$ multiplies all elements of Q_j by the j-th entry of u^0, that is, u^0_j.)

(C3) Assume $Q_j \subseteq \mathbb{R}^{d_j}$ for $j = 1, \ldots, k$ are treeplexes. Then the following set (their Cartesian product) is a treeplex

$$\left\{ \left(u^1, \ldots, u^k\right) \in \mathbb{R}^{d_1+\cdots+d_k} : u^j \in Q_j,\ j = 1, \ldots, k \right\}.$$

Note that any simplex is a treeplex: Δ_k is obtained by applying **(C2)** with $Q_j = \emptyset,\ j = 1, \ldots, k$.

A slightly different definition of treeplexes was given in [6]. It is easy to see that the two definitions are equivalent, but the above definition is better suited for the purposes of this paper.

Given a treeplex $Q \subseteq \mathbb{R}^d$ and a vector $g \in \mathbb{R}^d$, define the *value function* $v_{Q,g} : \mathbb{R}_+ \to \mathbb{R}$ as

$$v_{Q,g}(t) := \min \left\{ \frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in t \cdot Q \right\}.$$

Theorem 5 below shows that $v_{Q,g}$ is differentiable in $\mathbb{R}_+$ and its derivative is strictly increasing in $\mathbb{R}_+$. Let $\lambda_{Q,g} = v'_{Q,g}$ and let $\theta_{Q,g}$ be the inverse function of $\lambda_{Q,g}$. Since $\lambda_{Q,g}$ is strictly increasing in $\mathbb{R}_+$, its minimum value is $\lambda_{Q,g}(0)$. The function $\theta_{Q,g}$ can be defined in all of $\mathbb{R}$ by putting $\theta_{Q,g}(\lambda) := 0$ for all $\lambda \leq \lambda_{Q,g}(0)$. Finally, define the minimizer function $u_{Q,g} : \mathbb{R}_+ \to Q$ as

$$u_{Q,g}(t) := \operatorname{argmin} \left\{ \frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in t \cdot Q \right\}.$$

The recursive algorithm **TreeplexSubproblem** below computes the functions $v_{Q,g}, \lambda_{Q,g}, \theta_{Q,g}$, and $u_{Q,g}$ for any given treeplex Q. In particular, it computes the solution $u_{Q,g}(1)$ to the subproblem (19). The algorithm assumes that either Q is as in **(C2)** and $g = \left(g^0, g^1, \ldots, g^k\right) \in \mathbb{R}^{k+d_1+\cdots+d_k}$, or Q is as in **(C3)** and $g = \left(g^1, \ldots, g^k\right) \in \mathbb{R}^{d_1+\cdots+d_k}$.

TreeplexSubproblem(Q, g)

1. If Q is as in **(C2)** then

   (a) For $i = 1, \ldots, k$ let $\tilde{\lambda}_i : \mathbb{R}_+ \to \mathbb{R}$ and $\tilde{\theta}_i : \mathbb{R} \to \mathbb{R}_+$ be

   $$\tilde{\lambda}_i(t) := t - g^0_i + \lambda_{Q_i,g^i}(t), \qquad \tilde{\theta}_i := \tilde{\lambda}_i^{-1}.$$

   (b) Let $\theta_{Q,g} := \sum_{i=1}^k \tilde{\theta}_i$ and $\lambda_{Q,g} := \theta^{-1}_{Q,g}$

   (c) Let $u_{Q,g} : \mathbb{R}_+ \to Q$ be

   $$u_{Q,g}(t)^0_i := \tilde{\theta}_i(\lambda_{Q,g}(t))$$

and

$$u_{Q,g}(t)^i := u_{Q_i,g^i}\left(u_{Q,g}(t)_i^0\right)$$

for $i = 1, \ldots, k$.

2. If Q is as in **(C3)** then

   (a) Let $\lambda_{Q,g} := \sum_{i=1}^k \lambda_{Q_i,g^i}$ and $\theta_{Q,g} = \lambda_{Q,g}^{-1}$

   (b) Let $u_{Q,g} : \mathbb{R}_+ \to Q$ be

$$u_{Q,g}(t)^i := u_{Q_i,g^i}(t)$$

   for $i = 1, \ldots, k$.

While we presented Algorithm **TreeplexSubproblem** in recursive form for pedagogical reasons, for efficiency purposes we implemented it as a dynamic program. The implementation first performs a bottom-up pass that computes and stores the functions $\lambda_{Q,g}$. Subsequently a top-down pass computes the components of the minimizer $u_{Q,g}(t)$.

Theorem 5 *Algorithm* **TreeplexSubproblem** *is correct. In particular, the function $\lambda_{Q,g} = v'_{Q,g}$ exists and is piecewise linear. Furthermore, if Q is as in* **(C2)** *or is as in* **(C3)**, *then the total number of breakpoints $B(Q,g)$ of $\lambda_{Q,g}$ is at most*

$$\sum_{i=1}^k \max\{B(Q_i, g_i), 1\}.$$

If the breakpoints of λ_{Q_i,g^i} are available, then the breakpoints of $\lambda_{Q,g}$ can be constructed in

$$\mathcal{O}(B(Q,g)\ln(B(Q,g)))$$

steps, i.e., *this is the run time of Algorithm* **TreeplexSubproblem**.

Proof. First, assume that Q is as in **(C2)**. Then the value function $v_{Q,g}(t)$ can be written as

$$v_{Q,g}(t) = \min\left\{\frac{1}{2}\left\|u^0\right\|^2 - \left(g^0\right)^{\mathrm{T}} u^0 + \sum_{j=1}^k v_{Q_j,g^j}\left(u_j^0\right) : u^0 \in t \cdot \Delta_k\right\}. \tag{20}$$

This is a constrained optimization problem in the variables u^0. Its Karush-Kuhn-Tucker optimality conditions are

$$\begin{aligned} u_j^0 - g_j^0 + \lambda_{Q_j,g^j}\left(u_j^0\right) &= \lambda + \mu_j, \\ \lambda \in \mathbb{R},\ \mu &\in \mathbb{R}_+^k, \\ u^0 \in t \cdot \Delta_k,\ \ \mu^{\mathrm{T}} u^0 &= 0. \end{aligned} \tag{21}$$

In particular, the multiplier $\lambda \in \mathbb{R}$ is the unique solution to

$$\sum_{j=1}^{k} \max\{g_j^0 - \lambda_{Q_j,g^j}(u_j^0) + \lambda, 0\} = t. \tag{22}$$

Since this λ is unique, by basic differentiability properties from convex analysis (see, e.g., [5, Chapter D]), it follows that $v'_{Q,g}(t) = \lambda_{Q,g}(t)$ exists and is precisely the value of λ that solves (22). Furthermore, from (22) it readily follows that $\lambda_{Q,g}$ is strictly increasing. From the optimality conditions (21), we get $u_j^0 = \tilde{\theta}_j(\lambda),\ j = 1, \ldots, k$ for the functions $\tilde{\theta}_j$ constructed in step 1 of Algorithm **TreeplexSubproblem**. Hence

$$t = \sum_{j=1}^{k} u_j^0 = \sum_{j=1}^{k} \tilde{\theta}_j(\lambda).$$

Therefore, $\theta_{Q,g} = \sum_{j=1}^{k} \tilde{\theta}_j$. This shows the correctness of Steps 1.a and 1.b of Algorithm **TreeplexSubproblem**. Finally, the correctness of Step 1.c follows from (20) and (21).

On the other hand, if Q is as in **(C3)** then the value function $v_{Q,g}(t)$ can be decoupled as follows

$$v_{Q,g}(t) = \sum_{i=1}^{k} \min\left\{ \frac{1}{2} \left\|u^i\right\|^2 - \left(g^i\right)^{\mathrm{T}} u^i : u^i \in t \cdot Q_i \right\} = \sum_{i=1}^{k} v_{Q_i,g^i}(t). \tag{23}$$

This yields the correctness of Steps 2.a and 2.b.

The piecewise linearity of $\lambda_{Q,g}$ readily follows from the correctness of Algorithm **TreeplexSubproblem**. As for the number of breakpoints, consider first the case when Q is as in **(C2)**. Observe that the number of breakpoints of $\tilde{\theta}_i$ is the same as that of $\tilde{\lambda}_i$, which is either the same as that of λ_{Q_i,g^i} (if $Q_i \neq \emptyset$) or 1 (if $Q_i = \emptyset$). To get the bound on $B(Q,g)$, note that the total number of breakpoints of $\lambda_{Q,g}$ is the same as that of $\theta_{Q,g}$, which is at most the sum of the number of breakpoints of all $\tilde{\theta}_i,\ i = 1, \ldots, k$. The breakpoints of $\theta_{Q,g}$ can be obtained by sorting the breakpoints of all of the θ_i together. This can be done in $\mathcal{O}(B(Q,g)\ln(B(Q,g)))$ steps. In the case when Q is as in **(C3)** the number of breakpoints of $\lambda_{Q,g}$ is at most the sum of the number of breakpoints of all $\lambda_i,\ i = 1, \ldots, k$. The breakpoints of $\lambda_{Q,g}$ can be obtained by sorting the breakpoints of all of the λ_i together. This can be done in $\mathcal{O}(B(Q,g)\ln(B(Q,g)))$ steps.

□

3.2 *TreeplexSubproblem* Example

We include a simple example to illustrate Algorithm **TreeplexSubproblem**, as well as the use of our recursive definition of treeplexes. For simplicity of the example, let $Q_1 = \Delta_2$ and $Q_2 = \emptyset$. Then applying the recursive definition of treeplexes, **(C2)**, we get that Q is the set

$$\left\{ \left(u^0, u^1\right) : u^0 \in \Delta_2, u^1 \in u_1^0 \cdot Q_1 \right\}.$$

In a sequential game corresponding to this set of realization plans, the player first chooses among actions a_1^0 and a_2^0, with probabilities u_1^0 and u_2^0, respectively, and conditioned on choosing action a_1^0, the player may be asked to choose among actions a_1^1 and a_2^1, which are played with probabilities u_1^1/u_1^0 and u_2^1/u_1^0, respectively. (Note that there is no u^2 in the above equation since $Q_2 = \emptyset$, *i.e.*, if the agent plays a_2^0, he will have no further actions.) The treeplex Q can also be written as $\{x \geq 0 : Ex = e\}$ for

$$E = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 1 \end{bmatrix}, \; e = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Now, given input vector $g^1 \in \mathbb{R}^2$, we define the value function for Q_1 as

$$v_{Q_1,g^1}\left(u_1^0\right) := \min\left\{\frac{1}{2}\left\|u^1\right\|^2 - \left(g^1\right)^{\mathrm{T}} u^1 : u^1 \in u_1^0 \cdot Q_1\right\}.$$

Then, as was done in the proof of Theorem 5, we can write the value function for Q as

$$v_{Q,g}(t) := \min\left\{\frac{1}{2}\left\|u^0\right\|^2 - \left(g^0\right)^{\mathrm{T}} u^0 + v_{Q_1,g^1}\left(u_1^0\right) : u^0 \in t \cdot \Delta_2\right\}$$

for $g = \left(g^0, g^1\right) \in \mathbb{R}^4$. This is the problem that **TreeplexSubproblem** is trying to solve in our example.

We first demonstrate the algorithm as it executes **TreeplexSubproblem**(Q_1, g^1), *i.e.*, the bottom of the recursion. Since Q_1 has no "sub-treeplexes", we have

$$\begin{aligned} \tilde{\lambda}_1(t) &:= t - g_1^1, \\ \tilde{\lambda}_2(t) &:= t - g_2^1. \end{aligned}$$

The equations are graphed on the left in Figure 1. Step 1 of the algorithm constructs the $\tilde{\theta}_i$ functions to be the inverse of the $\tilde{\lambda}_i(t)$ functions. Once these inverses are computed, Step 2 of the algorithm adds the $\tilde{\theta}_i$ functions to obtain the θ_{Q_1,g^1} function, which is in turn inverted to construct the λ_{Q_1,g^1} function. This process of inverting, adding, and inverting again has a more intuitive description in the form of a "horizontal addition" operation on the $\tilde{\lambda}_i$ functions. In such an operation, two functions are added as normal, except we flip the axis of the graph so that the x-axis and y-axis are switched. This operation is illustrated in Figure 1. The graph on the left in Figure 1 contains the $\tilde{\lambda}_i(t)$ functions. These functions are "horizontally added" to obtain λ_{Q_1,g^1} on the right in Figure 1.

At non-bottom parts of the recursion ($\lambda_{Q,g}$ in our example) we construct the piecewise linear functions similarly, except that we have to take into account subsequent actions using the piecewise linear functions (function $\lambda_{Q_1,g^1}(t)$ in our example) already computed for the nodes below the

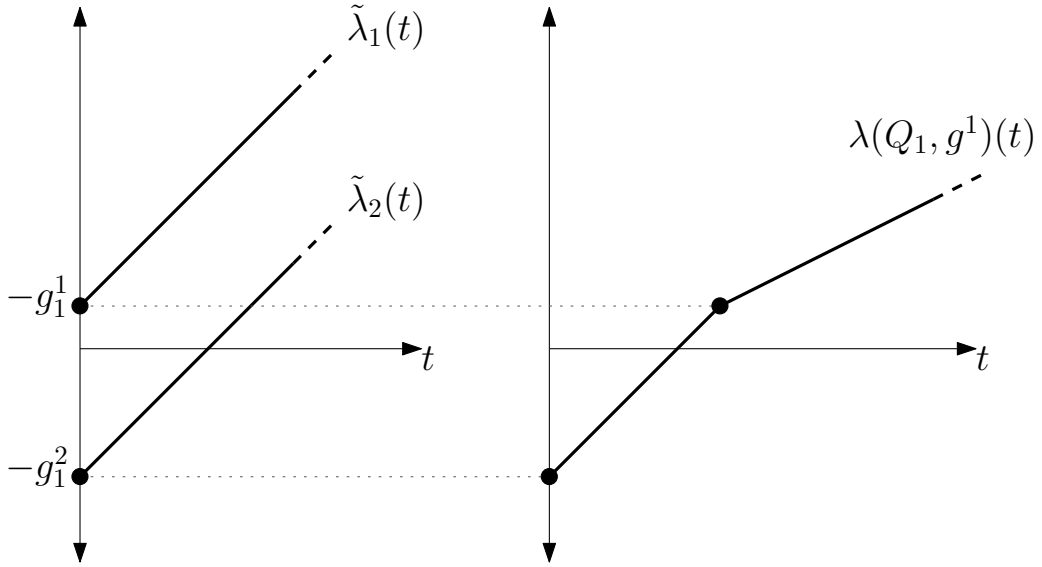

Figure 1: An illustration of Steps 1 and 2 of Algorithm **TreeplexSubproblem** applied to Q_1 and g^1.

current node in the recursion tree:

$$\begin{aligned}\tilde{\lambda}_1(t) &:= t - g_1^0 + \lambda_{Q_1,g^1}(t),\\ \tilde{\lambda}_2(t) &:= t - g_2^0\end{aligned}$$

The "horizontal addition" operation for this case is depicted in Figure 2.

Since $\lambda_{Q_1,g^1}(t)$ and $\lambda_{Q,g}$ are piecewise linear, our implementation simply represents them as a set of breakpoints, which are represented by solid circles in Figures 1 and 2. Given that we have finally constructed the piecewise linear function at the root, we can determine the values of u^0 and u^1 that solve the optimization problem in (19) as described in Step 3 of Algorithm **TreeplexSubproblem**. Specifically, we first take $t = 1$ and solve for u^0. To do this, we evaluate $\lambda_{Q,g}(1)$. Then we find the values of u_1^0 and u_2^0 such that

$$\begin{aligned}\tilde{\lambda}_1(u_1^0) &= \lambda_{Q,g}(1),\\ \tilde{\lambda}_2(u_2^0) &= \lambda_{Q,g}(1).\end{aligned}$$

This last operation is straightforward since the functions in question are monotonically increasing and piecewise linear.

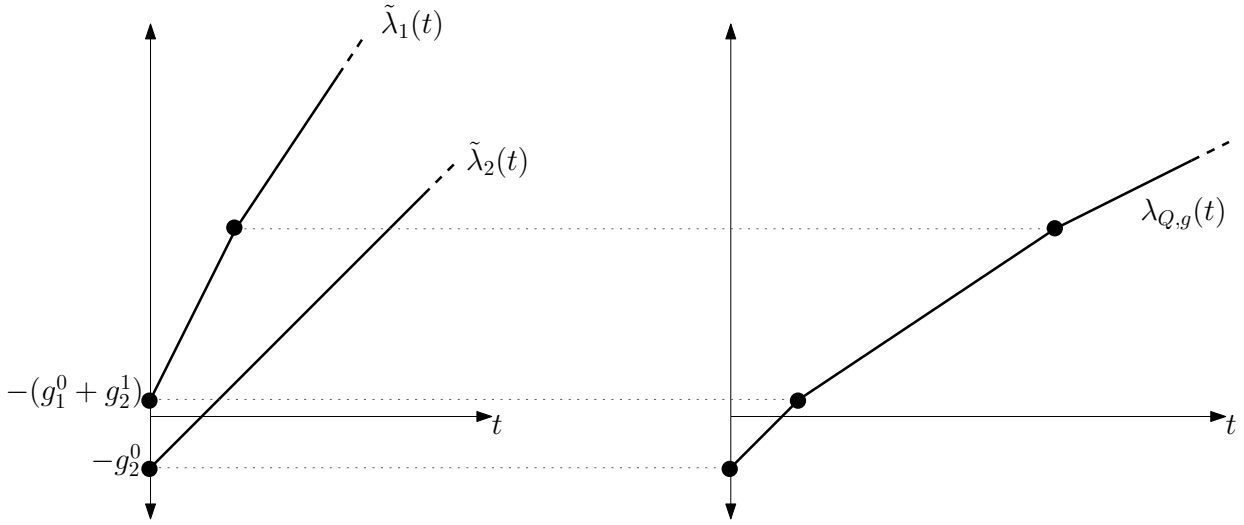

Figure 2: An illustration of Steps 1 and 2 of Algorithm **TreeplexSubproblem** applied to Q and g.

Once we have computed u_1^0, we can evaluate $\lambda_{Q_1,g^1}(u_1^0)$ and find u_1^1 and u_2^1 that satisfy

$$\begin{aligned} \tilde{\lambda}_1(u_1^1) &= \lambda_{Q_1,g^1}(u_1^0), \\ \tilde{\lambda}_2(u_2^1) &= \lambda_{Q_1,g^1}(u_1^0). \end{aligned}$$

Again, this operation is easy due to the functions being monotonically increasing and piecewise linear. This completes the execution of Algorithm **TreeplexSubproblem** on our example.

4 Computational Experiments

In this section we report on our computational experience with our new method. We compared our **iterated** algorithm against the basic **smoothing** algorithm. We tested the algorithms on matrix games as well as sequential games.

For matrix games, we generated 100 games of three different sizes where the payoffs are drawn uniformly at random from the interval $[-1, 1]$. This is the same instance generator as in Nesterov's [13] experiments.

For sequential games, we used the benchmark instances `81`, `10k`, and `160k` which have been used in the past for benchmarking equilibrium-finding algorithms for sequential imperfect-information games [6]. These instances are all abstracted versions of Rhode Island Hold'em poker [16], and they are named to indicate the number of variables in each player's strategy vector.

Figure 3 displays the results. Each graph is plotted with ϵ on the x-axis (using an inverse logarithmic scale). The y-axis is the number of seconds (using a logarithmic scale) needed to find

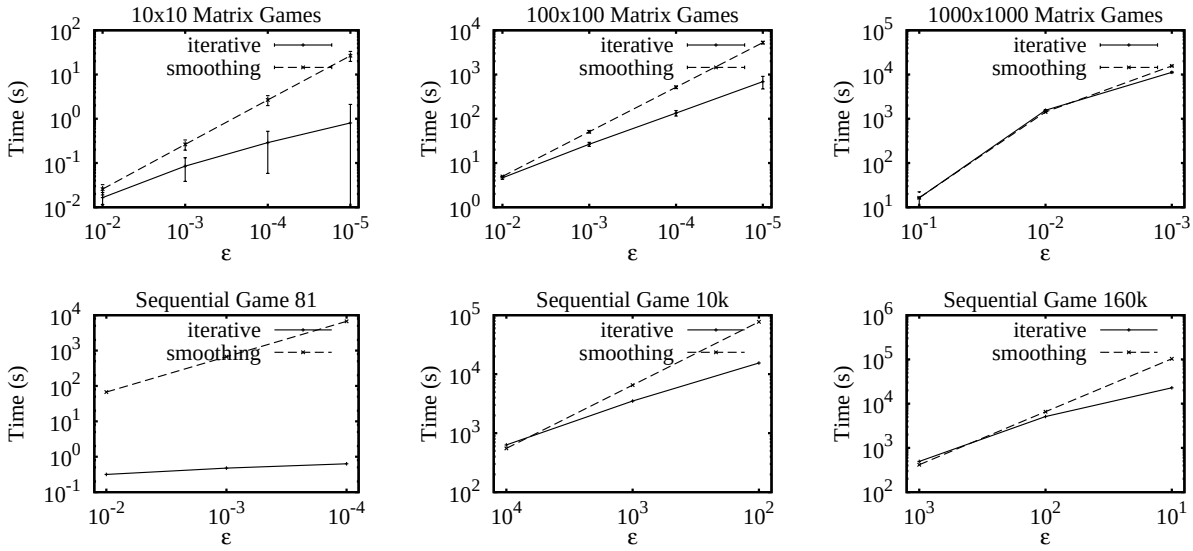

Figure 3: Time taken (in seconds) for each algorithm to find an ϵ-equilibrium for various values of ϵ.

ϵ-equilibrium for the given ϵ. The matrix game graphs also display the standard deviation.

In all settings we see that our **iterated** algorithm indeed outperforms the **smoothing** algorithm (as the worst-case complexity results would suggest). In fact, as the desired accuracy increases, the relative speed difference also increases.

We also tested a version of our iterated smoothing algorithm that used the Lan *et al.* [8] variant of Nesterov's algorithm for smooth optimization (subroutine **smoothing**). The only difference in that subroutine is in Step 2(d) of **smoothing**. Although the guarantee of Theorem 2 does not hold, that version performed almost identically.

5 Conclusions

We presented a new algorithm for finding ϵ-equilibria in two-person zero-sum games. It applies to both matrix and sequential games. The algorithm has convergence rate $\mathcal{O}\left(\frac{\|A\|}{\delta(A)}\ln(1/\epsilon)\right)$, where $\delta(A)$ is a condition measure of the matrix A. In terms of the dependence on ϵ, this matches the complexity of interior-point methods and is exponentially faster than prior first-order methods. Furthermore, our algorithm, like other first-order methods, uses dramatically less memory than interior-point methods, indicating that it can scale to games much larger than previously possible.

Our scheme supplements Nesterov's first-order smoothing method with an outer loop that lowers the target ϵ between iterations (this target affects the amount of smoothing in the inner loop). We find it surprising that such a simple modification yields an exponential speed improvement, and

wonder whether a similar phenomenon might occur in other optimization settings as well. Finally, computational experiments both in matrix games and sequential games show that a significant speed improvement is obtained in practice as well, and the relative speed improvement increases with the desired accuracy (as suggested by the complexity bounds).

Some recent work by Mordukhovich, Peña, and Roshchina [10] gives additional insight into the condition measure $\delta(A)$. In particular, this article gives an explicit characterization of $\delta(A)$ by relying on tools from variational analysis. This may lead to further understanding of the condition measure $\delta(A)$ such as lower bounds on $\delta(A)$ for particular classes of matrices A, or lower bounds on the expected value of $\delta(A)$ for randomly chosen A.

6 Acknowledgments

Research for the first and third authors was supported by the National Science Foundation under grants IIS-0427858, IIS-0905390, and IIS-0964579.

Research for the second author was supported by the National Science Foundation under grants CMMI-0855707 and CCF-0830533.

References

[1] D. Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems*. Kluwer International Series, Dordrecht, 2002.

[2] G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[3] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 50–57, Vancouver, Canada, 2007. AAAI Press.

[4] J.-L. Goffin. On the convergence rate of subgradient optimization methods. *Mathematical Programming*, 13:329–347, 1977.

[5] J. Hirriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer-Verlag, Berlin, 2001.

[6] S. Hoda, A. Gilpin, J. Peña, and T. Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010.

[7] Daphne Koller and Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, October 1992.

[8] Guanghui Lan, Zhaosong Lu, and Renato D. C. Monteiro. Primal-dual first-order methods with $O(1/\epsilon)$ iteration-complexity for cone programming. *To Appear in Mathematical Programming*, 2010.

[9] H. Brendan McMahan and Geoffrey J. Gordon. A fast bundle-based anytime algorithm for poker and other convex games. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS)*, San Juan, Puerto Rico, 2007.

[10] B. Mordukhovich, J. Peña, and V. Roshchina. Computation of a condition measure of a smoothing algorithm for matrix games. *To Appear in SIAM Journal on Optimization*, 2010.

[11] Yurii Nesterov. A method for unconstrained convex minimization problem with rate of convergence $O(1/k^2)$. *Doklady AN SSSR*, 269:543–547, 1983. Translated to English as *Soviet Math. Docl.*

[12] Yurii Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal of Optimization*, 16(1):235–249, 2005.

[13] Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[14] M. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, Cambridge, MA, 1994.

[15] I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

[16] Jiefu Shi and Michael Littman. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, pages 333–345, London, UK, 2002. Springer-Verlag.

[17] Alexander J. Smola, S. V. N. Vishwanathan, and Quoc Le. Bundle methods for machine learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2007.

[18] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[19] S. J. Wright. *Primal-Dual Interior-Point Methods.* SIAM, Philadelphia, PA, 1997.

[20] Y. Ye, M. Todd, and S. Mizuno. An $o(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.

[21] Martin Zinkevich, Michael Bowling, and Neil Burch. A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Vancouver, Canada, 2007.

[22] Martin Zinkevich, Michael Bowling, Michael Johanson, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2007.