A Whole New Ball Game: A Primal Accelerated Method for Matrix Games and Minimizing the Maximum of Smooth Functions

Yair Carmon* Arun Jambulapati† Yujia Jin‡ Aaron Sidford§

Abstract

We design algorithms for minimizing $\max_{i\in[n]} f_i(x)$ over a d-dimensional Euclidean or simplex domain. When each f_i is 1-Lipschitz and 1-smooth, our method computes an ϵ-approximate solution using $\widetilde{O}(n\epsilon^{-1/3}+\epsilon^{-2})$ gradient and function evaluations, and $\widetilde{O}(n\epsilon^{-4/3})$ additional runtime. For large n, our evaluation complexity is optimal up to polylogarithmic factors. In the special case where each f_i is linear—which corresponds to finding a near-optimal primal strategy in a matrix game—our method finds an ϵ-approximate solution in runtime $\widetilde{O}(n(d/\epsilon)^{2/3}+nd+d\epsilon^{-2})$. For $n>d$ and $\epsilon=1/\sqrt{n}$ this improves over all existing first-order methods. When additionally $d=\omega(n^{8/11})$ our runtime also improves over all known interior point methods.

Our algorithm combines three novel primitives: (1) A dynamic data structure which enables efficient stochastic gradient estimation in small ℓ_2 or ℓ_1 balls. (2) A mirror descent algorithm tailored to our data structure implementing an oracle which minimizes the objective over these balls. (3) A simple ball oracle acceleration framework suitable for non-Euclidean geometry.

1 Introduction

Consider the optimization problem

$$\underset{x\in\mathcal{X}}{\text{minimize}}\left\{\max_{i\in[n]} a_i^\top x=\max_{y\in\Delta^n} x^\top Ay\right\}, \tag{1.1}$$

where $\mathcal{X}\subset\mathbb{R}^d$ is closed and convex, Δ^n is the probability simplex in n dimensions, and $A\in\mathbb{R}^{d\times n}$ has columns $a_1,\ldots,a_n$. We consider two settings of $\mathcal{X}$: (1) the ℓ_1 setting where $\mathcal{X}\subseteq\Delta^n$ and we measure distance with the 1-norm, and (2) the ℓ_2 setting where $\mathcal{X}$ is a subset of the unit Euclidean ball and we measure distance with the Euclidean norm. The first setting encompasses finding an optimal strategy for one side of a matrix game, which is sufficient for linear programming [24, 1]. The second setting includes important problems in machine learning and computational geometry: hard-margin support vector machines [41] and minimum enclosing and maximum inscribed ball [21].

Due to its fundamental nature, many algorithms have been developed to solve (1.1). The frontier of the best performing algorithms comprises efficiently-implemented second-order interior point methods [22, 64] and stochastic first-order methods [28, 21, 13]. We are interested in methods of the second type, which currently obtain preferable runtimes as we fix the solution accuracy and let the problem dimensions n and d grow. The best existing methods of this type jointly evolve the primal x and dual y variables via stochastic mirror descent; it not clear if additional runtime improvements are possible with this approach.

In this work we adopt a different approach, and design a primal stochastic first-order method that evolves the variable x by directly sampling from an (approximate) best-response distribution y at each step.[1] Our method

*Tel Aviv University, ycarmon@tauex.tau.ac.il.
†University of Washington, jmblpati@uw.edu.
‡Stanford University, yujiajin@stanford.edu.
§Stanford University, sidford@stanford.edu.

The "ball" in the title refers to ball oracle acceleration [14] at the heart of our results; no balls are placed into bins in this paper.

This paper is available as a preprint at https://arxiv.org/abs/2311.10886.

[1] It is not clear whether our method can efficiently extract the solution to the dual problem $\text{maximize}_{y\in\Delta^n}\min_{x\in\mathcal{X}} x^\top Ay$ without simply swapping the role of y and x. Nevertheless, in many applications finding an approximately-optimal x suffices.

solves the more general problem

$$\underset{x\in\mathcal{X}}{\text{minimize}}\left\{ f_{\max}(x) \coloneqq \max_{i\in[n]} f_i(x) = \max_{y\in\Delta^n} \sum_{i\in[n]} y_i f_i(x) \right\}, \tag{1.2}$$

where $f_1, \ldots, f_n$ are convex, L_f-Lipschitz, and L_g-smooth with respect to the norm of interest. The problem (1.1) corresponds to $f_i(x) = a_i^\top x$ and $L_g = 0$.

Method	$f_i, \nabla f_i$ evaluation complexity	Additional runtime	Simplex guarantees?
Subgradient method	$n\left(\frac{L_f}{\epsilon}\right)^2$	-	✔
AGD on softmax [47]	$n\left(\frac{L_f}{\epsilon}\right)$	-	✔
"Thinking inside the ball" [16]	$n\left(\frac{L_f}{\epsilon}\right)^{2/3} + \sqrt{n}\left(\frac{L_f}{\epsilon}\right)$	-	✗
AGD on linearization [49, 16]	$n\left(\frac{L_g}{\epsilon}\right)^{1/2}$	$\sqrt{nd(n+d)}\frac{L_f\sqrt{L_g}}{\epsilon^{3/2}}$	✗
Proposed method	$n\left(\frac{L_g}{\epsilon}\right)^{1/3} + \left(\frac{L_f}{\epsilon}\right)^2$	$n\frac{L_f^2}{L_g^{2/3}\epsilon^{4/3}}$	✔
Lower bound [16]	$n\left(\frac{L_g}{\epsilon}\right)^{1/3} + \sqrt{n}\left(\frac{L_g}{\epsilon}\right)^{1/2}$	N/A	✗

Table 1. Complexity guarantees for solving the problem (1.2) to ϵ accuracy. Parameters n and d denote the number of functions and domain dimensions, respectively, while L_f and L_g are the respective Lipschitz constants of f_i and ∇f_i. Expressions in the table omit constant and polylogarithmic factors. We assume that each f_i and ∇f_i evaluation takes time $\Omega(d)$ so that the "additional runtime" column only includes terms that are not dominated by d times the evaluation complexity. For simplicity, we also assume that $\epsilon \le L_g \le L_f^2/\epsilon$. The final column indicates whether the method has proven guarantees for the ℓ_1/simplex setting.

Method	Runtime for general parameters	Runtime for $n > d$ and $\epsilon = \frac{1}{\sqrt{n}}$
Stochastic primal-dual [28, 21]	$(n+d)\epsilon^{-2}$	n^2
Exact gradient primal-dual [45, 48]	$nd\epsilon^{-1}$	$n^{3/2}d$
Variance-reduced primal-dual [13]	$nd + \sqrt{nd(n+d)}\epsilon^{-1}$	$n^{3/2}d^{1/2}$
Proposed method	$nd + n(d/\epsilon)^{2/3} + d\epsilon^{-2}$	$n^{4/3}d^{2/3}$
Interior point [resp., 22, 64] †	$\max\{n,d\}^\omega$ $nd + \min\{n,d\}^{5/2}$	n^ω $nd + d^{5/2}$

Table 2. Runtime bounds for solving the problem (1.1) to ϵ accuracy, omitting constant and polylogarithmic factors. The bounds assume a unit Lipschitz constant, i.e., $\|a_i\|_* \le 1$ for all i, where the dual norm $\|\cdot\|_*$ is the ∞-norm in the ℓ_1 setting and the 2-norm ins the ℓ_2 setting. †To our knowledge the runtime bound $nd + \min\{n,d\}^{5/2}$ is proven only in the ℓ_1 setting.

Our methods builds upon previous work [16, 3, 11] that develop *ball oracles* which approximately minimize $f_{\max}$ in a small ball around a reference point, and then apply *ball oracle acceleration* [14, 17] to globally minimize the objective in a small number of ball oracle calls. These methods have two key shortcomings that prevent them from providing better runtimes for matrix games: (1) the ball oracles they implement have too small ball radii and (2) they do not apply to ℓ_1 geometry. This work overcomes the first shortcoming by designing data structures that, using sketching and sampling techniques, maintain linear approximations of the functions $\{f_i\}$ which facilitate efficient gradient estimation at larger distance from the reference point. To overcome the second challenge we redesign the ball oracle acceleration framework using a novel accelerated proximal point method formulation,

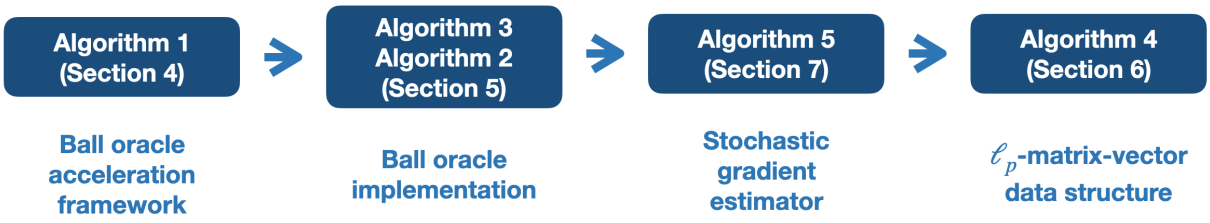

Figure 1: A diagram of the main components of our algorithm and their location in the paper.

and implement an approximate non-Euclidean ball oracle using a careful mirror descent scheme that provides a fine-grained control of the amount of iterate movement which our data structures require.

Tables 1 and 2 summarize the complexity guarantees of our method and compare them to prior work. We measure complexity as either the runtime or the number of evaluations of $f_i(x)$ and $\nabla f_i(x)$ (for some $x \in \mathcal{X}$ and $i \in [n]$) required to produce x such that $\mathbb{E} f_{\max}(x) - \min_{x_\star \in \mathcal{X}} f_{\max}(x_\star) \leq \epsilon$. For the problem (1.2) in the regime where n is large (e.g., $n \geq L_f^2 \epsilon^{-2}$) we obtain the optimal evaluation complexity with a modest additional computational cost due to our data structures, which becomes negligible as d grows. For problem (1.1), in the regime $d < \min\{\epsilon^{-2}, n\epsilon^2\}$ (which implies $d < n$) our bounds improve on all previous first-order methods. This regime includes $\epsilon = \frac{1}{\sqrt{n}}$, which is standard for empirical risk minimization problems, where statistical errors are typically also of order $\frac{1}{\sqrt{n}}$. Since we consider maximum (rather that mean) risk minimization, statistical errors (if they exist) will likely be higher for the problems we study. Additionally, our runtime improves over all known methods (including interior point methods) when n is not too much larger than d and ϵ lies in some range, namely $d < n < d^{3/2}$, $\epsilon \leq \frac{1}{\sqrt{d}}$, and $\epsilon > \max\{\frac{1}{d^{3/4}}, \frac{n^{3/2}}{d^{11/4}}, \frac{d^{1/2}}{n}\}$. For $\epsilon = \frac{1}{\sqrt{n}}$ we improve over all methods when $n^{8/11} < d < n$.

Our results also directly lead to an algorithm for finding the minimum Euclidean ball enclosing points $a_1, \ldots, a_n \in \mathbb{R}^d$, a fundamental problem in computational geometry [58, 21]. Our algorithm finds an ϵ-accurate solution in $\widetilde{O}(nd + d/\epsilon + nd^{2/3}\epsilon^{-1/3})$ time, while the previous best known runtime obtained by first-order methods is $\widetilde{O}(nd + nd^{1/2}\epsilon^{-1/2})$ [2, 15]. This is an improvement for a range of parameter values including $n > 1/\epsilon > d$.

Paper organization (see also Figure 1). In Section 1.1 we discuss related work. In Section 2 we provide a detailed overview of our key technical contributions. Section 3 introduces the general notation and conventions of the paper. In Section 4, we describe the main acceleration framework building on a ball-restricted proximal oracle, followed by the implementation of restricted oracle in Section 5. In Section 6, we build the main data structure used for ℓ_p-matrix-vector maintenance, which we then use to build an efficient stochastic gradient estimator in Section 7. In Section 8, we combine our developments and obtain guarantees for solving problem (1.2) and, as special cases, problem (1.1) and minimum enclosing ball.

1.1 Related work.

We now review several additional closely related lines of research.

Minimizing the maximum of linear functions. Research on algorithms for solving problems of the form (1.1), particularly in the context of linear programming, has a long and celebrated history in computer science [24]. The best existing methods fall on a spectrum of trade-offs between per-iteration cost and number of iterations. At one end of the spectrum lie second-order, interior-point methods [33, 51], whose iterations are expensive (usually requiring a linear system solution) but the number of iterations depends only logarithmically on the desired accuracy ϵ^{-1}; recent years saw much progress at making the iterations of these methods more efficient [e.g., 39, 22, 60, 63, 64, 61, 32]. Next come first-order methods that use exact gradients [e.g., 47, 45, 48] whose per-iteration cost is linear in the problem size, but whose iteration complexity typically scales as ϵ^{-1}. Finally, at the other end of the spectrum are stochastic first-order methods [e.g., 28, 21] whose per iteration cost is sublinear in the problem size—and sometimes even near-constant [15, 66]—but whose iteration complexity typically scales as ϵ^{-2}. In addition, variance reduction techniques [e.g., 5, 13, 15, 56, 57] use a mix of exact and stochastic gradient computation to obtain a faster rate of convergence in terms of ϵ while maintaining a sublinear per-iteration cost.

It is possible to view our ball oracle approach as a hybrid of stochastic and exact gradient queries, though the

way we leverage the exact gradient queries is quite different from variance reduction: we query exact gradients to increase the efficiency of nearby stochastic gradient estimates, while variance reduction methods seek to make them more accurate. Carmon et al. [16] (discussed at length in the following section) combine a ball oracle and variance reduction for minimizing the maximum of Lipschitz, slightly smooth functions. However, to do so they rely on an "exponentiated softmax" technique that is not compatible with the larger balls we consider in this paper. Enhancing our method using variance reduction is a promising direction for future work.

Minimizing the maximum of general convex functions. The general problem (1.2) has seen less research than the matrix games problem (1.1). The exact-gradient first order methods mentioned above [e.g., 47, 45, 48] also apply in the general case, and Nesterov [49, Section 2.3.1] shows how to reduce the general cases to a sequence of matrix games. However, stochastic gradient methods typically exploit the matrix structure in (1.1) and do not extend to the general case. Indeed, stochastic methods for the problem (1.2) typically have high variance gradient estimators, leading to an iteration count that depends on the number of functions n [44, 54, 46, 11]. The work [16] made significant progress in reducing the number of full-data passes required to solve the problem (1.2), and we improve it further to obtain (for large n) the optimal number of data passes for smooth problems.

Accelerated approximate proximal point methods. The accelerated proximal point method [29, 53] is a powerful and versatile building block for convex optimization algorithms, owing to the fact that the proximal point operation admits several approximate solution criteria that preserve the accelerated rate of convergence [26, 40, 18]. In particular, the approximate solution notion due to Monteiro and Svaiter [43] has led to a plethora of accelerated optimization methods [e.g., 27, 9, 31, 10, 8, 34] including the ball oracle acceleration framework [14, 16, 3, 11] at the core of our algorithm. We contribute to this line of research by designing a new approximate accelerate proximal point method that is suitable for non-Euclidean geometry and allows efficient oracle implementation using stochastic gradient methods; our technique also borrows the momentum damping technique from [17] for improving the simplicity and efficiency of the Monteiro-Svaiter method.

Our acceleration scheme also bears a strong resemblance to gradient sliding [35, 59, 36]: both techniques efficiently approximate the accelerated proximal point method by making use of both the averaged and final iterates of stochastic gradient descent. Since our method is based on a simple approximation condition for an exact proximal point problem, it provides insight into the efficacy of this approach.

Data structures for optimization. Optimization algorithms often rely on data structures for leveraging iterate sparsity and efficiently computing projections [38, 55, 25, 15]. However, randomized data structures—such as the matrix-vector maintainer we employ—are notoriously difficult to use in the context of optimization, since the iterative nature of the algorithm could make the sequence of data structure queries non-oblivious, thus invalidating the data structure's guarantees. We address this difficulty using rejection sampling, which ensures that the distribution of consecutive queries is the same regardless of the data structure's random state.

Our matrix-vector maintenance data structure is closely related to data structures designed in recent works on efficient interior point methods for linear programming [63, 62, 64], e.g., the "vector maintenance data structure" in [63]. The interior point methods using these data structures also take care to ensure that their queries remain oblivious, though not always via rejection sampling. Similar to our data structure, the ones in [63, 62, 64] also maintain an approximation to the products of a sequence of query vector with a given matrix, and they use a linear sketch similar to the one we use for the Euclidean case (but not the ℓ_1 case). Our data structure differs in the type of approximation maintained, the norms considered, and the assumptions on the query sequence. Moreover, our technique of supporting a long query sequence by instantiating multiple simpler data structures at different scales is well known [see, e.g., 4].

2 Technical overview

In this section we provide a detailed overview of our technical contribution. Section 2.1 describes the initial setup proposed in [16, 3]. In Section 2.2 we explain how we use linear approximations and data structures to increase the size of the ball for which we can implement an optimization oracle. Then, in Section 2.3 we explain how to extend ball oracle acceleration to non-Euclidean geometry, in Section 2.4 we describe the ball oracle implementation, and in Section 2.5 we put the components of our algorithm together and derive its complexity bounds.

2.1 Preliminaries. To begin the technical exposition, we first explain the key components of the "thinking inside the ball" approach [16, 3] to solving the problem (1.2), which we build upon to obtain our results. The first

step at tackling the problem is the standard "softmax" trick of smoothing the maximum operation by considering

$$f_{\text{smax}}(x) \coloneqq \max_{y \in \Delta^n} \left\{ \sum_{i \in [n]} [y_i f_i(x) - \epsilon' y_i \log y_i] \right\} = \epsilon' \log \left(\sum_{i \in [n]} e^{f_i(x)/\epsilon'} \right), \text{ with } \epsilon' = \frac{\epsilon}{2 \log n}, \tag{2.3}$$

which is a uniform $\frac{\epsilon}{2}$-approximation to $f_{\max}$ and therefore minimizing it to accuracy $\frac{\epsilon}{2}$ solves the problem (1.2) to accuracy ϵ.

Next, we design an oracle that approximately minimizes f_{smax} in a ball of radius r around a query point $y \in \mathcal{X}$. Roughly speaking, the implementation consists of stochastic gradient descent (SGD) with an unbiased estimator for $\nabla f_{\text{smax}}(x) = \mathbb{E}_{i \sim e^{f_i(x)/\epsilon'}} \nabla f_i(x)$. Naively computing the distribution proportional to $e^{f_i(x)/\epsilon'}$ requires n function/gradient evaluations, which is as expensive as computing ∇f_{smax} exactly. Instead, the estimator proposed in [3] uses rejection sampling to efficiently draw $i \sim e^{f_i(x)/\epsilon'}$, and then returns $\nabla f_i(x)$. Given a query point x and a reference point y, the rejection sampling operates by drawing $i \sim e^{\tilde{f}_i(x;y)/\epsilon'}$, where $\tilde{f}_i(x;y)$ is an approximation of $f_i(x)$ for x close to y, and then accepting with probability $\exp\big((f_i(x) - \tilde{f}_i(x;y) - C)/\epsilon'\big)$ for C such that $\left|f_i(x) - \tilde{f}_i(x;y)\right| \le C$ for all $\|x - y\| \le r$. For an approximation $\tilde{f}_i$ with $C = O(\epsilon')$, this rejection sampling routine returns a valid sample from $e^{f(x)/\epsilon'}$ using an expected $O(1)$ draws from $e^{\tilde{f}_i(x;y)/\epsilon'}$. Asi et al. [3] simply perform n evaluations to precompute $f_1(y), \ldots, f_n(y)$ and then take $\tilde{f}_i(x;y) = f_i(y)$, for which $C = L_f r$ by the Lipschitz continuity of the f_i. Taking $r = \epsilon'/L_f$ ensures that each ∇f_{smax} estimation takes $O(1)$ expected additional evaluations. Thus, the overall expected evaluation complexity of minimizing f_{smax} inside a ball of radius $r = O(\epsilon/L_f)$ is $n + O(T)$, where the SGD iteration number T is sublinear in n.

Finally, we make efficient use of the ball oracle to globally minimize f_{smax}. To this end, we rely on the ball oracle acceleration technique proposed by Carmon et al. [14] and refined in [16, 3, 11, 17], which we further improve in this work. The technique, a type of accelerated proximal point method [29, 53, 42] finds an ϵ-accurate minimizer in $O(r^{-2/3} \log(1/\epsilon))$ ball oracle calls. Combining these ingredients yields a gradient evaluation complexity bound whose leading term in n is $\widetilde{O}(nr^{-2/3}) = \widetilde{O}(n(L_f/\epsilon)^{2/3})$.

2.2 Increasing the ball size by linear approximation data structures.

Exact linear approximation. The main limitation of the softmax gradient estimation procedure described above is that it only works for fairly small balls of radius $\widetilde{O}(\epsilon/L_f)$. To increase the ball size, we leverage smoothness to build better function value approximations $\tilde{f}_i(x;y)$. As a starting point, consider the linear approximation

$$\tilde{f}_i^{\text{lin}}(x;y) \coloneqq f_i(y) + \langle \nabla f_i(y), x - y \rangle .$$

When each f_i is L_g-smooth (i.e., ∇f_i is L_g-Lipschitz) then $\left|f(x) - \tilde{f}_i^{\text{lin}}(x;y)\right| \le \frac{1}{2} L_g \|x - y\|^2$ for all x and y. Therefore, we may increase the ball radius r from ϵ'/L_f to $\sqrt{\epsilon'/L_g}$. Since computing $\tilde{f}_1^{\text{lin}}(\cdot;y), \ldots, \tilde{f}_n^{\text{lin}}(\cdot;y)$ requires only n function and gradient evaluations, substituting this improved approximation into the acceleration framework described above yields a leading order evaluation complexity term of $\widetilde{O}(nr^{-2/3}) = \widetilde{O}(n(L_g/\epsilon)^{1/3})$.

However, sampling $i \sim e^{\tilde{f}_i^{\text{lin}}(x;y)/\epsilon'}$ is computationally expensive, since exactly computing the inner products $\langle \nabla f_1(y), x - y \rangle, \ldots, \langle \nabla f_n(y), x - y \rangle$ takes $\Theta(nd)$ time. In some cases, including bilinear problems (1.1), this is as expensive as calculating ∇f_{smax} exactly, undoing the efficiency gains of rejection sampling using $\tilde{f}^{\text{lin}}$.

Matrix-vector estimation data structure. We address this challenge by replacing $\tilde{f}_i^{\text{lin}}(x;y)$ with an efficient randomized approximation, denoted $\tilde{f}_i^{\text{est}}(x;y)$, such that $\left|\tilde{f}_i^{\text{est}}(x;y) - \tilde{f}_i^{\text{lin}}(x;y)\right| \le \epsilon'$ with high probability. We construct *matrix-vector estimation data structures* that, after $O(nd)$ preprocessing time, for query x and reference y, compute $\{\tilde{f}_i^{\text{est}}(x;y)\}_{i \in [n]}$ in time $\widetilde{O}\left(n(L_f \|x - y\|/\epsilon')^2\right)$: in the ℓ_2 setting we achieve this using CountSketch [19, 37], while in the ℓ_1 setting we simply approximate $\langle \nabla f_i(y), x - y \rangle$ by sampling entries of $\nabla f_i(y)$ from a distribution proportional to $|x - y|$, a technique similar to "sampling from the difference" used for variance reduction in matrix games [13].

From matrix-vector estimation to maintenance. If we were to implement the ball oracle using the estimate described above, the additional runtime cost would be $\widetilde{O}(n(L_f r/\epsilon)^2 T)$, where T is the SGD iteration count. While independent of d, such runtime would have a large dependence on the desired accuracy ϵ, again rendering the approach unhelpful for matrix games. To further improve efficiency, we design *matrix-vector maintenance data*

structures that allow evaluating $\tilde{f}^{\text{est}}$ at a series of query points $x_1, \ldots, x_T$ with additional runtime

$$\widetilde{O}\left(nd + n\left(\frac{L_f \sum_{i\in[T]} \|x_i - x_{i-1}\|}{\epsilon'}\right)^2\right).$$

As we explain in more detail below, we design a careful stochastic gradient method for which the queries satisfy $\sum_{i\in[T]} \|x_i - x_{i-1}\| = \widetilde{O}(r)$, leading to the $\widetilde{O}\left(r^{-2/3} n (L_f r/\epsilon)^2\right) = \widetilde{O}\left(n L_f^2 / L_g^{2/3} \epsilon^{4/3}\right)$ additional runtime shown in Table 1.

Our matrix-vector maintenance data structure solves the more general problem of approximately maintaining the value of Ax for a suitably bounded matrix $A \in \mathbb{R}^{n\times d}$ and a changing x that is guaranteed not to move too much. Specialized to our applications, this data structure essentially maintains approximations to $\langle \nabla f_i(y), v\rangle$ for vectors v of different (exponentially spaced) distances from the current point. Given a new query x, the data structure simply updates the v vectors and the approximations to $\langle \nabla f_i(y), v\rangle$ to preserve the exponentially space distance invariant. This update is made efficient by estimating the values of $\langle \nabla f_i(y), v\rangle$ for the update v in terms of their difference in value from the closest (to the query point) non-updated v and using a matrix-vector estimator. Finally, the data structure outputs the approximation to $\langle \nabla f_i(y), v\rangle$ for the closest v.

By carefully choosing and reusing approximations to the $\langle \nabla f_i(y), v_j\rangle$ over time, we are able to guarantee the claimed runtime bound. Essentially, we obtain a runtime for maintaining approximation over a whole sequence of queries in essentially the same complexity a matrix-vector estimation data structure would naturally use for answering one query whose distance form y is the sum of all query movements. We design our matrix-vector maintenance data structure via a reduction to matrix-vector estimation, which; this more general framework could be of utility in other geometries.

A note on obliviousness. Our use of efficient randomized data structures hinges on a subtle yet crucial property of our method: our data structure query sequences do not depend on its random state, and hence the probabilistic approximation guarantees remain valid throughout. At first glance this might appear to be false, since we use the output of the data structure to draw random indices that define the stochastic gradient estimate and hence influence the next SGD iterate and data structure query point. However, due to rejection sampling, the *distribution* of the rejection sampling output is proportional to $e^{f(x)/\epsilon'}$, without any dependence on the random bits of the data structure.[2]

2.3 Accelerating entropy ball oracles.

We now shift our focus to the ball oracle acceleration algorithm that takes in an (approximate) radius-r ball oracle and returns an approximate minimizer in $\widetilde{O}(r^{-2/3})$ oracle calls. Here, the main challenge is extending the algorithm to support a non-Euclidean domain geometry. Specifically, the difficulty lies in coming up with an approximate oracle notion that supports efficient implementation via stochastic gradient methods while still allowing acceleration.

Prior idealized scheme. To explain our developments, it is instructive to first consider idealized acceleration schemes using exact ball oracles, and contrast the idealized scheme of prior work to the one proposed here. Previous ball acceleration methods [14, 16, 3, 17][3] maintain a parameter sequence $a_1, \ldots, a_T$ and its running sums $A_t = \sum_{i\le t} a_i$, and construct an iterate sequence (x_t, v_t) according to

$$x_{t+1} = \operatorname*{argmin}_{x\in\mathcal{X}:\|x-\Phi_t(v_t)\|_2\le r} \left\{ f(x) + \frac{A_{t+1}}{2a_{t+1}^2}\|x - \Phi_t(v_t)\|_2^2 \right\} \quad \text{where } \Phi_t(z) \coloneqq \frac{A_t}{A_{t+1}} x_t + \frac{a_{t+1}}{A_{t+1}} z \tag{2.4}$$

$$v_{t+1} = \operatorname*{argmin}_{v\in\mathcal{X}} \left\{ \langle \nabla f(x_{t+1}), v\rangle + \frac{1}{2a_{t+1}}\|v - v_t\|_2^2 \right\}. \tag{2.5}$$

The step (2.4) calls a radius-r ball oracle with center point $\Phi_t(v_t)$, while the step (2.5) executes a mirror descent iteration using the gradient of f at the output of the ball oracle. Proper setting of a_t ensures that for all t we have

[2] More precisely, the distribution of the next iterate is the same for all possible random bits, except for a low-probability set of random bits for which the approximation condition $\left|\tilde{f}_i^{\text{est}}(x;y) - \tilde{f}_i^{\text{lin}}(x;y)\right| \le \epsilon'$ fails for some $i \in [n]$.

[3] In order to ensure correctness, these ball acceleration methods must either choose a_t such that x_{t+1} has $\|x_{t+1} - \Phi_t(v_t)\| \in [r/2, r)$ (which necessitates a bisection to solve an implicit equation) or modify their iterates through a momentum damping scheme [17]. We ignore this point throughout the overview, and use momentum damping in our full method.

$f(x_t) - f(x_\star) \le \frac{\|x_0 - x_\star\|_2^2 - S_t}{A_t}$ for some $S_t \ge 0$, and that after $T = O\left(\left(\frac{\|x - x_\star\|_2}{r}\right)^{2/3} \log \frac{f(x_0) - f(x_\star)}{\epsilon}\right)$ iterations either $A_T \ge \frac{\|x_0 - x_\star\|_2^2}{\epsilon}$ or $S_T \ge \|x_0 - x_\star\|_2^2$.

To move to general norms, we use the standard technique of introducing a Bregman divergence $V_a(b)$ induced by a 1-strongly-convex distance generating function, so that $V_a(b) \ge \frac{1}{2}\|b - a\|^2$; in the Euclidean we simply have $V_a(b) = \frac{1}{2}\|a - b\|_2^2$, while for the simplex setting we use the KL divergence $V_a(b) = \sum_{i \in [d]} b_i \log \frac{b_i}{a_i}$ (see Section 3 for more details).

A straightforward generalization of the idealized method above exists, but is not conducive to approximation. Such generalization consists of replacing $\|\cdot\|_2$ in step (2.4) with a general norm $\|\cdot\|$, and replacing $\frac{1}{2}\|v - v_k\|_2^2$ with $V_{v_k}(v)$ in step (2.5). It can be shown that $f(x_t) - f(x_\star) \le \epsilon$ after $O\left(\left(\frac{V_{x_0}(x_\star)}{r}\right)^{2/3} \log \frac{f(x_0) - f(x_\star)}{\epsilon}\right)$ iterations. However, it is not clear how to efficiently approximate the non-Euclidean ball oracle computation in this method. In particular, in order to approximate the step (2.5), Asi et al. [3] design a multilevel Monte Carlo (MLMC) estimator that is nearly unbiased for the exact ball oracle output (2.4), and the analysis of this technique appears to strongly rely on properties that are unique to the Euclidean norm.

New idealized scheme. To address this challenge, we redesign the acceleration method with Bregman divergences and efficient approximation in mind. Our new idealized method is

$$v_{t+1} = \operatorname*{argmin}_{v \in \mathcal{X} : V_{v_t}(v) \le \frac{1}{2}\rho_{t+1}^2} \{A_{t+1} f(\Phi_t(v)) + V_{v_t}(v)\} \quad \text{where } \rho_{t+1} \coloneqq \frac{A_{t+1}}{a_{t+1}} r \tag{2.6}$$

$$x_{t+1} = \Phi_t(v_{t+1}) \tag{2.7}$$

and Φ_t is as defined in (2.4). In the unconstrained Euclidean case (i.e., when $\mathcal{X} = \mathbb{R}^d$ and the a_t sequence is such that $\|x_{t+1} - \Phi_t(v_t)\| < r$ for all t), straightforward algebra shows that the old and new idealized schemes are exactly equivalent. However, outside that setting—and particularly in the non-Euclidean case—the two methods produce different iterates. Nonetheless, both methods enjoy the same $O\big(\big(\frac{V_{x_0}(x_\star)}{r}\big)^{2/3} \log \frac{f(x_0) - f(x_\star)}{\epsilon}\big)$ iteration complexity guarantee. Moreover, the constraint $V_{v_t}(v) \le \frac{1}{2}\rho_{t+1}^2$ and the definition of ρ_{t+1} implies that every feasible point v in step (2.6) satisfies $\|\Phi_t(v) - \Phi_t(v_t)\| \le \frac{a_{t+1}}{A_{t+1}}\|v - v_t\| \le \frac{a_{t+1}}{A_{t+1}}\rho_{t+1} = r$. This justifies considering (2.6) a call to a radius-r optimization oracle centered at $\Phi_t(v_t)$.

Defining the approximate ball oracle. We now briefly derive our approximation condition for step (2.6). To lighten notation, let $y \coloneqq v_t$, let $\rho \coloneqq \rho_{t+1}$, let $h(v) \coloneqq A_{t+1} f(\Phi_t(v))$ and $v_\star \coloneqq v_{t+1}$ (i.e., the exact ball oracle output). Note that $v_\star$ is the global minimizer of $H(v) \coloneqq h_t(v) + cV_y(v)$, for some $c \ge 1$ which enforces the constraint $V_y(v) \le \frac{1}{2}\rho^2$. Therefore, by convexity we have $H(v_\star) - H(u) \le -cV_{v_\star}(u)$ for all $u \in \mathcal{X}$. Substituting the definition of H and dividing through by c gives

$$\frac{h(v_\star) - h(u)}{c} \le V_y(u) - V_{v_\star}(u) - V_y(v_\star) \le V_y(u) - V_{v_\star}(u) - \frac{1}{2}\rho^2 \mathbb{1}_{\{c > 1\}} \text{ for all } u \in \mathcal{X}, \tag{2.8}$$

where the final inequality holds since $V_y(v_\star) = \frac{1}{2}\rho^2$ when $c > 1$ due to complementary slackness.

To further relax the condition (2.8), we allow the approximate ball oracle to *return two points* $z, w \in \mathcal{X}$ such that $h(z)$ replaces $h(v_\star)$ and $V_w(u)$ replaces $V_{v_\star}(u)$. We further replace $\frac{1}{2}\rho^2 \mathbb{1}_{\{c>1\}}$ with $\gamma\rho^2 \mathbb{1}_{\{c>2\}}$ for some $\gamma \le \frac{1}{2}$, and we allow $\gamma\rho^2$ additive error for $c \le 2$. Finally, we allow randomization by requiring that bound holds only in expectation. The resulting relaxed output condition is

$$\mathbb{E}\frac{h(z) - h(u)}{c} \le \mathbb{E}[V_y(u) - V_w(u)] - \gamma\rho^2 \mathbb{E}\big[\mathbb{1}_{\{c>2\}} - \mathbb{1}_{\{c \le 2\}}\big] \text{ for all } u \in \mathcal{X}. \tag{2.9}$$

In the acceleration framework, we approximate v_{t+1} with w, and x_{t+1} with $\Phi_t(z)$, and show that the resulting sequence still satisfies (up to constants) the same error bound as the exact proximal method. The key advantage of the two-point approximation condition (2.9) is that SGD naturally achieves it, with z and w being the average and final SGD iterates respectively. This "two outputs" property of SGD has been leveraged before in the literature on gradient sliding methods in structured convex optimization [35, 59, 36]. It allows us to sidestep the need for Multilevel Monte-Carlo [7, 3], which appears challenging to use in the non-Euclidean setting.

2.4 Implementing entropy-ball oracles. We now explain the key components in constructing an approximate ball oracle meeting the condition (2.9) using our data structure-based gradient estimator. There are two main challenges in designing this oracle. First, the inequality (2.9) needs to hold for all $u \in \mathcal{X}$ rather than just in a ball of radius ρ around y; this prevents us from using standard constrained optimization techniques. Second, our matrix-vector maintenance data structure requires that the total movement in the SGD iterates sum to $\widetilde{O}(\rho)$, a guarantee which standard SGD does not provide. We explain our solution to each challenge in turn.

Implicitly-constrained SGD. To obtain a guarantee valid for any comparator point $u \in \mathcal{X}$, we approximately find the Lagrange multiplier for the constraint $V_y(v) \le \frac{1}{2}\rho^2$ and apply unconstrained SGD, taking careful care to show that its iterates nevertheless stay close to the reference point y. First, we perform bisection to find a Lagrange multiplier $\lambda \ge 1$ such that $v_\lambda = \operatorname{argmin}_{v\in\mathcal{X}}\{h(v) + \lambda V_y(v)\}$ satisfies $V_y(v_\lambda) \in [\frac{\alpha}{2}\rho^2, \frac{\beta}{2}\rho^2]$ for some $\alpha, \beta = \widetilde{\Theta}(1)$, where we use SGD to approximate $V_y(v_\lambda)$. Second, having found a suitable λ, we apply (unconstrained) SGD once more to obtain the global guarantee (2.9) with $c \approx \lambda$. However, removing the explicit ball constraint introduces another difficulty: SGD could potentially query iterates outside the ball, where our gradient estimator is inefficient. To address this concern we use techniques introduced in [12, 30] to show that, with high probability, SGD never leaves a ball of radius $O(\|v_\lambda - y\|)$ around y. Since λ satisfies $V_y(v_\lambda) = O(\rho^2)$, the SGD iterates remain (with high probability) in the region where our gradient estimator is efficient.

A relaxed triangle inequality of KL divergence. Before proceeding to the next challenge we highlight a technical point of potential broader interest. To establish the correctness of the procedures described above, we need to assume that the Bregman divergence satisfies a relaxed triangle inequality of the form

$$V_a(b) + V_b(a) \le \tau\Big(\tilde{V}(a,c) + \tilde{V}(c,b)\Big) \text{ where } \tilde{V}(x,y) = \min\{V_x(y), V_y(x)\}$$

for all $a, b, c \in \mathcal{X}$. In the Euclidean case where $V_a(b) = \frac{1}{2}\|a-b\|_2^2$, this holds for $\tau = 4$. However, when $\mathcal{X}$ is the simplex and V is the KL divergence, this inequality is false for any τ. Nevertheless we show that for a *truncated simplex* $\Delta_\nu^n = \{p \in \Delta^n \mid p_i \ge \nu \text{ for all } i\}$, the relaxed triangle inequality holds with $\tau = O(\log \frac{1}{\nu})$. This observation is new to the best of our knowledge, and potentially of independent interest. The Lipschitz continuity of our objective functions means that its optimal value in $\mathcal{X}$ and $\mathcal{X} \cap \Delta_\nu^n$ differ by at most $O(L_f\nu)$. Therefore, truncating the simplex with $\nu = \mathsf{poly}(\epsilon/L_f)$ allows us to use the relaxed triangle inequality with $\tau = \widetilde{O}(1)$ without significantly changing the solution quality.

Controlling the sum of query movement sizes. Next, we address the challenge introduced by our matrix-vector maintenance data structure. This data structure enables us to generate stochastic gradients for SGD at a computational cost proportional to the sum of distances between consecutive SGD queries. For standard SGD using T iterations, this sum is $\Omega(\sqrt{T})$, resulting in a bad complexity bound. To address this, we employ a variant of SGD due to Cutkosky [23] which enables much tighter control over total query movement. This variant applies mirror descent updates on the gradient estimated on the running average of its iterates, computing

$$w_{t+1} = \operatorname*{argmin}_{w\in\mathcal{X}}\left\{\langle \mathcal{G}(x_t), w\rangle + \frac{1}{\eta}V_{w_t}(w)\right\},$$

where η is a step size, $\mathcal{G}$ is the gradient estimator, and $x_t = \frac{1}{t}\sum_{i\le t} w_i = \frac{t-1}{t}x_{t-1} + \frac{1}{t}w_t$. Therefore, we have $\sum_{t\le T}\|x_t - x_{t-1}\| = \sum_{t\le T}\frac{1}{t}\|w_t - x_{t-1}\|$. Since we guarantee that $\|w_t - w_0\| = O(\rho)$ for all $t \le T$ with high probability, we have $\|w_t - x_t\| = O(\rho)$ as well. This implies the movement bound $\sum_{t\le T}\frac{1}{t}\|w_t - x_t\| = O(\rho \log T)$ that is sufficient for our purposes.

2.5 Putting it all together. Having described our main algorithmic ingredients, we now briefly derive the runtime bounds shown in Tables 1 and 2.

Acceleration framework setup. We begin by considering our accelerated proximal point method applied on the function f_{smax}. We stop the method at the first time T in which $A_T = \Omega(\epsilon^{-1})$, where its potential analysis guarantees $\mathbb{E}f_{\text{smax}}(x_T) - f_{\text{smax}}(x_\star) = O(\epsilon)$. Roughly speaking, our algorithm sets the a_t sequence such that $\frac{a_{t+1}}{A_t} = \widetilde{\Theta}(r^{2/3})$ is constant for all iterations. We show that with an appropriate damping scheme, our algorithm will either grow A_{t+1} by a multiplicative $1 + \widetilde{\Theta}(r^{2/3})$ factor or decrease a nonnegative potential function with initial value 1 by $\widetilde{\Theta}(r^{2/3})$: this implies that A_T exceeds the stopping threshold in $T = \widetilde{O}(r^{-2/3})$ steps.

Our setting of a_t means that

$$\rho_t = \frac{A_t}{a_t} r = \widetilde{\Theta}(r^{1/3})$$

is also constant for all the iterations. At step t we apply our approximate ball oracle on $h_t(v) = A_{t+1} f_{\text{smax}}(\Phi_t(v))$. Noting that the Jacobian of Φ_t is $\frac{a_{t+1}}{A_{t+1}} I$, we have $\nabla h_t(v) = a_{t+1} \nabla f_{\text{smax}}(\Phi_t(v))$ by the chain rule. Therefore, to estimate $\nabla h_t(v)$ we simply apply our estimator for ∇f_{smax} at the point $\Phi_t(v)$ and multiply the resulting vector by a_{t+1}. Since our estimates for ∇f_{smax} are always of the form $\nabla f_i(\Phi_t(v))$ for some $i \in [N]$, they are bounded by L_f. The gradients estimates for h_t are therefore bounded by

$$\Gamma = a_{t+1} L_f = \widetilde{O}\left(r^{2/3} A_{t+1} L_f\right) = \widetilde{O}\left(\frac{r^{2/3}}{\epsilon} L_f\right),$$

where the last transition holds since $A_t = O(\epsilon^{-1})$ for all iterations before stopping.

Iteration and evaluation complexity. Next, we bound the iteration count of all ball oracle calls and the total gradient evaluation complexity. For a function h with stochastic gradients bounded by Γ and target movement ρ, the approximate ball oracle requires $\widetilde{O}(\Gamma^2/\rho^2)$ iterations; the complexity of finding a point that is $\widetilde{O}(\rho)$ away from the optimum of a 1-strongly-convex function using stochastic gradients bounded by Γ. Substituting the above bounds for Γ and ρ, the iteration complexity per oracle call is $\widetilde{O}\left(r^{2/3} L_f^2 \epsilon^{-2}\right)$. For each ball oracle call we require n individual function and gradient evaluations to set up the data structure, and (for $r = \widetilde{O}(\sqrt{\epsilon/L_g})$) an additional $\widetilde{O}(1)$ gradient evaluations per step with high probability, giving $\widetilde{O}(n + r^{2/3} L_f^2 \epsilon^{-2})$ evaluations overall. Since the expected number of ball oracle calls is $\widetilde{O}(r^{-2/3})$, with constant probability the total evaluation complexity is $\widetilde{O}\left(nr^{-2/3} + L_f^2 \epsilon^{-2}\right)$.

Runtime complexity. To account for the runtime complexity of our method, we make the simplifying assumption that each function/gradient evaluation takes $\Omega(d)$ time. In this case, the only term not subsumed by the function/gradient evaluation cost comes from the matrix-vector maintenance" data structure. Our oracle implementation makes sequences of queries to our ∇f_{smax} estimator, whose total movement is $\widetilde{O}(r)$. Therefore, the additional runtime of a single oracle call is $\widetilde{O}\left(nL_f^2 r^2/\epsilon^2\right)$, and for the whole algorithm the cost is $\widetilde{O}\left(nL_f^2 r^{4/3}/\epsilon^2\right)$.

Choosing the ball radius r. Finally, we discuss the optimal choice of the parameter r. For general problems (1.2) with $L_g > 0$, a simple strategy is to choose the highest value of r for which the linear approximation is sufficiently accurate, i.e, $\widetilde{\Theta}(\sqrt{\epsilon/L_g})$. This yields the complexity bounds in Table 1. However, when L_g is very small it is more computationally efficient to choose a smaller value of r. Letting $\mathcal{T} = \Omega(d)$ denote the runtime of an individual function/gradient evaluation, the value of r that minimizes the runtime terms $n\mathcal{T} r^{-2/3} + nr^{4/3} L_f^2/\epsilon^2$ is $r = \epsilon\sqrt{\mathcal{T}}/L_f$, and the minimal value is $n(\mathcal{T} L_f/\epsilon)^{2/3}$. For $L_g < L_f^2/\mathcal{T}\epsilon$ this optimal r is permissible (i.e., smaller than $\sqrt{\epsilon/L_g}$), and the total runtime of the method is $\widetilde{O}\left(n\left(\frac{\mathcal{T} L_f}{\epsilon}\right)^{2/3} + \mathcal{T}\left(\frac{L_f}{\epsilon}\right)^2\right)$. In particular, for matrix games (where $\mathcal{T} = \Theta(d)$ and $L_g = 0$) we obtain the runtimes listed in Table 2.

3 Notation and conventions

General. We use $\mathcal{X}$ to denote a general closed convex set. We use $\Delta^d \coloneqq \{x \in \mathbb{R}^d, x \geq 0, \sum_i x_i = 1\}$ to denote the simplex, $\Delta_\nu^d \coloneqq \{x \in \Delta^d, x \geq \nu \mathbf{1}\}$ to denote the truncated simplex, and $\mathbb{B}^d \coloneqq \{x \in \mathbb{R}^d, \|x\|_2 \leq 1\}$ to denote the unit Euclidean ball. We denote the binary indicator of event $\mathfrak{E}$ by $\mathbb{1}_{\{\mathfrak{E}\}}$.

Vector, matrix and norm. We use $\|\cdot\|$ to denote a general norm on $\mathcal{X}$ and $\|\cdot\|_* = \sup_{\|x\| \leq 1} \langle x, \cdot \rangle$ to denote its dual norm on the dual space $\mathcal{X}^*$. For any vector $v \in \mathbb{R}^d$ and $p \geq 1$ we denote the ℓ_p norm by $\|v\|_p \coloneqq \left(\sum_{i \in [d]} |v_i|^p\right)^{1/p}$ with $\|v\|_\infty = \max_{i \in [d]} |v_i|$. For any $p \geq 1$ we let $p^* = (1 - \frac{1}{p})^{-1}$ be such that $\|\cdot\|_{p^*}$ is dual to $\|\cdot\|_p$. For any matrix $A \in \mathbb{R}^{n \times d}$, we write A_{ij} for the (i,j) entry, $A_{i:}$ for the i-th row as a row vector, and $A_{:j}$ for the j-th column as a column vector. Given $p, q \geq 1$, we write the matrix norm $\|A\|_{p \to q} \coloneqq \max_{v \in \mathbb{R}^d, v \neq 0} \frac{\|Av\|_q}{\|v\|_p}$.

Functions. We work with convex, differentiable functions f on domain $\mathcal{X}$ throughout the paper. We say a function f is L_f-Lipschitz with respect to $\|\cdot\|$ if and only if $|f(x) - f(y)| \leq L_f \|x - y\|$ for all $x, y \in \mathcal{X}$. A function f is L_g-smooth with respect to $\|\cdot\|$ if and only if $\|\nabla f(x) - \nabla f(y)\|_* \leq L_g \|x - y\|$ for all $x, y \in \mathcal{X}$. A convex function

f is μ-strongly convex with respect to $\|\cdot\|$ if and only if for any $x, y \in \mathcal{X}$, $f(x) - f(y) - \langle \nabla f(y), x - y\rangle \geq \frac{\mu}{2}\|x-y\|^2$. We call a random point x an ϵ-optimal minimizer of f in expectation if $\mathbb{E}f(x) - \min_{x' \in \mathcal{X}} f(x') \leq \epsilon$.

Bregman divergences. Given a distance-generating function (dgf) $\varphi : \mathcal{X} \to \mathbb{R}$, we define its induced *Bregman divergence* $V_x^r(y) := \varphi(y) - \varphi(x) - \langle \nabla\varphi(x), y - x\rangle$, and drop the superscript φ when clear from context. Within the paper, for Euclidean space equipped with $\|\cdot\|_2$, we use $\varphi(x) = \frac{1}{2}\|x\|^2$ and its induced Bregman divergence is $V_x(y) = \frac{1}{2}\|x-y\|_2^2$, which is 1-strongly convex in $\|\cdot\|_2$. For the simplex (or a closed convex subset thereof) equipped with $\|\cdot\|_1$, we use $\varphi(x) = \sum_i x_i \log x_i$ and its induced Bregman divergence is the Kullback–Leibler (KL) divergence $V_x(y) = \sum_i y_i \log(y_i/x_i)$, which is 1-strongly convex in $\|\cdot\|_1$ by Pinsker's inequality.

Runtime. To simplify the presentation of our runtime bounds we use the following conventions throughout. We assume that the number of non-zero elements in matrix $A \in \mathbb{R}^{d\times n}$, denoted $\mathrm{nnz}(A)$, satisfies $\mathrm{nnz}(A) = \Omega(d+n)$. This holds for any matrix without empty rows or columns. In similar vein, we assume that the number of non-zero elements in any vector x satisfies $\mathrm{nnz}(x) = \Omega(1)$.

We also assume that we are working in a computational model in which can pre-process any vector $v \in \mathbb{R}^n$ in $O(n)$ time and then be able to sample index i with probability proportional to $|v_i|$ in $O(1)$ time, e.g., as in [65]. If these costs are larger by multiplicative polylogarithmic factors then our final runtimes similarly grow by multiplicative polylogarithmic factors.

Throughout the paper, we use $\widetilde{O}$, $\widetilde{\Omega}$ and $\widetilde{\Theta}$ to hide poly-logarithmic factors in problem parameters, e.g. dimension, smoothness, Lipschitz constant, domain size, and desired accuracy ϵ and probability factor $1/\delta$.

4 Non-Euclidean ball oracle acceleration

In this section, we describe our main acceleration framework leveraging a non-Euclidean ball oracle. The main result proved in this section is the following.

THEOREM 4.1. *Let $f : \mathcal{X} \to \mathbb{R}$ be a convex function which supports a gradient oracle $\mathcal{G}$ with $\|\mathcal{G}(x)\|_* \leq G$ for all $x \in \mathcal{X}$. For some $\mathcal{E}_0, R > 0$, let $x_0, v_0 \in \mathcal{X}$ satisfy $f(x_0) - f(x_\star) \leq \mathcal{E}_0$ and $V_{v_0}(x_\star) \leq R^2$, where $x_\star$ is a minimizer of f. For any ball radius $r \leq R$, oracle approximation parameter $\gamma < 1/2$, and error tolerance $\epsilon > 0$, Algorithm 1 has the following guarantees:*

- *The algorithm outputs a point x_T such that $\mathbb{E}\left[f(x_T)\right] - f(x_\star) \leq \epsilon$.*
- *The algorithm terminates after $O(\gamma^{-1/3}R^{2/3}r^{-2/3}\log(\mathcal{E}_0/\epsilon))$ iterations in expectation.*
- *Each iteration of the algorithm performs $O(1)$ arithmetic operations on elements of $\mathcal{X}$ and makes a single call to a ball-restricted proximal oracle (Definition 4.1 below) with parameter $\rho = \Theta\left(\gamma^{-1/3}R^{2/3}r^{1/3}\right)$ for a convex function h_t that supports a gradient estimator $\mathcal{G}_t$ with $\|\mathcal{G}_t(x)\|_* = O\left(\frac{\gamma^{1/3}r^{2/3}R^{4/3}}{\epsilon}G\log(\mathcal{E}_0/\epsilon)\right)$.*

Our result in this section follows the outline in Section 2.3. Algorithm 1 chooses parameter sequences A'_t, a_t and in each iteration calls an oracle that attempts to solve the optimization problem

$$\underset{V_{v_t}(z)\leq\rho^2}{\text{minimize}}\,\{h_t(z) + V_{v_t}(v)\}, \quad \text{where} \quad h_t(z) = A'_{t+1}f(\Phi_t(z)) \quad \text{and} \quad \Phi_t(z) = \frac{A_t}{A'_{t+1}}x_t + \frac{a_{t+1}}{A'_{t+1}}z. \tag{4.10}$$

We consider an approximate oracle that relaxes the exact solution to (4.10) in three critical ways:

- We allow the oracle to return a parameter c, which corresponds to the Lagrange multiplier on the domain constraint,
- We let the oracle return two points—each used for a different purpose in our final algorithm,
- We allow the oracle's output guarantee to hold in expectation and to tolerate some additive error.

Formally, we define this relaxed oracle as follows.

DEFINITION 4.1. (BALL-RESTRICTED PROXIMAL ORACLE) *Let $h : \mathcal{X} \to \mathbb{R}$ be a convex function with gradient estimator $\mathcal{G}$. A $(\rho, \gamma, c_{\max})$-*restricted proximal oracle *takes as input $\mathcal{G}$, center point $y \in \mathcal{X}$ and points $z, w \in \mathcal{X}$ and a scalar $c \in [1, c_{\max}]$ satisfying*

$$\mathbb{E}\left[\frac{h(z) - h(u)}{c}\right] \leq \mathbb{E}\left[V_y(u) - V_w(u)\right] - \gamma\mathbb{E}\left[\mathbb{1}_{\{c\geq 2\}} - \mathbb{1}_{\{c<2\}}\right]\rho^2. \tag{4.11}$$

We note that our analysis only needs the oracle parameter $c_{\max}$ to be finite, since its only use is verifying a condition of the optional stopping theorem. We therefore omit it for Theorem 4.1 and the subsequent lemmas used to prove it, and argue that it is indeed finite for our oracle implementations.

We now describe a final component of Algorithm 1 that is omitted from the outline in Section 2.3: *momentum damping*. This mechanism, introduced in recent work on optimal methods for Monteiro-Svaiter acceleration [17], handles the fact the sequence c_t of regularization terms varies over time which introduces subtlety to the selection of a suitable sequence a_t. Given the outputs $z_{t+1}, w_{t+1}, c_{t+1}$ from the oracle in an iteration, we set $v_{t+1} = w_{t+1}$ and $x'_{t+1} = \Phi_t(z_{t+1})$. However, instead of returning $x_{t+1} = x'_{t+1}$ for the next iteration, we actually set

$$x_{t+1} = \frac{1}{c_{t+1}} x'_{t+1} + \frac{c_{t+1} - 1}{c_{t+1}} x_t \quad \text{and} \quad A_{t+1} = A_t + \frac{a_{t+1}}{c_{t+1}}.$$

To provide intuition for this, we consider the cases where $c_{t+1} \approx 1$ and $c_{t+1} \gg 1$. In the former case, the Lagrange multiplier on the domain constraint of eq. (4.10) is nearly inactive: thus our output $x_{t+1} \approx x'_{t+1}$ makes good progress. On the other hand, if $c_{t+1} \gg 1$ then the ball constraint on the proximal step is extremely active. In this case, we are unable to conclude that x'_{t+1} has good function error: we set $x_{t+1} \approx x_t$ and $A_{t+1} \approx A_t$ to prevent x'_{t+1} from destabilizing the algorithm. However, we show that c_{t+1} being very large implies that a natural potential function significantly decreases: this "win-win situation" enables us to guarantee progress regardless of the actual range of c_{t+1}.

Algorithm 1: Generalized ball acceleration framework

Input: Convex function f with gradient estimator $\mathcal{G}$
Input: $\mathcal{O}$, a $(\rho, \gamma, c_{\max})$-ball restricted proximal oracle
Input: Parameters $r, R, \mathcal{E}_0, \epsilon > 0$
Input: Input points $x_0, v_0 \in \mathbb{R}^n$ satisfying $f(x_0) - f(x_\star) \le \mathcal{E}_0$, $V_{v_0}(x_\star) \le R^2$

1 $A_0 = \frac{R^2}{\mathcal{E}_0}$
2 **while** $A_t < \frac{40R^2 \log(80\mathcal{E}_0/\epsilon)}{\epsilon}$ **do**
3 $\quad a_{t+1} = \left(\sqrt{\gamma} r / R\right)^{2/3} A_t$ and $A'_{t+1} = A_t + a_{t+1}$
4 $\quad \Phi_t(z) = \frac{A_t}{A'_{t+1}} x_t + \frac{a_{t+1}}{A'_{t+1}} z$ and $\rho = \frac{A'_{t+1}}{a_{t+1}} r = \left(1 + \left(\frac{R}{\sqrt{\gamma} r}\right)^{2/3}\right) r$ ▷ ρ is constant across iterations
5 $\quad h_t(z) \coloneqq A'_{t+1} f\left(\Phi_t(z)\right)$ ▷ $\|x - y\| \le \rho$ implies $\|\Phi_t(x) - \Phi_t(y)\| \le \frac{a_{t+1}}{A'_{t+1}} \rho = r$
6 $\quad \mathcal{G}_t(z) = a_{t+1} \mathcal{G}(\Phi_t(z))$ ▷ $\mathcal{G}_t$ is a stochastic gradient estimator for h_t
7 $\quad z_{t+1}, v_{t+1}, c_{t+1} = \mathcal{O}(\mathcal{G}_t, v_t, \rho)$
8 $\quad x_{t+1} = \frac{1}{c_{t+1}} \Phi_t(z_{t+1}) + \frac{c_{t+1} - 1}{c_{t+1}} x_t$
9 $\quad A_{t+1} = \frac{c_{t+1} - 1}{c_{t+1}} A_t + \frac{1}{c_{t+1}} A'_{t+1} = A_t + \frac{a_{t+1}}{c_{t+1}}$
10 $\quad t = t + 1$

Return: x_t

We begin the analysis by proving a potential decrease bound.

LEMMA 4.1. (POTENTIAL DECREASE) *Consider an execution of Algorithm 1. Let $x_\star \in \mathcal{X}$ be a minimizer of f and for each iteration t let*

$$E_t \coloneqq f(x_t) - f(x_\star) \quad \text{and} \quad D_t \coloneqq V_{v_t}(x_\star).$$

Let $P_t \coloneqq A_t E_t + D_t$ where A_t is defined on line 9. Then for any $t \ge 0$

$$\mathbb{E}\left[P_{t+1}\right] \le P_t - \gamma \mathbb{E}\left[\mathbb{1}_{\{c_{t+1} \ge 2\}} - \mathbb{1}_{\{c_{t+1} < 2\}}\right] \rho^2.$$

where the expectation is taken over the choice of randomness in a single iteration.

Proof. By the guarantee of the ball-restricted proximal oracle $\mathcal{O}$, we have

$$\mathbb{E}\left[\frac{h_t(z_{t+1}) - h_t(u)}{c_{t+1}}\right] \le \mathbb{E}\left[V_{v_t}(u) - V_{v_{t+1}}(u)\right] - \gamma \mathbb{E}\left[\mathbb{1}_{\{c_{t+1} \ge 2\}} - \mathbb{1}_{\{c_{t+1} < 2\}}\right] \rho^2$$

We will bound the left-hand side of this inequality. First, observe that for any choice of c_{t+1}, z_{t+1},

$$\begin{aligned}\frac{h_t(z_{t+1}) - h_t(u)}{c_{t+1}} &\geq \frac{A'_{t+1} f(\Phi_t(z_{t+1})) - A_t f(x_t) - a_{t+1} f(u)}{c_{t+1}} \\ &= \left(\frac{A'_{t+1}}{c_{t+1}} f(\Phi_t(z_{t+1})) + A_t \frac{c_{t+1} - 1}{c_{t+1}} f(x_t)\right) - A_t f(x_t) - \frac{a_{t+1}}{c_{t+1}} f(u) \\ &\geq A_{t+1} f(x_{t+1}) - A_t f(x_t) - \frac{a_{t+1}}{c_{t+1}} f(u) \\ &= A_{t+1} E_{t+1} - A_t E_t.\end{aligned}$$

where the inequalities follow from the convexity of f. Substituting this in yields

$$\mathbb{E}\left[A_{t+1} E_{t+1} - A_t E_t\right] \leq \mathbb{E}\left[D_t - D_{t+1}\right] - \gamma \mathbb{E}\left[\mathbb{1}_{\{c_{t+1} \geq 2\}} - \mathbb{1}_{\{c_{t+1} < 2\}}\right] \rho^2.$$

and rearranging gives the claim. ∎

Iterating this potential decrease lemma gives a full complexity bound.

LEMMA 4.2. *Let x_T be the output of the above algorithm. We have*

$$\mathbb{E}\left[f(x_T) - f(x_\star)\right] \leq \epsilon.$$

In addition, the algorithm performs at most

$$18 \left(\frac{R}{\sqrt{\gamma} r}\right)^{2/3} \log\left(\frac{80 \mathcal{E}_0}{\epsilon}\right)$$

iterations in expectation, where each iteration calls a ball-restricted proximal oracle (Definition 4.1).

Proof. Let T denote the (random) iteration where the algorithm returns x_T, i.e., the first T for which $A_T \geq \frac{40 R^2 \log(80\mathcal{E}_0/\epsilon)}{\epsilon}$. Define the random process

$$Q_t = P_t + \sum_{i=1}^{t} \gamma \left(\mathbb{1}_{\{c_i \geq 2\}} - \mathbb{1}_{\{c_i < 2\}}\right) \rho^2.$$

We recall that

$$A_{t+1} = A_t + \frac{a_{t+1}}{c_{t+1}} = A_t \left(1 + \frac{1}{c_{t+1}} \left(\frac{\sqrt{\gamma} r}{R}\right)^{2/3}\right) \implies A_{t+1} \geq A_t \exp\left(\frac{1}{2 c_{t+1}} \left(\frac{\sqrt{\gamma} r}{R}\right)^{2/3}\right). \tag{4.12}$$

As $c_{t+1} \leq c_{\max}$ by the definition of the ball-restricted proximal oracle, we observe that with probability 1

$$A_t \geq A_0 \exp\left(\frac{1}{2 c_{\max}} \left(\frac{\sqrt{\gamma} r}{R}\right)^{2/3} t\right).$$

As we terminate when $A_T \geq \frac{40 R^2 \log(80\mathcal{E}_0/\epsilon)}{\epsilon}$, this implies that T is finite with probability 1. Lemma 4.1 implies that Q_t is a supermartingale and therefore, by the optional stopping theorem, we have

$$\mathbb{E}\left[Q_T\right] \leq Q_0 \leq 2R^2. \tag{4.13}$$

Now define

$$T_1 = \sum_{i=1}^{T} \mathbb{1}_{\{c_i \geq 2\}} \quad \text{and} \quad T_2 = \sum_{i=1}^{T} \mathbb{1}_{\{c_i < 2\}}.$$

By definition, we have $T = T_1 + T_2$ and

$$Q_T = P_T + \gamma\rho^2(T_1 - T_2).$$

Now for any iteration with $c_{t+1} < 2$, eq. (4.12) implies

$$A_{t+1} \geq A_t \exp\left(\frac{1}{4}\left(\frac{\sqrt{\gamma}r}{R}\right)^{2/3}\right) \implies A_T \geq A_0 \exp\left(\frac{T_2}{4}\left(\frac{\sqrt{\gamma}r}{R}\right)^{2/3}\right).$$

As $A_{T-1} < \frac{40R^2\log(80\mathcal{E}_0/\epsilon)}{\epsilon}$ and $A_T \leq \left(1 + (\sqrt{\gamma}r/R)^{2/3}\right) A_{T-1} < \frac{80R^2\log(80\mathcal{E}_0/\epsilon)}{\epsilon}$, this implies that with probability 1

$$\begin{aligned}\frac{80R^2\log(80\mathcal{E}_0/\epsilon)}{\epsilon} \geq \frac{R^2}{\mathcal{E}_0}\exp\left(\frac{1}{4}\left(\frac{\sqrt{\gamma}r}{R}\right)^{2/3}T_2\right) \implies T_2 &\leq 4\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0\log(80\mathcal{E}_0/\epsilon)}{\epsilon}\right)\\ &\leq 8\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right)\end{aligned}$$

where the last inequality follows from $\alpha\log\alpha \leq \alpha^2$ for any $\alpha > 0$. This implies

$$\mathbb{E}\left[P_T + \gamma\rho^2 T_1\right] = \mathbb{E}[Q_T + \gamma\rho^2 T_2] \leq 2R^2 + 8\gamma\rho^2\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right) \tag{4.14}$$

where the inequality follows from the above bound on T_2 and eq. (4.13). Now, note that

$$\rho = \left(1 + \left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\right) r < \frac{2R^{2/3}r^{1/3}}{\gamma^{1/3}}$$

as $r < R$. Substituting this into eq. (4.14), we obtain

$$\begin{aligned}\frac{40R^2\log(80\mathcal{E}_0/\epsilon)}{\epsilon}\mathbb{E}\left[f(x_T) - f(x_\star)\right] &\leq \mathbb{E}\left[A_T E_T\right]\\ &\leq \mathbb{E}\left[P_T + \gamma\rho^2 T_1\right] \leq 2R^2 + 8\gamma\rho^2\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right)\\ &\leq 2R^2 + 32R^2\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right) \leq 34R^2\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right)\end{aligned}$$

and therefore $\mathbb{E}\left[f(x_T) - f(x_\star)\right] \leq \epsilon$. In addition, eq. (4.14) also yields

$$\begin{aligned}\mathbb{E}[T_1] &\leq \frac{2R^2}{\gamma\rho^2} + 4\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0\log(80\mathcal{E}_0/\epsilon)}{\epsilon}\right)\\ &\leq \frac{2\gamma^{2/3}R^2}{\gamma R^{4/3}r^{2/3}} + 8\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right) \leq 10\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right).\end{aligned}$$

Thus, the expected number of iterations of the method satisfies

$$\mathbb{E}[T] = \mathbb{E}[T_2] + \mathbb{E}[T_1] \leq 18\left(\frac{R}{\sqrt{\gamma}r}\right)^{2/3}\log\left(\frac{80\mathcal{E}_0}{\epsilon}\right).$$

□

We combine these facts to prove Theorem 4.1.

Proof of Theorem 4.1. Lemma 4.2 implies the first two items in Theorem 4.1. For the third item, we observe that the only nontrivial step of the while loop is on line 7, which performs a single call to $\mathcal{O}$ with gradient estimator $\mathcal{G}_t$ and parameter $\rho = O(r^{1/3}R^{2/3})$. Any t prior to terminating has $A_t = O\left(\frac{R^2 \log(\mathcal{E}_0/\epsilon)}{\epsilon}\right)$. Thus, for any $x \in \mathcal{X}$

$$\begin{aligned}\|\mathcal{G}_t(x)\|_* = a_{t+1}\|\mathcal{G}(\Phi_t(x))\|_* &\le \left(\frac{\sqrt{\gamma}r}{R}\right)^{2/3} A_t G \\ &= O\left(\gamma^{1/3}\frac{r^{2/3}}{R^{2/3}} \cdot \frac{R^2\log(\mathcal{E}_0/\epsilon)}{\epsilon} G\right) = O\left(\frac{\gamma^{1/3}R^{4/3}r^{2/3}\log(\mathcal{E}_0/\epsilon)}{\epsilon} G\right).\end{aligned}$$

□

5 Ball oracle implementation

In this section, we develop Algorithm 2 which implements a ball-restricted proximal introduced in Definition 4.1 in the previous section. The algorithm combines last-iterate proximal mirror descent (LI-MD, Algorithm 3) with a careful bisection procedure (λ-BISECTION in Algorithm 2). For high-level description of the algorithm, see Section 2.4.

Let us briefly describe Algorithms 2 and 3. The λ-BISECTION procedure tries (with high probability) $\widetilde{O}(1)$ values of λ and finds one for which $u^\star_\lambda = \arg\min_{x\in\mathcal{X}} h(x) + \lambda V_y(x)$ satisfies $V_y(u^\star_\lambda) = \widetilde{\Theta}(\rho^2)$. Using this λ we call Algorithm 3 once again to obtain random outputs z, w independent of the random bits that produce λ. By properly choosing the step sizes and number of iterations, we argue that the results satisfy the restricted proximal ball oracle condition (4.11).

Our algorithm has additional properties that enable efficient gradient estimation for the problems we study. First, all iterations stay within a radius-ρ norm ball centered at y, as LI-MD aborts whenever going outside the radius. This enables efficiently sampling from softmax distribution using linear approximation and rejection sampling, see Section 7. Second, due to the "last-iterate" mechanism (which performs iterate averaging before the stochastic gradient queries), the total movement of iterates throughout Algorithm 2 is also bounded by $\widetilde{O}(\rho)$. This movement bound is used for bounding the runtime when querying the data structure that we designed in Algorithm 4 for constructing $\mathcal{G}$.

The formal guarantees of our algorithm require the following notion of *τ-triangle inequality* for Bregman divergences.

DEFINITION 5.1. (τ-TRIANGLE INEQUALITY) *For any $\tau \ge 1$, a domain $\mathcal{X}$ and Bregman divergence V satisfy a τ-triangle inequality, for all $x, y, z \in \mathcal{X}$,*

$$V_x(z) + V_z(x) \le \tau(\min\{V_x(y), V_y(x)\} + \min\{V_y(z), V_z(y)\}). \tag{5.15}$$

With this definition in hand, we state the main guarantees of Algorithm 2.

THEOREM 5.1. *Let $\mathcal{X}$ be a closed convex set, let $h : \mathcal{X} \to \mathbb{R}$ be a convex function with gradient estimator $\mathcal{G}$ that satisfies $\|\mathcal{G}(x)\|_* \le \Gamma$ with probability 1, and let $\mathcal{X}$ and V satisfy a $\tau \ge 4$ triangle inequality (Definition 5.1) as well as $\max_{x,y\in\mathcal{X}} V_x(y) \le R^2$. Let $\hat{T}_k$ to be the number of iterations in the k'th call to LI-MD. Then, for any radius $\rho > 0$, center point $y \in \mathcal{X}$, for error probability $\delta \le \frac{\rho^2}{2^{14}(\sqrt{2}R\Gamma + 3R^2)\tau^5}$, the following holds:*

1. *Algorithm 2 implements a $(\rho, \gamma, c_{\max})$ restricted proximal oracle for function h, with $\gamma = \frac{1}{2^{13}\tau^5}$ and $c_{\max} = \frac{32\tau\Gamma}{\rho}$. That is, the outputs z, w and c of Algorithm 2 satisfy*

   $$\mathbb{E}\frac{h(z) - h(u)}{c} \le \mathbb{E}[V_y(u) - V_w(u)] - \frac{1}{2^{13}\tau^5}\mathbb{E}\left(\mathbb{1}_{\{c\ge 2\}} - \mathbb{1}_{\{c<2\}}\right)\rho^2 \quad \text{for all } u \in \mathcal{X} \tag{5.16}$$

   and $c \le \frac{32\tau\Gamma}{\rho}$ with probability 1.

2. *With probability 1, the queries $x_1^{(k)}, \ldots, x_{\hat{T}_k}^{(k)}$ that Algorithm 3 makes to $\mathcal{G}$ when called in the k'th iteration of Algorithm 2 satisfy*

   $$\|x_t^{(k)} - y\| \le \rho \quad \text{for all } t \le \hat{T}_k \text{ and } k \le K.$$

Algorithm 2: Projection-free ball oracle implementation $\mathcal{O}(\mathcal{G}, y, \rho)$

Input: Objective $h : \mathcal{X} \to \mathbb{R}$ with gradient estimator $\mathcal{G}$, center point $y \in \mathcal{X}$, radius ρ

Parameters: Gradient bound Γ, 1-strongly-convex dgf φ and associated Bregman divergence V, triangle inequality factor τ

Parameters: constant $C = 66 \cdot 2^{12}$, error probability $\delta \le \frac{\rho^2}{2^{14}(\sqrt{2}R\Gamma + R^2)\tau^5}$

$\lambda \leftarrow \lambda$-BISECTION$(\mathcal{G}, y, \rho)$, $\delta_0 \leftarrow \delta$, $\eta \leftarrow \frac{\rho^2 \lambda}{C \cdot \log(16/\delta_0) \cdot \tau^5 \Gamma^2}$, $T \leftarrow \frac{4\tau}{\eta\lambda}$

$(z, w, \mathsf{OutOfBound}) \leftarrow \mathsf{LI\text{-}MD}(\mathcal{G}, \varphi, y, \rho, \lambda, \eta, T)$

return z, w and $c = \lambda + \frac{1}{\eta T}$

function λ-BISECTION$(\mathcal{G}, y, \rho)$

  Set $\lambda_{\max} = \frac{16\tau\Gamma}{\rho}$, $\lambda_0 = \lambda_{\min} = 1$, $\eta_0 = \frac{\rho^2 \lambda_{\min}}{C \cdot \log(16/\delta) \cdot \tau^5 \Gamma^2}$, $T_0 = \frac{4\tau}{\eta_0 \lambda_{\min}}$ and $K_{\max} = \left\lceil \log \frac{9600\tau^3\Gamma^3}{\rho^3} \right\rceil + 1$

  $(z^{(0)}, w^{(0)}, \mathsf{OutOfBound}^{(0)}) \leftarrow \mathsf{LI\text{-}MD}(\mathcal{G}, \varphi, y, \rho, \lambda_0, \eta_0, T_0)$

  if $V_y(z^{(0)}) < \frac{\rho^2}{64\tau}$ **then return** $\lambda_{\min}$

  for $k = 1, \ldots, K_{\max}$ **do**

    $\lambda_k = \frac{1}{2}(\lambda_{\max} + \lambda_{\min})$, $\delta_k = \frac{\delta}{8k^2}$, an $\eta_k = \frac{\rho^2 \lambda_k}{C \cdot \log(16/\delta_k) \cdot \tau^5 \Gamma^2}$ ▷ satisfying $\frac{C \log \frac{16}{\delta_k} \eta_k \Gamma^2}{\lambda_k} = \frac{\rho^2}{\tau^5}$

    $T_k = \frac{4\tau}{\eta_k \lambda_k}$ ▷ satisfying $\frac{2}{\lambda_k \eta_k T_k} = \frac{1}{2\tau}$

    $(z^{(k)}, w^{(k)}, \mathsf{OutOfBound}^{(k)}) \leftarrow \mathsf{LI\text{-}MD}(\mathcal{G}, \varphi, y, \rho, \lambda_k, \eta_k, T_k)$

    if $\mathsf{OutOfBound}^{(k)}$ or $V_y(z^{(k)}) > \frac{\rho^2}{64\tau}$ **then** $\lambda_{\min} = \lambda_k$

    else if $V_y(z^{(k)}) < \frac{\rho^2}{256\tau^3}$ **then** $\lambda_{\max} = \lambda_k$

    else return λ_k

  return $\lambda_{K_{\max}}$ ▷ the probability of reaching this line is less than $\delta/2$

Algorithm 3: Last-iterate proximal mirror descent $\mathsf{LI\text{-}MD}(\mathcal{G}, \varphi, y, \rho, \lambda, \eta, T)$

Input: Objective function $h : \mathcal{X} \to \mathbb{R}$ with gradient estimator $\mathcal{G}$, 1-strongly-convex dgf φ (and associated Bregman divergence V), center point $y \in \mathcal{X}$, radius ρ, regularization parameter $\lambda \ge 0$, step size η, iteration budget T

Set $w_0 = x_0 = y$

Set $\mathsf{OutOfBound} = \mathsf{False}$ ▷ monitor if iterations go out of ρ-radius from center y

for $t = 1, \ldots, T$ **do**

  $x_t = \frac{1}{t}\sum_{i=0}^{t-1} w_i = \frac{t-1}{t} x_{t-1} + \frac{1}{t} w_{t-1}$

  if $\|x_t - y\| \ge \rho$ **then** $\mathsf{OutOfBound} = \mathsf{True}$ **break**

  $\hat{g}_t = \mathcal{G}(x_t)$ ▷ $g_t := \mathbb{E}[\hat{g}_t \mid x_t] \in \partial h(x_t)$

  $w_t = \operatorname{argmin}_{w \in \mathcal{X}} \left\{ \eta[\langle \hat{g}_t, w \rangle + \lambda V_y(w)] + V_{w_{t-1}}(w) \right\}$

$\tilde{w}_T = \operatorname{argmax}_{v \in \mathcal{X}} \left\{ \left\langle \frac{\sum_{t=1}^T \nabla\varphi(w_t) + \frac{1}{\lambda\eta}\nabla\varphi(w_T)}{T + \frac{1}{\lambda\eta}}, v \right\rangle - \varphi(v) \right\} = \nabla\varphi^* \left(\frac{\sum_{t=1}^T \nabla\varphi(w_t) + \frac{1}{\lambda\eta}\nabla\varphi(w_T)}{T + \frac{1}{\lambda\eta}} \right)$

if $\mathsf{OutOfBound} = \mathsf{False}$ **then return** x_T, $\tilde{w}_T$, $\mathsf{OutOfBound} = \mathsf{False}$.

$z = y + \rho \frac{x_t - y}{\|x_t - y\|}$ ▷ Arbitrarily selecting a point with distance ρ from y

return z, z, $\mathsf{OutOfBound} = \mathsf{True}$ ▷ return arbitrary point if outside radius ρ

3. *With probability* 1, *the sequences* $\{x_1^{(k)}, \dots, x_{\hat{T}_k}^{(k)}\}_{k=0}^K$ *defined above satisfy*

$$\sum_{k=0}^{K}\sum_{t=1}^{\hat{T}_k}\|x_t^{(k)} - x_{t-1}^{(k)}\| \le \rho \cdot 2K_{\max}\log\frac{4C\log(16K_{\max}^2/\delta)\tau^6\Gamma^2}{\rho^2},$$

where $K_{\max} = \lceil \log\frac{9600\tau^3\Gamma^3}{\rho^3}\rceil + 1$.

4. *Algorithm* 2 *makes at most* $O\Big(\frac{\Gamma^2}{\rho^2}\cdot\tau^6\Big(\log\frac{1}{\delta} + \log\log\frac{\tau\Gamma}{\rho}\Big)\cdot\log\frac{\tau\Gamma}{\rho}\Big)$ *calls to* $\mathcal{G}$ *and the same number of mirror-descent steps.*

The remainder of this section is organized as follows. First, in Section 5.1 we give two examples of Bregman divergences satisfying the τ-triangle inequality and calculate the particular values of τ in difference cases. Then we analyze Algorithm 3 and the λ-BISECTION procedure in Sections 5.2 and 5.3, respectively. Finally, in Section 5.4 we combine those results to prove the main proposition of the section.

5.1 Divergences satisfying τ-triangle inequality. Throughout the paper we mainly consider two divergences, $V_x(y) = \frac{1}{2}\|x-y\|^2$ for the ball setup and $V_x(y) = \sum_{i\in[d]} y_i\log(y_i/x_i)$ for the simplex setup. In this section we show both divergences satisfy τ-triangle inequality with $\tau = \widetilde{\Theta}(1)$.

EXAMPLE 5.1. (EUCLIDEAN SETUP WITH ℓ_2-NORM-SQUARED) *Any* $\mathcal{X}\subseteq\mathbb{R}^d$ *and* $V_x(y) = \frac{1}{2}\|x-y\|_2^2$, *satisfy a* τ*-triangle inequality with* $\tau = 4$.

EXAMPLE 5.2. (TRUNCATED SIMPLEX SETUP WITH KL-DIVERGENCE) *For any* $\nu\in(0,1/4]$, *the simplex* $\Delta_\nu^d := \{x\in\Delta^d, x\ge\nu\mathbf{1}\}$ *and KL-divergence* $V_x(y) = \sum_{i\in[d]} y_i\log\frac{y_i}{x_i}$ *satisfy a* τ*-triangle inequality with* $\tau = 6\log(\nu^{-1})$.

Example 5.1 is an immediate consequence of the standard triangle inequality, but Example 5.2 is less obvious and stems from the following connection between KL-divergence to squared Hellinger distance.

LEMMA 5.1. *Given* $\nu\in(0,1/4]$ *and any* $x,y\in\Delta_\nu^d$, *consider the KL-divergence* $V_x(y) = \sum_{i\in[d]} y_i\log\frac{y_i}{x_i}$ *and squared Hellinger distance* $H^2(x,y) = \frac{1}{2}\|\sqrt{x}-\sqrt{y}\|^2$. *Then*

$$V_x(y) + V_y(x) \le \big(6\log\tfrac{1}{\nu}\big)H^2(x,y).$$

Proof. We have

$$V_x(y)+V_y(x) = \sum_{i\in[d]}(y_i-x_i)\log\frac{y_i}{x_i} = \frac{1}{2}\sum_{i\in[d]} f\left(\frac{y_i}{x_i}\right)\left(\sqrt{y_i}-\sqrt{x_i}\right)^2, \quad \text{where} \quad f(t) = \frac{2(t-1)\log t}{(\sqrt{t}-1)^2}.$$

The lemma follows from noting that $\max_{t\in[\nu,1/\nu]} f(t) = \frac{2(\frac{1}{\nu}-1)\log\frac{1}{\nu}}{\left(\sqrt{\frac{1}{\nu}}-1\right)^2} \le 6\log\frac{1}{\nu}$. ∎

Proof of Example 5.2. We first use the AM-GM inequality of Hellinger distance, which gives $2H^2(x,z)+2H^2(z,y)\ge H^2(x,y)$, and consequently we have

$$\min(V_x(z),V_z(x)) + \min(V_y(z),V_z(y)) \overset{(i)}{\ge} 2H^2(x,z)+2H^2(z,y) \ge H^2(x,y) \overset{(ii)}{\ge} \frac{1}{6\log(\nu^{-1})}(V_x(y)+V_y(x)).$$

Here we use (i) the well-established inequality $V_x(z)\ge 2H^2(x,z)$ (see, e.g. Reiss [50]), and (ii) the inequality shown in Lemma 5.1. This proves the desired claim. ∎

We remark that any divergence V satisfying the τ-triangle inequality on $\mathcal{X}$ is also symmetric in its arguments up to factor τ, formally stated as follows.

COROLLARY 5.1. *For any closed convex set* $\mathcal{X}$ *and some Bregman divergence* V *on* $\mathcal{X}$ *satisfying a* τ*-triangle inequality, then* $\frac{1}{\tau}V_x(y)\le V_y(x)\le\tau V_x(y)$.

Proof. We can apply the definition of τ-triangle inequality with $z=y$ to get that $\min\{V_x(y),V_y(x)\} + \min\{V_y(y),V_y(y)\} \ge \frac{1}{\tau}(V_x(y)+V_y(x)) \ge \frac{1}{\tau}V_x(y)$, which implies the first inequality. The second inequality follows by symmetry. ∎

5.2 Analysis of Algorithm 3. In this section, we provide the main analysis and guarantees for LI-MD (Algorithm 3). The first lemma is a deterministic error bound for last-iterate proximal mirror descent. Throughout the analysis we use

$$H_\lambda(x) \coloneqq h(x) + \lambda V_y(x) \quad \text{and} \quad u_\lambda^\star \coloneqq \arg\min_{x\in\mathcal{X}} H_\lambda(x)$$

for the regularized objective function and its minimizer, respectively.

LEMMA 5.2. *Let $\mathcal{X}$ be a closed convex set, and $h : \mathcal{X} \to \mathbb{R}$ be a convex function with gradient estimator $\mathcal{G}$ that satisfies $\|\mathcal{G}(x)\|_* \le \Gamma$ with probability 1, and let $y \in \mathcal{X}$ and $\lambda \ge 0$. The iterates of Algorithm 3 satisfy, for all $u \in \mathcal{X}$,*

$$H_\lambda(x_T) - H_\lambda(u) \le -\frac{\lambda}{T}\sum_{t=1}^{T} V_{w_t}(u) + \frac{V_y(u) - V_{w_T}(u)}{\eta T} + \frac{\eta}{2}\Gamma^2 + \frac{1}{T}\sum_{t=1}^{T}\langle g_t - \hat{g}_t, w_{t-1} - u\rangle . \tag{5.17}$$

Proof. At iteration $t \in [T]$, the optimality condition for each iteration of Line 7 gives, for any $u \in \mathcal{X}$,

$$\left\langle \eta\hat{g}_t + \eta\lambda\nabla V_y(w_t) + \nabla V_{w_{t-1}}(w_t), w_t - u\right\rangle \le 0,$$

which by rearranging terms implies

$$\langle \hat{g}_t + \lambda\nabla V_y(w_t), w_t - u\rangle \le \frac{1}{\eta}\left\langle -\nabla V_{w_{t-1}}(w_t), w_t - u\right\rangle = \frac{1}{\eta}\left(V_{w_{t-1}}(u) - V_{w_t}(u) - V_{w_{t-1}}(w_t)\right), \tag{5.18}$$

where we use the three-point equality following the definition of Bregman divergence for the last equality.

Now, for the terms on the LHS of (5.18), by applying three-point equality again,

$$\lambda\langle\nabla V_y(w_t), w_t - u\rangle = \lambda\left(V_{w_t}(u) + V_y(w_t) - V_y(u)\right). \tag{5.19}$$

By rearranging terms

$$\begin{aligned}
\langle \hat{g}_t, w_t - u\rangle &= \langle \hat{g}_t, w_t - w_{t-1}\rangle + \langle g_t, w_{t-1} - u\rangle + \langle \hat{g}_t - g_t, w_{tc-1} - u\rangle \\
&\overset{(i)}{=} \langle \hat{g}_t, w_t - w_{t-1}\rangle + \langle g_t, tx_t - (t-1)x_{t-1} - u\rangle + \langle \hat{g}_t - g_t, w_{t-1} - u\rangle \\
&\overset{(ii)}{\ge} -\frac{\eta}{2}\|\hat{g}_t\|_*^2 - \frac{1}{2\eta}\|w_t - w_{t-1}\|^2 + (t-1)\left(h(x_t) - h(x_{t-1})\right) + \left(h(x_t) - h(u)\right) + \langle \hat{g}_t - g_t, w_{t-1} - u\rangle \\
&\overset{(iii)}{\ge} -\frac{\eta}{2}\|\hat{g}_t\|_*^2 - \frac{1}{\eta}V_{w_{t-1}}(w_t) + (t-1)\left(h(x_t) - h(x_{t-1})\right) + \left(h(x_t) - h(u)\right) + \langle \hat{g}_t - g_t, w_{t-1} - u\rangle .
\end{aligned} \tag{5.20}$$

Here we use (i) the relation that $tx_t = (t-1)x_{t-1} + w_{t-1}$, (ii) the AM-GM inequality and convexity of h, and (iii) the 1-strong-convexity of the distance generating function.

Plugging Equations (5.19) and (5.20) back into Equation (5.18) and rearranging terms,

$$\begin{aligned}
&t(h(x_t) - h(u)) - (t-1)(h(x_{t-1}) - h(u)) \\
&\qquad \le \lambda\left(V_y(u) - V_{w_t}(u) - V_y(w_t)\right) + \frac{1}{\eta}\left(V_{w_{t-1}}(u) - V_{w_t}(u) - V_{w_{t-1}}(w_t)\right) \\
&\qquad\quad + \frac{\eta}{2}\|\hat{g}_t\|_*^2 + \frac{1}{\eta}V_{w_{t-1}}(w_t) + \langle g_t - \hat{g}_t, w_{t-1} - u\rangle \\
&\qquad \le \lambda\left(V_y(u) - V_{w_t}(u) - V_y(w_t)\right) + \frac{1}{\eta}\left(V_{w_{t-1}}(u) - V_{w_t}(u)\right) + \frac{\eta}{2}\Gamma^2 + \langle g_t - \hat{g}_t, w_{t-1} - u\rangle .
\end{aligned}$$

Here for the last inequality we use $\|\hat{g}_t\|_* \le \Gamma$ by definition of the gradient estimator.

Averaging over $t \in [T]$, we have for any $u \in \mathcal{X}$,

$$\begin{aligned} h(x_T) - h(u) \leq & \lambda V_y(u) - \frac{\lambda}{T}\sum_{t\in[T]} V_{w_t}(u) - \frac{\lambda}{T}\sum_{t\in[T]} V_y(w_t) \\ & + \frac{1}{\eta T}\left(V_y(u) - V_{w_T}(u)\right) + \frac{\eta}{2}\Gamma^2 + \frac{1}{T}\sum_{t\in[T]} \langle g_t - \hat{g}_t, w_{t-1} - u\rangle, \\ \leq & \lambda V_y(u) - \frac{\lambda}{T}\sum_{t\in[T]} V_{w_t}(u) - \lambda V_y(x_T) \\ & + \frac{1}{\eta T}\left(V_y(u) - V_{w_T}(u)\right) + \frac{\eta}{2}\Gamma^2 + \frac{1}{T}\sum_{t\in[T]} \langle g_t - \hat{g}_t, w_{t-1} - u\rangle, \end{aligned}$$

where the last inequality is due to the convexity of $V_y(\cdot)$, $V_y(w_0) = 0$ and $x_T = \frac{1}{T}\sum_{t=0}^{T-1} w_t$ so that $\frac{1}{T}\sum_{t\in[T]} V_y(w_t) \geq \frac{1}{T}\sum_{t=0}^{T-1} V_y(w_t) \geq V_y(x_T)$. Rearranging terms concludes the proof. ∎

Combining with τ-triangle inequality of divergence V, we can get the following in-expectation progress guarantee (5.21).

COROLLARY 5.2. *In the setting of Theorem 5.1, the outputs z, w and* OutOfBound *of Algorithm 3 satisfy*

$$\begin{aligned} \mathbb{E}h(z) - h(u) \leq & \left(\lambda + \frac{1}{\eta T}\right)\mathbb{E}[V_y(u) - V_w(u)] + \eta\Gamma^2 - \left(\frac{\lambda}{\tau} - \frac{1}{\eta T}\right)\mathbb{E}V_y(u_\lambda^\star) \\ & + \left(\sqrt{2}R\Gamma + \left(\lambda + \frac{1}{\eta T}\right)R^2\right)\mathbb{P}(\textsf{OutOfBound} = \textsf{True}). \end{aligned} \tag{5.21}$$

Proof. We first consider an alternative "imaginary" algorithm which continues even if OutOfBound becomes True (i.e., we go outside of radius-ρ ball) and deterministically terminate after T iterations, outputting $x_T, \tilde{w}_T$. For such an "imaginary" algorithm we have $\mathbb{E}[\langle g_t - \hat{g}_t, w_{t-1} - u\rangle \,|w_{t-1}, x_{t-1}] = 0$, thus by taking expectation on Lemma 5.2,

$$\mathbb{E}h(x_T) - h(u) \leq \mathbb{E}\left[\left(\lambda + \frac{1}{\eta T}\right)V_y(u) - \left(\frac{\lambda}{T}\sum_{t\in[T]} V_{w_t}(u) + \frac{1}{\eta T}V_{w_T}(u)\right) + \frac{\eta}{2}\Gamma^2 - \lambda V_y(x_T)\right]. \tag{5.22}$$

Standard tools from convex analysis imply that φ^* (the dual function of φ), and its induced Bregman divergence $V_a^{\varphi^*}(a') = \varphi^*(a') - \varphi^*(a) - \langle\nabla\varphi^*(a), a' - a\rangle$ satisfy

$$V_a(b) = V_{\nabla\varphi(b)}^{\varphi^*}(\nabla\varphi(a))$$

for any $a, a' \in \mathcal{X}^*$ [52]. Now,

$$\begin{aligned} \frac{\lambda}{T}\sum_{t\in[T]} V_{w_t}(u) + \frac{1}{\eta T}V_{w_T}(u) & \overset{(i)}{=} \frac{\lambda}{T}\sum_{t\in[T]} V_{\nabla\varphi(u)}^{\varphi^*}(\nabla\varphi(w_t)) + \frac{1}{\eta T}V_{\nabla\varphi(u)}^{\varphi^*}(\nabla\varphi(w_T)) \\ & \overset{(ii)}{\geq} \left(\lambda + \frac{1}{\eta T}\right)\cdot V_{\nabla\varphi(u)}^{\varphi^*}\left(\frac{\frac{\lambda}{T}\sum_{t\in[T]}\nabla\varphi(w_t) + \frac{1}{\eta T}\nabla\varphi(w_T)}{\lambda + \frac{1}{\eta T}}\right) \\ & \overset{(iii)}{=} \left(\lambda + \frac{1}{\eta T}\right)V_{\nabla\varphi(u)}^{\varphi^*}(\nabla\varphi(\tilde{w}_T)) = \left(\lambda + \frac{1}{\eta T}\right)V_{\tilde{w}_T}(u). \end{aligned}$$

Here we use (i) the equality $V_a(b) = V_{\nabla\varphi(b)}^{\varphi^*}(\nabla\varphi(a))$, (ii) the convexity of $V_x(\cdot)$ and (iii) the definition of $\tilde{w}_T$ as in line 8 of Algorithm 3. Plugging this back into Equation (5.22) proves the expected guarantee for the "imaginary" algorithm:

$$\mathbb{E}h(x_T) - h(u) \leq \left(\lambda + \frac{1}{\eta T}\right)\mathbb{E}[V_y(u) - V_{\tilde{w}_T}(u)] + \frac{\eta}{2}\Gamma^2 - \lambda\mathbb{E}V_y(x_T).$$

Further, applying strong convexity of H_λ, we have $H_\lambda(x_T) - H_\lambda(u_\lambda^\star) \geq \lambda V_{u_\lambda^\star}(x_T)$. Combining it with Equation (5.17) (where we choose $u = u_\lambda^\star$), we have

$$\lambda V_{u_\lambda^\star}(x_T) \leq \frac{V_y(u_\lambda^\star)}{\eta T} + \frac{\eta}{2}\Gamma^2 + \frac{1}{T}\sum_{t\in[T]} \langle g_t - \hat{g}_t, w_{t-1} - u\rangle .$$

Taking expectation yields,

$$\lambda \mathbb{E} V_{u_\lambda^\star}(x_T) \leq \frac{V_y(u_\lambda^\star)}{\eta T} + \frac{\eta}{2}\Gamma^2. \tag{5.23}$$

By the τ-triangle inequality, we also have

$$\frac{\lambda}{\tau} V_y(u_\lambda^\star) \leq \lambda V_y(x_T) + \lambda V_{u_\lambda^\star}(x_T). \tag{5.24}$$

Combining Equations (5.23) and (5.24) we have

$$\frac{\lambda}{\tau} V_y(u_\lambda^\star) \leq \lambda \mathbb{E} V_y(x_T) + \frac{V_y(u_\lambda^\star)}{\eta T} + \frac{\eta}{2}\Gamma^2 \quad \Longrightarrow \quad -\lambda \mathbb{E} V_y(x_T) \leq -\left(\frac{\lambda}{\tau} - \frac{1}{\eta T}\right) V_y(u_\lambda^\star) + \frac{\eta}{2}\Gamma^2.$$

Plugging this back into Equation (5.21) proves the following guarantee for the output $x_T, \tilde{w}_T$ of the "imaginary" algorithm.

$$\mathbb{E} h(x_T) - h(u) \leq \left(\lambda + \frac{1}{\eta T}\right)\mathbb{E}[V_y(u) - V_{\tilde{w}_T}(u)] + \eta\Gamma^2 - \left(\frac{\lambda}{\tau} - \frac{1}{\eta T}\right)\mathbb{E} V_y(u_\lambda^\star). \tag{5.25}$$

Now considering the original algorithm, the iterates will behave exactly the same when $\mathsf{OutOfBound} = \mathsf{False}$ for all iterations. When $\mathsf{OutOfBound} = \mathsf{True}$ the actual algorithm returns an arbitrary point x_t incurs a loss bounded by $h(x_t) - h(x_T) + (\lambda + \frac{1}{\eta T}) V_{x_t}(u) \leq \sqrt{2} R\Gamma + (\lambda + \frac{1}{\eta T}) R^2$. Thus, we have for the claimed bound for the actual algorithm's output iterates z, w, i.e.,

$$\begin{aligned}\mathbb{E} h(z) - h(u) \leq & \left(\lambda + \frac{1}{\eta T}\right)\mathbb{E}[V_y(u) - V_w(u)] + \eta\Gamma^2 - \left(\frac{\lambda}{\tau} - \frac{1}{\eta T}\right)\mathbb{E} V_y(u_\lambda^\star) - \lambda \mathbb{E} V_y(x_T) \\ & + \left(\sqrt{2} R\Gamma + \left(\lambda + \frac{1}{\eta T}\right) R^2\right) \mathbb{P}(\mathsf{OutOfBound} = \mathsf{True}).\end{aligned}$$

□

Next, we bound the term $\sum_{t=1}^T \langle g_t - \hat{g}_t, w_{t-1} - u\rangle$ on the RHS of Equation (5.17) using concentration of measure. This is formally stated in the next lemma; we defer its proof to the end of this subsection.

LEMMA 5.3. *In the setting of Lemma 5.2, for any $\delta, \varepsilon \in (0,1)$ and $u \in \mathcal{X}$, we have*

$$\mathbb{P}\left(\mathcal{E}(\delta) \coloneqq \left\{ \max_{1\leq t\leq T} \left| \sum_{i=1}^t \langle \hat{g}_i - g_i, w_{i-1} - u\rangle \right| \leq \Gamma \max_{0\leq i<T} \|w_i - u\| \sqrt{32T \log\frac{2}{\delta}} \right\}\right) \geq 1 - \delta. \tag{5.26}$$

Combining Equation (5.17) in Lemma 5.2 with the concentration guarantees in Lemma 5.3, we show the iteration $\{w_t\}_{t\in[T]}$ and x_T stay relatively close to the true optimizer $u_\lambda^\star$ in the following.

LEMMA 5.4. *In the setting of Lemma 5.2 and Lemma 5.3, let $u_\lambda^\star \coloneqq \operatorname{argmin}_{x\in\mathcal{X}} H_\lambda(x)$. For any $\delta \in (0,1)$ and $T \geq 1$, when event $\mathcal{E}(\delta)$ happens,*

$$\max_{0\leq t\leq T} V_{w_t}(u_\lambda^\star) \leq 2V_y(u_\lambda^\star) + \left(65 \log\frac{2}{\delta}\right)\eta^2\Gamma^2 T$$

and

$$\lambda V_{u_\lambda^\star}(x_T) \leq \frac{2V_y(u_\lambda^\star)}{\eta T} + \left(66 \log\frac{2}{\delta}\right)\eta\Gamma^2.$$

Proof. For the first inequality, we follow Equation (5.17), due to $H_\lambda(x_T) - H_\lambda(u_\lambda^\star) \geq 0$ and the non-negativity of Bregman divergences, we have

$$V_{w_T}(u_\lambda^\star) \leq V_y(u_\lambda^\star) + \frac{\eta^2}{2}\Gamma^2 T + \eta \sum_{t\in[T]} \langle g_t - \hat{g}_t, w_{t-1} - u_\lambda^\star \rangle .$$

Applying the same argument for all $t \in [T]$ gives

$$V_{w_t}(u_\lambda^\star) \leq V_y(u_\lambda^\star) + \frac{\eta^2}{2}\Gamma^2 t + \eta \sum_{i\in[t]} \langle g_i - \hat{g}_i, w_{i-1} - u_\lambda^\star \rangle , \quad \text{for all } t \in [T].$$

Applying Lemma 5.3 with $u = u_\lambda^\star$, we have under the event $\mathcal{E}(\delta)$,

$$\begin{aligned}
\max_{t\in[T]} V_{w_t}(u_\lambda^\star) &\leq V_y(u_\lambda^\star) + \frac{\eta^2}{2}\Gamma^2 T + \eta \max_{t\in[T]} \left| \sum_{i\in[t]} \langle g_i - \hat{g}_i, w_{i-1} - u_\lambda^\star \rangle \right| \\
&\leq V_y(u_\lambda^\star) + \frac{\eta^2}{2}\Gamma^2 T + \eta\Gamma \max_{0\leq t\leq T} \|w_t - u_\lambda^\star\| \sqrt{8T \log\frac{2}{\delta}} \\
&\overset{(i)}{\leq} V_y(u_\lambda^\star) + \frac{\eta^2}{2}\Gamma^2 T + \eta^2\Gamma^2 \cdot (32T \log\frac{2}{\delta}) + \max_{0\leq t\leq T} \frac{1}{4}\|w_t - u_\lambda^\star\|^2 \\
&\overset{(ii)}{\leq} V_y(u_\lambda^\star) + \left(\frac{65}{2}\log\frac{2}{\delta}\right)\eta^2\Gamma^2 T + \max_{0\leq t\leq T} \frac{1}{2} V_{w_t}(u_\lambda^\star).
\end{aligned} \tag{5.27}$$

Here we use (i) the AM-GM inequality and (ii) the strong convexity of Bregman divergence by definition. Note the RHS in Equation (5.27) also upper bounds $V_{w_0}(u_\lambda^\star)$ since $w_0 = y$ in the initialization of Algorithm 3. Combining these together and rearranging terms,

$$\max_{0\leq t\leq T} V_{w_t}(u_\lambda^\star) \leq 2V_y(u_\lambda^\star) + \left(65\log\frac{2}{\delta}\right)\eta^2\Gamma^2 T,$$

thus proving the first inequality.

For the second inequality, we note by strong convexity, $H_\lambda(x_T) - H_\lambda(u_\lambda^\star) \geq \lambda V_{u_\lambda^\star}(x_T)$, plugging this back into Equation (5.17) and again using non-negativity of Bregman divergences and similar arguments following Lemma 5.3, we have when event $\mathcal{E}(\delta)$ happens,

$$\begin{aligned}
\lambda V_{u_\lambda^\star}(x_T) &\leq \frac{V_y(u_\lambda^\star)}{\eta T} + \frac{\eta}{2}\Gamma^2 + \frac{1}{T}\sum_{t\in[T]} \langle g_t - \hat{g}_t, w_{t-1} - u_\lambda^\star \rangle \\
&\leq \frac{V_y(u_\lambda^\star)}{\eta T} + \frac{\eta}{2}\Gamma^2 + \frac{1}{T}\Gamma \max_{0\leq i\leq T} \|w_i - u_\lambda^\star\| \sqrt{32T\log\frac{2}{\delta}} \\
&\overset{(i)}{\leq} \frac{V_y(u_\lambda^\star)}{\eta T} + \frac{\eta}{2}\Gamma^2 + 32\left(\log\frac{2}{\delta}\right)\eta\Gamma^2 + \frac{1}{2\eta T}\max_{0\leq i\leq T} V_{w_i}(u_\lambda^\star) \\
&\overset{(ii)}{\leq} \frac{2V_y(u_\lambda^\star)}{\eta T} + 65\left(\log\frac{2}{\delta}\right)\eta\Gamma^2.
\end{aligned}$$

Here we use the Cauchy-Schwarz inequality for (i) and the first inequality proven for (ii). This concludes the proof for the second inequality. ∎

We use Lemma 5.4 to control the possibility of LI-MD going out of bounds when $u_\lambda^\star$ is not too far from the center point y.

LEMMA 5.5. *In the setting of Lemma 5.4, if* $V_y(u_\lambda^\star) \leq \frac{\rho^2}{16}$ *and for some* $\delta \in (0,1)$ *we have* $\left(\log\frac{2}{\delta}\right)\eta^2\Gamma^2 T \leq \frac{\rho^2}{65\cdot 16}$ *then the event* $\mathcal{E}(\delta)$ *implies that* OutOfBound = False.

Proof. We have $\mathsf{OutOfBound} = \mathsf{False}$ if and only if $\max_{t\le T}\|x_t - y\| \le \rho$. To derive a sufficient condition for this inequality we upper bound $\max_{t\le T}\|x_t - y\|$ as follows:

$$\max_{t\le T}\|x_t - y\| \overset{(i)}{\le} \max_{t<T}\|w_t - y\| \le \|u_\lambda^\star - y\| + \max_{t<T}\|w_t - u_\lambda^\star\| \overset{(ii)}{\le} \sqrt{2V_y(u_\lambda^\star)} + \sqrt{2\max_{t<T} V_{w_t}(u_\lambda^\star)},$$

where (i) follows by convexity and the definition of x_t as the averaging of $w_0, \dots, w_{t-1}$, and (ii) follows from the 1-strong-convexity of the distance generating function.

Next, we apply Lemma 5.4 and the assumption $\left(\log\frac{2}{\delta}\right)\eta^2\Gamma^2 T \le \frac{\rho^2}{65\cdot 16}$ to obtain that $\mathcal{E}(\delta)$ implies

$$\max_{t<T} V_{w_t}(u_\lambda^\star) \le 2V_y(u_\lambda^\star) + \frac{\rho^2}{16}.$$

Substituting $V_y(u_\lambda^\star) \le \frac{\rho^2}{16}$ and combining the above displays yields

$$\max_{t\le T}\|x_t - y\| \le \frac{\rho}{\sqrt{8}} + \sqrt{\frac{\rho^2}{4} + \frac{\rho^2}{8}} \le \rho$$

as required. ∎

Finally, the next lemma bounds the total movement of iterations $\{x_t\}_{t\in[T]}$:

LEMMA 5.6. *In the setting of Lemma 5.2, we have, for any $u \in \mathcal{X}$,*

$$\sum_{t=1}^{T}\|x_t - x_{t-1}\| \le 2(\log T + 1)\max_{0\le t\le T}\|w_t - u\|. \tag{5.28}$$

Proof. By definition of x_t, we have $x_t - x_{t-1} = \frac{1}{t}(w_{t-1} - x_{t-1})$, consequently by triangle inequality we have

$$\sum_{t=1}^{T}\|x_t - x_{t-1}\| = \sum_{t\in[T]}\frac{1}{t}\|w_{t-1} - x_{t-1}\| \le \sum_{t\in[T]}\frac{1}{t}\|w_{t-1} - u\| + \sum_{t\in[T-1]}\frac{1}{t}\|x_{t-1} - u\|.$$

We proceed to bound the two terms on the RHS respectively. For the first term,

$$\sum_{t\in[T]}\frac{1}{t}\|w_{t-1} - u\| \le \left(\sum_{t\in[T]}\frac{1}{t}\right)\max_{0\le t\le T-1}\|w_t - u\| \le (\log T + 1)\max_{0\le t\le T}\|w_t - u\|.$$

For the second term,

$$\sum_{t\in[T]}\frac{1}{t}\|x_{t-1} - u\| \le \left(\sum_{t\in[T]}\frac{1}{t}\right)\max_{0\le t\le T-1}\|x_t - u\| \overset{(\star)}{\le} (\log T + 1)\max_{0\le t\le T}\|w_t - u\|,$$

where we also use convexity of the norm function $\|\cdot\|$ and the fact that $x_{t-1} = \frac{1}{t-1}\sum_{i=0}^{t-2} w_i$ for $(\star)$. Summing the two terms proves the claimed bound. ∎

Proof of Lemma 5.3. We consider the random variable $X_i := \frac{1}{2\Gamma\max_{0\le j\le i-1}\|w_j - u\|}\langle g_i - \hat{g}_i, w_{i-1} - u\rangle$ and the filtration $\mathcal{F}_{i-1} := \sigma(x_0, w_0, x_1, w_1, \cdots, w_{i-1}, x_i)$. Note we have $\mathbb{E}[X_i|\mathcal{F}_{i-1}] = 0$ and additionally $|X_i| \le \frac{\|g_i - \hat{g}_i\|_*}{2\Gamma} \le 1$ with probability 1. Thus, applying Blackwell's inequality (cf. Blackwell [6] Theorem 1), we have for any $a, b > 0$,

$$\mathbb{P}\left(\exists\, t \in [T], \left|\sum_{i\in[t]} X_i\right| \le a + bt\right) \le 2e^{-2ab}.$$

Replacing $a = \sqrt{T\log(2/\delta)/2}$, $b = \sqrt{\log(2/\delta)/2T}$, with probability $1-\delta$, we have for all $t \in [T]$,

$$\left|\sum_{i\in[t]} X_i\right| \le \sqrt{T\log(2/\delta)/2} + \sqrt{\log(2/\delta)/2T}\cdot t \le \sqrt{2T\log(2/\delta)}.$$

Now applying Lemma 5 of Ivgi et al. [30] with $a_i = 2\Gamma \max_{0\le j\le i-1}\|w_j - u\|$ and $b_i = X_i$, we have

$$\begin{aligned}\left|\sum_{i\in[t]}\langle g_i - \hat{g}_i, w_{i-1}-u\rangle\right| &\le 4\Gamma \max_{0\le i\le t-1}\|w_i-u\|\cdot\max_{1\le i\le t}\left|\sum_{j\in[i]} X_j\right| \\ &\le \Gamma \max_{0\le i\le T-1}\|w_i-u\|\sqrt{32T\log\frac{2}{\delta}} \quad \text{for all } t\in[T].\end{aligned}$$

Taking maximum over all $t \in [T]$ gives the desired claim. ∎

5.3 Analysis of λ-Bisection. In this section, we prove the correctness and bound the number of iterations for λ-Bisection in Algorithm 2. We use $\mathcal{E}_k(\delta)$ to denote the probabilistic event described in Lemma 5.3 with parameter δ when calling LI-MD$(\mathcal{G}, y, \rho, \lambda_k, \eta_k, T_k)$, which according to that lemma happens with probability at least $1-\delta$. In the next lemma, we first show that if the stopping criterion of the binary search holds for some k and λ_k, then with high probability the value of $V_y(u^\star_{\lambda_k})$ is $\Theta(\rho^2/\text{poly}(\tau))$.

Lemma 5.7. *Assume $\mathcal{X}$ and V satisfy a τ-triangle inequality. For $\delta_k \in (0,1)$, under the event $\mathcal{E}_k(\delta_k/8)$, at iteration k of Algorithm 2 the call to* LI-MD *outputs $z^{(k)}$ such that if $V_y(z^{(k)}) \le \frac{\rho^2}{64\tau}$ then $V_y(u^\star_{\lambda_k}) \le \frac{\rho^2}{16}$ and if $V_y(z^{(k)}) \ge \frac{\rho^2}{256\tau^3}$ then $V_y(u^\star_{\lambda_k}) \ge \frac{\rho^2}{1024\tau^4}$.*

Proof. We begin by noting that $V_y(z^{(k)}) \le \frac{\rho^2}{64\tau}$ implies that $\|z^{(k)}-y\| < \rho$ and therefore that $\mathsf{OutOfBound}^{(k)} = \mathsf{False}$ and $z^{(k)} = x^{(k)}_{T_k}$, i.e., the last iterate of LI-MD. This allows us to apply Lemma 5.4 to bound, in the event $\mathcal{E}_k(\delta_k/8)$,

$$V_{u^\star_{\lambda_k}}(z^{(k)}) \le \frac{2V_y(u^\star_{\lambda_k})}{\lambda_k\eta_k T_k} + \left(66\log\frac{16}{\delta_k}\right)\frac{\eta_k}{\lambda_k}\Gamma^2 \le \frac{1}{2\tau}V_y(u^\star_{\lambda_k}) + \frac{\rho^2}{1024\tau^4}.$$

To upper bound $V_y(u^\star_{\lambda_k})$ we use the τ-triangle inequality, $V_y(z^{(k)}) \le \frac{\rho^2}{64\tau}$ and the bound on $V_{u^\star_{\lambda_k}}(z^{(k)})$ to write

$$V_y(u^\star_{\lambda_k}) \le \tau\left(V_y(z^{(k)}) + V_{u^\star_{\lambda_k}}(z^{(k)})\right) \le \frac{\rho^2}{36} + \frac{1}{2}V_y(u^\star_{\lambda_k}) + \frac{\rho^2}{1024}.$$

Rearranging yields $V_y(u^\star_{\lambda_k}) \le \frac{\rho^2}{16}$ as required.

To lower bound $V_y(u^\star_{\lambda_k})$ we combine the τ-triangle with the assumed lower bound on $V_y(z^{(k)})$,

$$\begin{aligned}V_y(u^\star_{\lambda_k}) &\ge \frac{1}{\tau}V_y(z^{(k)}) - V_{u^\star_{\lambda_k}}(z^{(k)}) \ge \frac{1}{\tau}V_y(z^{(k)}) - \frac{\rho^2}{1024\tau^4} - \frac{1}{2\tau}V_y(u^\star_{\lambda_k}) \ge \frac{\rho^2}{512\tau^4} - V_y(u^\star_{\lambda_k}) \\ &\implies V_y(u^\star_{\lambda_k}) \ge \frac{\rho^2}{1024\tau^4}.\end{aligned}$$

∎

The next lemma shows that there exists a nontrivial range of λ values for which the binary search will terminate with high probability. Here by overloading notations we let $\mathcal{E}_\lambda(\delta)$ to denote the probablistic event in Lemma 5.3 with parameter δ when calling LI-MD$(\mathcal{G}, y, \rho, \lambda, \eta, T)$ with η and T chosen as in λ-Bisection.

Lemma 5.8. *Assume $\mathcal{X}$ and V satisfy a τ-triangle inequality with $\tau \ge 4$. For $\delta \in (0,1)$ let $\eta \le \frac{\rho^2\lambda}{66\cdot1024\cdot\log(16/\delta)\tau^5\Gamma^2}$ and $T = \frac{4\tau}{\eta\lambda}$. Then under event $\mathcal{E}_\lambda(\delta/8)$ the output z of* LI-MD$(\mathcal{G}, y, \rho, \lambda, \eta, T)$ *satisfies if $V_y(u^\star_\lambda) \le \frac{\rho^2}{100\tau^2}$ then $V_y(z) \le \frac{\rho^2}{64\tau}$ and if $V_y(u^\star_\lambda) \ge \frac{\rho^2}{120\tau^2}$ then $V_y(z) \ge \frac{\rho^2}{256\tau^3}$.*

Proof. We begin by noting that by Lemma 5.5 the assumption $V_y(u_\lambda^\star) \le \frac{\rho^2}{48\tau^2}$, the event $\mathcal{E}_\lambda(\delta/8)$, and the choice of η implies that $\mathsf{OutOfBound} = \mathsf{False}$. Therefore, as in the proof of Lemma 5.7 above, we may use Lemma 5.4 and conclude that

$$V_{u_\lambda^\star}(z) \le \frac{2V_y(u_\lambda^\star)}{\lambda\eta T} + \left(66\log\frac{2}{\delta}\right)\frac{\eta}{\lambda}\Gamma^2 \le \frac{1}{2\tau}V_y(u_\lambda^\star) + \frac{\rho^2}{1024\tau^4}.$$

By the τ-triangle inequality,

$$V_y(z) \le \tau\left(V_y(u_\lambda^\star) + V_{u_\lambda^\star}(z)\right) \le \frac{3}{2}\tau V_y(u_\lambda^\star) + \frac{\rho^2}{1024\tau^3} \le \rho^2\left(\frac{3\tau}{2}\cdot\frac{1}{100\tau^2} + \frac{1}{1024\tau^4}\right) \le \frac{\rho^2}{64\tau}.$$

Applying the τ-triangle inequality in the other direction gives

$$V_y(z) \ge \frac{1}{\tau}V_y(u_\lambda^\star) - V_{u_\lambda^\star}(z) \ge \frac{1}{2\tau}V_y(u_\lambda^\star) - \frac{\rho^2}{1024\tau^4} \ge \rho^2\left(\frac{1}{2\tau}\cdot\frac{1}{120\tau^2} - \frac{1}{1024\cdot 4\tau^3}\right) \ge \frac{\rho^2}{256\tau^3}.$$

∎

The next lemma justifies the choice of the upper bisection limit $\lambda_{\max}$.

LEMMA 5.9. (UPPER BISECTION LIMIT) *Let $h : \mathcal{X} \to \mathbb{R}$ be convex and, for some $y \in \mathcal{X}$, let $H_\lambda(x) \coloneqq h(x) + \lambda V_y(x)$ with 1-strongly-convex $V_y(\cdot)$ and $u_\lambda^\star \coloneqq \operatorname{argmin}_{x\in\mathcal{X}} H_\lambda(x)$. If h is Γ-Lipschitz then for any $\lambda \ge 0$,*

$$V_y(u_\lambda^\star) \le \frac{\Gamma^2}{2\lambda^2}. \tag{5.29}$$

Consequently, for $\lambda_{\max} = \frac{16\tau\Gamma}{\rho}$ we have $V_y(u_{\lambda_{\max}}^\star) < \frac{\rho^2}{100\tau^2}$.

Proof. We may assume that $u_\lambda^\star$ is in the interior of $\mathcal{X}$, since otherwise $V_y(u_\lambda^\star) = V_y(u_{\lambda'}^\star)$ for some $\lambda' \ge \lambda$ such that for all $\lambda'' > \lambda'$ the point $u_{\lambda''}^\star$ is in the interior of $\mathcal{X}$, and we may apply the following considerations to $\lambda'' \downarrow \lambda'$ instead. We further assume without loss of generality that h and φ are differentiable, as otherwise we may unifromly approximate them with convex differentiable functions via Moreau envelopes.

These assumptions imply that

$$0 = \nabla H_\lambda(u_\lambda^\star) = \nabla h(u_\lambda^\star) + \lambda\nabla V_y(u_\lambda^\star).$$

Hence, the fact that h is Γ-Lipschitz implies that

$$\|\nabla V_y(u_\lambda^\star)\|_* = \frac{1}{\lambda}\|\nabla h(u_\lambda^\star)\|_* \le \frac{\Gamma}{\lambda}.$$

Finally, the 1-strong-convexity of $x \mapsto V_y(x)$ and the fact that its minimal value of 0 is obtained at y implies that

$$V_y(u_\lambda^\star) = V_y(u_\lambda^\star) - V_y(y) \le \frac{1}{2}\|\nabla V_y(u_\lambda^\star)\|_*^2 \le \frac{\Gamma^2}{2\lambda^2}$$

as required. ∎

The next lemma justifies the lower bisection limit $\lambda_{\min}$.

LEMMA 5.10. (LOWER BISECTION LIMIT) *Let $\lambda_{\min} = \lambda_0 = 1$ and $u_{\lambda_{\min}}^\star \coloneqq \operatorname{argmin}_{x\in\mathcal{X}} H_{\lambda_{\min}}(x)$ and assume that $\mathcal{X}$ and V satisfy a τ-triangle inequality with $\tau \ge 4$. Under the event $\mathcal{E}_0(\delta/8)$, if $V_y(z^{(0)}) \le \frac{\rho^2}{64\tau}$ then $V_y(u_{\lambda_{\min}}^\star) \le \frac{\rho^2}{16}$ and if $V_y(z^{(0)}) \ge \frac{\rho^2}{64\tau}$ then $V_y(u_{\lambda_{\min}}^\star) \ge \frac{\rho^2}{120\tau^2}$.*

Proof. Immediate from Lemmas 5.7 and 5.8. ∎

Finally, we bound the Lipschitz constant of $\lambda \mapsto V_y(u_\lambda^\star)$ and apply the above lemmas to conclude that λ-BISECTION returns a valid points within $\widetilde{O}(1)$ iterations.

PROPOSITION 5.1. *In the setting of Theorem 5.1, under the event* $\cap_{k=0}^{K_{\max}} \mathcal{E}_k(\delta_k/8)$, *with* $K_{\max} = \lceil \log_2 \frac{9600\tau^2\Gamma^3}{\rho^3} \rceil + 1$, *which happens with probabiltiy at least* $1 - \frac{\delta}{2}$, *the* λ-BISECTION *procedure in Algorithm 2 successfully returns at iteration* $K < K_{\max}$ *a value* λ_K *such that* $V_y(u^\star_{\lambda_K}) \le \frac{\rho^2}{16}$ *and, if* $K \ge 1$, *also* $V_y(u^\star_{\lambda_K}) \ge \frac{\rho^2}{1024\tau^4}$.

Proof. We begin by noting that $\mathbb{P}\Big(\cap_{k=0}^{K_{\max}} \mathcal{E}_k(\delta_k/8)\Big) \ge 1 - \frac{\delta}{2}$ by Lemma 5.3 and the union bound.

Next, Lemma 5.10 establishes the claims of the proposition in the edge case we return with $K = 0$.

Moving on to the main case we return with $K \ge 1$. If also $K < K_{\max}$ then Lemma 5.7 guarantees that the claim $V_y(u^\star_{\lambda_K}) \in \big[\frac{\rho^2}{1024\tau^4}, \frac{\rho^2}{16}\big]$ holds. It therefore remains to argue that the bisection does indeed terminate in less than $K_{\max}$ steps. Let $\lambda', \lambda'' \in (\lambda_{\min}, \lambda_{\max}]$ satisfy $V_y(u^\star_{\lambda'}) = \frac{\rho^2}{100\tau^2}$ and $V_y(u^\star_{\lambda''}) = \frac{\rho^2}{120\tau^2}$. By Lemmas 5.8 to 5.10, when $\cap_{k=0}^{K_{\max}} \mathcal{E}_k(\delta_k/8)$ holds then $[\lambda', \lambda''] \subseteq [\lambda_{\min}, \lambda_{\max}]$ is an invariant of the bisection and moreover the bisection terminates if we query $\lambda_K \in [\lambda', \lambda'']$. Since the bisection the search interval at every step, it must return in $\log_2 \frac{\lambda_{\max} - \lambda_{min}}{\lambda'' - \lambda'}$ steps. We have $\lambda_{\max} - \lambda_{\min} \le \frac{16\tau\Gamma}{\rho}$, so to conclude the proof we need only lower bound $\lambda'' - \lambda'$.

To do so, we write

$$\begin{aligned}
\frac{\rho^2}{600\tau^2} = V_y(u^\star_{\lambda'}) - V_y(u^\star_{\lambda''}) = \int_{\lambda=\lambda''}^{\lambda'} (V_y(u^\star_\lambda))' d\lambda &= \int_{\lambda=\lambda''}^{\lambda'} \langle \nabla V_y(u^\star_\lambda), \nabla_\lambda u^\star_\lambda \rangle \, d\lambda \\
&\overset{(i)}{=} -\int_{\lambda=\lambda''}^{\lambda'} \Big\langle \nabla V_y(u^\star_\lambda), \big(\nabla^2 h(u^\star_\lambda) + \lambda \nabla^2 V_y(u^\star_\lambda)\big)^{-1} \nabla V_y(u^\star_\lambda) \Big\rangle \, d\lambda \\
&\overset{(ii)}{\le} (\lambda'' - \lambda') \frac{\Gamma^2}{(\lambda')^3} \le (\lambda'' - \lambda')\Gamma^2.
\end{aligned}$$

Here for (i) we use $\nabla h(u^\star_\lambda) + \lambda \nabla V_y(u^\star_\lambda) = 0$ for all $\lambda \in [\lambda', \lambda'']$, which implies $\nabla_\lambda u^\star_\lambda = -(\nabla^2 h(u^\star_\lambda) + \nabla^2 V_y(u^\star_\lambda))^{-1} \nabla V_y(u^\star_\lambda)$ by taking derivatives with respect to λ and rearranging terms (we assume here that h and r are twice differentiable; this is again without loss of generality due to smoothing arguments). For (ii) we reuse $\|\nabla V_y(u^\star_\lambda)\| \le \frac{\Gamma}{\lambda}$ from the proof of Lemma 5.9. The above display implies that $\lambda'' - \lambda' \ge \frac{\rho^2}{600\tau^2\Gamma^2}$ and therefore our choice of $K_{\max}$ guarantees that $\log_2 \frac{\lambda_{\max} - \lambda_{min}}{\lambda'' - \lambda'} < K_{\max}$, concluding the proof. ∎

5.4 Proof of Theorem 5.1.

Proof. We prove each part of the proposition in turn.

For part 1, let $K \le K_{\max}$ be the final iteration of λ-BISECTION and let λ_K be its output. Recall from Corollary 5.2 that, for $\lambda = \lambda_K$, input parameters $\eta = \frac{\rho^2\lambda}{C \cdot \log(16/\delta)\tau^5\Gamma^2}$, $T = \frac{4\tau}{\eta\lambda}$, the outputs of LI-MD satisfy

$$\begin{aligned}
\mathbb{E}_\lambda h(z) - h(u) \le &\left(\lambda + \frac{1}{\eta T}\right)\lambda[V_y(u) - V_w(u)] + \eta\Gamma^2 - \left(\frac{\lambda}{\tau} - \frac{1}{\eta T}\right) V_y(u^\star_\lambda) \\
&+ \left(\sqrt{2}R\Gamma + \left(\lambda + \frac{1}{\eta T}\right)R^2\right)\mathbb{P}_\lambda(\mathsf{OutOfBound} = \mathsf{True}),
\end{aligned}$$

where $\mathbb{E}_\lambda$ and $\mathbb{P}_\lambda$ denote conditional expectation over random variable $\lambda = \lambda_K$.

Dividing both sides by $c = \lambda + (\eta T)^{-1} \ge 1$ and taking total expectation we have

$$\begin{aligned}
&\mathbb{E}\frac{h(z) - h(u)}{c} \\
&\quad \le \mathbb{E}[V_y(u) - V_w(u)] + \mathbb{E}\left[\frac{\eta}{\lambda}\Gamma^2 - \frac{\frac{\lambda}{\tau} - \frac{\lambda}{4\tau}}{\lambda + \frac{\lambda}{4\tau}} V_y(u^\star_\lambda) + \left(\frac{\sqrt{2}R\Gamma}{\lambda + (\eta T)^{-1}} + R^2\right)\mathbb{P}_\lambda(\mathsf{OutOfBound} = \mathsf{True})\right] \\
&\quad \le \mathbb{E}[V_y(u) - V_w(u)] + \frac{\rho^2}{C\log(16/\delta)\tau^5} - \frac{3}{5\tau}\mathbb{E}V_y(u^\star_\lambda) + (\sqrt{2}R\Gamma + R^2)\mathbb{P}(\mathsf{OutOfBound} = \mathsf{True}). \qquad (5.30)
\end{aligned}$$

Proposition 5.1 implies that $V_y(u^\star_\lambda) \ge \frac{\rho^2}{1024\tau^4}\mathbb{1}_{\{\lambda \ne \lambda_{\min}\}}$ holds with probability at least $1 - \frac{\delta}{2}$. Therefore, since Bregman divergences are nonnegative,

$$\mathbb{E}V_y(u^\star_\lambda) \ge \left(1 - \frac{\delta}{2}\right)\frac{\rho^2}{1024\tau^4}\mathbb{1}_{\{\lambda \ne \lambda_{\min}\}} \ge \frac{\rho^2}{2^{11}\tau^4}\mathbb{1}_{\{\lambda \ne \lambda_{\min}\}}.$$

By our choices of η and T, Lemma 5.5 tells us that $V_y(u_\lambda^\star) \le \frac{\rho^2}{16}$ and $\mathcal{E}_\lambda(\delta/8)$ imply that $\mathsf{OutOfBound} = \mathsf{False}$. Therefore,

$$\mathbb{P}_\lambda(\mathsf{OutOfBound} = \mathsf{True}) \le \mathbb{P}_\lambda(\neg\mathcal{E}_\lambda(\delta/8)) + \mathbb{1}_{\left\{V_y(u_\lambda^\star) > \frac{\rho^2}{16}\right\}} \le \frac{\delta}{8} + \mathbb{1}_{\left\{V_y(u_\lambda^\star) > \frac{\rho^2}{16}\right\}},$$

where the final inequality used Lemma 5.3. Taking expectation and invoking Proposition 5.1 again, we find that

$$\mathbb{P}(\mathsf{OutOfBound} = \mathsf{True}) \le \frac{\delta}{8} + \mathbb{P}\left(V_y(u_\lambda^\star) > \frac{\rho^2}{16}\right) \le \frac{\delta}{8} + \frac{\delta}{2} \le \delta.$$

Substituting the bounds on $\mathbb{E}V_y(u_\lambda^\star)$ and $\mathbb{P}(\mathsf{OutOfBound} = \mathsf{True})$ into Equation (5.30), noting that $c \ge 2$ only when $\lambda \ne \lambda_{\min}$ and recalling that $\delta \le \frac{\rho^2}{2^{14}(\sqrt{2}R\Gamma + R^2)\tau^5}$, we obtain the required ball-restricted proximal oracle bound (5.16). Additionally, we note for all possible choices of returned c it satisfies $c \le 2\lambda_{\max} = \frac{32\tau\Gamma}{\rho}$ with probability 1, giving the claimed value of $c_{\max}$.

Part 2 of the proposition is immediate from the definition of LI-MD, which always outputs points with distane at most ρ from y.

For part 3 we use Lemma 5.6 with $u = u_\lambda^\star$, which gives for all $k \le K$ that $\sum_{t\in[\hat{T}_k]} \|x_t^{(k)} - x_{t-1}^{(k)}\| \le 2\log(2T_k)\rho$. Summing these bounds gives

$$\sum_{k=0}^{K}\sum_{t\in[\hat{T}_k]} \|x_t^{(k)} - x_{t-1}^{(k)}\| \le 2\sum_{k=0}^{K}\log(2\hat{T}_k)\rho \le 2\rho K_{\max}\log\frac{4C\log(16K_{\max}^2/\delta)\tau^6\Gamma^2}{\rho^2}.$$

Finally, part 4 follows from the setting of the T_k and $K_{\max}$, since the total number of gradient queries and mirror descent steps is at most $\left(\sum_{k=0}^{K_{\max}} T_k\right)$. ∎

6 Matrix-vector maintenance data structures

In this section we formally define an *ℓ_p-matrix-vector maintenance data structures (abbreviated* MVM_p*)* and provide efficient algorithms for them for $p \in \{1, 2\}$. An MVM_p approximates the sequence $\{Ax_t\}$ to additive ϵ error in ℓ_∞, as long as the sum of the ℓ_p norm of the movements $\Delta_t = x_{t+1} - x_t$ does not exceed a given bound R. The data structure is formally defined below in Definition 6.1; for a brief description of these data structures and how they fit into our overall method, see Section 2.2. In the definition of an MVM_p, and throughout this section, for any $p \ge 1$ we let $p^* \ge 1$ be such that $\frac{1}{p} + \frac{1}{p^*} = 1$; if $p = 1$ then $p^* = \infty$. Furthermore, for any matrix $A \in \mathbb{R}^{n\times d}$ with rows $a_1, \ldots, a_n \in \mathbb{R}^d$ and $p \ge 1$ we let

$$\|A\|_{p\to\infty} := \sup_{x\in\mathbb{R}^n, \|x\|_p = 1} \|Ax\|_\infty = \max_{i\in[n]} \|a_i\|_{p^*}.$$

Definition 6.1. (Matrix-vector maintenance) *We call a data structure an* ℓ_p-matrix-vector maintenance data structure (MVM_p) *if it supports the following operations:*

- $\textsc{init}(A \in \mathbb{R}^{n\times d}, x_0 \in \mathbb{R}^d, R \in \mathbb{R}_{>0}, \epsilon \in \mathbb{R}_{>0})$*: initializes the data structure with a matrix A with $\|A\|_{p\to\infty} \le 1$, initial point x_0, movement range R, and accuracy $\epsilon \le R/2$.*[4] *Sets $t \leftarrow 0$.*
- $\textsc{query}(\Delta_t \in \mathbb{R}^d)$*: sets $x_{t+1} \leftarrow x_t + \Delta_t$, and $t \leftarrow t + 1$ and then outputs $y_t \in \mathbb{R}^n$ (or the coordinates which changed from the previous output if that is cheaper) with $\|y_t - Ax_t\|_\infty \le \epsilon$ provided that $\sum_{i\in[t]} \|\Delta_i\|_p \le R$.*

Our main results for designing MVM_p's are encapsulated in the following theorem.

Theorem 6.1. (Matrix-vector maintenance) *For both $p = 1$ and $p = 2$ and any $\delta > 0$, there is a* MVM_p *(Definition 6.1) that implements* INIT *and T* QUERY *operations with probability $1-\delta$ (against an oblivious adversary) in total time*

$$O\left(\sum_{t\in[T]} \mathrm{nnz}(\Delta_t) + \left(\mathrm{nnz}(A)\log^{p-1}\left(\frac{R}{\epsilon}\right) + d\cdot\frac{R}{\epsilon}\right)\log^{p-1}\left(\frac{nR}{\epsilon\delta}\right) + n\left(\frac{R}{\epsilon}\right)^2\log\left(\frac{nR}{\epsilon\delta}\right)\right).$$

[4] This can always be obtained by initializing the algorithm with a smaller value of ϵ or a larger value of R.

The runtime in Theorem 6.1 is nearly linear in input size $\text{nnz}(A) + \sum_{t\in[T]} \text{nnz}(\Delta_t)$ with an additive $\widetilde{O}(d(R/\epsilon))$ and $\widetilde{O}(n(R/\epsilon)^2)$ terms. When A is dense and R does not depend on T, this runtime considerably improves on the $\Omega(ndT)$ cost of naively implementing the data structure by computing Ax_t exactly for each $t \in [T]$.

Our data structures have similar runtime complexity for both $p = 1$ and $p = 2$ (up to additional logarithmic factors for $p = 2$), but potentially much smaller memory complexity for $p = 2$. As developed in the rest of this section, our data structure for $p = 1$ needs to store the entire input matrix A. In contrast, our data structure when $p = 2$ requires $\widetilde{O}(d + n(R/\epsilon)^2)$ space after initialization, which can be sublinear in $\text{nnz}(A)$.

Approach and section organization. We prove Theorem 6.1 in two steps. First, in Section 6.1, we consider the simpler problem of designing a data structure which supports preprocessing A and then outputting ℓ_∞ estimates for Ax for a single query x, under no movement bound assumptions. We call such a data structures an *ℓ_p-matrix-vector estimation data structure (abbreviated* MVE_p*)*, and provide efficient implementations for $p \in \{1, 2\}$. Our MVE_p when $p = 2$ is then based on linear sketching and our data structure when $p = 1$ is based on random sampling.

Second, in Section 6.2 we provide a general reduction from designing a MVM_p to designing MVE_p's. In particular, we provide an MVM_p which carefully uses $O(\log(R/\epsilon))$ copies of an MVE_p with different accuracy parameters. We use these MVE_p's approximately maintain $Ax_1^{\text{ref}}, \ldots, Ax_k^{\text{ref}}$ for $k = O(\log(R/\epsilon))$ reference points $x_1^{\text{ref}}, \ldots, x_k^{\text{ref}}$. By carefully updating these reference points when the movement is sufficient and using our MVE_p's, we prove Theorem 6.1.

Our runtimes for MVM_p's for $p \in \{1, 2\}$, i.e., Theorem 6.1, are ultimately the same as the cost of initializing our MVE_p's and performing a single query for a vector that has ℓ_p-norm at most R (up to logarithmic factors). In other words, even though an MVM_p needs to answer many queries, the computational cost we obtain is comparable to answering a single query to a vector that has ℓ_p distance R from the initial point.

6.1 Matrix-vector estimation.

We now formally define an MVM_p data structure (Definition 6.2) and efficiently implement it for $p \in \{1, 2\}$.

DEFINITION 6.2. (MATRIX-VECTOR ESTIMATION) *We call a data structure an ℓ_p-*matrix-vector estimation data structure (MVE_p) *if it supports the following operations (against an oblivious adversary):*

- $\text{INIT}(A \in \mathbb{R}^{n\times d}, \epsilon \in \mathbb{R}_{>0}, \delta > 0)$*: initialize the data structure with matrix A, accuracy parameter ϵ, and failure probability $\delta > 0$.*
- $\text{QUERY}(x \in \mathbb{R}^d)$*: outputs $y \in \mathbb{R}^n$ such that $\|y - Ax\|_\infty \le \epsilon \|A\|_{p\to\infty}\|x\|_p$ holds with probability at least $1 - \delta$ (for just this query).*

THEOREM 6.2. (ℓ_2-MATRIX-VECTOR ESTIMATION) *There is a* MVE_2 *(Definition 6.2) that implements* $\text{INIT}(A, \epsilon, \delta)$ *for in time $O((\text{nnz}(A) + d)\log(n/\delta))$ and subsequent* $\text{QUERY}(x)$ *operations in time $O((\text{nnz}(x) + n\epsilon^{-2})\log(n/\delta))$.*

Proof. Our data structure is a natural application of CountSketch matrices [19]. We use that, from the literature on CountSketch matrices (see e.g., [19, 37]), there exists a distribution, $\mathcal{M}$, on matrices in $\mathbb{R}^{s\times d}$ for $s = O(\epsilon^{-2}\log(n/\delta)$ that have the following properties:

- $Q \sim \mathcal{M}$ can be computed in $O(d\log(n/\delta))$ time and each column of Q has at most $O(\log(n/\delta)))$ non-zero entries.
- There is a procedure DECODE_Q that given any input Qx and Qy for $Q \sim \mathcal{M}$ drawn independently of $x, y \in \mathbb{R}^d$ outputs $\alpha = \text{DECODE}_Q(Qx, Qy)$ with $|\alpha - \langle x, y\rangle| \le \epsilon\|x\|_2\|y\|_2$ in $O(\epsilon^{-2}\log(n/\delta))$ time with probability at least $1 - (\delta/n)$.

To implement INIT our data structure draws $Q \sim \mathcal{M}$ and then computes $y^i = QA_{i:}^\top$ for all $i \in [n]$. To implement $\text{QUERY}(x)$ our data structure then outputs $v \in \mathbb{R}^n$ with each $v_i = \text{DECODE}_Q(y_i, Qx)$.

To see that our data structure is a MVE_2 note that $|v_i - \langle A_{i:}^\top, x\rangle| \le \epsilon\|A_{i:}^\top\|_2\|x\|_2$ with probability at least $1 - (\delta/n)$ by the properties of Q. Since $\|A_{i:}^\top\|_2 \le \|A\|_{2\to\infty}$ for all $i \in [n]$ by applying union bound for this event for all $i \in [n]$ we have the desired bound that $\|v - Ax\|_\infty \le \epsilon\|A\|_{2\to\infty}\|x\|_2$ with probability at least $1 - \delta$.

To bound the algorithm's runtime, first note that computing Qx for any vector x can be implemented in $O(\text{nnz}(x)\log(n/\delta))$ just by considering the $O(\log(n/\delta))$-sparse column of Q for each non-zero entry of x. The

runtime for INIT follows immediately from this and the time to compute Q. The runtime for QUERY$(\cdot)$ then follows by first computing Qx in time $O(\text{nnz}(x)\log(n/\delta))$ and then considering the cost of n-invocations of DECODE$_Q(\cdot,\cdot)$. ∎

THEOREM 6.3. (ℓ_1-MATRIX-VECTOR ESTIMATION) *For $p = 1$ there is a* MVE$_1$ *data structure (Definition 6.2) that implements* INIT(A, ϵ, δ) *for $A \in \mathbb{R}^{n\times d}$ in time $O(\text{nnz}(A))$ and* QUERY(x) *in time $O(\text{nnz}(x) + n\epsilon^{-2}\log(n/\delta))$.*[5]

Proof. Our data structure is a straightforward application random sampling and a Chernoff bound. For any $a, x \in \mathbb{R}^d$ we let SAMPLE(a, x) be a procedure that outputs independent, random $X \in \mathbb{R}$ by picking $i \in [d]$ with probability proportional to $|x_i|$ and then outputting $\|x\|_1 a_j \operatorname{sign}(x_j)$, i.e., for any $j \in [n]$

$$\mathbb{P}(X = \|x\|_1 a_j \operatorname{sign}(x_j)) = \frac{|x_j|}{\|x\|_1}, \text{ where } \operatorname{sign}(t) \coloneqq \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t = 0 \\ -1 & \text{if } t < 0 \end{cases}.$$

By design, $\mathbb{E}[X] = \langle a, x\rangle$ and by a Chernoff bound [see, e.g., 20] we have that for sufficiently large $T = O(\epsilon^{-2}\log(n/\delta))$ and $\alpha = \frac{1}{T}\sum_{t\in[T]}$ SAMPLE(a, x) it is the case that $|\alpha - \langle a, x\rangle| \le \epsilon\|a\|_\infty\|x\|_1$ with probability at least $1 - (\delta/n)$. To implement INIT our data structure simply saves A, ϵ, and R. To implement QUERY(x) the data structure then outputs $v \in \mathbb{R}^n$ with each $v_i = \frac{1}{T}\sum_{t\in[T]}$ SAMPLE$(A_{i:}^\top, x)$.

To see that our data structure is a MVE$_1$ note that $|v_i - \langle a, x\rangle| \le \epsilon\|A_{i:}^\top\|_\infty\|x\|_1$ with probability at least $1 - (\delta/n)$ by the properties of Q. Since $\|A_{i:}^\top\|_\infty \le \|A\|_{1\to\infty}$ for all $i \in [n]$ by applying union bound for this event for all $i \in [n]$ we have the desired bound that $\|v - Ax\|_\infty \le \epsilon\|A\|_{1\to\infty}\|x\|_1$ with probability at least $1 - \delta$.

To bound the algorithm's runtime, first note that, as discussed in Section 3, we assumed that we are in a computation model where can process the vector $|x|$ in $O(\text{nnz}(x))$ time to support sampling $i \propto |x_i|$ in time $O(1)$. Leveraging this, we have that with $O(\text{nnz}(x)))$ time spent all subsequent SAMPLE$(\cdot)$ operations can be performed in $O(1)$. Since there are $nT = O(n\epsilon^{-2}\log(n/\delta))$ sample operations the data structure has the desired running time. ∎

6.2 From estimation to maintenance. Now that we have established efficient MVE$_p$'s for $p = 2$ (Theorem 6.2) and $p = 1$ (Theorem 6.3), here we use these data structures to prove our main result on MVM$_p$'s (Theorem 6.1).

We provide a general reduction from MVM$_p$ to MVE$_p$. In particular, we provide a MVM$_p$ in Algorithm 4 that uses $k = O(\log(R/\epsilon))$ MVE$_p$'s for different accuracy parameters. In Theorem 6.4 we prove that for any such implementation and choice of input parameters $\alpha \in \Delta^k$, Algorithm 4 is indeed a MVM$_p$ and we analyze its runtime. We then prove Theorem 6.1 by setting α, using our MVE$_p$ implementations and applying an additional runtime improvement technique.

Designing and analyzing the data structure. Before providing these results and wrapping up the section, here we provide some additional intuition and information regarding Algorithm 4. In addition to the standard input for a MVM$_p$ and $\delta > 0$, the data structure is specified by k MVE$_p$'s and parameters $\alpha \in \Delta^k$. In INIT$(\cdot)$, the data structure initializes each MVE$_p$ —denoted $D_1, \ldots, D_k$—and stores $k+2$ reference vectors $x_0^{\text{ref}}, \ldots, x_{k+1}^{\text{ref}} \in \mathbb{R}^d$ all initialized to x_0 as well as $y_0^{\text{ref}}, \ldots, y_{k+1}^{\text{ref}} \in \mathbb{R}^n$ all initialized to Ax_0. The data structure maintains the invariant that $x_0^{\text{ref}} = x_t$ and $\|x_i^{\text{ref}} - x_{i-1}^{\text{ref}}\|_p \le \epsilon \cdot 2^{i-2}$ for all $1 \le i \le k+1$. It uses this invariant to efficiently maintain that $y_i^{\text{ref}} \approx Ax_i^{\text{ref}}$. It then holds that, at any given time, y_1^{ref} is a valid response to QUERY$(\cdot)$.

The challenge in designing and analyzing Algorithm 4 is then to maintain these invariants, bound the error in setting y_1^{ref} to be the response to QUERY$(\cdot)$, and analyzing the runtime. Maintaining that $x_0^{\text{ref}} = x_t$ and $\|x_i^{\text{ref}} - x_{i-1}^{\text{ref}}\|_p \le \epsilon \cdot 2^{i-2}$ is straightforward; after each QUERY$(\cdot)$ we simply set $x_0^{\text{ref}} = x_t$ and then update $x_i^{\text{ref}} = x_t$ all $i \in [j]$ for the smallest j for which this suffices to preserve the invariant. Due to the choice of 2^{i-2} and the bound on how much the x_t can change, it is straightforward to show that x_i^{ref} for $i \ge 1$ changes at most $O((R/\epsilon)2^{-i})$ times via this procedure. Furthermore, to update y_i^{ref} for all such $i \in [j]$ we simply estimate $A(x_i - x_{i+1})$ using D_i.QUERY$(\cdot)$ and add this estimate to y_{i+1}^{ref}. For appropriate choice of accuracies in the D_i (adjusted by the α_i) we show this algorithm works as desired. Further, by choosing α and the accuracies, we get a tradeoff between the cost of each D_i.QUERY$(\cdot)$ and the number of times it is invoked. Putting these pieces together and carefully reasoning about computational costs then yields our result.

[5] Each INIT can actually be implement in time $O(0)$, i.e., no initialization is required, provided that looking up entries A and the

Algorithm 4: ℓ_p matrix-vector maintenance meta-data structure

Input: Parameter $p \geq 1$, $\delta > 0$, and $\alpha \in \Delta^k$

State: $A \in \mathbb{R}^{n \times d}$, $x_0 \in \mathbb{R}^d$, $R \in \mathbb{R}_{>0}$, $\epsilon \in \mathbb{R}_{>0}$, $\delta > 0$

State: Current vector $x_t \in \mathbb{R}^d$, count $t \in \mathbb{R}_{\geq 0}$, and parameter $k \in \mathbb{Z}_{>0}$

State: ℓ_p matrix-vector estimation data structures, $D_1, \ldots, D_k$ // see Definition 6.2

State: Reference vectors $x_0^{\text{ref}}, \ldots, x_{k+1}^{\text{ref}} \in \mathbb{R}^d$ and $y_0^{\text{ref}}, \ldots, y_{k+1}^{\text{ref}} \in \mathbb{R}^n$ // $\|x_i^{\text{ref}} - x_{i-1}^{\text{ref}}\|_p \leq \epsilon \cdot 2^{i-2}$

function INIT($A \in \mathbb{R}^{n \times d}, x_0 \in \mathbb{R}^d, R \in \mathbb{R}_{>0}, \epsilon \in \mathbb{R}_{>0}$)

  Save A, x_0, R, and ϵ as part of data structure's state

  $t \leftarrow 0$ and $k \leftarrow \lceil \log_2(\lceil R/\epsilon \rceil) \rceil + 1$

  $x_i^{\text{ref}} \leftarrow x_0$ and $y_i^{\text{ref}} \leftarrow Ax_0$ for all $i \in \{0\} \cup [k+1]$

  Set $\epsilon_i \leftarrow \alpha_i 2^{-i}$ and call D_i.INIT$(A, \epsilon_i, \bar{\delta})$ for $\bar{\delta} \leftarrow \delta\epsilon/R$ and all $i \in [k]$

function QUERY($\Delta_t \in \mathbb{R}^d$)

  $x_{t+1} \leftarrow x_t + \Delta_t$ and $x_0^{\text{ref}} \leftarrow x_{t+1}$ and then $t \leftarrow t + 1$

  Let j be the minimum $i \in [k+1]$ such that $\|x_t - x_i^{\text{ref}}\|_p \leq \epsilon \cdot 2^{i-2}$

  for $i \in \{j-1, \ldots, 1\}$ **do** $x_i^{\text{ref}} \leftarrow x_0^{\text{ref}}$ and then $y_i^{\text{ref}} \leftarrow D_i$.QUERY$(x_i^{\text{ref}} - x_{i+1}^{\text{ref}}) + y_{i+1}^{\text{ref}}$

  return y_1^{ref}

THEOREM 6.4. (REDUCING MATRIX VECTOR MAINTENANCE TO ESTIMATION) *Algorithm 4 is an ℓ_p-matrix-vector maintenance data structure (Definition 6.1). If the runtime for each D_i.INIT$(A, \epsilon_i, \delta_i)$ is $\mathcal{T}_{\text{INIT}}(i)$ and the runtime for each subsequent D_i.QUERY$(\cdot)$ is $\mathcal{T}_{\text{QUERY}}(i)$ then Algorithm 4 can implement* INIT *and T* QUERY *operations in total time*

$$O\left(\text{nnz}(A) + d \cdot \frac{R}{\epsilon} + \sum_{t \in [T]} \text{nnz}(\Delta_t) + \sum_{i \in [k]} \left(\mathcal{T}_{\text{INIT}}(i) + \frac{R}{\epsilon \cdot 2^i} \cdot [\mathcal{T}_{\text{QUERY}}(i)] \right) \right).$$

Proof. We begin by showing that $\|x_t - x_{k+1}^{\text{ref}}\|_p \leq \epsilon \cdot 2^{k-1}$ in each execution of Line 12 and therefore the j on Line 12 is well-defined. To see this, note that in each execution of Line 12 we have

$$\|x_t - x_0\|_p = \left\| \sum_{i \in [t]} (x_i - x_{i-1}) \right\|_p \leq \sum_{i \in [t]} \|x_i - x_{i-1}\|_p = \sum_{i \in [t]} \|\Delta_i\|_p \leq R. \tag{6.31}$$

Since $\epsilon \cdot 2^{k-1} \geq \epsilon \cdot 2^{\log_2(R/\epsilon)} = R$ so long as $x_{k+1}^{\text{ref}} = x_0$ then $\|x_t - x_{k+1}^{\text{ref}}\|_p \leq \epsilon \cdot 2^{k-1}$. However, $x_{k+1}^{\text{ref}} = x_0$ is set in INIT and then never updated (since $i \leq k$ is on Line 13) and the claim follows.

Leveraging that j is well-defined on Line 12, we show that before and after each call to QUERY$(\cdot)$, $\|x_i^{\text{ref}} - x_{i-1}^{\text{ref}}\|_p \leq \epsilon \cdot 2^{i-2}$ for all $i \in [k]$. This invariant holds after INIT as each x_i^{ref} is initially set to x_0. Next, suppose the invariant holds before a call QUERY$(\cdot)$. x_i^{ref} are only changed on Line 13 and for $i \leq j-1$, in which case they are set to x_0^{ref}. However, $\|x_0^{\text{ref}} - x_j^{\text{ref}}\|_p \leq \epsilon \cdot 2^{j-2}$ by the definition of j (Line 13) and that j is well-defined. Therefore, after the call to QUERY$(\cdot)$ the invariant holds since $\|x_{j-1}^{\text{ref}} - x_j^{\text{ref}}\| \leq \epsilon \cdot 2^{j-2}$ and $\|x_i^{\text{ref}} - x_{i+1}^{\text{ref}}\| = 0 \leq \epsilon \cdot 2^{i-2}$ for all $i \in [j-2]$.

Next, we show that for all $i \in [k]$, throughout the use of Algorithm 4 as an MVM$_p$, D_i.QUERY$(\cdot)$ is called on Line 13 at most $R\epsilon^{-1}2^{-(i-2)}$ times. Whenever D_i.QUERY$(\cdot)$ is called on Line 13 it must be the case that $\|x_t - x_i^{\text{ref}}\|_p > \epsilon \cdot 2^{i-2}$ (as otherwise $j \leq i$ by the definition of j on Line 12). Let $v_0, ..., v_L$ denote the sequence of different x_i^{ref} vectors set on Line 13 (where $v_0 = x_0^{\text{ref}}$); we have just argued that $\|v_\ell - v_{\ell-1}\|_p > \epsilon \cdot 2^{i-2}$ for $\ell \in [L]$. Further, since the v_ℓ are a subsequence of the x_t, triangle inequality implies that

$$R \geq \sum_{t \in [T]} \|x_t - x_{t-1}\|_p \geq \sum_{\ell \in [L]} \|v_\ell - v_{\ell-1}\|_p > L \cdot \epsilon \cdot 2^{i-2}.$$

values of ϵ and R can all be performed in $O(1)$ during QUERY$(\cdot)$.

Since $D_i.\textsc{query}(\cdot)$ is invoked L times, the claim follows.

Leveraging the previous properties, we next establish that with probability at least $1-\delta$ before and after each call to $\textsc{query}(\cdot)$, we have that $\|y_i^{\text{ref}} - Ax_i^{\text{ref}}\|_\infty \le \sum_{j=i}^{k-1} \frac{\alpha_i \epsilon}{2} \le \frac{\epsilon}{2}$ for all $i \in [k+1]$. By the preceding paragraph, we know that the total umber of matrix-vector estimation queries on Line 13 is at most

$$\sum_{i\in[k]} \frac{R}{\epsilon \cdot 2^{i-2}} \le \frac{R}{2\epsilon} \sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{R}{2\epsilon}\,.$$

Further, by the definition of a MVE_p (Definition 6.2) and by the union bound with probability at least $1-(\bar{\delta}R/(2\epsilon)) \ge 1-\delta$ every call to $D_i.\textsc{query}(x_0^{\text{ref}} - x_{i+1}^{\text{ref}})$ on Line 13 outputs a vector z_i where

$$\|z_i - A(x_i^{\text{ref}} - x_{i+1}^{\text{ref}})\|_\infty \le \epsilon_i \|A\|_{p\to\infty} \|x_i^{\text{ref}} - x_{i+1}^{\text{ref}}\|_\infty \le \frac{\epsilon}{2} \cdot \alpha_i$$

where we used the definition of ϵ_i, that $\|A\|_{p\to\infty} \le 1$ by assumption and that $\|x_i^{\text{ref}} - x_{i+1}^{\text{ref}}\| \le \epsilon \cdot 2^{i-1}$ in the last inequality. Consequently, with probability $1-\delta$, before and after each call to $\textsc{query}(\cdot)$ we have that for all $i \in [k-1]$,

$$\|y_i^{\text{ref}} - Ax_i^{\text{ref}}\|_p \le \|z_i - A(x_i^{\text{ref}} - x_{i+1}^{\text{ref}})\|_p + \|y_{i+1}^{\text{ref}} - Ax_{i+1}^{\text{ref}}\|_p \le \frac{\epsilon \cdot \alpha_i}{2} + \|y_{i+1}^{\text{ref}} - Ax_{i+1}^{\text{ref}}\|_p\,.$$

The claim then follows by induction and the facts that $\|y_{k+1}^{\text{ref}} - Ax_{k+1}^{\text{ref}}\| = 0$ (they are never changed after initialization) and $R\epsilon^{-1}2^{-(k-2)} < 1$.

We now have everything necessary to prove that Algorithm 4 is a MVM_p (Definition 6.1). Note that with probability $1-\delta$ after each call to $\textsc{query}(\cdot)$ we have argued that $\|y_i^{\text{ref}} - Ax_i^{\text{ref}}\|_p \le \frac{\epsilon}{2}$ and that $\|y_0^{\text{ref}} - y_1^{\text{ref}}\|_p \le \epsilon \cdot 2^{-1}$. Consequently,

$$\|y_1^{\text{ref}} - Ax_t\|_p = \|y_1^{\text{ref}} - Ax_0^{\text{ref}}\|_p \le \|y_1^{\text{ref}} - Ax_1^{\text{ref}}\|_p + \|A(x_1^{\text{ref}} - x_0^{\text{ref}})\|_p \le \frac{\epsilon}{2} + \|A\|_{p\to\infty}\|x_1^{\text{ref}} - x_0^{\text{ref}}\|_p \le \epsilon\,.$$

To complete the proof, we need to bound the data structure's runtime. Note that $\textsc{init}$ can be implemented in time $O(\text{nnz}(A) + \sum_{i\in[k]} \mathcal{T}_{\textsc{init}}(i))$ by simply performing the operations (and saving multiple copies of vectors and matrices with pointers as needed). Next, note that changes to x_t, x_0^{ref}, and $x_0^{\text{ref}} - x_1^{\text{ref}}$ due to x_0^{ref} changing can be computed in $O(\sum_{t\in[T]} \text{nnz}(\Delta_t))$ time. With this, it is possible to keep track of the changes to $\|x_0^{\text{ref}} - x_1^{\text{ref}}\|_p$ due to x_0^{ref} changing in $O(\sum_{t\in[T]} \text{nnz}(\Delta_t))$ time as well. Whenever $j > 1$ in Line 12, if we spend $O(dj)$ time to implement Line 12 and $O(d)$ plus the $D_i.\textsc{query}(\cdot)$ costs in each iteration of Line 13 then the total additional cost of $\textsc{query}(\cdot)$ over all invocations is

$$O\left(d + k + \sum_{i\in[k]} \left[\frac{R}{\epsilon \cdot 2^i} \cdot [\mathcal{T}_{\textsc{query}}(i)] + d\right]\right) = O\left(d \cdot \left\lceil \frac{R}{\epsilon} \right\rceil + k + \sum_{i\in[k]} \left(\frac{R}{\epsilon \cdot 2^i} \cdot [\mathcal{T}_{\textsc{query}}(i)]\right)\right)\,.$$

provided that only changes to the output of $\textsc{query}(\cdot)$ are reported. ∎

We conclude the section by proving Theorem 6.1, the main result that we use in other sections.

Proof of Theorem 6.1. Apply Theorem 6.4 using Theorem 6.2 and Theorem 6.3 respectively. Using these algorithms for all $i \in [k]$

$$\mathcal{T}_{\textsc{init}}(i) = O\left((\text{nnz}(A) + d)\log^{p-1}\left(\frac{nR}{\epsilon\delta}\right)\right) \text{ and } \mathcal{T}_{\textsc{query}}(i) = O\left(d\log^{p-1}\left(\frac{nR}{\epsilon\delta}\right) + n\epsilon_i^{-2}\log\left(\frac{nR}{\epsilon\delta}\right)\right)\,.$$

Next, to optimize the contribution of the ϵ_i terms to to the final runtime, pick $\alpha_i \propto 2^{i/3}$, i.e. $\alpha_i = 2^{i/3}/(\sum_{j\in[k]} 2^{j/3})$. Using that $\epsilon_i = 2^{-i}\alpha_i$ this yields that

$$\sum_{i\in[k]} \frac{1}{2^i} \cdot \frac{1}{\epsilon_i^2} = \sum_{i\in[k]} \frac{2^i}{\alpha_i^2} = \left(\sum_{i\in[k]} 2^{i/3}\right)^3 = \left(\frac{2^{(k+1)/3} - 1}{2^{1/3} - 1}\right)^3 = O(2^k) = O\left(\frac{R}{\epsilon}\right)$$

where in the last step we used the definition of k. Combining with the facts that $\sum_{i\in[k]} \frac{R}{\epsilon 2^i} = O(\frac{R}{\epsilon})$ and $k = O(\log(R/\epsilon))$ yields that

$$\sum_{i\in[k]} \mathcal{T}_{\text{INIT}}(i) = O\left((\text{nnz}(A) + d)\log^{p-1}\left(\frac{nR}{\epsilon\delta}\right)\log\left(\frac{R}{\epsilon}\right)\right) \text{ and}$$

$$\sum_{i\in[k]} \frac{R}{\epsilon\cdot 2^i}\cdot[\mathcal{T}_{\text{QUERY}}(i)] = O\left(d\cdot\frac{R}{\epsilon}\log^{p-1}\left(\frac{nR}{\epsilon\delta}\right) + n\left(\frac{R}{\epsilon}\right)^2\log\left(\frac{nR}{\epsilon\delta}\right)\right).$$

The result for $p = 2$ then follows via Theorem 6.4 and the $\log(R/\epsilon) = O(R/\epsilon)$.

To obtain the result for $p = 1$ we proceed identically and add one further improvement on the algorithm's implementation. In the case of $p = 1$ that rather spending $O(\text{nnz}(A)\log(R/\epsilon))$ time in each $D_i.\text{INIT}(\cdot)$ we can simply save the matrix once and use it for each D_i. This removes the logarithmic factors on the $\text{nnz}(A)$ terms in the runtime for $p = 1$ and yields the desire result. ∎

Even though we only use Algorithm 4 to prove Theorem 6.1 and in turn only apply Theorem 6.1 in a restricted set of settings, we provide the more general algorithm and analysis as it may be useful in additional settings. In particular, we allowed α to be a parameter because if we were in a setting where the runtime of each $D_i.\text{QUERY}(\cdot)$ had a different dependence on ϵ, e.g., ϵ^{-1} rather than ϵ^{-2}, then other configurations of α might be preferable, e.g., uniform with $\alpha_i = \frac{1}{k}$. The particular choice of $\alpha_i \propto 2^{i/3}$ in the proof of Theorem 6.1 improves over $\alpha_i = \frac{1}{k}$ by logarithmic factors.

Note that, with more careful analysis, it may be possible to improve the dependence on d in Theorem 6.1, potentially at the cost of additional logarithmic factors. The current dependence arises by accounting for at least d time whenever $j > 1$ on Line 12. However, in the case that Δ_t are sparse one could instead maintain the difference from x_t to each x_i^{ref} and seek faster implementations of $D_i.\text{QUERY}(\cdot)$ provided that the input changes sparsely. We do not pursue such an improvement for simplicity and since the term proportional to d does not affect our final runtimes.

7 Efficient gradient estimation via matrix-vector maintenance

In this section, build upon the data structures developed in the previous section to provide an efficient stochastic gradient oracle for the "softmax" approximation of the original objective. Recall that the "softmax" of functions $f_1, \ldots, f_n$ is

$$f_{\text{smax}}(x) = \epsilon'\log\left(\sum_{i\in[n]}\exp\left(\frac{f_i(x)}{\epsilon'}\right)\right),$$

$$\text{and } \nabla f_{\text{smax}}(x) = \sum_{i\in[n]} p_i(x)\nabla f_i(x) \text{ where } p_i(x) = \frac{\exp(f_i(x)/\epsilon')}{\sum_{i\in[n]}\exp(f_i(x)/\epsilon')}.$$

Throughout this section we assume that each f_i is L_g-smooth and L_f-Lipschitz.

Algorithm 5 provides an unbiased estimator of $\nabla f_{\text{smax}}(x)$ by leveraging a matrix-vector maintenance data structure $\mathcal{M}$. The algorithm takes as input a sequence of query points $x_1, \ldots, x_T$ that satisfies $\|x_t - x_0\| \le r$ and $\sum_{t\le T}\|x_t - x_{t-1}\| \le r'$ for $r, r' > 0$ such that $\frac{1}{2}L_g r^2 \le \epsilon'$. It outputs a sequence of vectors $\mathcal{G}(x_1), \ldots, \mathcal{G}(x_T)$ such that (informally) $\mathbb{E}[\mathcal{G}(x_t) \mid \mathcal{M}, x_1, \ldots, x_t] = \nabla f_{\text{smax}}(x_t)$ for all $t \le T$ with high probability. To compute these estimates the algorithm requires, with high probability, $\widetilde{O}(n + T)$ individual function value and gradient calculations, as well as $\widetilde{O}\left((n+T)d + d(L_f r'/\epsilon') + n(L_f r'/\epsilon')^2\right)$ additional runtime. We state this guarantee in full detail in the following.

THEOREM 7.1. (SOFTMAX GRADIENT ESTIMATOR) *Let $p \in \{1, 2\}$ and let $\{f_i\}_{i\in[n]}$ be L_g-smooth and L_f-Lipschitz with respect to $\|\cdot\|_p$. For all $t \in [T]$ assume that input x_t to Algorithm 5 is a (deterministic) function of the previous outputs $\mathcal{G}(x_1), \ldots, \mathcal{G}(x_{t-1})$, and that $\|x_t - x_0\|_p \le r$ and $\sum_{t\le T}\|x_t - x_{t-1}\| \le r'$ hold for parameters $r, r' > 0$ such that $\frac{1}{2}L_g r^2 \le \epsilon'$ and $\epsilon' \le L_f r'/2$. Let $\mathcal{F}_t$ be the filtration induced by all the random bits Algorithm 5 draws up to iteration t and all those that may be used by $\mathcal{M}$. Then for any error tolerance $\delta \in (0, 1)$ there exists event $\mathcal{E}$ such that the following hold:*

Algorithm 5: Softmax gradient estimator

Input: $\{f_i\}_{i\in[n]}$, query sequence $\{x_t\}_{t\le T}$ such that x_t is a function of the previous outputs $\mathcal{G}(x_1),\dots,\mathcal{G}(x_{t-1})$ (i.e., x_0 and x_1 do not depend on any outputs).

Parameters: Softmax tolerance ϵ', movement bound r', Lipschitz constant L_f, error tolerate $\delta\in(0,1)$, ℓ_p-matrix-vector maintenance data structure $\mathcal{M}$.

Call $\mathcal{M}.\textsc{init}(A, 0, r', \frac{\epsilon'}{L_f}, \frac{\delta}{2})$ where $A = [\frac{1}{L_f}\nabla f_i(x_0)^\top]_{i\in[n]}$

for $t = 1, 2, \cdots, T$ **do**

  $y_t \leftarrow L_f \cdot \mathcal{M}.\textsc{query}(x_t - x_{t-1})$ ▷ maintain vector $y_t \approx L_f A(x_t - x_0) = [\langle \nabla f_i(x_0), x_t - x_0\rangle]_{i\in[n]}$

  accepted $\leftarrow$ False

  while not accepted **do**

    Draw $i \sim \exp\left(\frac{f_i(x_0)+[y_t]_i}{\epsilon'}\right)$

    With probability $\min\left\{\exp\left(\frac{f_i(x_t)-f_i(x_0)-[y_t]_i}{\epsilon'} - 2\right), 1\right\}$

      yield $i_t = i$ and $\mathcal{G}(x_t) = \nabla f_{i_t}(x_t)$

      accepted $\leftarrow$ True

- *We have* $\mathbb{P}(\mathcal{E}) \ge 1 - \delta$.
- *When* $\mathcal{E}$ *holds we have* $\mathbb{E}[\mathcal{G}(x_t) \mid \mathcal{F}_{t-1}] = \nabla f_{\text{smax}}(x_t)$ *for all* $t \in [T]$.
- *When* $\mathcal{E}$ *holds, Algorithm 5 makes* $O(n + T\log(1/\delta))$ *queries of the form* $\{f_i(x), \nabla f_i(x)\}$, *and requires additional runtime*

$$O\left(T\left(d + \log\left(\frac{1}{\delta}\right)\right) + \left(nd\log^{p-1}\left(\frac{L_f r'}{\epsilon'}\right) + d\left(\frac{L_f r'}{\epsilon'}\right)\right)\log^{p-1}\left(\frac{nL_f r'}{\epsilon'\delta}\right) + n\left(\frac{L_f r'}{\epsilon'}\right)^2 \log\frac{nL_f r'}{\epsilon'\delta}\right).$$

- *With probability 1 we have* $\|\mathcal{G}(x_t)\|_{p^\star} \le L_f$, *where* p^* *is such that* $\frac{1}{p} + \frac{1}{p^*} = 1$.

Proof. We prove the theorem by coupling Algorithm 5 to an "alternative" algorithm that uses $\mathcal{M}$ in a strictly oblivious manner, and produces a potentially different sequence of indices $i'_1,\dots,i'_T$, queries $x'_2,\dots,x'_T$ and matrix-vector estimates $y'_t = L_f \cdot \mathcal{M}.\textsc{query}(x'_t - x'_{t-1})$. The alternative algorithm proceeds exactly like Algorithm 5, except at every iteration it tests whether

$$\max_{i\in[n]}\left|[y'_t]_i - \langle \nabla f_i(x_0), x'_t - x_0\rangle\right| = L_f\left\|\tfrac{1}{L_f}y'_t - A(x'_t - x_0)\right\|_\infty \le \epsilon' \tag{7.32}$$

holds. As long as this condition holds, the algorithm produces i'_t using rejection sampling as in Algorithm 5 (and with the same random bits). If at any $t \le T$ the condition fails, the algorithm proceeds to directly draw $i'_t \sim e^{f_i(x'_t)/\epsilon'}$ at all subsequent iterations, ignoring the values of y'_t. Thus, both algorithms produce identical outputs $\mathcal{G}(x'_t) = \mathcal{G}(x_t)$ leading to identical queries $x'_t = x_t$ whenever Equation (7.32) holds for all $t \in [T]$.

For the alternative algorithm we have $i'_t \sim e^{f_i(x'_t)/\epsilon'}$ for all $t \in [T]$, regardless of randomness in $\mathcal{M}$. To see this, note that by smoothness of the f_i we have

$$\left|\frac{f_i(x'_t) - f_i(x_0) - \langle \nabla f_i(x_0), x'_t - x_0\rangle}{\epsilon'}\right| \le \frac{\frac{1}{2}L_g r^2}{\epsilon'} \le 1$$

for all $i \in [n]$, by our assumptions that $\|x'_t - x_0\|_p \le r$ and $\frac{1}{2}L_g r^2 \le \epsilon'$. Consequently, when Equation (7.32) holds we have

$$\left|\frac{f_i(x'_t) - f_i(x_0) - [y'_t]_i}{\epsilon'}\right| \le 1 + \left|\frac{\langle \nabla f_i(x_0), x'_t - x_0\rangle - [y'_t]_i}{\epsilon'}\right| \le 2$$

for all $i \in [n]$. Therefore, $\exp\left(\frac{f_i(x_t')-f_i(x_0)-[y_t']_i}{\epsilon'} - 2\right) \le 1$ and (by standard analysis of rejection sampling) we have $\mathbb{P}(i_t' = i) \propto \exp\left(\frac{f_i(x_0)+[y_t']_i}{\epsilon'}\right) \cdot \exp\left(\frac{f_i(x_t')-f_i(x_0)-[y_t']_i}{\epsilon'}\right) = e^{f_i(x_t')/\epsilon'}$. As a consequence, the alternative algorithm's outputs satisfy $\mathbb{E}\left[\mathcal{G}(x_t') \mid \mathcal{F}_{t-1}'\right] = \nabla f_{\mathrm{smax}}(x_t')$ for all $t \le T$ by definition of the softmax function.

Since the alternative algorithm's queries are oblivious to the data structure's randomness, we may apply[6] Theorem 6.1 to conclude that, with probability at least $1 - \frac{\delta}{2}$ we have $\left\| \frac{1}{L_f} y_t' - A(x_t' - x_0) \right\|_\infty \le \frac{\epsilon'}{L_f}$ for all $t \le T$, implying that the condition (7.32) holds for all $t \le T$ and therefore both algorithms produce identical outputs. This defines a probability $\ge 1 - \frac{\delta}{2}$ event under which $\mathbb{E}[\mathcal{G}(x_t) \mid \mathcal{F}_{t-1}] = \nabla f_{\mathrm{smax}}(x_t)$ for all $t \le T$, giving the first part of the theorem.

For the second part of the theorem, we note that smoothness and the condition (7.32) also imply that the rejection probability $\exp\left(\frac{f_i(x_t')-f_i(x_0)-[y_t']_i}{\epsilon'} - 2\right) \ge e^{-4}$. Therefore, the expected number of rejection sampling steps in the alternative algorithm is $O(1)$. By standard Chernoff bounds [see, e.g., 20], with probability at least $1 - \frac{\delta}{2}$ the alternative algorithms makes $O(T \log \frac{1}{\delta})$ rejection sampling steps throughout. Thus, by a union bound we have that with probability $1 - \delta$, Algorithm 5 and the alternative algorithm are identical, with each making $O(T \log \frac{1}{\delta})$ rejection sampling steps. Each rejection sampling step costs $O(1)$ function and gradient evaluations, and to construct the matrix A we require n additional evaluations. Additionally, given the computational model of this paper, with $O(n)$ preprocessing we can implement each random sampling of i in $O(1)$ time as discussed in Section 3. Altogether, this brings the overall cost to $O(n + T \log \frac{1}{\delta})$ and the bound on additional runtime follows immediately from Theorem 6.1.

Finally, the third part of the theorem is immediate from noting that $\mathcal{G}(x_t) = \nabla f_{i_t}(x_t)$ for some $i_t \in [n]$ and therefore $\|\mathcal{G}(x_t)\|_{p^*} \le L_f$ by the Lipschitz continuity of the f_i. ∎

8 Runtime bounds

We now put together the pieces constructed in the previous sections to obtain runtime bounds for minimizing the maximum of convex functions. In Section 8.1 we study general convex functions, in Section 8.2 we specialize our results to linear functions, and in Section 8.3 we specialize them further to the problem of finding a minimum enclosing ball.

8.1 General convex functions.

Recall the problem

$$\underset{x \in \mathcal{X}}{\text{minimize}} \left\{ f_{\max}(x) \coloneqq \max_{y \in \Delta^n} \sum_{i \in [n]} y_i f_i(x) \right\}. \tag{8.33}$$

We consider the problem in two different settings, which we call the ball setup or the simplex setup, formally defined as follows.

Definition 8.1. (Ball setup) *In the* ball setup, *the norm $\|\cdot\|$ is the Euclidean norm $\|\cdot\|_2$, the domain $\mathcal{X}$ is a closed and convex subset of the unit Euclidean ball $\mathbb{B}^d = \{x \in \mathbb{R}^d \mid \|x\|_2 \le 1\}$, and the Bregman divergence is $V_x(y) = \frac{1}{2}\|y - x\|_2^2$. Furthermore, we let $\mathcal{X}_\nu \coloneqq \mathcal{X}$ for all $\nu \ge 0$.*

Definition 8.2. (Simplex setup) *In the* simplex setup, *the norm $\|\cdot\|$ is the 1-norm $\|\cdot\|_1$, the domain $\mathcal{X}$ is a closed and convex subset of the probability simplex $\Delta^d = \{x \in \mathbb{R}^d_{\ge 0} \mid \sum_{i \in [d]} x_i = 1\}$, and the Bregman divergence is $V_x(y) = \sum_{i \in [d]} y_i \log \frac{y_i}{x_i}$. Furthermore, we let $\mathcal{X}_\nu \coloneqq \{x \in \mathcal{X} \mid x_i \ge \nu,\ \forall i \in [d]\}$ for all $\nu \ge 0$.*

We introduce the set $\mathcal{X}_\nu$ in the definitions above in order to satisfy the τ-triangle in the simplex setup; see Definition 5.1 and Example 5.2.

The following is our main result concerning the complexity of solving the problem (8.33).

Theorem 8.1. *Consider the problem (8.33) in either the ball or simplex setups (Definitions 8.1 and 8.2, respectively), where each function f_i is convex, L_f-Lipschitz, and L_g-smooth with respect to $\|\cdot\|$. Let $\epsilon > 0$,*

[6] The matrix A satisfies $\|A\|_{p \to \infty} \le 1$ since $\|\nabla f_i(x_0)\|_{p^*} \le L_f$ for all $i \in [n]$ by the Lipschitz continuity assumption.

let $\nu = \frac{\epsilon}{4dL_f}$*, and for initial point* $x_0 \in \mathcal{X}_\nu$ *let* $\max_{x\in\mathcal{X}_\nu} V_{x_0}(x) \le \frac{1}{2}R^2$*. Then, Algorithm 1 with parameters* $r \le \sqrt{\frac{\epsilon}{L_g \log n}}$*,* R*,* $\mathcal{E}_0 = L_f R$*, accuracy* $\frac{\epsilon}{8}$*, ball oracle implementation Algorithm 2 and gradient oracle implementation in Algorithm 5, return a point* x *such that*

$$\mathbb{E} f_{\max}(x) - \min_{x_\star \in \mathcal{X}} f_{\max}(x_\star) \le \epsilon.$$

Let $\mathcal{T}_{\text{eval}}$ *be the time to compute* $f_i(x), \nabla f_i(x)$ *for any* $x \in \mathcal{X}$ *and* $i \in [n]$*, and let* $\mathcal{T}_{\text{md}}$ *be the time to compute a mirror descent step of the form* $\operatorname{argmin}_{z\in\mathcal{X}_\nu}\{\langle g, z\rangle + \lambda V_y(z) + V_x(z)\}$ *for any* $g \in \mathbb{R}^d$ *and* $x, y \in \mathcal{X}$*. For* $\epsilon \le \min\left\{L_g R^2, L_f^2/L_g\right\}$ *and* $r = \min\left\{\sqrt{\frac{\epsilon}{L_g \log n}}, \frac{\epsilon\sqrt{\mathcal{T}_{\text{eval}}+d}}{L_f}\right\}$ *with probability at least* $\frac{9}{10}$*, the algorithm has runtime*

$$\widetilde{O}\left(n(\mathcal{T}_{\text{eval}}+d)\left(\frac{L_g R^2}{\epsilon}\right)^{1/3} + n\left(\frac{(\mathcal{T}_{\text{eval}}+d)L_f R}{\epsilon}\right)^{2/3} + (\mathcal{T}_{\text{eval}}+\mathcal{T}_{\text{md}}+d)\left(\frac{L_f R}{\epsilon}\right)^2\right). \tag{8.34}$$

Proof. We establish the theorem in four steps: reducing the objective to softmax on a truncated domain, describing the gradient oracle implementation, arguing the correctness of our methods, and finally bounding the runtime.

Reduction. We claim it suffices to solve to $\epsilon/4$ additive error the problem

$$\operatorname*{minimize}_{x\in\mathcal{X}_\nu} f_{\text{smax}}(x) \quad \text{where} \quad f_{\text{smax}}(x) = \epsilon' \log\left(\sum_{i\in[n]} \exp\left(\frac{f_i(x)}{\epsilon'}\right)\right), \ \epsilon' = \frac{\epsilon}{2\log n}, \ \nu = \frac{\epsilon}{4dL_f}. \tag{8.35}$$

To see the claim is true, note we have $|f_{\text{smax}}(x) - f_{\max}(x)| \le \epsilon/2$ for all $x \in \mathcal{X}$ and

$$\min_{x\in\mathcal{X}} f_{\max}(x) \le \min_{x\in\mathcal{X}_\nu} f_{\max}(x) \le \min_{x\in\mathcal{X}} f_{\max}(x) + \epsilon/4$$

due to L_f-Lipschitz continuity of f. Consequently, for any $\tilde{x}$ that is an $\epsilon/4$ approximate optimizer of the true minimizer of (8.35), we have

$$f_{\text{smax}}(\tilde{x}) \le \min_{x\in\mathcal{X}_\nu} f_{\text{smax}}(x) + \epsilon/4 \le \min_{x\in\mathcal{X}_\nu} f_{\max}(x) + \epsilon/2 + \epsilon/4 \le \min_{x\in\mathcal{X}} f_{\max}(x) + \epsilon/4 + \epsilon/2 + \epsilon/4.$$

This proves such $\tilde{x}$ is also an ϵ-additive minimizer of the original problem (8.35). We therefore focus on solving (8.35) to $\epsilon/4$ additive error.

Stochastic gradient oracle. At the t'th ball oracle call in the outer loop (Algorithm 1) we instantiate a gradient estimator $\mathcal{G}$ for ∇f_{smax} using Algorithm 5 with initial point $\Phi_t(v_t)$, parameters r and $r' = \widetilde{O}(r)$, and failure probability $\delta = \frac{\epsilon}{L_f R} \cdot \frac{1}{100 T_{\text{outer}}}$ with $T_{\text{outer}} = \widetilde{O}((R/r)^{2/3})$ such that Theorem 4.1 guarantees (via Markov's inequality) that Algorithm 1 requires at most T_{outer} iterations with probability at least $\frac{99}{100}$. Since Theorem 7.1 only guarantees that this gradient estimator is unbiased for ∇f_{smax} with high probability, we repeat the coupling argument from the proof of Theorem 7.1. Namely, we consider "alternative" completely unbiased gradient estimators that with probability at least $1-\delta$ produce identical outputs to Algorithm 5. We then analyze an alternative algorithm with the alternative estimators, and use the fact that with probability at least $1 - \delta T_{\text{outer}}$ it produces the same output as our algorithm. By our choice of δ and T_{outer}, we have that with probability at least $1 - \frac{\epsilon}{50 L_f R}$ the actual and alternative gradient estimators produce identical outputs for the entire duration of the algorithm.

Correctness. For the ball and simplex setups, our chosen Bregman divergence $V_x(y)$ is 1-strongly convex with respect to the ℓ_2 or ℓ_1 norm, respectively. The corresponding divergence also satisfies a τ-triangle inequality (Definition 5.1) with $\tau = \widetilde{\Theta}(1)$. For the k'th out loop iteration, let us argue that the stochastic gradient queries made the inner loop of Algorithm 2 satisfy the conditions of Theorem 7.1. Let $\mathcal{G}$ denote the estimator for ∇f_{smax} defined above, and let $\mathcal{G}_k(x) = a_{k+1}\mathcal{G}(\Phi_k(x))$ be the gradient estimator for h_k defined in Algorithm 1 and let $y = \Phi(v_k)$. Let $x_1, \ldots, x_T$ denote the sequence of queries to $\mathcal{G}_k$ made by Algorithm 2. Then Theorem 5.1 guarantees that $\|x_t - v_k\| \le \rho$ for all $t \in [T]$ and that $\sum_{t\in[T]} \|x_t - x_{t-1}\| = \widetilde{O}(\rho)$. By design of Algorithm 1 we have that $\Phi_k(z) - \Phi_k(z') = \frac{r}{\rho}(z - z')$ and therefore the queries to $\mathcal{G}$ satisfy $\|\Phi_k(x_t) - y\| \le r$ for all $t \in [T]$ and $\sum_{t\in[T]} \|\Phi_k(x_t) - \Phi_k(x_{t-1})\| = \widetilde{O}(r) = r'$ as required by Theorem 7.1.

With the conditions of Theorem 7.1 satisfied, we have a nearly unbiased gradient estimator for f_{smax}, that with probability at least $1 - \frac{\epsilon}{50L_f R}$ produce identical outputs to a completely unbiased gradient for the entire duration of the algorithm, as discussed above. Theorem 5.1 then guarantees that (using the alternative gradient estimator) Algorithm 2 implements a valid $(\rho, \gamma, c_{\max})$ restricted proximal oracle for $\rho = \widetilde{\Theta}(R^{2/3}r^{1/3})$, $\gamma = \widetilde{O}(1)$ and $c_{\max} < \infty$. We may therefore apply Theorem 4.1 (with $\epsilon \to \epsilon/8$) to conclude that with alternative gradient estimator we output x' such that

$$\mathbb{E} f_{\text{smax}}(x') \le \min_{x_\star \in \mathcal{X}_\nu} f_{\text{smax}}(x_\star) + \frac{\epsilon}{8}.$$

Letting x be the output of the algorithm using the actual gradient estimator, we have

$$\begin{aligned}
\mathbb{E} f_{\text{smax}}(x) &= \mathbb{E} f_{\text{smax}}(x') \mathbb{1}_{\{x = x'\}} + \mathbb{E} f_{\text{smax}}(x) \mathbb{1}_{\{x \neq x'\}} \\
&\overset{(i)}{\le} \mathbb{E} f_{\text{smax}}(x') + \mathbb{E} L_f \|x - x_0\| \mathbb{1}_{\{x \neq x'\}} \\
&\overset{(ii)}{\le} \mathbb{E} f_{\text{smax}}(x') + L_f R \cdot \mathbb{P}(x \neq x') \\
&\overset{(iii)}{\le} \min_{x_\star \in \mathcal{X}_\nu} f_{\text{smax}}(x_\star) + \frac{\epsilon}{8} + L_f R \cdot \frac{1}{50 L_f R} \le \min_{x_\star \in \mathcal{X}_\nu} f_{\text{smax}}(x_\star) + \frac{\epsilon}{4},
\end{aligned}$$

due to the (i) Lipschitz continuity of f_{smax}, (ii) the definition of R, and (iii) the bounds on $\mathbb{E} f(x')$ and the probability of $x = x'$ discussed above. This proves the correctness of our algorithm.

Complexity. By Theorem 4.1 and the discussion above, the outer loop (Algorithm 1) terminates in $T_{\text{outer}} = \widetilde{O}(R^{2/3} r^{-2/3})$ iterations with probability at least $\frac{99}{100}$. Each iteration of the outer loop performs $O(1)$ operations on d-dimensional vectors and makes one call to a ball restricted proximal oracle.

By Theorem 5.1, each restricted ball oracle call makes $\widetilde{O}\left(\Gamma^2/\rho^2\right)$ calls to the gradient estimator, mirror descent step computations, and d-dimensional vector arithmetic operations.[7] Recalling that $r' = \widetilde{O}(r)$ and $\epsilon' = \widetilde{O}(\epsilon)$, Theorem 7.1 gives that, with probability at least $1 - \frac{1}{100 T_{\text{outer}}}$ the runtime of a restricted oracle call is at most

$$T_{\text{inner}} = \widetilde{O}\left(n \frac{L_f r^2}{\epsilon^2} + d \frac{L_f r}{\epsilon} + n(\mathcal{T}_{\text{eval}} + d) + (\mathcal{T}_{\text{eval}} + \mathcal{T}_{\text{md}} + d) \frac{\Gamma^2}{\rho^2} \right).$$

By Theorems 4.1 and 7.1 we have that $\rho = \widetilde{\Theta}(R^{2/3} r^{1/3})$ and $\Gamma = \widetilde{O}\left(\frac{L_f r^{2/3} R^{4/3}}{\epsilon}\right)$. Moreover, the number of ball oracle calls is bounded by $T_{\text{outer}} = \widetilde{O}(R^{2/3} r^{-2/3})$ with probability at least $\frac{99}{100}$. Substituting and applying a union bound, we get that the total runtime of the algorithm is bounded by

$$T_{\text{outer}} \cdot T_{\text{inner}} = \widetilde{O}\left(n \frac{L_f^2 R^{2/3} r^{4/3}}{\epsilon^2} + n(\mathcal{T}_{\text{eval}} + d) \frac{R^{2/3}}{r^{2/3}} + (\mathcal{T}_{\text{eval}} + \mathcal{T}_{\text{md}} + d) \frac{L_f^2 R^2}{\epsilon^2} \right)$$

with probability at least $\frac{9}{10}$. Substituting $r = \min\left\{ \sqrt{\frac{\epsilon}{L_g \log n}}, \frac{\epsilon \sqrt{\mathcal{T}_{\text{eval}} + d}}{L_f} \right\}$ yields the claimed bound (8.34) and completes the proof. ∎

8.2 Matrix games. In the special case where $f_i(x) = [A^\top x]_i$ are linear functions, the ball and simplex setups reduce to ℓ_p-ℓ_1 matrix games with $p \in \{2, 1\}$, respectively. Formally, the problem definition is

$$\text{minimize}_{x \in \mathcal{X}} \left[\max_{y \in \Delta^n} x^\top A y \right], \quad \text{where } \mathcal{X} = \Delta^d \text{ for } \ell_1\text{-}\ell_1 \text{ and } \mathcal{X} = \mathbb{B}^d \text{ for } \ell_2\text{-}\ell_1. \tag{8.36}$$

To simplify expressions, we assume that each $f_i(x)$ is 1-Lipschitz in $\|\cdot\|_p$, which is equivalent to assuming that

$$\|A\|_{p \to \infty} = \begin{cases} \max_{j,i} |A_{ji}| & \text{for } \ell_1\text{-}\ell_1 \text{ games} \\ \max_{i \in [n]} \|A_{:i}\|_2 & \text{for } \ell_2\text{-}\ell_1 \text{ games} \le 1 \end{cases} \tag{8.37}$$

Our runtime guarantees are as follows.

[7] Logarithmic factors in Theorem 5.1 depend on a bound for $\max_{x,y \in \mathcal{X}_\nu} V_x(y)$ whereas we only assumed $\max_{y \in \mathcal{X}_\nu} V_{x_0}(y) \le R^2/2$. However, a τ-triangle inequality with $\tau = \widetilde{O}(1)$ implies that $\max_{x,y \in \mathcal{X}_\nu} V_x(y) = \widetilde{O}(\max_{y \in \mathcal{X}_\nu} V_{x_0}(y))$.

COROLLARY 8.1. (MATRIX GAMES) *For $p \in \{1,2\}$, consider the problem of ℓ_p-ℓ_1 matrix games (8.36) under the assumption (8.37). For $\epsilon \in (0,1)$ and $\nu = \epsilon/(4d)$, Algorithm 1 with parameters $r = \min(1, \sqrt{d}\epsilon)$, $R = \widetilde{O}(1)$, $\mathcal{E}_0 = R$, accuracy $\epsilon/4$, ball oracle implementation in Algorithm 2 and gradient oracle implementation in Algorithm 5, return a point x such that*

$$\mathbb{E} \min_{x \in \mathcal{X}} \left[\max_{y \in \Delta^n} x^\top A y \right] - \min_{x_\star \in \mathcal{X}} \left[\max_{y_\star \in \Delta^n} x_\star^\top A y_\star \right] \le \epsilon.$$

With probability at least $\frac{9}{10}$ the runtime of the algorithm is

$$\widetilde{O}\left(nd + nd^{2/3} \frac{1}{\epsilon^{2/3}} + d \frac{1}{\epsilon^2} \right).$$

Proof. We invoke Theorem 8.1 with $L_f = 1$ (by assumption) and $L_g = 0$ (since each function is linear). For matrix games we have $\mathcal{T}_{\text{eval}} = O(d)$. Let us also argue that $\mathcal{T}_{\text{md}} = \widetilde{O}(d)$, recalling that $\mathcal{T}_{\text{md}}$ is the time to find $w = \operatorname{argmin}_{w \in \mathcal{X}} \{\langle g, w \rangle + \lambda V_y(w) + V_z(w)\}$ for some $y, z \in \mathcal{X}_\nu$ and $g \in \mathbb{R}^d$. In the ball setup we simply have

$$w = \Pi_{\mathbb{B}^d}\left(\frac{z + \lambda y - g}{1 + \lambda} \right) \quad \text{where} \quad \Pi_{\mathbb{B}^d}(x) = \frac{x}{\max\{1, \|x\|\}}$$

is the Euclidean projection onto $\mathbb{B}^d$. Therefore, $\mathcal{T}_{\text{md}} = O(d)$ in the ball setup.

In the (truncated) simplex setup $\mathcal{X} = \Delta_\nu^d$ with some $\nu \in (0, 1/2d]$, we can implement the mirror descent step as follows. Let $\xi = z^{\frac{1}{1+\lambda}} \circ y^{\frac{\lambda}{1+\lambda}} \circ \exp(-\frac{1}{1+\lambda} g)$, where we use $\circ$ to represent element-wise product. Let σ be a permutation of $(1, \dots, d)$ such that ξ_{σ_i} is the i-th largest entry of ξ (breaking ties arbitrarily). Now define $\alpha_i = \frac{\nu \sum_{j \le i} \xi_{\sigma_j}}{1 - \nu(d-i)}$ (so that $\frac{\alpha_i}{\sum_{j \le i} \xi_{\sigma_j} + \alpha_i (d-i)} = \nu$), and the cutoff index $i' \in [d]$ to be the largest $i \in [d]$ such that $\frac{\xi_{\sigma_i}}{\sum_{j \le i} \xi_{\sigma_j}} \ge \frac{\nu}{1 - \nu(d-i)}$. Such $i' \in [d]$ must be well-defined as the inequality is satisfied when $i = 1$. It is then straightforward to verify that $w \in \mathbb{R}^d$ such that for all $i \in [d]$

$$w_i = \begin{cases} \frac{\nu}{\alpha_{i'}} \cdot \xi_{\sigma_i} & \text{if } i \le i', \\ \nu & \text{if } i > i' \end{cases}$$

is the solution to the problem defining the mirror descent step. Computing ξ takes $O(d)$ time, sorting it takes $O(d \log d)$ time, and finding i' and calculating w each take additional $O(d)$ time, so overall $\mathcal{T}_{\text{md}} = \widetilde{O}(d)$ in the simplex setup.

Plugging $L_f = 1$, $L_g = 0$, and $\mathcal{T}_{\text{eval}}, T_{\text{outer}} = \widetilde{O}(d)$ into Equation (8.34) yields the claimed runtime bound. ∎

8.3 Minimum Enclosing Ball. In this section, we apply our method to solving the minimum enclosing ball problem, defined as follows. Given data points $a_1, \dots, a_n \in \mathbb{R}^d$ such that $a_1 = 0$ and $\max_{i \in [n]} \|a_i\|_2 = 1$, the goal is to find the minimum radius $R_\star$ ball containing all data points. That is,

$$\frac{1}{2} R_\star^2 = \min_{x \in \mathbb{R}^d} \max_{y \in \Delta^n} f_i(x) \quad \text{where} \quad f_i(x) = \frac{1}{2} \|x - a_i\|_2^2. \tag{8.38}$$

The problem is also equivalent to an ℓ_2-ℓ_1 matrix game with a quadratic regularization term, but for our purpose the natural formulation above is more convenient. Letting $x_\star := \operatorname{argmin}_{x \in \mathbb{R}^d} \max_{y \in \Delta^n} f_i(x)$, it holds without loss of generality that $\|x_\star\|_2 \le 1$ and $R_\star \in [\frac{1}{2}, 1]$ (see Allen-Zhu et al. [2] for detailed explanation). Under these assumptions, we obtain the following runtime guarantee.

COROLLARY 8.2. (MINIMUM ENCLOSING BALL) *Consider the problem (8.38) with $a_1 = 0$ and $\max_{i \in [n]} \|a_i\|_2 \le 1$ (so that $\|x_\star\| \le 1$ and $R_\star \ge 1/2$). For any $\epsilon \in (0,1)$, there is an algorithm that makes $\widetilde{O}(1)$ calls to Algorithm 1 with ball oracle implementation Algorithm 2 and gradient oracle implementation in Algorithm 5 and, with probability at least $\frac{9}{10}$ returns a point x such that*

$$\frac{1}{2} \|x - x_\star\|_2^2 \le \epsilon \cdot R_\star^2$$

with total runtime

$$\widetilde{O}\left(nd + nd^{2/3}\epsilon^{-1/3} + d\epsilon^{-1}\right).$$

Proof. Let $K = \log_2 \frac{4}{\epsilon}$. We use Theorem 8.1 with $f_i(x) = \frac{1}{2}\|x - a_i\|_2^2$ defined above, and boost its result to failure probability $\frac{1}{10K}$ by repeatedly calling the algorithm $\widetilde{O}(1)$ times, cutting it off whenever it exceeds the runtime bound, and selecting the best result in $\widetilde{O}(nd)$ time. We apply this high-probability solver recursively, generating a sequence of solutions $x^{(0)}, \ldots, x^{(K)}$ that satisfies, with probability at least $\frac{9}{10}$,

$$\frac{1}{2}\|x^{(k)} - x_\star\|_2^2 \le 2^{-(k+1)} \le 2^{-k} 2R_\star^2 \quad \text{for all} \quad k \le K,$$

so that $x = x^{(K)}$ satisfies $\frac{1}{2}\|x - x_\star\|_2^2 \le \epsilon \cdot R_\star^2$ as required.

To generate $x^{(0)}, \ldots, x^{(K)}$, we start with $x^{(0)} = 0$, which satisfies $\frac{1}{2}\|x^{(0)} - x_\star\|_2^2 \le \frac{1}{2} \le 2R_\star^2$ by assumption. To produce $x^{(k)}$ for $k \ge 1$ we apply our algorithm on with parameters $R_k = 2^{-(k-1)/2}$, $\epsilon_k = 2^{-(k+1)}$, $L_g = 1$ and $L_f = O(1)$ on the domain $\mathcal{X}_k = \left\{x \mid \|x - x^{(k-1)}\| \le 2^{-(k-1)/2}\right\}$, which contains $x_\star$ by the inductive assumption that $\|x^{(k-1)} - x_\star\|_2 \le 2^{-(k-1)/2}$. The 1-strong-convexity of our objective function then guarantees (with the appropriate probability) that $\frac{1}{2}\|x^{(k)} - x_\star\|_2^2 \le \epsilon_k = 2^{-(k+1)}$, completing the induction. The runtime to produce $x^{(k)}$ is

$$\begin{aligned}
&\widetilde{O}\left(n(\mathcal{T}_{\text{eval}} + d)\left(\frac{R_k^2}{\epsilon_k}\right)^{1/3} + n\left(\frac{(\mathcal{T}_{\text{eval}} + d)R_k}{\epsilon_k}\right)^{2/3} + (\mathcal{T}_{\text{eval}} + \mathcal{T}_{\text{md}} + d)\left(\frac{R_k}{\epsilon_k}\right)^2\right) \\
&\qquad = \widetilde{O}\left(nd + nd^{2/3} \cdot 2^{k/3} + d \cdot 2^k\right),
\end{aligned}$$

where the transition follows from substituting R_k, ϵ_k, and plugging in $\mathcal{T}_{\text{eval}} = \mathcal{T}_{\text{md}} = O(d)$. Summing this over $k \in [K]$ and recalling that $2^K = O(\frac{1}{\epsilon})$ yields the claimed runtime bound. ∎

Acknowledgments

We thank Kfir Levy for suggesting the work [23] may be useful for ball oracle acceleration, and the anonymous reviewers for their helpful feedback.

YC was supported in part by the Israeli Science Foundation (ISF) grant no. 2486/21 and the Len Blavatnik and the Blavatnik Family foundation. YJ was supported in part by a Stanford Graduate Fellowship and the Dantzig-Lieberman Fellowship. AS was supported in part by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship.

References

[1] I. Adler. The equivalence of linear programs and zero-sum games. *International Journal of Game Theory*, 42: 165–177, 2013.

[2] Z. Allen-Zhu, Z. Liao, and Y. Yuan. Optimization algorithms for faster computational geometry. In *International Colloquium of Automata, Languages and Programming*, 2016.

[3] H. Asi, Y. Carmon, A. Jambulapati, Y. Jin, and A. Sidford. Stochastic bias-reduced gradient methods. *Advances in Neural Information Processing Systems*, 34, 2021.

[4] B. Axelrod, Y. P. Liu, and A. Sidford. Near-optimal approximate discrete and continuous submodular function minimization. In *Symposium on Discrete Algorithms, (SODA)*, 2020.

[5] P. Balamurugan and F. Bach. Stochastic variance reduction methods for saddle-point problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[6] D. Blackwell. Large deviations for martingales. In *Festschrift for Lucien Le Cam*, 1997.

[7] J. H. Blanchet and P. W. Glynn. Unbiased Monte Carlo for optimization and functions of expectations via multi-level randomization. In *2015 Winter Simulation Conference (WSC)*, pages 3656–3667, 2015.

[8] S. Bubeck, Q. Jiang, Y. T. Lee, Y. Li, and A. Sidford. Complexity of highly parallel non-smooth convex optimization. *arXiv:1906.10655*, 2019.

[9] S. Bubeck, Q. Jiang, Y. T. Lee, Y. Li, and A. Sidford. Near-optimal method for highly smooth convex optimization. In *Proceedings of the Thirty Second Annual Conference on Computational Learning Theory*, pages 492–507, 2019.

[10] B. Bullins. Highly smooth minimization of non-smooth problems. In *Conference on Learning Theory*, pages 988–1030, 2020.

[11] Y. Carmon and D. Hausler. Distributionally robust optimization via ball oracle acceleration. *arXiv:2203.13225*, 2022.

[12] Y. Carmon and O. Hinder. Making SGD parameter-free. In *Conference on Learning Theory (COLT)*, 2022.

[13] Y. Carmon, Y. Jin, A. Sidford, and K. Tian. Variance reduction for matrix games. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[14] Y. Carmon, A. Jambulapati, Q. Jiang, Y. Jin, Y. T. Lee, A. Sidford, and K. Tian. Acceleration with a ball optimization oracle. In *Advances in Neural Information Processing Systems*, 2020.

[15] Y. Carmon, Y. Jin, A. Sidford, and K. Tian. Coordinate methods for matrix games. In *Symposium on Foundations of Computer Science (FOCS)*, 2020.

[16] Y. Carmon, A. Jambulapati, Y. Jin, and A. Sidford. Thinking inside the ball: Near-optimal minimization of the maximal loss. In *Conference on Learning Theory*, 2021.

[17] Y. Carmon, D. Hausler, A. Jambulapati, Y. Jin, and A. Sidford. Optimal and adaptive Monteiro-Svaiter acceleration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[18] Y. Carmon, A. Jambulapati, Y. Jin, and A. Sidford. RECAPP: Crafting a more efficient catalyst for convex optimization. In *International Conference on Machine Learning (ICML)*, 2022.

[19] M. Charikar, K. C. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium of Automata, Languages and Programming*, 2018.

[20] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

[21] K. L. Clarkson, E. Hazan, and D. P. Woodruff. Sublinear optimization for machine learning. *Journal of the ACM (JACM)*, 59(5):1–49, 2012.

[22] M. B. Cohen, Y. T. Lee, and Z. Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

[23] A. Cutkosky. Anytime online-to-batch, optimism and acceleration. In *International Conference on Machine Learning (ICML)*, 2019.

[24] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1953.

[25] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *International Conference on Machine Learning (ICML)*, 2008.

[26] R. Frostig, R. Ge, S. Kakade, and A. Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In *International Conference on Machine Learning (ICML)*, 2015.

[27] A. V. Gasnikov, P. E. Dvurechensky, E. Gorbunov, E. A. Vorontsova, D. Selikhanovych, and C. A. Uribe. Optimal tensor methods in smooth convex and uniformly convex optimization. In *Proceedings of the Thirty Second Annual Conference on Computational Learning Theory*, pages 1374–1391, 2019.

[28] M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.

[29] O. Güler. New proximal point algorithms for convex minimization. *SIAM Journal on Optimization*, 2(4): 649–664, 1992.

[30] M. Ivgi, O. Hinder, and Y. Carmon. DoG is SGD's best friend: A parameter-free dynamic step size schedule. In *International Conference on Machine Learning (ICML)*, 2023.

[31] B. Jiang, H. Wang, and S. Zhang. An optimal high-order tensor method for convex optimization. In *Proceedings of the Thirty Second Annual Conference on Computational Learning Theory*, pages 1799–1801, 2019.

[32] S. Jiang, Z. Song, O. Weinstein, and H. Zhang. A faster algorithm for solving general lps. In S. Khuller and V. V. Williams, editors, *Proceedings of the Fifty-Third Annual ACM Symposium on the Theory of Computing*, pages 823–832. ACM, 2021.

[33] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Symposium on Theory of Computing (STOC)*, pages 302–311, 1984.

[34] D. Kovalev and A. Gasnikov. The first optimal acceleration of high-order methods in smooth convex optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[35] G. Lan. Gradient sliding for composite optimization. *Mathematical Programming*, 159(1):201–235, 2016.

[36] G. Lan and Y. Ouyang. Accelerated gradient sliding for structured convex optimization. *Computational Optimization and Applications*, 82(2):361–394, 2022.

[37] K. G. Larsen, R. Pagh, and J. Tetek. CountSketches, feature hashing and the median of three. In *International Conference on Machine Learning (ICML)*, 2021.

[38] Y. T. Lee and A. Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 147–156, 2013.

[39] Y. T. Lee and A. Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *Symposium on Foundations of Computer Science (FOCS)*, 2015.

[40] H. Lin, J. Mairal, and Z. Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[41] M. Minsky and S. Papert. *Perceptrons: An introduction to computational geometry*. MIT Press, 1988.

[42] R. D. Monteiro and B. F. Svaiter. Iteration-complexity of a Newton proximal extragradient method for monotone variational inequalities and inclusion problems. *SIAM Journal on Optimization*, 22(3):914–935, 2012.

[43] R. D. C. Monteiro and B. F. Svaiter. An accelerated hybrid proximal extragradient method for convex optimization and its implications to second-order methods. *SIAM Journal on Optimization*, 23(2):1092–1125, 2013.

[44] H. Namkoong and J. C. Duchi. Stochastic gradient methods for distributionally robust optimization with f-divergences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[45] A. Nemirovski. Prox-method with rate of convergence $o(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[46] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[47] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical programming*, 103:127–152, 2005.

[48] Y. Nesterov. Dual extrapolation and its applications to solving variational inequalities and related problems. *Mathematical Programming*, 109(2-3):319–344, 2007.

[49] Y. Nesterov. *Lectures on convex optimization*. Springer, 2018.

[50] R.-D. Reiss. *Approximate distributions of order statistics: with applications to nonparametric statistics*. Springer science & business media, 2012.

[51] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1-3):59–93, 1988.

[52] R. T. Rockafellar. *Convex analysis*. Princeton university press, 1997.

[53] S. Salzo and S. Villa. Inexact and accelerated proximal point algorithms. *Journal of Convex analysis*, 19(4): 1167–1192, 2012.

[54] S. Shalev-Shwartz and Y. Wexler. Minimizing the maximal loss: How and why? In *International Conference on Machine Learning (ICML)*, 2016.

[55] A. Sidford and K. Tian. Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow. In *Symposium on Foundations of Computer Science (FOCS)*, 2018.

[56] C. Song, S. J. Wright, and J. Diakonikolas. Variance reduction via primal-dual accelerated dual averaging for nonsmooth convex finite-sums. In *International Conference on Machine Learning*, 2021.

[57] C. Song, C. Y. Lin, S. Wright, and J. Diakonikolas. Coordinate linear variance reduction for generalized linear programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[58] J. J. Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1(1):79–80, 1857.

[59] K. K. Thekumparampil, P. Jain, P. Netrapalli, and S. Oh. Projection efficient subgradient method and optimal nonsmooth frank-wolfe method. In *Advances in Neural Information Processing Systems*, 2020.

[60] J. van den Brand. A deterministic linear program solver in current matrix multiplication time. pages 259–278. SIAM, 2020.

[61] J. van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In H. V. Le and V. King, editors, *4th Symposium on Simplicity in Algorithms (SOSA)*, pages 1–13. SIAM, 2021.

[62] J. van den Brand, Y. T. Lee, D. Nanongkai, R. Peng, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *Symposium on Foundations of Computer Science (FOCS)*, 2020.

[63] J. van den Brand, Y. T. Lee, A. Sidford, and Z. Song. Solving tall dense linear programs in nearly linear time. In K. Makarychev, Y. Makarychev, M. Tulsiani, G. Kamath, and J. Chuzhoy, editors, *Proceedings of the Fifty-Second Annual ACM Symposium on the Theory of Computing*, 2020.

[64] J. Van Den Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang. Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances. In *Symposium on Theory of Computing (STOC)*, 2021.

[65] M. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, 1991.

[66] M. Wang. Randomized linear programming solves the Markov decision problem in nearly linear (sometimes sublinear) time. *Mathematics of Operations Research*, 45(2):517–546, 2020.