A Practical Adaptive Subgame Perfect Gradient Method

Alan Luner* Benjamin Grimmer†

Abstract

We present a performant gradient method for smooth convex optimization, drawing inspiration from several recent advances in the field. Our algorithm, the Adaptive Subgame Perfect Gradient Method (ASPGM) is based on the notion of subgame perfection, attaining a dynamic strengthening of minimax optimality. At each iteration, ASPGM makes a momentum-type update, optimized dynamically based on a (limited) memory/bundle of past first-order information. ASPGM is linesearch-free, parameter-free, and adaptive due to its use of recently developed auto-conditioning, restarting, and preconditioning ideas. We show that ASPGM is competitive with state-of-the-art L-BFGS methods on a wide range of smooth convex problems. Unlike quasi-Newton methods, however, our core algorithm underlying ASPGM has strong, subgame perfect, non-asymptotic guarantees, providing certificates of solution quality, resulting in simple stopping criteria and restarting conditions.

1 Introduction

There is a common trade-off in optimization between strong convergence guarantees and practical speed. For first-order convex optimization, linesearch methods, in particular quasi-Newton methods, are well-established and popular algorithms. When measured by practical performance, L-BFGS [1] and its variants (e.g., [2, 3]) are state-of-the-art for many smooth convex problems, often converging superlinearly. However, despite this empirical performance, these methods have rather limited non-asymptotic guarantees. At the other extreme, accelerated gradient methods with comprehensive convex optimization theory are often outperformed by these linesearch methods. For example, the Optimized Gradient Method (OGM) [4] is known to provide the minimax optimal convergence guarantee for smooth convex problems but can fail to converge faster than its worst-case rate even on very simple instances[1].

In light of this dichotomy, there has been a recent renewed focus in smooth optimization on building performant methods with explicit non-asymptotic convergence rates. In this work, we bring together two separate recent directions of this movement: (i) adaptive methods that learn problem parameters and conditioning while running and (ii) subgame perfect methods that optimally (in a strong game-theoretic sense) leverage the bundle of first-order information collected at runtime.

Regarding (i), good performance of first-order methods often relies on an estimate of the problem's level of smoothness and its overall conditioning. Estimating smoothness is often addressed by backtracking to ensure an estimated constant satisfies the needed conditions for progress at each iteration [6, 7]. Recently, however, many works [8–18] have produced linesearch-free methods with strong supporting convergence theory, capable of estimating problem smoothness while avoiding the cost of backtracking. Problem conditioning can be addressed through careful selection of a

*Johns Hopkins University, Department of Applied Mathematics and Statistics, aluner1@jhu.edu
†Johns Hopkins University, Department of Applied Mathematics and Statistics, grimmer@jhu.edu

[1]See the example of minimizing $f(x) = \frac{1}{2}x^2$ in [5] where OGM attains the minimax worst-case rate.

preconditioner to improve the problem's overall condition number. BFGS-type methods could be viewed as providing such a live preconditioner through their maintenance of a second-order approximation. Recently, an alternative online preconditioner was developed by [19], extending the ideas of [20], in the framework of online learning to optimize performance.

Regarding (ii), the performance of first-order methods has long been known to improve if methods are allowed to maintain an additional memory. Bundle methods [21–24] have a long history of success, maintaining a bundle of many past first-order evaluations for use in its iterative update. The works [25, 26], among others, introduce several "gradient methods with memory", using a memory bundle to improve the progress of their estimate sequence and construct strategic, tight smooth lower bounds. Recently, the subgame perfect approach of [5] provided a theoretical framework describing the optimal dynamic usage of memory in a gradient method, strengthening the typical standard of minimax optimality. Their resulting Subgame Perfect Gradient Method (SPGM) solves a planning subproblem at each iteration to optimally leverage a bundle of past information—although to do so optimally requires exact knowledge of the given convex problem's smoothness constant.

This work provides a new performant method, which we call the Adaptive Subgame Perfect Gradient Method (ASPGM), combining the benefits of adaptive linesearch-free methods, preconditioning, and a subgame perfect usage of memory.

1.1 Related Literature

Optimized Gradient Method (OGM) First proposed numerically by Drori and Teboulle [27] and later formalized by Kim and Fessler [4], OGM possesses an optimal $O(1/N^2)$ convergence guarantee for smooth convex minimization. This is the same order of convergence as Nesterov's classic Fast Gradient Method [28] but improves asymptotically by a constant factor of two. Drori [29] showed that OGM's convergence guarantee is exactly minimax optimal among all first-order methods. That is, among all gradient-span methods $A \in \mathcal{A}$ producing a final iterate x_N, OGM attains the minimum value of

$$\min_{A\in\mathcal{A}} \max_{(f,x_0)\in\mathcal{F}_L} \frac{f(x_N)-f(x_\star)}{\frac{L}{2}\|x_0-x_\star\|^2} \tag{1.1}$$

where $\mathcal{F}_L$ denotes the family of all L-smooth convex problems with a minimizer $x_\star$ and initialization x_0. Note OGM is not adaptive or parameter-free as it relies on knowledge of the smoothness constant L. OGM and its theory are formally introduced in Section 2.1.1.

Subgame Perfect Gradient Method (SPGM) SPGM [5] improves OGM to attain a stronger notion of optimality than minimax optimality in the sense of (1.1) called subgame perfection. Consider a gradient method that sees iterates and first-order information of $\{(x_i, f_i, g_i)\}_{i=0}^N$ where $f_i = f(x_i)$ and $g_i = \nabla f(x_i)$. At any iteration $n \in \{1,\dots,N\}$ having seen a history of $\mathcal{H} = \{(x_i, f_i, g_i)\}_{i=0}^{n-1}$, denote the subclass of L-smooth convex problems agreeing with all observed first-order information by $\mathcal{F}_L^{\mathcal{H}}$ and the subclass of gradient-span methods required to reproduce $x_0,\dots,x_{n-1}$ by $\mathcal{A}^{\mathcal{H}}$. Then we say an algorithm is subgame perfect if for any iteration n and observed history $\mathcal{H}$, the method attains the minimum value of

$$\min_{A\in\mathcal{A}^{\mathcal{H}}} \max_{(f,x_0)\in\mathcal{F}_L^{\mathcal{H}}} \frac{f(x_N)-f(x_\star)}{\frac{L}{2}\|x_0-x_\star\|^2}. \tag{1.2}$$

SPGM, formally introduced in Section 2.1.2, attains this heightened standard. At each iteration, it solves a simple low-dimensional second-order cone program (SOCP) to compute an optimal next step. This update rule is optimal with respect to the stored first-order information in memory.

Numerically, [5] observed that when applied to a non-adversarial problem instance, SPGM greatly improves upon the performance of non-dynamic methods like OGM. However, like OGM, SPGM is not adaptive as it remains dependent on a smoothness parameter as input.

BFGS/L-BFGS Quasi-Newton methods [1–3, 30–33] attempt to find a local minimum for a function by applying Newton's method with an approximation of the inverse Hessian. At each iteration n, one performs a low-rank update to the inverse Hessian approximation, B_n, and a search direction is computed as $v = -B_n \nabla f(x_n)$. Once a descent direction is found, a linesearch method is applied to determine a stepsize and update the current iterate. Common options for linesearch methods include Armijo backtracking [34,35], Strong-Wolfe conditions [35,36], and a modified robust method introduced by Hager and Zhang in [37]. In L-BFGS [1], instead of storing the full matrix B_n, one stores only a limited memory of past data. This allows the matrix update and search direction calculation to be performed using only vector-vector products, preventing any large matrix storage. While L-BFGS performs extremely well in practice and often has superlinear convergence in a neighborhood of the solution, it only provides limited non-asymptotic guarantees on solution quality.

Preconditioning Preconditioning can be viewed as selecting an inner product via a preconditioner B to reduce the problem's difficulty, with respect to some measure. In smooth convex problems, a good preconditioner would reduce $\frac{L_B}{2}\|x_0 - x_\star\|_B^2$ as this controls convergence (where L_B is a global smoothness constant of f with respect to a modified inner product $\langle \cdot, \cdot \rangle_B$). In smooth strongly convex problems, one would like a preconditioner B minimizing the condition number L_B/μ_B. For $f \in C^2$, selecting B as the inverse of $\nabla^2 f(x_\star)$ drives this condition number to one in a neighborhood of $x_\star$. One can view quasi-Newton methods then as gradient descent with its preconditioner being adjusted at runtime to roughly approximate this inverse Hessian. The work [20] introduces a multidimensional backtracking technique to update a diagonal preconditioner at runtime. As a modern advancement of this idea, the Online Gradient Scaling Method of [19] uses the online learning regret bound toolbox to adjust the preconditioner at runtime. Asymptotically, this ensures an optimal preconditioned performance measure for gradient descent.

Adaptive Methods Adaptive methods offer strategies to address the issue of unknown function parameters. Backtracking is perhaps the most common and simple approach for smooth convex optimization when the smoothness parameter L is unknown. At each iteration, the algorithm evaluates the current estimate of L, and backtracks ($L \leftarrow 2L$) if certain smoothness inequalities are not satisfied. In [6], Beck and Teboulle extended the accelerated rates of Nesterov's Fast Gradient Method [28] to a backtracking linesearch setting. This approach was subsequently applied to broader settings, including [7] proposing a more general universal backtracking method for the broader class of functions with Hölder-continuous gradient. In [38], Park and Ryu propose the Optimal Backtracking Linesearch (OBL) method, which they show is minimax optimal against a certain generalized class of smooth convex gradient oracles.

While effective, a downside of backtracking is that it can be "wasteful" with respect to oracle evaluations. When a backtrack occurs, the new oracle information computed at the candidate iterate is effectively discarded. Linesearch-free methods manage to avoid this small inefficiency through various techniques. In bundle methods [21–24], this is handled with a "null step" in which the function value and gradient information are still saved in the bundle to inform future iterations. Starting with the introduction of adaptive gradient descent by [9], many recent works [8, 10, 11, 13, 16–18] adaptively learn the smoothness constant without backtracking, while also providing a non-asymptotic

convergence guarantee. The task of designing methods that estimate an unknown level of strong convexity is typically much harder than adapting to smoothness. Recently, [12] introduced a method for adaptively learning strong convexity constants while guaranteeing (suboptimal) linear global convergence. Our ASPGM will employ a similar estimate as a heuristic to benefit from strong convexity.

1.2 Our Contributions

This work provides a new performant gradient method bringing together the benefits from adaptive linesearch-free methods, preconditioning, and subgame perfect usage of memory. Our three main contributions are:

- **A Backtracking-free Subgame Perfect Gradient Method (BSPGM).** We first design a core parameter-free algorithm, BSPGM, that is subgame perfect against an appropriate class of first-order oracles. Although friendly to backtracking, BSPGM is linesearch-free; instead, it uses an efficient null-step, auto-conditioning approach, while providing dynamic, non-asymptotic convergence guarantees.
- **An Adaptive Subgame Perfect Gradient Method (ASPGM).** The strong non-asymptotic convergence guarantees and certificates resulting from BSPGM's subgame perfection provide a strong restarting condition for the algorithm, ensuring a contraction under strong convexity. Combining this restarting with preconditioning gives our full algorithm, ASPGM.
- **A Numerical Survey.** We conduct a numerical survey comparing BSPGM and ASPGM with a range of existing first-order methods from the literature across a range of smooth convex optimization problems. Our methods consistently outperform existing adaptive methods with strong theoretical support and perform comparably to state-of-the-art L-BFGS methods that lack such theory. These findings hold whether performance is measured in terms of oracle complexity or wall clock time.

Outline Section 2 introduces key concepts for building our algorithm and theory. Section 3 gradually presents the full algorithm and its various elements, while demonstrating convergence theory and the subgame perfection of the core procedure underlying ASPGM. Section 4 presents numerical experiments, comparing with a wide range of algorithms. Finally, Section 5 contains deferred proofs of our theoretical results.

2 Preliminaries

We consider unconstrained minimization problems of the form

$$\min_{x\in\mathbb{R}^d} f(x) \tag{2.1}$$

where f is convex and differentiable with gradient ∇f locally Lipschitz and with a minimizer $x_\star$. We highlight the fact that a global Lipschitz constant on the gradient L need not be known, or even exist, for our algorithm to be well-defined and progress. We denote I as the identity matrix, e_i as the ith standard basis vector, and $\mathbf{1}$ as the all ones vector, where the dimension of each should be clear from context.

Throughout, we will use $\langle \cdot, \cdot \rangle$ to denote the standard Euclidean inner product and $\| \cdot \|$ its corresponding norm. However, we will often use the modified inner products and norms defined by

$$\langle x, y \rangle_B = \langle x, B^{-1} y \rangle \qquad \|x\|_B = \sqrt{\langle x, x \rangle_B} \tag{2.2}$$

for a given symmetric positive definite matrix B. Accordingly, we will use $\nabla_B f(x)$ to denote the gradient of f at x, under the inner product $\langle \cdot, \cdot \rangle_B$. One can verify that $\nabla_B f(x) = B \nabla f(x)$. Our nonstandard choice of notation will become relevant when we apply preconditioning to our problem in Section 3.5. However, for much of our analysis, one can assume that $B = I$, and the standard Euclidean forms are recovered.

We recall two key inequalities for smooth convex functions. First, we have

$$f(y) \geq f(x) + \langle \nabla_B f(x), y - x \rangle_B \qquad \forall x, y \tag{2.3}$$

which we will refer to as the *convexity inequality*. Second, if f has locally L-Lipschitz gradient near x, often referred to as local L-smoothness, we have

$$f(y) \geq f(x) + \langle \nabla_B f(x), y - x \rangle_B + \frac{1}{2L} \|\nabla_B f(x) - \nabla_B f(y)\|_B^2 \qquad \forall y \text{ s.t. } \|x - y\|_B < R \tag{2.4}$$

which we will refer to as the *cocoercivity inequality*. We also define the function

$$\tilde{L}_B(x, y) = \frac{f(y) - f(x) - \langle \nabla_B f(x), y - x \rangle_B}{\frac{1}{2} \|\nabla_B f(x) - \nabla_B f(y)\|_B^2} \tag{2.5}$$

which returns the smallest L such that (2.4) holds from x to y. We use the convention that $0/0 = 0$ to resolve the edge case where both numerator and denominator are zero, as any positive L then satisfies (2.4).

We will often consider a discretized set of data for our function. Letting $\mathcal{I} = \{0, 1, \ldots, N, \star\}$ for some N, we consider $\{(x_i, f_i, g_i)\}_{i \in \mathcal{I}}$, where $f_i = f(x_i)$ and $g_i = \nabla_B f(x_i)$. We define the following values, corresponding to our inequalities (2.3) and (2.4) above: for $i, j \in \mathcal{I}$

$$W_{i,j} := f_i - f_j - \langle g_j, x_i - x_j \rangle_B, \tag{2.6}$$

$$Q_{i,j}(L) := f_i - f_j - \langle g_j, x_i - x_j \rangle_B - \frac{1}{2L} \|g_i - g_j\|_B^2. \tag{2.7}$$

Thus, for a locally smooth convex function and an appropriate L, it holds that $W_{i,j}$ and $Q_{i,j}(L)$ are nonnegative for all $i, j \in \mathcal{I}$.

We will also consider the case of smooth, strongly convex functions. Recall that any function f that is μ-strongly convex with respect to $\langle \cdot, \cdot \rangle_B$ has

$$f(y) \geq f(x) + \langle \nabla_B f(x), y - x \rangle_B + \frac{\mu}{2} \|x - y\|_B^2 \qquad \forall x, y. \tag{2.8}$$

Similar to the smooth case, we define the function $\tilde{\mu}_B(x, y) = \frac{f(y) - f(x) - \langle \nabla_B f(x), y - x \rangle_B}{\frac{1}{2} \|x - y\|_B^2}$ which returns the largest μ such that (2.8) holds from x to y.

2.1 Design of Optimized Methods

The derivation of many first-order methods can be written as an inductive argument in which one builds a nonnegative quantity at each iteration. A useful review of these arguments, also called potential-function proofs, can be found in [39, 40]. This inductive view of algorithm design and analysis will be central to our development herein. Below, we introduce three methods from the recent literature in this style: the Optimized Gradient Method (OGM) [4], the Subgame Perfect Gradient Method (SPGM) [5], and the Optimized Backtracking Linesearch method (OBL) [38]. Our method, ASPGM, will build upon these methods and, in particular, their inductions.

2.1.1 OGM - An Inductive View Assume for now that the smoothness constant L with respect to an inner product $\langle \cdot, \cdot \rangle_B$ is known. The Optimized Gradient Method [4] can be viewed as a method for maintaining an induction of the form

$$H_n := \tau_n \left(f_\star - f_n + \frac{1}{2L} \|g_n\|_B^2 \right) + \frac{L}{2} \|x_0 - x_\star\|_B^2 - \frac{L}{2} \|z_{n+1} - x_\star\|_B^2 \geq 0$$

for some $\tau_n > 0$ and iterate sequences x_n, z_{n+1}. The base case of this induction can be established for any given x_0 by setting $z_1 = x_0 - \frac{2}{L} g_0,\ \tau_0 = 2$ since we then have $H_0 = Q_{\star,0}(L) \geq 0$. Then OGM iterates by setting

$$(\tau_n, x_n, z_{n+1}) = \texttt{OGM\_Update}(\tau_{n-1}, x_{n-1}, g_{n-1}, z_n) \tag{2.9}$$

according to Algorithm 1, where the modified final step formula applies either when the given iteration budget is exhausted ($n = N$) or when some stopping/restarting criterion is met.

Algorithm 1: OGM Update Procedure

Function `OGM_Update`($\hat{\tau}$, x, g, z)**:**

$$\tau_n = \begin{cases} \hat{\tau} + \frac{1+\sqrt{1+4\hat{\tau}}}{2} & \text{if } \textbf{final step} \\ \hat{\tau} + 1 + \sqrt{1 + 2\hat{\tau}} & \text{otherwise} \end{cases}$$

$$x_n = \tfrac{\hat{\tau}}{\tau_n}(x - \tfrac{1}{L} g) + \tfrac{\tau_n - \hat{\tau}}{\tau_n} z$$

$$z_{n+1} = z - \tfrac{\tau_n - \hat{\tau}}{L} g$$

return (τ_n, x_n, z_{n+1})

These choices precisely ensure that the induction $H_n \geq 0$ continues, as one can verify that

$$H_n = H_{n-1} + \tau_{n-1} Q_{n-1,n}(L) + (\tau_n - \tau_{n-1}) Q_{\star,n}(L) \geq 0. \tag{2.10}$$

Equality can be shown above by expanding the polynomial $H_{n-1} + \tau_{n-1} Q_{n-1,n}(L) + (\tau_n - \tau_{n-1}) Q_{\star,n}(L)$ and collecting terms, and nonnegativity follows since each term in the sum is nonnegative. A slight modification to this induction and algorithm is needed at the final iteration N, as one instead defines

$$H_N := \tau_N (f_\star - f_N) + \frac{L}{2} \|x_0 - x_\star\|_B^2 - \frac{L}{2} \|z_{N+1} - x_\star\|_B^2 \geq 0. \tag{2.11}$$

Rearranging (2.11), one arrives at a convergence rate of $f_N - f_\star \leq \frac{\frac{L}{2}\|x_0 - x_\star\|_B^2}{\tau_N}$. This convergence guarantee is exactly minimax optimal among all gradient-span methods in the sense of (1.1), proven by the matching lower bound of Drori [29].

2.1.2 SPGM - A Dynamically Optimized Induction Continuing to assume that the smoothness constant L with respect to an inner product $\langle \cdot, \cdot \rangle_B$ is known, the Subgame Perfect Gradient Method [5], and its limited memory variant, can be viewed as a dynamic improvement of OGM to use a stronger inductive hypothesis at each step. Rather than building $H_n \geq 0$ using the previous hypothesis $H_{n-1} \geq 0$, one could imagine an enhanced induction, constructing and using the strongest hypothesis of the form

$$H' = \tau' \left(f_\star - f_m + \frac{1}{2L} \|g_m\|_B^2 \right) + \frac{L}{2} \|x_0 - x_\star\|_B^2 - \frac{L}{2} \|z' - x_\star\|_B^2 \tag{2.12}$$

for some τ', z' and index m. Given a limited memory of size k, SPGM constructs such an aggregate nonnegative hypothesis by setting

$$H' = \sum_{i=n-k}^{n-1} \rho_i H_i + \sum_{i=n-k}^{n-1} \gamma_i Q_{\star,i}(L) + \epsilon \tag{2.13}$$

for nonnegative $\rho, \gamma \in \mathbb{R}^k$ and $\epsilon \geq 0$.

Let $Z \in \mathbb{R}^{d\times k}$ be the matrix with columns given by $z_{i+1} - x_0$ and indexed by $[n-k, n-1]$. Similarly, let $G \in \mathbb{R}^{d\times k}$ be the matrix with columns given by $\frac{g_i}{L}$ and indexed by $[n-k, n-1]$. Additionally, let $\tau = (\tau_{n-k}, \ldots, \tau_{n-1})$ and let $v \in \mathbb{R}^k$ be defined componentwise by

$$v_i := f_i - \frac{1}{2L}\|g_i\|_B^2.$$

Combining (2.12) and (2.13), one finds that any feasible hypothesis must set $z' = x_0 + Z\rho - G\gamma$. Then, to maximize the value of τ' while keeping $H' \geq 0$, one should set $m \in \operatorname{argmin}_{i\in[n-k,n-1]}\{f_i - \frac{1}{2L}\|g_i\|_B^2\}$. Using this index m, we further define vectors $q, r \in \mathbb{R}^k$ with components

$$\begin{aligned} q_i &:= \tau_i(f_i - \frac{1}{2L}\|g_i\|_B^2) + \frac{L}{2}\|z_{i+1}\|_B^2 - \frac{L}{2}\|x_0\|_B^2 - v_m\tau_i - \langle L(z_{i+1} - x_0), x_0\rangle_B \\ r_i &:= f_i - \langle g_i, x_i - x_0\rangle_B - \frac{1}{2L}\|g_i\|_B^2 - v_m. \end{aligned}$$

Under this selection, rearranging (2.13) to solve for ϵ, one obtains that

$$\epsilon(\rho, \gamma) = \sum_{i=n-k}^{n-1} \rho_i q_i + \sum_{i=n-k}^{n-1} \gamma_i r_i - \frac{L}{2}\|Z\rho - G\gamma\|_B^2.$$

Then the strongest such hypothesis (i.e., the one maximizing τ') is attained by solving the second-order cone program of

$$\tau' = \begin{cases} \max & \sum_{i=n-k}^{n-1} \rho_i\tau_i + \sum_{i=n-k}^{n-1} \gamma_i \\ \text{s.t.} & \epsilon(\rho, \gamma) \geq 0 \\ & \rho, \gamma \geq 0. \end{cases} \tag{2.14}$$

Note that setting $\rho = (0, \ldots, 0, 1)$, $\gamma = 0$, $\epsilon = 0$, is feasible and recovers $H' = H_{n-1}$ when $m = n-1$, i.e., the hypothesis used by OGM. Thus $\tau' \geq \tau_{n-1}$, ensuring SPGM's intermediate aggregate hypothesis is always at least as strong as the minimax optimal induction of OGM.

SPGM then iterates by repeatedly constructing this optimized aggregate hypothesis and then applying the OGM step (Algorithm 1) from it to construct $H_n \geq 0$. This is formalized in Algorithm 2. Grimmer, Shu, and Wang [5] established that SPGM's overall induction is at least as strong as OGM's and therefore SPGM is also exactly minimax optimal. Further when a full memory is used, they constructed a dynamic adversarial strategy for revealing first-order information, proving a matching lower bound on the possible performance of any first-order method. Since this dynamic lower bound exactly matched SPGM's induction for every n, the authors conclude its dynamic guarantees are the strongest possible on the final objective gap given any history of first-order observations. That is, SPGM is subgame perfect in the sense of (1.2).

Algorithm 2: SPGM

Input : Convex function f, memory size $k \in \mathbb{N}$, $x_0 \in \mathbb{R}^d$, $L > 0$, $B \succ 0$
Set $\tau_0 = 2,\ z_1 = x_0 - \frac{2}{L}\nabla_B f(x_0)$
for $n = 1, \dots, N$ **do**
  Compute $f_{n-1} = f(x_{n-1})$, $g_{n-1} = \nabla_B f(x_{n-1})$
  Update memory $\mathcal{H} = \{(x_i, f_i, g_i, z_{i+1}, \tau_i)\}_{i=n-k}^{n-1}$ and construct Z and G
  Set $m \in \operatorname{argmin}_{i \in [n-k, n-1]}\{f_i - \frac{1}{2L}\|g_i\|_B^2\}$
  if (2.14) *is unbounded* **then**
    Set $x_n = x_m - \frac{1}{L}g_m$ and **break**
  end
  Compute ρ, γ by solving (2.14) and set $z' = x_0 + Z\rho - G\gamma$ and
   $\tau' = \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i$
  Set $(\tau_n, x_n, z_{n+1}) = \texttt{OGM\_Update}(\tau', x_m, g_m, z')$ according to Algorithm 1
end
Output : x_n

2.1.3 OBL - A Backtracking-Friendly Induction Both of the above methods relied on knowledge of the global smoothness constant L, which can be both unrealistic in practice and lead to overly conservative algorithms. One classic approach to avoiding this is the use of backtracking, where at each iteration an estimate L_n is utilized to compute a step. Then, whichever inequalities are needed by the proof are checked; if they hold the step is accepted, otherwise L_n is increased, typically to $2L_n$, and the process repeats.

To apply this reasoning to OGM, one would need to verify at each step the nonnegativity of $Q_{n-1,n}(L_n)$ and $Q_{\star,n}(L_n)$. This first quantity, $Q_{n-1,n}(L_n)$, is easy to compute at runtime. However, the second quantity $Q_{\star,n}(L_n)$ requires knowledge of $x_\star$ to be evaluated and hence prevents backtracking. The work of Park and Ryu [38] showed this issue can be circumvented by replacing the cocoercivity inequality $Q_{\star,n}(L_n)$ in OGM's induction with the weaker convexity inequality $W_{\star,n}$, which is independent of L_n. Carrying out this slightly weaker induction, one arrives at the backtracking-friendly OBL algorithm of [38].

Each iteration maintains an inductive hypothesis of the form

$$U_n := \tau_n \left(f_\star - f_n + \frac{1}{2L_n}\|g_n\|_B^2 \right) + \frac{L_n}{2}\|x_0 - x_\star\|_B^2 - \frac{L_n}{2}\|z_{n+1} - x_\star\|_B^2 + \Delta_n \geq 0 \tag{2.15}$$

where the additional nonnegative constant Δ_n above accumulates error terms due to changing L_n. As a base case, one can initialize this induction with $\tau_0 = 1, z_1 = x_0 - \frac{1}{L_0}g_0$ to obtain $U_0 = W_{\star,0} \geq 0$. Then, given $U_{n-1} \geq 0$ for some $\tau_{n-1}, x_{n-1}, z_n, \Delta_{n-1}$, OBL generates a test iterate by setting

$$(\tau_n, x_n, z_{n+1}, \Delta_n) = \texttt{OBL\_Update}(\tau_{n-1}, x_{n-1}, g_{n-1}, z_n, L_n, \frac{L_n}{L_{n-1}}\Delta_{n-1}, \delta_n) \tag{2.16}$$

according to Algorithm 3, where $\delta_n = L_n \tau_{n-1} \left(\frac{1}{L_{n-1}^2} - \frac{1}{L_n^2} \right) \frac{1}{2}\|g_{n-1}\|_B^2$. If $Q_{n-1,n}(L_n) \geq 0$, then the iterate is accepted. Otherwise, one backtracks, setting $L_n \leftarrow 2L_n$, discarding the computed (x_n, f_n, g_n), and repeating Algorithm 3 until an iterate is accepted.

Following this, one has that (see [38, Theorem 6])

$$U_n = \frac{L_n}{L_{n-1}}U_{n-1} + \tau_{n-1}Q_{n-1,n}(L_n) + (\tau_n - \tau_{n-1})W_{\star,n} + \tau_{n-1}\left(\frac{L_n}{L_{n-1}} - 1\right)(f_{n-1} - f_\star) \geq 0$$

Algorithm 3: OBL Update Procedure

Function `OBL_Update`($\hat{\tau}$, x, g, z, L, $\hat{\Delta}$, $\hat{\delta}$)**:**

$$\tau_n = \begin{cases} \hat{\tau} + \sqrt{\hat{\tau}} & \text{if } \textbf{final step} \\ \hat{\tau} + \frac{1+\sqrt{1+8\hat{\tau}}}{2} & \text{otherwise} \end{cases}$$

$x_n = \frac{\hat{\tau}}{\tau_n}(x - \frac{1}{L}g) + \frac{\tau_n - \hat{\tau}}{\tau_n} z$

$z_{n+1} = z - \frac{\tau_n - \hat{\tau}}{L} g$

$\Delta_n = \hat{\Delta} + \hat{\delta}$

return $(\tau_n, x_n, z_{n+1}, \Delta_n)$

where the equality is verified by expanding polynomial terms, and the inequality follows as each term is nonnegative. Hence, after N iterations (and again modifying our final induction U_N) this backtracking provides a guarantee of $f_N - f_\star \leq \frac{\frac{L_N}{2}\|x_0 - x_\star\|_B^2 + \Delta_N}{\tau_N}$. If a valid smoothness constant $L_0 = L$ is given initially, one will have $\Delta_N = 0$. In this setting, [38] proved this is the best possible convergence rate any first-order method can attain against a first-order oracle whose responses satisfy the inequalities $\{Q_{i-1,i}\}_{i=1}^n \cup \{W_{\star,i}\}_{i=0}^n$.

3 Algorithm

In this section, we gradually build our algorithm ASPGM. In Section 3.1, we build a dynamic optimization approach improving OBL's induction, mirroring that of SPGM, but with L unknown. In Section 3.2, we demonstrate how this can be extended to a linesearch-free method we call the Backtracking-free Subgame Perfect Gradient Method (BSPGM). Section 3.3 presents convergence guarantees for BSPGM, including its subgame perfection. Finally, in Section 3.4 and Section 3.5, we develop adaptive restarting and preconditioning and present our full algorithm ASPGM, using BSPGM as a subroutine.

3.1 Dynamically Optimized Induction

We consider the case of minimizing a locally smooth function, with unknown smoothness constant L. Additionally, we suppose that we have access to a memory of size k of past algorithm information: $\mathcal{H} = \{(x_i, f_i, g_i, z_{i+1}, \tau_i, L_i, \Delta_i)\}_{i=n-k}^{n-1}$.

We begin by introducing some notation. Given a vector $\tau = (\tau_{n-k}, \ldots, \tau_{n-1})$ and $L_n > 0$, define the vector $v \in \mathbb{R}^k$ by its components:

$$v_i := f_i - \frac{1}{2L_n}\|g_i\|_B^2 \tag{3.1}$$

for $i \in [n-k, n-1]$. Next, letting $J = \{i \in [n-k, n-1] \mid \tau_i > 0\}$, define two special indices m and s according to

$$m \in \operatorname*{argmin}_{i \in J} v_i, \tag{3.2}$$

$$s = \max\{i \in J\}. \tag{3.3}$$

Colloquially, m represents the "best" iterate so far, with respect to our induction U_i. As we will see, x_m will be important in determining our next iterate x_n. On the other hand, s corresponds to the most recent iterate for which our induction U_i was nontrivial ($\tau_i \neq 0$). This index will play a

key role in our induction and error term collection. Note that for each $i \notin J$, we will have $U_i \equiv 0$. Further define δ_n as

$$\delta_n := L_n \tau_s \left(\frac{1}{L_s^2} - \frac{1}{L_n^2} \right) \frac{1}{2} \|g_s\|_B^2 \tag{3.4}$$

and the vectors $a, b \in \mathbb{R}^k$ with components

$$a_i := \tau_i (f_i - \frac{1}{2L_i} \|g_i\|_B^2) + \frac{L_i}{2} \|z_{i+1}\|_B^2 - \frac{L_i}{2} \|x_0\|_B^2 - v_m \tau_i - \langle L_i (z_{i+1} - x_0), x_0 \rangle_B \tag{3.5}$$

$$b_i := f_i - \langle g_i, x_i - x_0 \rangle_B - v_m \tag{3.6}$$

for $i \in [n-k, n-1]$. Note that v_i is dependent on the current L_n, while a_i and b_i also depend on the past value L_i.

We recall the inductive quantity U_i (2.15) from OBL and now apply the dynamic induction strategy introduced in Section 2.1.3. We define

$$U' = \tau' \left(f_\star - f_m + \frac{1}{2L_n} \|g_m\|_B^2 \right) + \frac{L_n}{2} \|x_0 - x_\star\|_B^2 - \frac{L_n}{2} \|z' - x_\star\|_B^2 + \Delta' + \delta_n \tag{3.7}$$

for some τ', z', and $\Delta' \geq 0$. We construct our enhanced hypothesis U' by setting

$$U' = \sum_{i=n-k}^{n-1} \rho_i U_i + \sum_{i=n-k}^{n-1} \gamma_i W_{\star,i} + \epsilon \tag{3.8}$$

for nonnegative $\rho, \gamma \in \mathbb{R}^k$ and $\epsilon \geq 0$. Proceeding as in SPGM, we seek to maximize τ' while keeping the error term Δ' bounded. Here we briefly summarize how this dynamic optimization is achieved, but we include a more detailed derivation in Section 5.2.

Let $Z \in \mathbb{R}^{d \times k}$ be the matrix with columns now given by $\frac{L_i}{L_n}(z_{i+1} - x_0)$ and indexed by $[n-k, n-1]$. Similarly, let $G \in \mathbb{R}^{d \times k}$ be the matrix with columns now given by $\frac{g_i}{L_n}$ and indexed by $[n-k, n-1]$. Once again, by rearranging (3.8) and applying a bound on Δ', we find that the following constraints must be satisfied:

$$z' = x_0 + Z\rho - G\gamma, \tag{3.9}$$

$$\tau' = \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i, \tag{3.10}$$

$$\epsilon = \epsilon(\rho, \gamma) = \sum_{i=n-k}^{n-1} \rho_i a_i + \sum_{i=n-k}^{n-1} \gamma_i b_i + \delta_n - \frac{L_n}{2} \|Z\rho - G\gamma\|_B^2. \tag{3.11}$$

In order to guarantee the nonnegativity of U' we must have $\epsilon(\rho, \gamma) \geq 0$. Thus, our dynamic optimization becomes the following subproblem with a single second-order cone constraint:

$$\tau' = \left\{ \begin{array}{ll} \max_{\rho,\gamma} & \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i \\ \text{s.t.} & \epsilon(\rho, \gamma) \geq 0 \\ & \rho,\, \gamma \geq 0. \end{array} \right. \tag{3.12}$$

Observe that maximizing τ' in (3.12) once again amounts to solving a low-dimensional second order cone program (SOCP). We emphasize that this SOCP is independent of the problem dimension d. Specifically, the subproblem (3.12) has dimension $2k$, where k is the chosen memory size. By appropriately tracking and storing $O(k^2)$ past vector-vector products (i.e., $Z^T Z, G^T G, G^T Z$), we

eliminate any matrix-vector products with full-dimension matrices. Additionally, the problem has a linear objective with a single quadratic constraint, further reducing the complexity of the subproblem. Thus, for high-dimensional ($k \ll d$) problems with moderate costs for gradient and/or function evaluations, the per iteration computational cost of (3.12) becomes negligible.

3.2 A Backtracking-free Subgame Perfect Gradient Method (BSPGM)

This dynamic induction already gives way to a reasonable backtracking method. Given τ' and z' after solving (3.12), generate x_n (along with $\tau_n, z_{n+1}, \Delta_n$) according to (2.16). We then check if the current estimate of L_n is valid, i.e., if $Q_{m,n}(L_n) \geq 0$. If so, accept x_n and continue. If not, reject x_n, increment $L_n \leftarrow 2L_n$, and repeat the update rule. Note that when x_n is accepted, this indicates that our hypothesis $U_n \geq 0$ is valid, so the induction can continue.

While this approach represents an effective algorithm, it is inefficient in terms of how it uses new oracle information. When a proposed next iterate x_n is rejected, the new gradient g_n and function value f_n are effectively discarded without providing additional information other than the fact that the current estimate L_n is too small. For high-dimensional problems, first-order oracles can have a large computational cost; we therefore want to learn as much information as possible from each oracle call.

In the case that $Q_{m,n}(L_n) < 0$, the computed first-order information still satisfies convexity, ensuring $W_{\star,n} = f_\star - f_n - \langle g_n, x_\star - x_n \rangle \geq 0$. As an alternate strategy to backtracking in this case, we can accept the computed iterate x_n, adding its function value and gradient to our history, but modify our induction to avoid usage of $Q_{m,n}(L_n)$. In particular, we set $\tau_n = 0, z_{n+1} = x_0, \Delta_n = 0$ to ensure that $U_n \geq 0$ holds vacuously, as

$$\begin{aligned} U_n &= \tau_n \left(f_\star - f_n + \frac{1}{2L_n}\|g_n\|_B^2 \right) + \frac{L_n}{2}\|x_0 - x_\star\|_B^2 - \frac{L_n}{2}\|z_{n+1} - x_\star\|_B^2 + \Delta_n \\ &= 0 + \frac{L_n}{2}\|x_0 - x_\star\|_B^2 - \frac{L_n}{2}\|x_0 - x_\star\|_B^2 + 0 \\ &= 0. \end{aligned}$$

We therefore continue our induction of $U_n \geq 0$ with no dependence on the negative term $Q_{m,n}(L_n)$.

This strategy is similar to the typical bundle method approach of allowing both "serious steps" where the main iterate sequence updates and "null steps" where only the bundle of first-order information is improved. Algorithm 4 formalizes our use of this conditional approach. We emphasize that while there are remnants of a backtracking mechanism in how we iteratively increase L_n, this method is notionally distinct as we are not discarding any computed x_n and associated first-order oracle values when L_n increases. Rather, (x_n, f_n, g_n) are incorporated into our bundle of historical information via the nonnegative quantity $W_{\star,n}$. We also note the use of $\tilde{L}_B(x_m, x_n)$ to increment L_n matches the auto-conditioning approach used by [8, 9, 11] and others.

Remark 1. To guarantee Algorithm 4 is well-defined, it is necessary to ensure that $J \neq \emptyset$. This is the role of our "strategic update" to our memory $\mathcal{H}$. In the case that $\tau_{n-k-1} > 0$ but $\tau_i = 0$ for $i \in [n-k, n-1]$, we discard from our memory the data indexed by $n-k$ rather than $n-k-1$ (resulting in a minor abuse of notation). In this rare scenario, this step prevents the last nonzero τ_i from being discarded and enables our continued induction with U_i without losing progress.

Remark 2. Recall that for $i \notin J$, we have $\tau_i = 0$ and $\Delta_i = 0$. Consequently, the corresponding coefficients ρ_i in (3.12) have no effect on the subproblem's solution. We can therefore fix $\rho_i = 0$ for all $i \notin J$ to reduce the number of variables in (3.12) for a minor computational improvement.

Algorithm 4: BSPGM: A Backtracking-free Subgame Perfect Gradient Method

Input : Convex, locally smooth function f, subgame memory size $k \in \mathbb{N}$, $x_0 \in \mathbb{R}^d$, $L_0 > 0$, $B \succ 0$

Set $\tau_0 = 1, z_1 = x_0 - \frac{1}{L_0}\nabla_B f(x_0), L_1 = L_0$

for $n = 1, 2, \ldots$ **do**

  Compute $f_{n-1} = f(x_{n-1})$, $g_{n-1} = \nabla_B f(x_{n-1})$

  Strategically update memory $\mathcal{H} = \{(x_i, f_i, g_i, \tau_i, z_{i+1}, L_i, \Delta_i)\}_{i=n-k}^{n-1}$, construct Z and G

  Select indices s, m according to (3.2) and (3.3)

  if (3.12) *is unbounded* **then**

    Set $x_n = x_m - \frac{1}{L_n} g_m$ and **break**

  end

  Compute ρ, γ by solving (3.12) with δ_n set according to (3.4), and set $z' = x_0 + Z\rho - G\gamma$, $\tau' = \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i$, and $\Delta' = \sum_{i=n-k}^{n-1} \rho_i \Delta_i$

  Set $(\tau_n, x_n, z_{n+1}, \Delta_n) = \texttt{OBL\_Update}(\tau', x_m, g_m, z', L_n, \Delta', \delta_n)$ according to Algorithm 3

  if $\tilde{L}_B(x_m, x_n) > L_n$ **then**

    $L_{n+1} = \max\{\tilde{L}_B(x_m, x_n), 2L_n\}, \quad \tau_n = 0, \quad z_{n+1} = x_0, \quad \Delta_n = 0$

  else if *Stopping condition* **then**

    break

  end

end

Output : x_n, $\mathcal{H}$

In the following lemma, we argue that our subproblem (3.12) is feasible and well-defined. We defer the proof to Section 5.1 and Section 5.3.

Lemma 3.1. *Consider any sequences $x_n, \tau_n, z_{n+1}, \Delta_n$ generated by BSPGM. Then*

(i) The problem (3.12) *is feasible.*

(ii) If (3.12) *is bounded, then it is strictly feasible and strong duality holds. Moreover, the optimal value τ' is attained and satisfies $\tau' \geq \tau_s$.*

(iii) If (3.12) *is unbounded, then $x_m - \frac{1}{L_n} g_m \in \operatorname{argmin} f$.*

3.3 Convergence Guarantees and Subgame Perfection of BSPGM

To maintain our inductive hypothesis $U_n \geq 0$, we simply have to verify that

$$U_n = U' + \tau' Q_{m,n}(L_n) + (\tau_n - \tau') W_{\star,n}. \tag{3.13}$$

The result then follows by the nonnegativity of our terms, provided that $Q_{m,n}(L_n) \geq 0$. This immediately provides the following convergence guarantee at each "serious" iteration where $Q_{m,n}(L_n) \geq 0$. We include a detailed proof in Section 5.2.

Theorem 3.2. *Consider any sequences $x_n, \tau_n, z_{n+1}, \Delta_n$ generated by Algorithm 4. Then each serious step $n \in \{0, \ldots, N\}$ (that is, with $\tau_n \neq 0$) has*

$$f(x_n) - \frac{1}{2L_n}\|g_n\|_B^2 - f(x_\star) \leq \frac{L_n \|x_0 - x_\star\|_B^2 + \Delta_n}{2\tau_n} \qquad \text{if } n < N \tag{3.14}$$

$$f(x_N) - f(x_\star) \leq \frac{L_N \|x_0 - x_\star\|_B^2 + \Delta_N}{2\tau_N} \qquad \text{if } n = N. \tag{3.15}$$

At each "null" step, where $\tau_i = 0$, the value of L_n at least doubles. If the function f is L-smooth with respect to $\langle \cdot, \cdot \rangle_B$ on a convex region containing all of the iterates, then there can be at most $\log_2(L/L_0)$ null steps. A simple calculation guarantees that τ_n grows at least quadratically in the number of "serious" steps[2]. As a result, when applied to L-smooth convex functions, the above guarantees decrease at the minimax optimal big-O rate of $O(L\|x_0 - x_\star\|_B^2/N^2)$.

Remark 3. We argue informally that our error term Δ_N should be small relative to τ_N. At each step of Algorithm 4, we have $\delta_n > 0$ if and only if $L_n > L_s$ (i.e., we took a null step). Therefore, Δ_N is a function of the number of null steps and should be proportional to $\log_2(\frac{L_N}{L_0})$. Next, since τ_N grows with order $O(N^2)$ and our gradient norms $\|g_i\|_B^2$ are generally decreasing, we expect Δ_N to be negligible.

Through induction, for a given n, we can express Δ_n as a recursive product of past ρ values and gradient norms, but it is difficult to meaningfully bound the value of Δ_n. However, a critical benefit to our dynamically optimized update steps is that Δ_n is dynamic as well. While the error term in OBL is monotonically nondecreasing with each iteration, our Δ_n is not necessarily monotone. In practice, it is not uncommon for (3.12) to place heavy weight in the γ variables and minimal weight on the ρ variables at a particular iteration. This effectively resets the accumulated error $\Delta' \approx 0$ and sets $\Delta_n \approx \delta_n$. Thus, at points in our algorithm, we can obtain nearly tight guarantees when the value of Δ_N "resets" (See an example in Figure 3).

Minimax Optimality and Subgame Perfection without Null Steps. Stronger theoretical guarantees can be stated for BSPGM if one restricts to first-order oracles where no null steps occur. Formally, denote a first-order oracle by a mapping $O\colon \{(x_i, f_i, g_i)\}_{i=0}^{n-1} \times \{x_n\} \mapsto (f_n, g_n)$ sending each possible nth iterate to a response and $O\colon \{(x_i, f_i, g_i)\}_{i=0}^{N} \mapsto (x_\star, f_\star, g_\star)$ with $g_\star = 0$. Let $\mathcal{O}_L$ denote the set of all oracles satisfying $Q_{i,j}(L) \geq 0$ holds for every $i < j$ and $W_{\star,i} \geq 0$ for every i. The use of such oracles follows the ideas of [38], where they considered oracles only required to satisfy a subset of the typical inequalities in smooth convex optimization.

In such settings, the dynamic optimization of BSPGM's induction ensures that its τ_n sequence grows at each iteration at least as fast as the sequence generated by the static OBL update. Then one can generate a worst-case final convergence rate for BSPGM at any iteration n by considering the final τ_N produced by $N - n$ subsequent iterations of OBL (instead of BSPGM), continuing the induction. Denote the hypothetical sequence of τ produced by this procedure by $\tau_{n,n} = \tau_n$ and $\tau_{n,i}$ for $i > n$ defined recursively via (2.16). Hence, we have the following sequence of dynamic upper bounds on BSPGM's performance. The proof is deferred to Section 5.4.

Theorem 3.3. *Consider any oracle $O \in \mathcal{O}_L$ and let $\{(x_i, f_i, g_i, \tau_i, z_{i+1})\}_{i=0}^{N}$ denote the results of running BSPGM with $L_0 = L$. Then one has the sequence of dynamic guarantees*

$$\frac{f_N - f_\star}{\frac{L}{2}\|x_0 - x_\star\|_B^2} \leq \frac{1}{\tau_{N,N}} \leq \frac{1}{\tau_{N-1,N}} \leq \cdots \leq \frac{1}{\tau_{1,N}} \leq \frac{1}{\tau_{0,N}} = \frac{2}{N(N+1) + \sqrt{2N(N+1)}}.$$

The final bound above is minimax optimal among all gradient-span methods for minimization against such an oracle and is also attained by the simpler OBL method, see [38, Theorem 4]. That is, letting $\mathcal{A}$ denote the set of gradient span methods producing a final iterate after N iterations,

$$\min_{A \in \mathcal{A}} \max_{O \in \mathcal{O}_L} \frac{f_N - f_\star}{\frac{L}{2}\|x_0 - x_\star\|_B^2} = \frac{2}{N(N+1) + \sqrt{2N(N+1)}}.$$

[2]This can be seen since $\hat{\tau} \geq \frac{1}{2}n^2$ implies $\hat{\tau} + \frac{1+\sqrt{1+8\hat{\tau}}}{2} \geq \frac{1}{2}(n+1)^2$

Beyond this minimax optimality, BSPGM is subgame perfect against such oracles. Given a history $\mathcal{H} = \{(x_i, f_i, g_i, \tau_i, z_{i+1})\}_{i=0}^{n-1}$ produced by BSPGM, let $\mathcal{O}_L^{\mathcal{H}}$ denote the subclass of oracles in $\mathcal{O}_L$ that agree with the history produced by the first $n-1$ iterations and $\mathcal{A}^{\mathcal{H}}$ denote the subset of algorithms producing the same first $n-1$ iterations. For any observed intermediate history, BSPGM provides a minimax optimal algorithm for the remaining subgame. The proof is deferred to Appendix A.2.

Theorem 3.4. *For any history $\mathcal{H} = \{(x_i, f_i, g_i, \tau_i, z_{i+1})\}_{i=0}^{n-1}$ observed by BSPGM with $L = L_0$ and memory $k \geq n$ up to iteration $n \leq N$, we have*

$$\min_{A \in \mathcal{A}^{\mathcal{H}}} \max_{O \in \mathcal{O}_L^{\mathcal{H}}} \frac{f_N - f_\star}{\frac{L}{2}\|x_0 - x_\star\|_B^2} = \frac{1}{\tau_{n,N}}.$$

3.4 Adaptive Restart

Restarting is a well-known effective strategy for attaining linear rates for smooth, strongly convex problems [41, 42]. We apply this approach to improve the performance of our ultimate ASPGM, adaptively restarting an inner loop of the previously introduced BSPGM in Algorithm 4. Provided a strong convexity constant μ with respect to $\langle \cdot, \cdot \rangle_B$ is known, the dynamic induction underlying BSPGM provides a provably-good dynamic restarting condition, stated below. After developing this restart condition, we briefly introduce a heuristic for estimating μ with respect to B used in our final, parameter-free method, similar in spirit to the approach of [12].

Going forward, we will refer to each inner loop of Algorithm 4 between restarts as an epoch, and we will use $x_i^{(\ell)}$, $g_i^{(\ell)}$, etc. to refer to the values of each variable at the ith iteration of the ℓth epoch. Suppose for the sake of our theoretical development that in epoch ℓ, a strong convexity constant $\mu^{(\ell)} > 0$ for f with respect to $\langle \cdot, \cdot \rangle_{B^{(\ell)}}$ is known. Our restart condition to be checked at each serious step is

$$\tau_n^{(\ell)} \geq \frac{2L_n^{(\ell)}}{\mu^{(\ell)}} + \frac{L_n^{(\ell)} \Delta_n^{(\ell)}}{f(x_0^{(\ell)}) - f(x_n^{(\ell)})}, \qquad f(x_0^{(\ell)}) - f(x_n^{(\ell)}) > 0. \tag{3.16}$$

Once (3.16) is satisfied, the algorithm restarts after the next "serious step", taken using the final step version of the OBL update (Algorithm 3). To restart our method, we initialize a new instance of Algorithm 4 at the current iterate $x_n^{(\ell)}$, with $x_0^{(\ell+1)} \leftarrow x_n^{(\ell)}$ and some $L_0^{(\ell)}$ and $B^{(\ell)}$.

Whenever (3.16) is satisfied by the inner loop executing Algorithm 4, the objective gap will have been reduced by at least a factor of two. Hence, by restarting the algorithm at this point (technically, at the next serious step), only a logarithmic number of restarts will be required to reach any given target accuracy. The following theorem formally states this contraction guarantee and a constant bound on the number of iterations needed to attain (3.16), with proof deferred to Section 5.5.

Theorem 3.5. *Suppose that f is μ-strongly convex with respect to $\langle \cdot, \cdot \rangle_B$. Further suppose that at some iteration n of Algorithm 4, the restart condition* (3.16) *is satisfied. If $x_{n+1}^{(\ell)}$ is a serious step and $\Delta_{n+1}^{(\ell)} \leq \Delta_n^{(\ell)}$, then*

$$f(x_{n+1}^{(\ell)}) - f(x_\star) \leq \frac{1}{2}\left(f(x_0^{(\ell)}) - f(x_\star)\right). \tag{3.17}$$

Further, the restart condition (3.16) *must be attained whenever*

$$n \geq \sqrt{\frac{4L_n^{(\ell)}}{\mu} + \frac{2L_n^{(\ell)}\Delta_n^{(\ell)}}{f(x_0^{(\ell)}) - f(x_\star)}} + \log_2\left(\frac{L_n^{(\ell)}}{L_0{}^{(\ell)}}\right).$$

Remark 4. In the case that $\Delta_{n+1}^{(\ell)} > \Delta_n^{(\ell)}$ or the next serious step does not occur until $x_{n+r}^{(\ell)}$ for some $r > 1$, we cannot guarantee the exact contraction in (3.17), since $\tau_{n+r}^{(\ell)}$ may no longer satisfy (3.16) due to increases in $\Delta_{n+r}^{(\ell)}$ and $L_{n+r}^{(\ell)}$. However, in such cases, we expect a contraction approximately matching (3.17), which was sufficient in our numerical evaluations.

In practice, accurately estimating $\mu^{(\ell)}$ (at each epoch) can be difficult and lead algorithms to be overly conservative. Numerically, we found strong practical performance by constructing a live estimate of μ similar to the approach of [12]. We initially set $\mu_0^{(\ell)} = \infty$ and update at each iteration according to

$$\mu_n^{(\ell)} = \min\{\mu_{n-1}^{(\ell)},\ \tilde{\mu}_{B^{(\ell)}}(x_m^{(\ell)}, x_n^{(\ell)})\}. \tag{3.18}$$

Then our ASPGM algorithm triggers a restart whenever (3.16) holds with $\mu^{(\ell)} = \mu_n^{(\ell)}$.

Remark 5. As additional heuristics, our implementation requires that each epoch perform a minimum number of iterations before restarting and forces a restart after a maximum number of iterations. This can prevent premature or delayed restarts due to a poor initial estimate of $\mu^{(\ell)}$ or $L^{(\ell)}$, respectively. In practice, we found that with a minimum iteration requirement of 20 and maximum iteration setting of 100, performance was slightly improved.

3.5 Preconditioning

In quasi-Newton methods, one uses an inverse Hessian approximation as a preconditioner to a gradient descent step. In L-BFGS, given a memory size t of points x_i and corresponding gradients g_i, the inverse Hessian approximation and Hessian approximation are formed recursively as follows. Letting $B_1 = B_1^{-1} = I$, the BFGS update sets

$$B_{i+1} = (I - \frac{s_i y_i^T}{y_i^T s_i})B_i(I - \frac{s_i y_i^T}{y_i^T s_i}) + \frac{s_i s_i^T}{y_i^T s_i}, \qquad B_{i+1}^{-1} = B_i^{-1} - \frac{B_i^{-1} s_i s_i^T B_i^{-1}}{s_i^T B_i^{-1} s_i} + \frac{y_i y_i^T}{y_i^T s_i} \tag{3.19}$$

where $s_i = x_{i+n-t} - x_{i+n-t-1}$, $y_i = g_{i+n-t} - g_{i+n-t-1}$ for $i = 1, \ldots, t$. We then define $B = B_{t+1}$ and $B^{-1} = B_{t+1}^{-1}$.

In ASPGM, we use the inverse Hessian approximation, derived from L-BFGS, to precondition our problem. At each restart, we "construct" our preconditioner B by saving the last t differences $s_i = x_{i+n-t} - x_{i+n-t-1}$ and $y_i = g_{i+n-t} - g_{i+n-t-1}$ into a separate memory storage, $\{(s_i, y_i)\}_{i=1}^t$. We then define B according to the induction (3.19). With this formula, one can efficiently compute Bv and $B^{-1}v$ for any vector v, using only vector-vector products; we therefore never have to store B or B^{-1} explicitly. For completeness, we include Algorithm 6 and Algorithm 7 in Appendix A to show the standard efficient computation of Bv and $B^{-1}v$ (both products will be needed since $\langle x, y\rangle_B = \langle x, B^{-1}y\rangle$ and $\nabla_B f(x) = B\nabla f(x)$).

Finally, we can present our complete algorithm.

Remark 6. In total, ASPGM requires the storage of $3k + 2t$ additional vectors of size d (beyond the current x_n and g_n), where k is the memory size used in the BSPGM subalgorithm, and t is the memory size used for preconditioning. All other storage requirements are negligible at high dimensions. In comparison, the storage requirements of SPGM and L-BFGS are $2k$ and $2t$, respectively.

4 Computational Benchmarks

In this section, we demonstrate the performance of BSPGM and ASPGM across a wide range of problem classes. In both oracle complexity and wall-clock performance, our methods using modest

Algorithm 5: ASPGM

Input : Convex, locally smooth function f, subgame memory size $k \in \mathbb{N}$, preconditioning memory size $t \in \mathbb{N}$, $x_0 \in \mathbb{R}^d$

Set $B^{(1)} = I$, $x_0^{(1)} = x_0$

for $\ell = 1, \dots$ **do**

$y = x_0^{(\ell)} + 10^{-4} \cdot \mathcal{N}(0, 1)$

$L_0^{(\ell)} = \tilde{L}_B^{(\ell)}(x_0^{(\ell)}, y)$

$x_n^{(\ell)}, \mathcal{H} =$ Output of Algorithm 4 given $k, x_0^{(\ell)}, L_0^{(\ell)}, B^{(\ell)}$

Construct $B^{(\ell+1)}$ based on the last t steps in $\mathcal{H}$ via (3.19)

$x_0^{(\ell+1)} = x_n^{(\ell)}$

end

memory sizes reach state-of-the-art performance in comparison with adaptive first-order methods and quasi-Newton methods. In Section 4.1, we discuss our implemented versions of the algorithm, along with a large sample of competing algorithms from the literature. In Section 4.2, we test ASPGM on a set of synthetic, randomly generated problems. Then in Section 4.3 and Section 4.4, we apply ASPGM to problems derived from real data sets. Finally, in Section 4.5, we examine the performance of ASPGM on several difficult, poorly conditioned problems.

ASPGM is implemented in Julia [43] and uses MOSEK [44] via JuMP [45] to solve the subproblem (3.12) at each iteration. Our implementation along with code used to run our experiments is available at

`github.com/alanluner/ASPGM`.

Experiments were run locally on an Intel i7 processor with 64GB of memory.

4.1 Algorithm Varieties and Benchmarks

We investigate the performance of three versions of our subgame perfect methods. First, we consider the full method, ASPGM (Algorithm 5), including adaptive restarting and preconditioning both with fixed memory sizes of $k = t = 1$ (`ASPGM-1-1`) and $k = t = 5$ (`ASPGM-5-5`). Additionally, we consider the core method BSPGM (Algorithm 4), with memory $k = 7$ (`BSPGM-7`), which does not implement any adaptive restart or preconditioning.

At the end of our section, we compare in Figure 6 the performance of ASPGM with different memory sizes. As expected, we see diminishing returns as the memory size increases. This is consistent with many memory-based methods including SPGM and L-BFGS. These results justify our use of `ASPGM-5-5` as our default method. Even at small memory, we see performance is generally maintained, motivating our inclusion of `ASPGM-1-1` in our results as a very low-memory alternative.

Benchmark Algorithms We compare the performance of our base BSPGM method and our full ASPGM method with a wide range of competing algorithms. We focus primarily on adaptive methods and thus exclude methods requiring smoothness estimates, like OGM and SPGM, from our survey. Many of these methods require an unspecified initial estimate of L_0; we initialize $L_0 = \tilde{L}_I(x_0, x_0 + 10^{-4} \cdot \xi)$ for $\xi \in \mathcal{N}(0, 1)^d$. L-BFGS is implemented in Julia using the Optim [46] package. All other algorithms are implemented directly in Julia.

- `OBL`: Optimized Backtracking Linesearch [38].

- `UFGM`: Universal Fast Gradient Method [7].
- `AdaNAG`: AdaNAG-G_{12} from [8, Corollary 7].
- `ACFGM`: Auto-Conditioned Fast Gradient Method [11]. We follow the implementation of the authors and set parameters $\eta_1 = 5/(2L_0)$, $\beta = 1 - \sqrt{6}/3$, and $\alpha = 0.1$.
- `AdGD`: Adaptive Gradient Descent [10]. We use the improved adaptive gradient descent method from [10, Algorithm 2].
- `NAGF`: NAG-free algorithm from [12, Algorithm 2].
- `OSGMR`: Online Scaled Gradient Method with ratio surrogate [19], with AdaGrad as the preconditioner update algorithm. For the stepsize, we conducted a parameter sweep across a subset of problems and selected $\eta = 0.1$ as the value with the best overall performance. Improved performance would follow from tuning this parameter per problem instance. We follow the authors' approach for handling an unknown lower bound (See [19, Section B.1]).
- `LBFGS-BL`: L-BFGS with backtracking linesearch. We use memory size $t = 10$.
- `LBFGS-HZ`: L-BFGS with the robust linesearch introduced by Hager and Zhang [37]. Note that this is the default option when using L-BFGS in the `Optim` Julia package. We use memory size $t = 10$.

Among these, the first seven provide comparisons with adaptive methods that possess strong theoretical support. Our `ASPGM-1-1` is the most comparable to these in computation/storage per iteration. The latter two L-BFGS methods are state-of-the-art in performance (although lacking non-asymptotic theory) and have similar computational/storage costs to `ASPGM-5-5` and `BSPGM-7`. Hence they set the baseline we seek to match.

4.2 Synthetic Smooth Convex Problems

We generate random problem instances for several problem classes discussed below, all parameterized by $A \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^m$, and $x_0 \in \mathbb{R}^d$. For each problem instance we set $m = 4d$, $x_0 = 0$, generate b element-wise from $\mathcal{N}(0,1)$, and generate c element-wise from $\mathcal{U}\{0,1\}$. Since the conditioning of each problem class is determined by the condition number of A^TA, we construct A so that A^TA has approximate condition numbers of $\kappa = 10^2$ and $\kappa = 10^4$. We also distribute the singular values of A in two different ways:

- Uniform distribution: $\sigma_i \sim \mathcal{U}(1, \sqrt{\kappa})$ for $i = 1, \ldots, d$
- Bimodal distribution: $\sigma_1, \ldots, \sigma_{9d/10} \sim \mathcal{U}(1, 1.1)$ and $\sigma_{9d/10+1}, \ldots, \sigma_d \in \mathcal{U}(0.9\sqrt{\kappa}, \sqrt{\kappa})$.

We generate 4 instances for each condition number and each distribution above; using dimensions $d = 1000, 2000, 4000, 8000$ and 6 problem classes below, this yields 384 synthetic problem instances.

In Figure 1, we present the overall performance of ASPGM, BSPGM, and competing methods across all of our problem classes. We also include in Appendix A.3 these same performance results stratified by problem class. We see ASPGM with both memory size 5 and a minimal size of 1 is highly performant on our synthetic problem set, outperforming other adaptive methods, as well as `LBFGS-HZ`. This advantage is maintained when considering wall clock time. We also note the strong performance of BSPGM for attaining low-accuracy solutions, especially in terms of wall-clock time. This is particularly impressive since BSPGM also provides an explicit non-asymptotic convergence guarantee and corresponding stopping criteria via L_n/τ_n.

We introduce each problem class below.

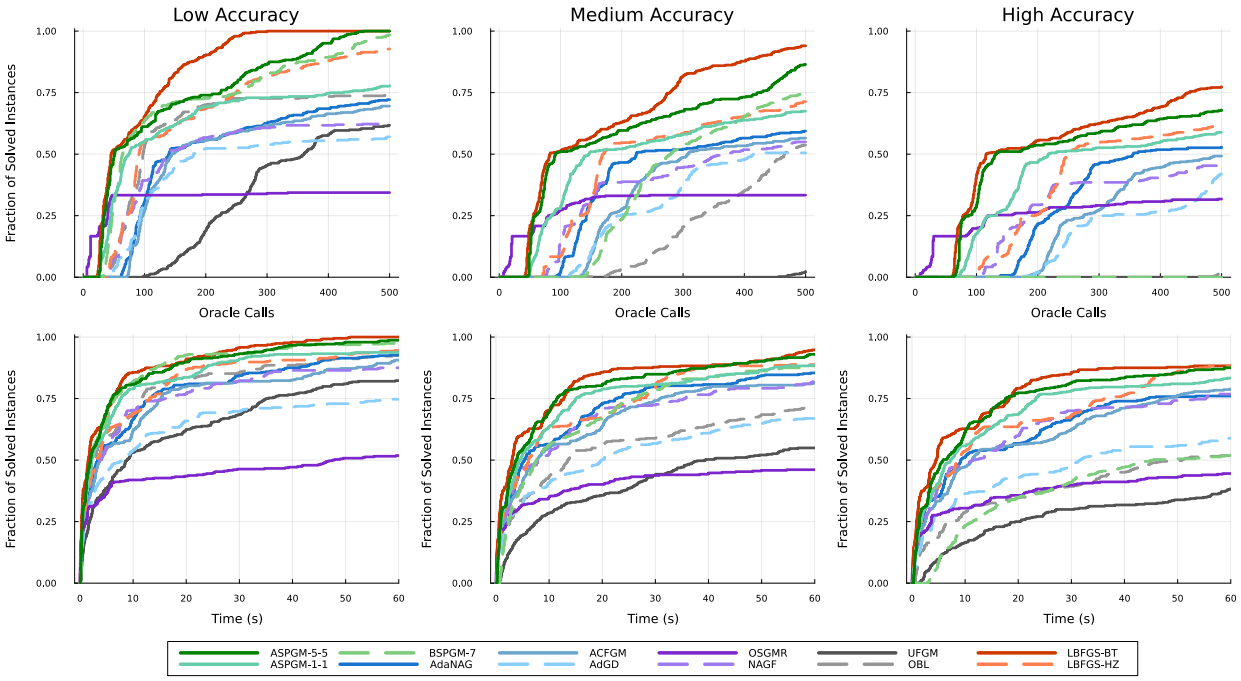

Figure 1: Performance results over synthetic problem instances, in terms of oracle calls and wall clock time. The performance is measured by $(f(x_n) - f_\star)/(f(x_0) - f_\star)$ and the target accuracies from left to right are $10^{-4}, 10^{-7}, 10^{-10}$.

Regression We consider least squares regression and logistic regression with L2 regularization. This amounts to the following objective functions:

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2, \tag{4.1}$$

$$f(x) = \sum_{i=1}^{m} \log(1 + \exp(c_i \cdot a_i^T x)) + \frac{1}{2m}\|x\|_2^2 \tag{4.2}$$

where a_i denotes the ith row of A.

Smoothings of Functions Next, we consider two methods of smoothing approximations of the feasibility problem seeking some x with $Ax - b \leq 0$. We define the following objectives,

$$f(x) = \log\left(1 + \sum_{i=1}^{m} \exp(a_i^T x - b_i)\right), \tag{4.3}$$

$$f(x) = \sum_{i=1}^{m} (a_i^T x - b_i)_+^2 \tag{4.4}$$

where v_+ denotes the positive part of v i.e., $v_+ = \max\{v, 0\}$ and a_i denotes the ith row of A.

Locally but not Globally Smooth Functions Lastly, we consider convex functions that are locally, but not globally smooth. We define the objective functions

$$f(x) = \frac{1}{4}\|Ax - b\|_4^4, \tag{4.5}$$

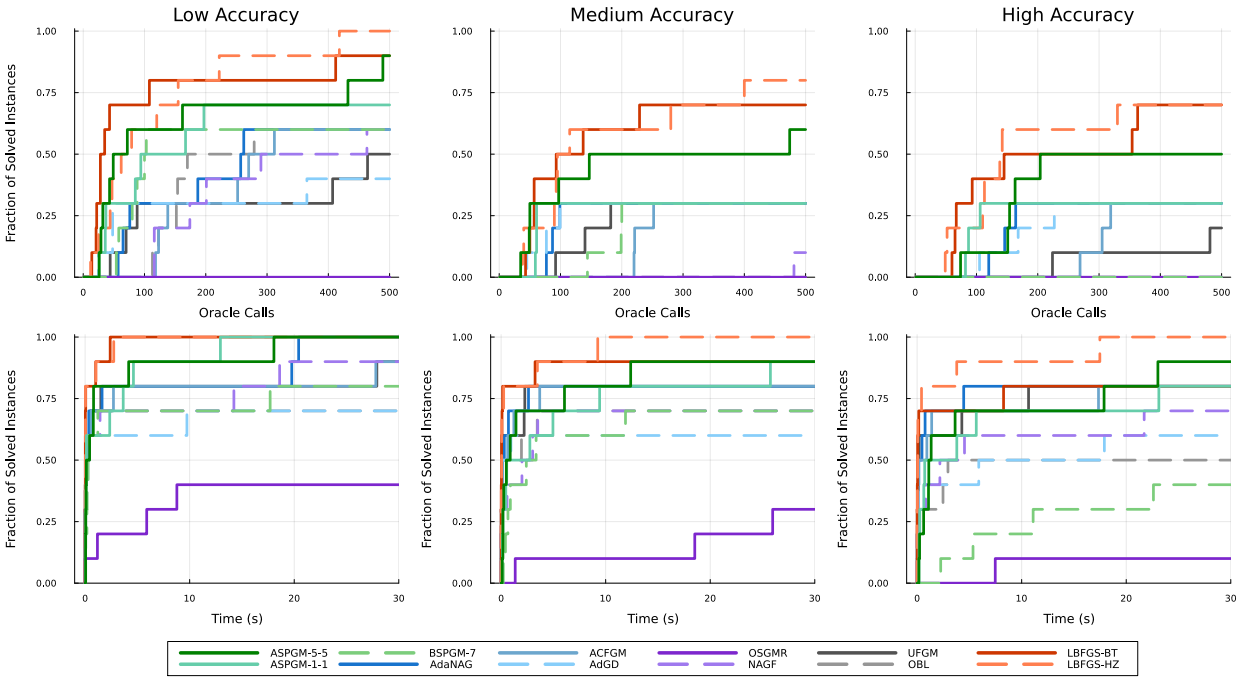

Figure 2: Performance results for least squares regression (4.1) and logistic regression (4.2) problems from LIBSVM [47]. The performance is measured by $(f(x_n) - f_\star)/(f(x_0) - f_\star)$ and the target accuracies from left to right are $10^{-4}, 10^{-7}, 10^{-10}$.

$$f(x) = \frac{1}{2}\|Ax\|_2^2 + b^T x + \frac{1}{6m}\|x\|_2^3. \tag{4.6}$$

4.3 Real-Data Regression Problems

We also present performance results for problem instances on real data. Using the LIBSVM [47] library, we apply least squares regression (4.1) to the data sets `bodyfat`, `eunite2001`, `pyrim`, `triazines`, and `yearPredictionMSD` and logistic regression with L2 regularization (4.2) to the data sets `colon-cancer`, `duke-breast-cancer`, `gisette`, `leukemia`, and `madelon`. Results are shown in Figure 2. We see similar trends as in our synthetic problems, but see that ASPGM no longer outperforms `LBFGS-HZ`. We also notice a stronger difference for ASPGM between oracle performance and real time performance. This can likely be partially attributed to these problems having lower dimension, thereby causing our subproblem to have a larger proportional impact on computation time. In Figure 3, we show detailed results for the `colon-cancer` logistic regression problem. Here we show the modified performance measure $(f(x_n) - f_\star)/(\frac{1}{2}\|x_0 - x_\star\|^2)$, so we can directly compare the performance with the guarantees (ignoring error terms) of `BSPGM-7`, `OBL`, and `AdaNAG` in the center plot. We also show the value of the error term Δ_n for `BSPGM-7` and `OBL`. In particular, this illustrates our dynamic theory allowing Δ_n to briefly fluctuate and then returns to nearly zero for `BSPGM-7`, unlike `OBL`.

4.4 Real-Data Feasibility Problems

As another set of problems derived from real data, we consider the search for a feasible point of an LP. Given an LP with the constraints $Ax \leq b$, one can frame this objective as minimizing

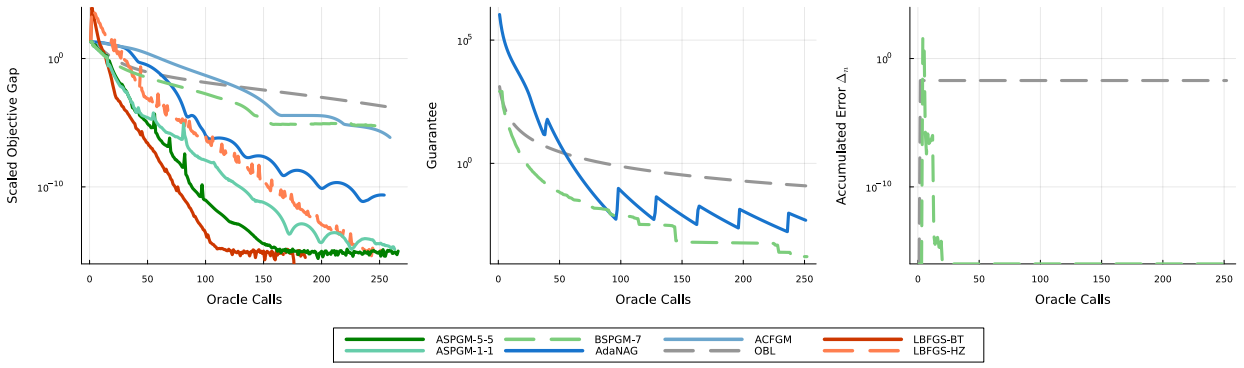

Figure 3: Performance data from logistic regression (4.2) run on the `colon-cancer` dataset. On the left, we show the alternate performance measure $(f(x_n) - f_\star)/(\frac{1}{2}\|x_0 - x_\star\|^2)$. In the middle plot, we compare the guarantees of `BSPGM-7`, `OBL`, and `AdaNAG` (ignoring error terms). For `BSPGM-7` and `OBL`, this corresponds to L_n/τ_n. The right plot shows the accumulated error Δ_n for `BSPGM-7` and `OBL`.

$\max_i(Ax-b)_i$. If we apply either of our smoothing approaches (4.3), (4.4) from Section 4.2, we obtain a smooth problem suitable for our set of algorithms. We apply this method to the following problems from the "LPfeas Benchmark" dataset [48]: `brazil3`, `chromaticindex1024-7`, `ex10`, `graph40-40`, `qap15`, `savsched1`, `scpm1`, `set-cover-model`, and `supportcase10`. These include high-dimensional ($d \approx 10^5$) sparse datasets. We apply both smoothing approaches to each data file, obtaining 18 total problem instances. We see similar trends in oracle-call performance but diminished real-time performance. We attribute this to the sparsity of data matrices, making gradient calculations more efficient and increasing the relative cost of our subproblem calculations.

4.5 Poorly Conditioned Problems

Lastly, to compare performances in highly degenerate settings, we consider several known poorly conditioned problems from the literature. Each problem is a quadratic of the form $f(x) = \frac{1}{2}x^TAx + b^Tx$, with specially selected $A \in \mathbb{R}^{d\times d}$, $b \in \mathbb{R}^d$, and starting point $x_0 \in \mathbb{R}^d$. First, we consider the classic hard instance [49] with $x_0 = 0$, $b = (-\frac{1}{2}, 0, \ldots, 0)$ and A as the tridiagonal matrix with diagonal $(1, \ldots, 1)$ and super- and sub-diagonals $(-\frac{1}{2}, \ldots, -\frac{1}{2})$. Next, we consider from [25], the case where $b = 0$, A is diagonal entries with entries $a_{ii} = \sin^2(\frac{\pi i}{2d})$, and $x_0 = (1/a_{11}, \ldots, 1/a_{dd})$. Finally, we consider the instance with $x_0 = 0$, $b = (1, \ldots, 1)$ and A as the diagonal matrix with diagonal $(1, 2, \ldots, d)$. We refer to these problem instances as `Problem A`, `Problem B`, and `Problem C`, respectively. In Figure 5, we show our methods remain competitive in these degenerate settings with $d = 1000$.

5 Theory

This section contains proofs of our earlier theoretical results. In Section 5.1 and Section 5.3, we prove Lemma 3.1. In Section 5.2, we prove the convergence rate of our core algorithm BSPGM, stated in Theorem 3.2; this includes a more detailed derivation of the subproblem (3.12). Section 5.4 and Section 5.5 present proofs of Theorem 3.3 and Theorem 3.5. To simplify notation, we will omit the subscript B throughout the section on our inner products and norms, but the analysis is unchanged.

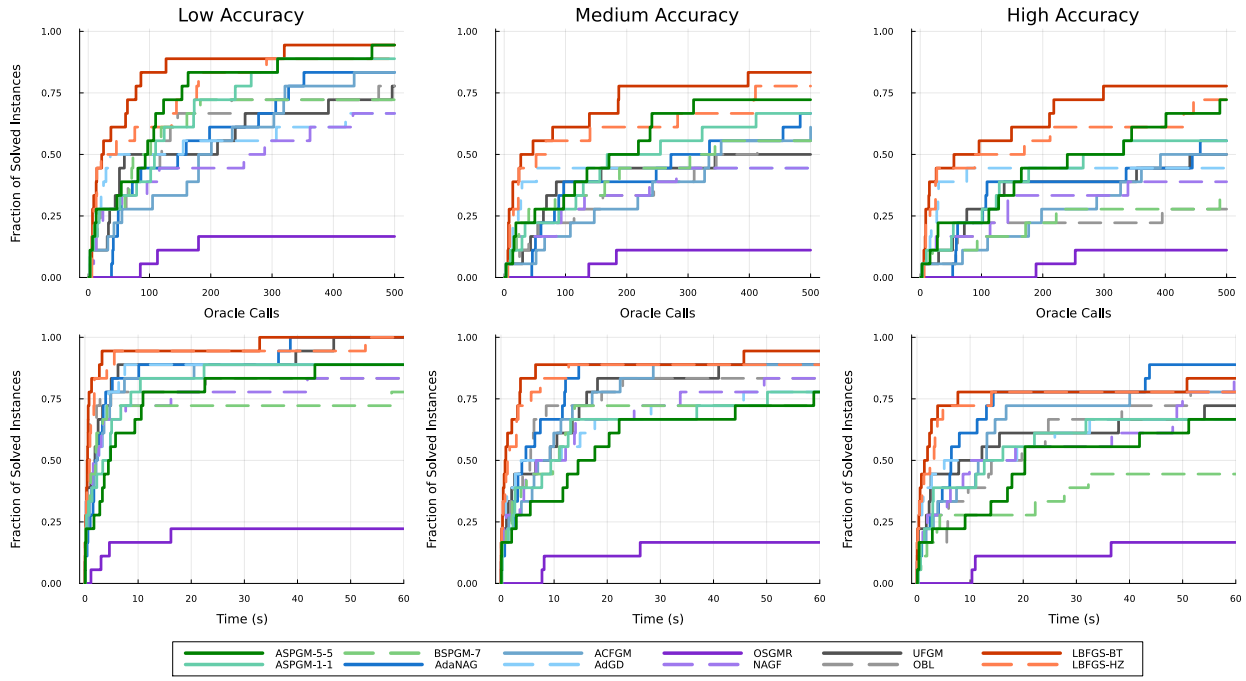

Figure 4: Performance results for feasibility problems over LP instances from [48]. The performance is measured by $(f(x_n)-f_\star)/(f(x_0)-f_\star)$ and the target accuracies from left to right are $10^{-4}, 10^{-7}, 10^{-10}$.

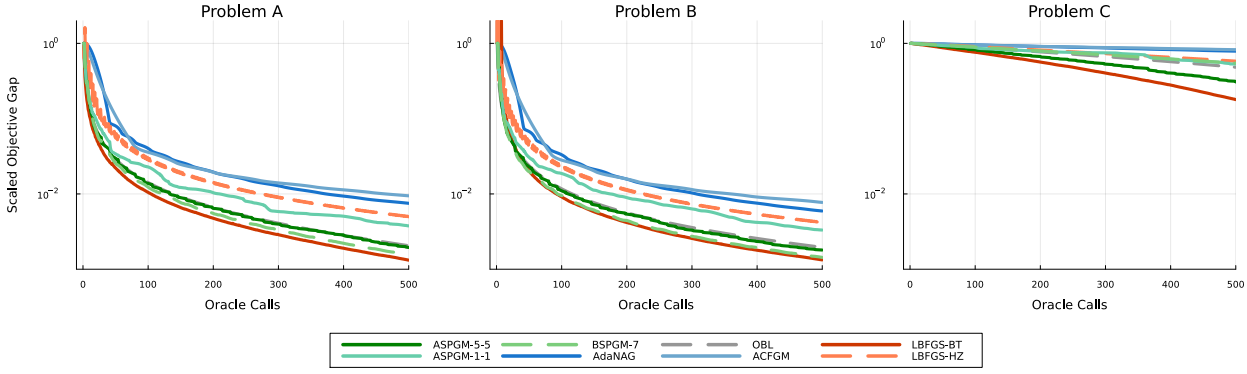

Figure 5: Performance on poorly conditioned problem instances with $d = 1000$, as measured by $(f(x_n) - f_\star)/(f(x_0) - f_\star)$.

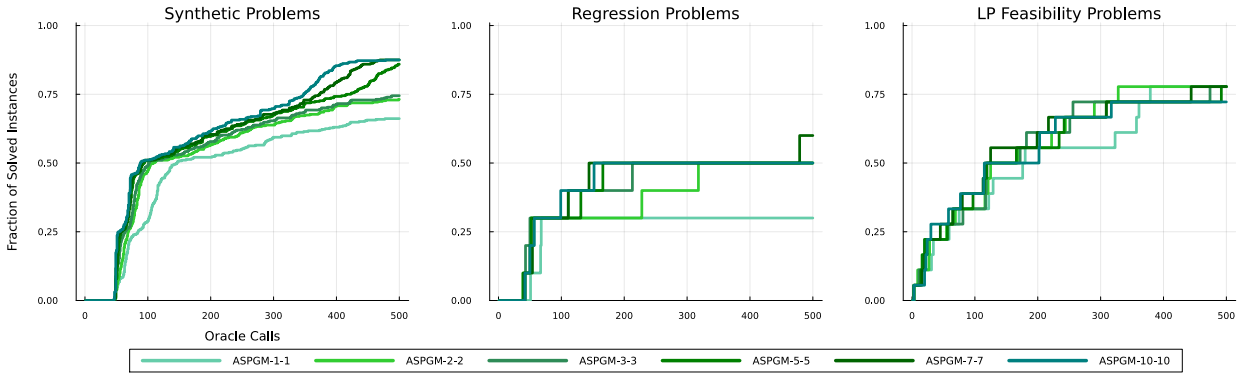

Figure 6: Performance comparison of ASPGM with different memory sizes. From left to right, we show results with performance measured by $(f(x_n) - f_\star)/(f(x_0) - f_\star)$ at accuracy 10^{-7}, for synthetic problems, regression problems, and LP feasibility problems.

5.1 Proof of Lemma 3.1 (i) and (ii)

Here we prove parts (i) and (ii) separately from part (iii). This is because the true sequence of logic is that (i) and (ii) imply Theorem 3.2, which in turn implies the result of (iii).

We first claim that if (3.12) is bounded, then we must have $z_{s+1} \neq x_0$ or $L_s \neq L_n$. Proving by contrapositive, if $z_{s+1} = x_0$ and $L_s = L_n$, then the solution $\rho = ce_s$, $\gamma = 0$ is feasible for all $c > 0$, as

$$\epsilon(ce_s, 0) = ca_s + \delta_n - \frac{L_n}{2}\|z_{s+1} - x_0\|^2 = c\tau_s(f_s - \frac{1}{2L_s}\|g_s\|^2 - v_m) + \delta_n = c\tau_s(v_s - v_m) + \delta_n \geq 0$$

and thus (3.12) is unbounded since $\tau' \geq c\rho_s\tau_s$.

By construction of Algorithm 4, we know that $L_n \geq L_i$ for all $i \in [n-k, n-1]$. Let $\kappa = c\frac{L_n}{L_s}$ for some $c \in [0, 1]$, and take $\rho = \kappa e_s$ and $\gamma = 0$. We clearly satisfy $\rho, \gamma \geq 0$. Then, verifying our main constraint, we have

$$\begin{aligned}
\epsilon(\rho, \gamma) &= \sum_{i=n-k}^{n-1} \rho_i a_i + \sum_{i=n-k}^{n-1} \gamma_i b_i + \delta_n - \frac{L_n}{2}\|Z\rho - G\gamma\|^2 \\
&= \kappa a_s + L_n\tau_s\left(\frac{1}{L_s^2} - \frac{1}{L_n^2}\right)\frac{1}{2}\|g_s\|^2 - \frac{L_n}{2}\|c(z_{s+1} - x_0)\|^2 \\
&= \kappa\tau_s f_s - \kappa\tau_s\frac{1}{2L_s}\|g_s\|^2 - \frac{\kappa L_s}{2}\|x_0\|^2 + \frac{\kappa L_s}{2}\|z_{s+1}\|^2 - \kappa v_m\tau_s - L_n c\langle z_{s+1} - x_0, x_0\rangle \\
&\qquad + L_n\tau_s\left(\frac{1}{L_s^2} - \frac{1}{L_n^2}\right)\frac{1}{2}\|g_s\|^2 - \frac{L_n}{2}\|c(z_{s+1} - x_0)\|^2 \\
&\geq \kappa\tau_s f_s - \kappa\tau_s\frac{1}{2L_n}\|g_s\|^2 - \kappa\tau_s v_m + \tau_s\left(\frac{1}{L_s} - \frac{1}{L_n}\right)\frac{1}{2}\|g_s\|^2 + \frac{L_n(c - c^2)}{2}\|z_{s+1} - x_0\|^2 \\
&\geq \kappa\tau_s(v_s - v_m) + \tau_s\left(\frac{1}{L_s} - \frac{1}{L_n}\right)\frac{1}{2}\|g_s\|^2 + \frac{L_n(c - c^2)}{2}\|z_{s+1} - x_0\|^2 \\
&\geq 0
\end{aligned}$$

where the final inequality applies the definition of m in (3.2). This shows feasibility of (ρ, γ).

Suppose (3.12) is bounded. Then by our earlier argument either $z_{s+1} \neq x_0$ or $L_s \neq L_n$, and the expression above is strictly positive for any $c \in (0, 1)$ (unless $g_s = 0$, in which case $x_s \in \operatorname{argmin} f$).

This shows there exists a strictly feasible point for (3.12), and therefore strong duality holds. Computing the objective function with our feasible solution and $c = 1$, we obtain

$$\tau' = \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i = \frac{L_n}{L_s} \tau_s \geq \tau_s.$$

Then we have $\tau' \geq \tau_s$. Moreover, our feasible region must be compact, since all terms in our objective function are nonnegative with positive coefficients. Therefore, the optimal value τ' is attained.

5.2 Proof of Theorem 3.2

We recall our relevant definitions:

$$\begin{aligned}
U_i &= \tau_i \left(f_\star - f_i + \frac{1}{2L_i} \|g_i\|^2 \right) + \frac{L_i}{2} \|x_0 - x_\star\|^2 - \frac{L_i}{2} \|z_{i+1} - x_\star\|^2 + \Delta_i, \\
W_{i,j} &= f_i - f_j - \langle g_j, x_i - x_j \rangle, \\
Q_{i,j}(L) &= f_i - f_j - \langle g_j, x_i - x_j \rangle - \frac{1}{2L} \|g_i - g_j\|^2, \\
v_i &= f_i - \frac{1}{2L_n} \|g_i\|^2
\end{aligned}$$

and we introduce the notation

$$\begin{aligned}
h_i &:= \tau_i (f_i - \frac{1}{2L_i} \|g_i\|^2) + \frac{L_i}{2} \|z_{i+1}\|^2 - \frac{L_i}{2} \|x_0\|^2, \\
w_i &:= f_i - \langle g_i, x_i \rangle.
\end{aligned}$$

Our algorithm data $\mathcal{H}$ is stored as before, and we have the columns of Z given by $\frac{L_i}{L_n}(z_{i+1} - x_0)$ and the columns of G given by $\frac{g_i}{L_n}$. Additionally, we will assign to z_{n+1} the form $z_{n+1} = z' - \frac{\alpha}{L_n} g_n$ for some $z' \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$.

We first claim that $U_0 \geq 0$. Since $z_1 = x_0 - \frac{1}{L_0} g_0$, and $\tau_0 = 1$, this can be seen as

$$\begin{aligned}
U_0 &= \tau_0 \left(f_\star - f_0 + \frac{1}{2L_0} \|g_0\|^2 \right) + \frac{L_0}{2} \|x_0 - x_\star\|^2 - \frac{L_0}{2} \|z_1 - x_\star\|^2 \\
&= f_\star - f_0 - \langle g_0, x_\star - x_0 \rangle \\
&= W_{\star,0} \geq 0
\end{aligned}$$

where the inequality follows from the convexity of our function f.

Let $J = \{i \in [n-k, n-1] \mid \tau_i > 0\}$ and recall that our strategic memory update in Algorithm 4 ensures that $J \neq \emptyset$ for any iteration n. We can therefore set $s = \max\{i \in J\}$ and $m \in \operatorname{argmin}_{i \in J} v_i$, and then U' from (3.7) is properly defined for some z' and $\tau', \Delta' \geq 0$. The unknown variables are $f_\star$, f_n, g_n, and $x_\star$, so we can view (3.8) in terms of a polynomial of these variables. Accordingly, we rewrite (3.7) and (3.8) as

$$U' = \left(\frac{L_n}{2} \|x_0\|^2 - \frac{L_n}{2} \|z'\|^2 - \tau' v_m + \Delta' + \delta_n \right) + \tau' f_\star + \langle L_n (z' - x_0), x_\star \rangle, \tag{3.7}$$

$$U' = \left(- \sum_{i=n-k}^{n-1} \rho_i h_i - \sum_{i=n-k}^{n-1} \gamma_i w_i + \sum_{i=n-k}^{n-1} \rho_i \Delta_i + \epsilon \right) + \left(\sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i \right) f_\star + \langle L_n Z \rho - L_n G \gamma, x_\star \rangle. \tag{3.8}$$

Setting these equal, we obtain the equations

$$\frac{L_n}{2}\|x_0\|^2 - \frac{L_n}{2}\|z'\|^2 - \tau' v_m + \Delta' + \delta_n = -\sum_{i=n-k}^{n-1} \rho_i h_i - \sum_{i=n-k}^{n-1} \gamma_i w_i + \sum_{i=n-k}^{n-1} \rho_i \Delta_i + \epsilon, \tag{5.1}$$

$$\tau' = \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i, \tag{5.2}$$

$$L_n(z' - x_0) = L_n Z\rho - L_n G\gamma. \tag{5.3}$$

Next, we claim

$$U_n = U' + \tau' Q_{m,n}(L_n) + (\tau_n - \tau')W_{\star,n} \tag{3.13}$$

for some value τ_n. We can rewrite U_n in terms of z' as

$$U_n = \left(\frac{L_n}{2}\|x_0\|^2 - \frac{L_n}{2}\|z'\|^2 + \Delta_n\right) + \tau_n f_\star - \tau_n f_n + \langle \alpha z', g_n\rangle + \langle L_n(z' - x_0), x_\star\rangle$$
$$+ (\tau_n 1_{n<N} - \alpha^2)\frac{1}{2L_n}\|g_n\|^2 - \alpha\langle g_n, x_\star\rangle$$

where $1_{n<N}$ is the indicator function for $n < N$. Then expanding terms of (3.13), we have

$$\Delta_n = \Delta' + \delta_n, \tag{5.4}$$

$$\alpha z' = \tau'\left(x_n - (x_m - \frac{1}{L_n}g_m)\right) + (\tau_n - \tau')x_n, \tag{5.5}$$

$$\tau_n 1_{n<N} - \alpha^2 = -\tau', \tag{5.6}$$

$$\alpha = \tau_n - \tau'. \tag{5.7}$$

The conditions (5.6) and (5.7) exactly recover the construction of τ_n in Algorithm 3. We can simplify (5.3) and (5.5) to obtain explicit expressions for z' and x_n:

$$z' = x_0 + Z\rho - G\gamma, \tag{5.8}$$

$$x_n = \frac{1}{\tau_n}\left(\tau'(x_m - \frac{1}{L_n}g_m) + (\tau_n - \tau')z'\right). \tag{5.9}$$

Our goal is to obtain the best inductive hypothesis $U' \geq 0$. Thus, we want to maximize τ' while keeping Δ' bounded, and subject to our constraints (5.1)-(5.7) and the nonnegativity of our terms ρ, γ, and ϵ. Recall from Section 5.1, that the choice $\rho = \frac{L_n}{L_s}e_s, \gamma = 0$ is feasible. Moreover, this aligns with the standard induction for OBL (2.16), for which $\Delta' = \frac{L_n}{L_s}\Delta_s = \sum_{i=n-k}^{n-1}\Delta_i\rho_i$. Therefore, it is valid to bound[3]

$$\Delta' \leq \sum_{i=n-k}^{n-1} \rho_i \Delta_i. \tag{5.10}$$

This yields the optimization problem:

$$\begin{cases} \max_{\rho,\gamma,\Delta',\epsilon} & \sum_{i=n-k}^{n-1}\rho_i\tau_i + \sum_{i=n-k}^{n-1}\gamma_i \\ \text{s.t.} & \frac{L_n}{2}\|x_0\|^2 - \frac{L_n}{2}\|x_0 + Z\rho - G\gamma\|^2 = \sum_{i=n-k}^{n-1}\rho_i(-h_i + v_m\tau_i) \\ & \qquad + \sum_{i=n-k}^{n-1}\gamma_i(-w_i + v_m) + \sum_{i=n-k}^{n-1}\rho_i\Delta_i - \Delta' - \delta_n + \epsilon \\ & 0 \leq \Delta' \leq \sum_{i=n-k}^{n-1}\rho_i\Delta_i \\ & \rho,\ \gamma,\ \epsilon \geq 0. \end{cases} \tag{5.11}$$

[3]While there are several options for bounding Δ', we choose this bound because it allows Δ_n to not be monotonically increasing (See Remark 3).

We can take several steps to simplify (5.11). We can remove ϵ from our first constraint and replace it with an inequality. Next, due to the role of Δ' in the first constraint and its absence from the objective, one can see it will be pushed to its maximum value. We therefore fix $\Delta' = \sum_{i=n-k}^{n-1} \rho_i \Delta_i$. With these simplifications, and rearranging our first constraint, we obtain

$$\begin{cases} \max_{\rho,\gamma} & \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i \\ \text{s.t.} & \frac{L_n}{2}\|Z\rho - G\gamma\|^2 \leq \sum_{i=n-k}^{n-1} \rho_i (h_i - v_m \tau_i) + \sum_{i=n-k}^{n-1} \gamma_i (w_i - v_m) - \langle L_n Z\rho, x_0\rangle + \langle L_n G\gamma, x_0\rangle + \delta_n \\ & \rho,\, \gamma \geq 0. \end{cases} \tag{5.12}$$

Recalling our definition of $\epsilon(\rho, \gamma)$ from (3.11), this is exactly our optimization problem (3.12). Finally, with τ' as the solution to (5.12), i.e. $\tau' = \lambda_n = \sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i$, and τ_n set according to Algorithm 3, we recover the exact form of Algorithm 4.

All that remains to be shown is that $U_n \geq 0$. By assumption, we have that $U_i \geq 0$ for all $i \in J$ and $Q_{m,n}(L_n) \geq 0$. Combining with the nonnegativity of our terms $W_{\star,i}$, equations (3.8) and (3.13) yield the fact that $U_n \geq 0$. Rearranging $U_n \geq 0$ and $U_N \geq 0$ yield the final equations in our theorem.

5.3 Proof of Lemma 3.1 (iii)

Suppose that (3.12) is unbounded. Then its feasible domain must be unbounded. By the convexity of our feasible region, the nonnegativity of ρ, γ and the feasibility of $(\rho, \gamma) = (0, 0)$ in (3.12), this means there must exist nonzero (ρ, γ) such that $(c\rho, c\gamma)$ is feasible for all $c \geq 0$. Since our quadratic constraint must hold as $c \to \infty$, we must have $Z\rho - G\gamma = 0$. Then, with $\tau' = c(\sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i)$ and $z' = x_0 + c(Z\rho - G\gamma) = x_0$, we apply our OBL update from Algorithm 4. Observe that $\tau' \to \infty$ as c increases since $\sum_{i=n-k}^{n-1} \rho_i \tau_i + \sum_{i=n-k}^{n-1} \gamma_i > 0$. Moreover, we see that $\frac{\tau'}{\tau_n} \to 1$ as $\tau' \to \infty$. Thus, as $c \to \infty$, we get $x_n \to x_m - \frac{1}{L_n} g_m$.

Applying our guarantee from (3.14), we have

$$f(x_n) - f_\star \leq \frac{1}{\tau_n}\left(\frac{L_n}{2}\|x_0 - x_\star\|^2 + \Delta_n\right).$$

Taking the limit of both sides using our above arguments, we obtain $f(x_m - \frac{1}{L_n} g_m) \leq f_\star$ and therefore $x_m - \frac{1}{L_n} g_m \in \operatorname{argmin} f$.

5.4 Proof of Theorem 3.3

We will assume that at each iteration, (3.12) is bounded; otherwise, from Lemma 3.1 we would have found a minimizer of f and the algorithm would terminate. We show by induction that $\tau_{i,i} \geq \tau_{n,i}$ for all $i \in [n, N]$. For our base case, we have by definition $\tau_n = \tau_{i,i} = \tau_{n,i}$ at $i = n$.

Note that in the setting of $\mathcal{O}_L^{\mathcal{H}}$, at any iteration n we have $s = n - 1$. Then from Lemma 3.1, we know that at iteration $i > n$, Algorithm 4 will select τ' such that $\tau' \geq \tau_{i-1,i-1}$. Applying our update from Algorithm 4, we have for $n < i < N$,

$$\tau_{i,i} = \tau' + \frac{1 + \sqrt{1 + 8\tau'}}{2} \geq \tau_{i-1,i-1} + \frac{1 + \sqrt{1 + 8\tau_{i-1,i-1}}}{2} = \tau_{i-1,i}$$

and similar for the case $i = N$. Then by induction, using our standard guarantee from U_N, we have

$$\frac{f_N - f_\star}{\frac{L}{2}\|x_0 - x_\star\|^2} \leq \frac{1}{\tau_{N,N}} \leq \frac{1}{\tau_{N-1,N}} \leq \cdots \leq \frac{1}{\tau_{1,N}} \leq \frac{1}{\tau_{0,N}}.$$

The final equality of our theorem $\frac{1}{\tau_{0,N}} = \frac{2}{N(N+1)+\sqrt{2N(N+1)}}$ comes directly from [38, Theorem 4].

5.5 Proof of Theorem 3.5

Since $x_{n+1}^{(\ell)}$ is a serious step, we know that $L_{n+1}^{(\ell)} = L_n^{(\ell)}$ and we assume $\Delta_{n+1}^{(\ell)} \leq \Delta_n^{(\ell)}$. Moreover, from Lemma 3.1, we know $\tau_{n+1}^{(\ell)} \geq \tau_n^{(\ell)}$.

We therefore have

$$\tau_{n+1}^{(\ell)} \geq \tau_n^{(\ell)} \geq \frac{2L_n^{(\ell)}}{\mu} + \frac{L_n^{(\ell)}\Delta_n^{(\ell)}}{f(x_0^{(\ell)}) - f(x_n^{(\ell)})} \tag{5.13}$$

$$\geq \frac{2L_{n+1}^{(\ell)}}{\mu} + \frac{L_{n+1}^{(\ell)}\Delta_{n+1}^{(\ell)}}{f(x_0^{(\ell)}) - f(x_\star)} \tag{5.14}$$

$$= \frac{2L_{n+1}^{(\ell)}(f(x_0^{(\ell)}) - f(x_\star)) + \mu L_{n+1}^{(\ell)}\Delta_{n+1}^{(\ell)}}{\mu(f(x_0^{(\ell)}) - f(x_\star))}. \tag{5.15}$$

Then by our convergence rate from (3.15), having applied the "final step" update to obtain $x_{n+1}^{(\ell)}$, we can write

$$\begin{aligned}
f(x_{n+1}^{(\ell)}) - f(x_\star) &\leq \frac{L_{n+1}^{(\ell)}}{2\tau_{n+1}^{(\ell)}}\left(\|x_0^{(\ell)} - x_\star\|^2 + \Delta_{n+1}^{(\ell)}\right) \\
&\leq \frac{L_{n+1}^{(\ell)}}{\tau_{n+1}^{(\ell)}}\left(\frac{f(x_0^{(\ell)}) - f(x_\star)}{\mu} + \frac{\Delta_{n+1}^{(\ell)}}{2}\right) \\
&\leq \left(\frac{2(f(x_0^{(\ell)}) - f(x_\star)) + \mu\Delta_{n+1}^{(\ell)}}{2\mu}\right)\left(\frac{\mu(f(x_0^{(\ell)}) - f(x_\star))}{2(f(x_0^{(\ell)}) - f(x_\star)) + \mu\Delta_{n+1}^{(\ell)}}\right) \\
&= \frac{f(x_0^{(\ell)}) - f(x_\star)}{2}
\end{aligned}$$

where the second inequality comes from the strong convexity inequality (2.8) applied to $x = x_\star$ and $y = x_0^{(\ell)}$.

Recall that there can be at most $\log_2(\frac{L_n^{(\ell)}}{L_0^{(\ell)}})$ null steps in epoch ℓ by iteration n. Accounting for null steps in the result of Theorem 3.3, we have that $\tau_n^{(\ell)} \geq \frac{1}{2}(n - \log_2(\frac{L_n^{(\ell)}}{L_0^{(\ell)}}))^2$. Setting $\tau_n^{(\ell)}$ according to our restart condition and rearranging, we get

$$n \geq \sqrt{2\tau_n^{(\ell)}} + \log_2\left(\frac{L_n^{(\ell)}}{L_0^{(\ell)}}\right) = \sqrt{\frac{4L_n^{(\ell)}}{\mu} + \frac{2L_n^{(\ell)}\Delta_n^{(\ell)}}{f(x_0^{(\ell)}) - f(x_\star)}} + \log_2\left(\frac{L_n^{(\ell)}}{L_0^{(\ell)}}\right).$$

Acknowledgments. This work was supported in part by the Air Force Office of Scientific Research under award number FA9550-23-1-0531. Benjamin Grimmer was additionally supported as a fellow of the Alfred P. Sloan Foundation.

References

[1] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[2] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.

[3] Nicol N. Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 436–443, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.

[4] Donghwan Kim and Jeffrey A. Fessler. Optimized first-order methods for smooth convex minimization. *Mathematical Programming*, 159(1–2):81–107, 2016.

[5] Benjamin Grimmer, Kevin Shu, and Alex L. Wang. Beyond minimax optimality: A subgame perfect gradient method. *arXiv:2412.06731*, 2025.

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[7] Yu Nesterov. Universal gradient methods for convex optimization problems. *Mathematical Programming*, 152:381–404, 2015.

[8] Jaewook J. Suh and Shiqian Ma. An adaptive and parameter-free nesterov's accelerated gradient method for convex optimization. *arXiv:2505.11670*, 2025.

[9] Yura Malitsky and Konstantin Mishchenko. Adaptive gradient descent without descent. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[10] Yura Malitsky and Konstantin Mishchenko. Adaptive proximal gradient method for convex optimization. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc.

[11] Tianjiao Li and Guanghui Lan. A simple uniformly optimal method without line search for convex optimization. *Mathematical Programming*, 2025.

[12] Joao V. Cavalcanti, Laurent Lessard, and Ashia C. Wilson. Adaptive acceleration without strong convexity priors or restarts. *arXiv:2506.13033*, 2025.

[13] Puya Latafat, Andreas Themelis, Lorenzo Stella, and Panagiotis Patrinos. Adaptive proximal algorithms for convex optimization under local lipschitz continuity of the gradient. *Mathematical Programming*, 213:433–471, 2025.

[14] Maria-Luiza Vladarean, Yura Malitsky, and Volkan Cevher. A first-order primal-dual method with adaptivity to local smoothness. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

[15] Puya Latafat, Andreas Themelis, and Panagiotis Patrinos. On the convergence of adaptive first order methods: proximal gradient and alternating minimization algorithms. In *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*, volume 242 of *Proceedings of Machine Learning Research*, pages 197–208. PMLR, 15–17 Jul 2024.

[16] Shotaro Yagishita and Masaru Ito. Simple linesearch-free first-order methods for nonconvex optimization. *arXiv:2509.14670*, 2025.

[17] Konstantinos Oikonomidis, Emanuel Laude, Puya Latafat, Andreas Themelis, and Panagiotis Patrinos. Adaptive proximal gradient methods are universal without approximation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 38663–38682. PMLR, 21–27 Jul 2024.

[18] Guanghui Lan, Tianjiao Li, and Yangyang Xu. Projected gradient methods for nonconvex and stochastic optimization: new complexities and auto-conditioned stepsizes. *arXiv:2412.14291*, 2024.

[19] Wenzhi Gao, Ya-Chi Chu, Yinyu Ye, and Madeleine Udell. Gradient methods with online scaling. In *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pages 2192–2226. PMLR, 30 Jun–04 Jul 2025.

[20] Frederik Kunstner, Victor S. Portella, Mark Schmidt, and Nick Harvey. Searching for optimal per-coordinate step-sizes with multidimensional backtracking. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[21] Krzysztof C. Kiwiel. An aggregate subgradient method for nonsmooth and nonconvex minimization. *Journal of Computational and Applied Mathematics*, 14(3):391–400, 1986.

[22] Krzysztof C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46:105–122, 1990.

[23] Claude Lemaréchal, Arkadii Nemirovskii, and Yurii Nesterov. New variants of bundle methods. *Mathematical Programming*, 69:111–147, 1995.

[24] Guanghui Lan. Bundle-level type methods uniformly optimal for smooth and nonsmooth convex optimization. *Mathematical Programming*, 149:1–45, 2015.

[25] Mihai I. Florea and Yurii Nesterov. An optimal lower bound for smooth convex functions. *Foundations of Computational Mathematics*, 2025.

[26] Yurii Nesterov and Mihai I. Florea. Gradient methods with memory. *Optimization Methods and Software*, 37(3):936–953, 2021.

[27] Yoel Drori and Marc Teboulle. Performance of first-order methods for smooth convex minimization: A novel approach. *Mathematical Programming*, 145:451–482, 2014.

[28] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[29] Yoel Drori. The exact information-based complexity of smooth convex minimization. *Journal of Complexity*, 39:1–16, 2017.

[30] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. General considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[31] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[32] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[33] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[34] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[35] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer New York, NY, 2006.

[36] Philip Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.

[37] William W. Hager and Hongchao Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, 16(1):170–192, 2005.

[38] Chanwoo Park and Ernest K. Ryu. Optimal first-order algorithms as a function of inequalities. *Journal of Machine Learning Research*, 25:2557–2625, 2024.

[39] Nikhil Bansal and Anupam Gupta. Potential-function proofs for first-order methods. *Theory of Computing*, 15:1–32, 2019.

[40] Alexandre d'Aspremont, Damien Scieur, and Adrien Taylor. Acceleration methods. *Foundations and Trends® in Optimization*, 5(1–2):1–245, 2021.

[41] A.S. Nemirovskii and Yu.E. Nesterov. Optimal methods of smooth convex minimization. *USSR Computational Mathematics and Mathematical Physics*, 25(2):21–30, 1985.

[42] Brendan O'Donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15:715–732, 2015.

[43] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[44] MOSEK ApS. *MOSEK Optimizer API for Julia manual. Version 10.0.*, 2024.

[45] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[46] Patrick K. Mogensen and Asbjørn N. Riseth. Optim: A mathematical optimization package for julia. *Journal of Open Source Software*, 3(24):615, 2018.

[47] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[48] H.D. Mittelmann. Decision tree for optimization software. `https://plato.asu.edu/guide.html`, 2024.

[49] Yurii Nesterov. *Lectures on Convex Optimization*. Springer Publishing Company, 2nd edition, 2018.

[50] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

A Appendix

A.1 Preconditioning Algorithms

We briefly present standard algorithms for efficient computation of matrix-vector products with the inverse Hessian approximation B and the Hessian approximation B^{-1}. Algorithm 6 follows the standard two-loop recursion method [1]. Algorithm 7 simply uses the compact representation [50] of the Hessian approximation B^{-1},

$$B^{-1} = \theta I - \begin{bmatrix}\theta S & Y\end{bmatrix} \begin{bmatrix}\theta S^T S & T \\ T^T & -D\end{bmatrix}^{-1} \begin{bmatrix}\theta S^T \\ Y^T\end{bmatrix}$$

to compute $B^{-1}v$ without storing any full-dimension matrices (see Algorithm 7 for variable definitions).

Algorithm 6: Efficient calculation of Bv [1]

Input : $\mathcal{M} := \{(s_i, y_i)\}_{i=1}^t$, $v \in \mathbb{R}^d$

$q = v$

for $i = t, t-1, \ldots, 1$ **do**

  $\eta_i = \frac{1}{y_i^T s_i}$

  $\alpha_i = \eta_i (s_i^T q)$

  $q = q - \alpha_i y_i$

end

$r = \frac{s_t^T y_t}{y_t^T y_t} q$

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

  $\beta_i = \eta_i (y_i^T r)$

  $r = r + (\alpha_i - \beta_i) s_i$

end

Output : r

Algorithm 7: Efficient calculation of $B^{-1}v$ [50]

Input : $\mathcal{M} := \{(s_i, y_i)\}_{i=1}^t$, $v \in \mathbb{R}^d$

$\theta = \frac{y_t^T y_t}{s_t^T y_t}$

$D = \operatorname{diag}(s_1^T y_1, \ldots, s_t^T y_t)$

$S = \begin{bmatrix}s_1 & \ldots & s_t\end{bmatrix} \quad Y = \begin{bmatrix}y_1 & \ldots & y_t\end{bmatrix}$

$T_{i,j} = \begin{cases} s_i^T y_j & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}$

$M = \begin{bmatrix}\theta S^T S & T \\ T^T & -D\end{bmatrix}$

$q = M^{-1} \begin{bmatrix}\theta S^T v \\ Y^T v\end{bmatrix}$

$r = \theta v - \begin{bmatrix}\theta S & Y\end{bmatrix} q$

Output : r

A.2 Proof of Theorem 3.4

From Theorem 3.3, we have the upper bound

$$\min_{A\in\mathcal{A}^{\mathcal{H}}}\max_{O\in\mathcal{O}_L^{\mathcal{H}}}\frac{f(x_N)-f(x_\star)}{\frac{L}{2}\|x_0-x_\star\|^2}\leq\frac{1}{\tau_{n,N}}. \tag{A.1}$$

Thus, to demonstrate the minimax optimality we have to provide a matching lower bound. To do so, we will construct a particular oracle that is hard for all $A\in\mathcal{A}^{\mathcal{H}}$.

First, due to our oracle class $\mathcal{O}_L^{\mathcal{H}}$, we know $L=L_0=\ldots,L_n$. In this setting, we immediately have $\Delta'=0$ and $\delta_n=0$. Our subproblem therefore reduces to the following:

$$\tau'=\left\{\begin{array}{ll}\max_{\rho,\gamma} & \sum_{i=n-k}^{n-1}\rho_i\tau_i+\sum_{i=n-k}^{n-1}\gamma_i\\ \text{s.t.} & \sum_{i=n-k}^{n-1}\rho_i a_i+\sum_{i=n-k}^{n-1}\gamma_i b_i-\frac{L}{2}\|Z\rho-G\gamma\|^2\geq 0\\ & \rho,\ \gamma\geq 0.\end{array}\right. \tag{A.2}$$

Next, we introduce some notation. Let ϕ_n be the solution to (3.12) at iteration n, (i.e., $\phi_n=\sum_{i=n-k}^{n-1}\rho_i\tau_i+\sum_{i=n-k}^{n-1}\gamma_i$) and define $\tau_{n,i}$ as in Theorem 3.3. Further define

$$\begin{aligned}\phi_i&=\tau_{n,i-1}\quad\forall i\in[n+1,N]\\ \psi_i&=\begin{cases}\frac{1+\sqrt{1+8\phi_i}}{2} & \text{if } n\leq i\leq N-1\\ \sqrt{\phi_i} & \text{if } i=N\end{cases}\\ \tau_{n,i}&=\phi_i+\psi_i.\end{aligned}$$

We now consider the dual program to (A.2) and obtain the following lemma. Recall from Lemma 3.1, that a bounded solution to (3.12) implies that strong duality holds. The derivation of the dual problem below is effectively identical to that in [5, Lemma 3].

Lemma A.1. *The dual program to* (A.2) *is*

$$\inf_{\xi,y}\left\{\begin{array}{ll}\frac{2\|y\|^2}{\xi L}+\frac{\xi}{2}\|x_0\|^2+2\langle x_0,y\rangle: & \xi(-h+v_m\tau)-2Z^Ty\geq\tau\\ & \xi(-w+v_m\mathbf{1})+2G^Ty\geq 1\\ & \xi>0\end{array}\right\} \tag{A.3}$$

Strong duality holds between (A.2) *and* (A.3). *If* (ρ,γ) *and* (ξ,y) *are optimal solutions to* (3.12) *and* (A.3) *respectively, then*

$$\frac{-2y}{\xi L}=x_0+Z\rho-G\gamma=z'.$$

We continue following the approach of [5] and define $f_\star=v_m-\frac{1}{\xi}$. Then our feasibility constraints for ξ can be rewritten as

$$\tau_i\left(f_\star-f_i+\frac{1}{2L}\|g_i\|^2\right)+\frac{L}{2}\|x_0\|^2-\frac{L}{2}\|z_{i+1}\|^2+L\langle z_{i+1}-x_0,z'\rangle\geq 0\qquad\forall i\in[n-k,n-1] \tag{A.4}$$

$$f_\star-f_i-\langle g_i,z'-x_i\rangle\geq 0\qquad\forall i\in[n-k,n-1]. \tag{A.5}$$

Letting $\sigma=\|z'-x_0\|^2$ and applying strong duality we have

$$\phi_n=\frac{2\|y\|^2}{\xi L}+\frac{\xi L}{2}\|x_0\|^2+2\langle x_0,y\rangle$$

$$= \frac{\xi L}{2}\left\|\frac{2y}{\xi L} + x_0\right\|^2$$
$$= \frac{\xi L \sigma}{2}$$

from which we obtain $f_\star = v_m - \frac{L\sigma}{2\phi_n}$.

Let $e_0, e_1, \ldots$ be an orthonormal basis. We assume without loss of generality that $g_i \in \text{span}(e_0, \ldots, e_i)$ for all $i < n$. We now define the responses of our oracle $x_i \mapsto (f_i, g_i)$ for $i = n, n+1, \ldots, N$ by

$$f_i = f_\star + L\sigma\psi_i\beta_i = v_m + \frac{L\sigma}{2}(2\psi_i\beta_i - \frac{1}{\phi_n}), \tag{A.6}$$
$$g_i = L\sqrt{\sigma\beta_i}e_i \tag{A.7}$$

where β_i is defined inductively as $\beta_i = \frac{1}{\phi_i(2\psi_i+1)}\left(1 + \sum_{j=n}^{i-1}\psi_j^2\beta_j\right)$ with $\beta_n = \frac{1}{\phi_n(2\psi_n+1)}$. Note that this oracle selection is independent of the value of x_i; this is reasonable since our oracle is not restricted to return values corresponding to an actual function[4]. It is only required to satisfy $\{Q_{i,j}\}_{i<j<n} \cup \{W_{\star,i}\}_{i=0}^{n-1}$. Lastly, we set $x_\star = z_{N+1}$.

We claim that this oracle does indeed satisfy our inequality set $\{Q_{i,j}\}_{i<j<n} \cup \{W_{\star,i}\}_{i=0}^{n-1}$. We will verify each inequality directly, but to do so, we first need a few useful lemmas.

Lemma A.2. *Let $j > i \geq n$. Then the following hold*

$$\beta_i \leq \frac{1}{\phi_n(2\psi_i + 1)} \tag{A.8}$$
$$(2\psi_j + 1)\beta_j \leq (2\psi_i - 1)\beta_i. \tag{A.9}$$

Proof. Note from our definition of ψ_i that $\psi_i^2 = 2\tau_{n,i} - \psi_i = \phi_i + \tau_{n,i}$ for $n \leq i < N$ and $\psi_N^2 = \tau_{n,N} - \psi_N = \phi_N$. From this we have

$$\tau_{n,i}(2\psi_i - 1) = \begin{cases} \phi_i(2\psi_i + 1) + \psi_i^2 & \text{if } i \in [n, N-1] \\ \phi_N(2\psi_N + 1) + \psi_N^2 - \tau_{n,N} & \text{if } i = N. \end{cases}$$

Using this relation, we can write

$$\begin{aligned}\phi_{i+1}(2\psi_{i+1} + 1)\beta_{i+1} &= 1 + \sum_{\ell=n}^{i}\psi_\ell^2\beta_\ell \\ &= 1 + \sum_{\ell=n}^{i-1}\psi_\ell^2\beta_\ell + \psi_i^2\beta_i \\ &= \phi_i(2\psi_i + 1)\beta_i + \psi_i^2\beta_i \\ &= \tau_{n,i}(2\psi_i - 1)\beta_i.\end{aligned}$$

Recognizing that $\phi_{i+1} = \tau_{n,i}$, we rearrange to obtain the identity

$$(2\psi_{i+1} + 1)\beta_{i+1} = (2\psi_i - 1)\beta_i. \tag{A.10}$$

[4]Indeed, this construction even allows the oracle to be "inconsistent" over time: If an algorithm selects $x_1 = x_0$, the oracle can return (f_1, g_1) with $f_1 \neq f_0$ and $g_1 \neq g_0$, as long as $Q_{0,1}(L) \geq 0$ still holds.

We now prove (A.8) by induction. By definition, $\beta_n = \frac{1}{\phi_n(2\psi_n+1)}$, so our base case holds. Then for $i > n$, using (A.10) we can write

$$\beta_i = \frac{(2\psi_{i-1}-1)\beta_{i-1}}{2\psi_i+1} \leq \frac{2\psi_{i-1}-1}{2\psi_i+1}\frac{1}{\phi_n(2\psi_{i-1}+1)} \leq \frac{1}{\phi_n(2\psi_{i-1}+1)}$$

where we use our induction hypothesis and then the fact that ψ_i is monotonically increasing.

For the second inequality (A.9), applying our identity (A.10) yields

$$(2\psi_j+1)\beta_j = (2\psi_{j-1}-1)\beta_{j-1} \leq (2\psi_{j-1}+1)\beta_{j-1}$$

and the result follows from induction.

□

Now we begin verifying our inequalities. Since we are restricted to gradient span algorithms, we know that $x_i \in \text{span}\{g_0, \ldots, g_{i-1}\}$ for all i. We will frequently use the fact from our orthonormal construction of g_i that for any $j \geq n$, if $i < j$ we have $\langle g_j, g_i\rangle = 0$ and $\langle g_j, x_i - x_j\rangle = 0$. We begin with our inequalities $Q_{i,j}(L)$ for $i < j$.

- Let $i < j < n$. Then by the definition of our oracle family $\mathcal{O}_L^{\mathcal{H}}$, we know $Q_{i,j}(L) \geq 0$.
- Let $i < n \leq j$. Then we have

$$\begin{aligned}
Q_{i,j}(L) &= f_i - f_j - \langle g_j, x_i - x_j\rangle - \frac{1}{2L}\|g_i - g_j\|^2 \\
&= f_i - \frac{1}{2L}\|g_i\|^2 - f_j - \frac{1}{2L}\|g_j\|^2 \\
&\geq v_m - f_j - \frac{1}{2L}\|g_j\|^2 \\
&= \frac{L\sigma}{2}\left(\frac{1}{\phi_n} - (2\psi_j+1)\beta_j\right) \\
&\geq 0
\end{aligned}$$

  where the first inequality follows from the definition of m and the second inequality follows from Lemma A.2. Note that in the special case $i = m$ and $j = n$, we have equality: $Q_{m,n}(L) = 0$.

- Let $n \leq i < j$. Then we have

$$\begin{aligned}
Q_{i,j}(L) &= f_i - f_j - \langle g_j, x_i - x_j\rangle - \frac{1}{2L}\|g_i - g_j\|^2 \\
&= f_i - \frac{1}{2L}\|g_i\|^2 - f_j - \frac{1}{2L}\|g_j\|^2 \\
&= \frac{L\sigma}{2}\left((2\psi_i-1)\beta_i - (2\psi_j+1)\beta_j\right) \\
&\geq 0
\end{aligned}$$

  where the inequality follows from Lemma A.2. Note that in the special case $i = j-1$, from (A.10) we have equality: $Q_{j-1,j}(L) = 0$.

Now consider the inequalities $W_{\star,i}$.

- Let $i < n$. Then

$$\begin{aligned} W_{\star,i} &= f_\star - f_i - \langle g_i, x_\star - x_i \rangle \\ &= f_\star - f_i - \langle g_i, z' - x_i \rangle \\ &\geq 0. \end{aligned}$$

  In the second line, we use the fact that $x_\star = z_{N+1} = z' - \sum_{\ell=n}^{N} \frac{\psi_\ell}{L} g_\ell$, along with the orthogonality of our g_ℓ vectors, and the inequality follows from our dual feasibility expression (A.5).

- Let $i \geq n$. Then we have

$$\begin{aligned} W_{\star,i} &= f_\star - f_i - \langle g_i, x_\star - x_i \rangle \\ &= f_\star - f_i - \langle g_i, z_{N+1} - x_i \rangle \\ &= f_\star - f_i + \frac{\psi_i}{L} \|g_i\|^2 \\ &= -L\sigma\psi_i\beta_i + \frac{\psi_i}{L} L^2 \sigma \beta_i \\ &= 0. \end{aligned}$$

Next, we verify that our inductive hypotheses $U_i \geq 0$ hold at each step in our algorithm history. For $i < n$, we have

$$\begin{aligned} U_i &= \tau_i \left(f_\star - f_i + \frac{1}{2L} \|g_i\|^2 \right) + \frac{L}{2} \|x_0 - x_\star\|^2 - \frac{L}{2} \|z_{i+1} - x_\star\|^2 \\ &= \tau_i \left(f_\star - f_i + \frac{1}{2L} \|g_i\|^2 \right) + \frac{L}{2} \|x_0\|^2 - \frac{L}{2} \|z_{i+1}\|^2 + L \langle z_{i+1} - x_0, x_\star \rangle \\ &= \tau_i \left(f_\star - f_i + \frac{1}{2L} \|g_i\|^2 \right) + \frac{L}{2} \|x_0\|^2 - \frac{L}{2} \|z_{i+1}\|^2 + L \langle z_{i+1} - x_0, z' \rangle \\ &\geq 0. \end{aligned}$$

The third equality again follows from $x_\star = z_{N+1} = z' - \sum_{\ell=n}^{N} \frac{\psi_\ell}{L} g_\ell$ and the orthogonality of our vectors g_ℓ, and the inequality follows from our dual feasibility expression (A.4).

Thus we have shown that $O \in \mathcal{O}_L^{\mathcal{H}}$. All that remains to be shown is our lower bound. We claim that it is sufficient to show that $U_N = 0$. By the definition of U_N we can then rearrange to obtain

$$\begin{aligned} f_N - f_\star &= \frac{1}{\tau_{n,N}} \left(\frac{L}{2} \|x_0 - x_\star\|^2 - \frac{L}{2} \|z_{N+1} - x_\star\|^2 \right) \\ &= \frac{1}{\tau_{n,N}} \left(\frac{L}{2} \|x_0 - x_\star\|^2 - \frac{L}{2} \|x_\star - x_\star\|^2 \right) \\ &= \frac{L \|x_0 - x_\star\|^2}{2\tau_{n,N}}. \end{aligned}$$

Therefore, the lower bound

$$\min_{A \in \mathcal{A}^{\mathcal{H}}} \max_{O \in \mathcal{O}_L^{\mathcal{H}}} \frac{f(x_N) - f(x_\star)}{\frac{L}{2} \|x_0 - x_\star\|^2} \geq \frac{1}{\tau_{n,N}} \tag{A.11}$$

holds.

To show that $U_N = 0$, we first look at U':

$$\begin{aligned}
U' &= \phi_n \left(f_\star - f_m + \frac{1}{2L}\|g_m\|^2 \right) + \frac{L}{2}\|x_0 - x_\star\|^2 - \frac{L}{2}\|z' - x_\star\|^2 \\
&= -\frac{L}{2}\|z' - x_0\|^2 + \frac{L}{2}\|x_0 - x_\star\|^2 - \frac{L}{2}\|z' - x_\star\|^2 \\
&= L\langle x_0 - z', z' - x_\star \rangle \\
&= L\langle x_0 - z', \sum_{\ell=n}^{N} \frac{\psi_\ell}{L} g_\ell \rangle \\
&= 0
\end{aligned}$$

where the final equality comes from the fact that $x_0 - z' \in \text{span}(g_0, \ldots, g_{n-1})$ and the orthogonality of g_ℓ. Finally, as noted above, we have $Q_{m,n}(L) = 0$, and $Q_{i-1,i}(L) = W_{\star,i} = 0$ for all $i \geq n$. Then following the standard OBL induction, we have

$$\begin{aligned}
U_N &= U_n + \sum_{\ell=n+1}^{N} \left(\phi_\ell Q_{\ell-1,\ell}(L) + \psi_\ell W_{\star,\ell} \right) \\
&= U' + \phi_n Q_{m,n}(L) + \psi_n W_{\star,n} + \sum_{\ell=n+1}^{N} \left(\phi_\ell Q_{\ell-1,\ell}(L) + \psi_\ell W_{\star,\ell} \right) \\
&= 0
\end{aligned}$$

and our proof is complete.

A.3 Additional Numerical Results

For reference, we include performance plots for each individual problem type considered in Section 4.2.

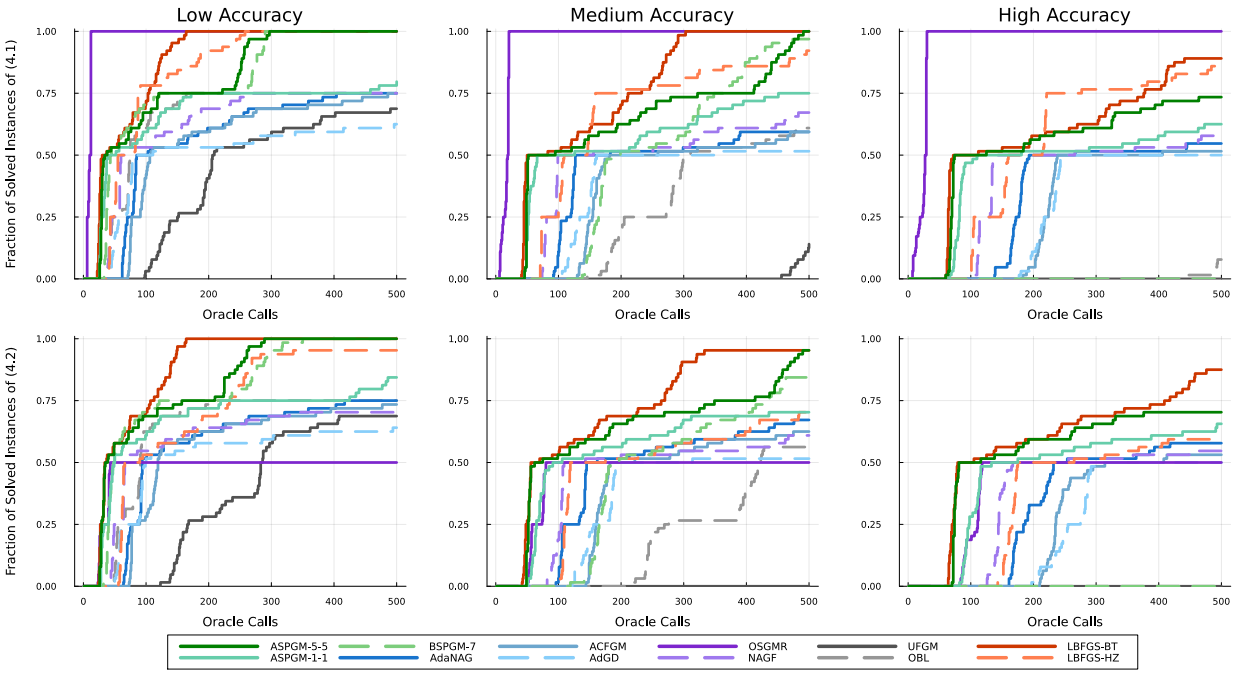

Figure 7: Performance comparison for least squares regression (4.1) and logistic regression (4.2). The performance is measured by $(f(x_n) - f_\star)/(f(x_0) - f_\star)$ and the target accuracies from left to right are $10^{-4}, 10^{-7}, 10^{-10}$.

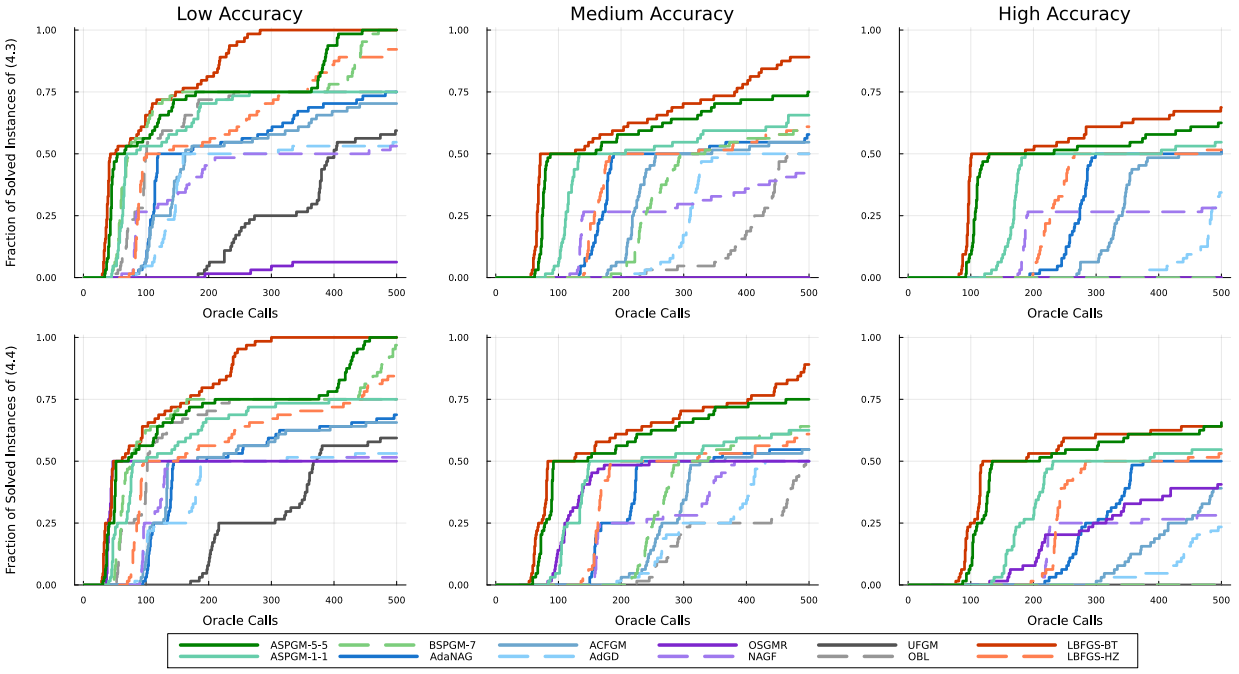

Figure 8: Performance comparison for smooth feasibility problems (4.3) and (4.4). The performance is measured by $(f(x_n)-f_\star)/(f(x_0)-f_\star)$ and the target accuracies from left to right are $10^{-4}, 10^{-7}, 10^{-10}$.

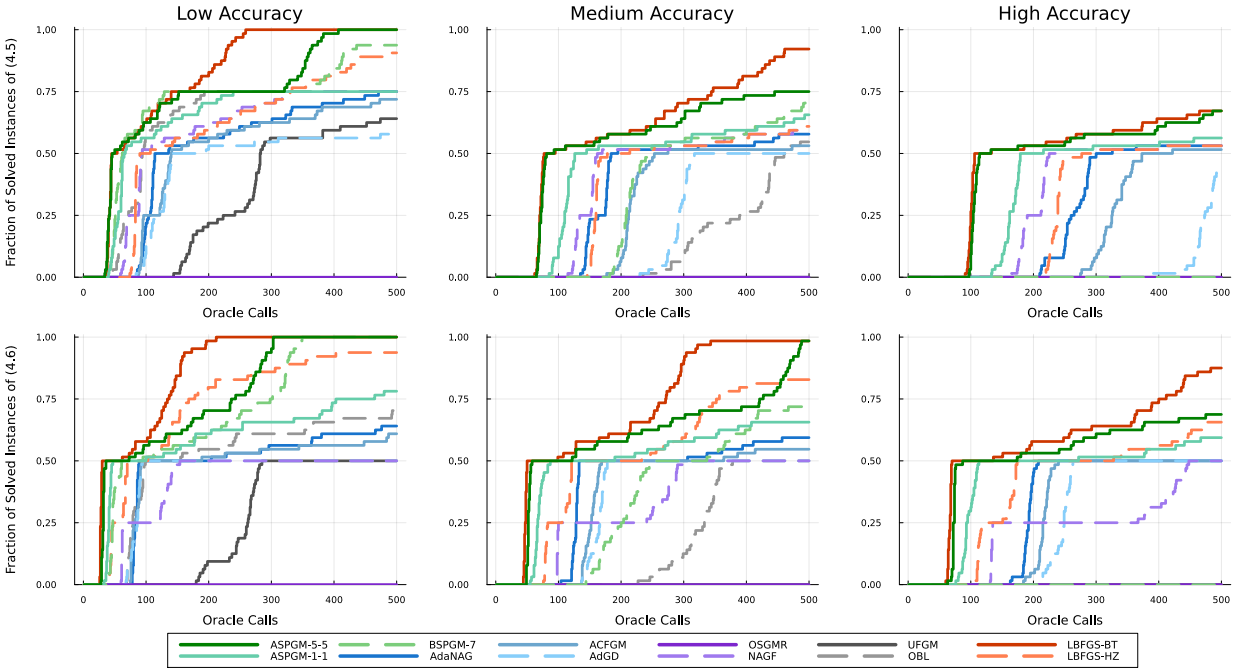

Figure 9: Performance comparison for locally smooth problems (4.5) and (4.6). The performance is measured by $(f(x_n)-f_\star)/(f(x_0)-f_\star)$ and the target accuracies from left to right are $10^{-4}, 10^{-7}, 10^{-10}$.