Optimal and Adaptive Monteiro-Svaiter Acceleration

Anonymous Author(s)
Affiliation
Address
email

Abstract

We develop a variant of the Monteiro-Svaiter (MS) acceleration framework that removes the need to solve an expensive implicit equation at every iteration. Consequently, for any $p \geq 2$ we improve the complexity of convex optimization with Lipschitz pth derivative by a logarithmic factor, matching a lower bound. We also introduce an MS subproblem solver that requires no knowledge of problem parameters, and implement it as either a second- or first-order method by solving linear systems or applying MinRes, respectively. On logistic regression problems our method outperforms previous momentum-based second-order methods, but underperforms Newton's method; simply iterating our first-order adaptive subproblem solver is comparable to L-BFGS.

1 Introduction

We consider the problem of minimizing a convex function $f : \mathcal{X} \to \mathbb{R}$ over closed convex set $\mathcal{X} \subseteq \mathbb{R}^d$, given access to an oracle $\mathcal{O} : \mathcal{X} \to \mathcal{X}$ that minimizes some local model of f around a given query point. A primary motivating example of such an oracle is the cubic-regularized Newton step

$$\mathcal{O}_{\mathsf{cr}}(y) = \operatorname*{argmin}_{x \in \mathcal{X}} \left\{ f(y) + \nabla f(y)^T (x - y) + \frac{1}{2}(x-y)^T \nabla^2 f(y)(x-y) + \frac{M}{6}\|x-y\|^3 \right\}, \quad (1)$$

i.e., minimizing the second-order Taylor approximation of f around y plus a cubic regularization term. However, our results apply to several more oracles including a simple gradient step, regularized higher-order Taylor expansions [4, 18, 6, 22, 7, 30, 20, 34], ball-constrained optimization [10], and new adaptive oracles that we develop.

A seminal work by Monteiro and Svaiter [27] (MS) shows how to accelerate the basic oracle iteration $x_{t+1} = \mathcal{O}(x_t)$. Their algorithm is based on the observation that many oracles, including $\mathcal{O}_{\mathsf{cr}}$, implicitly approximate proximal points. That is, for every y and $x = \mathcal{O}(y)$, there exists $\lambda_{x,y} > 0$ such that $x \approx \operatorname{argmin}_{x' \in \mathcal{X}} \left\{ f(x') + \frac{1}{2}\lambda_{x,y}\|x' - y\|^2 \right\}$, with the approximation error controlled by a specific condition they define. MS then prove that, under this condition, the accelerated proximal point method [21, 33] (with dynamic regularization parameter) maintains its rate of convergence. Applying their framework to $\mathcal{O}_{\mathsf{cr}}$ assuming $\nabla^2 f$ is Lipschitz, they achieve error bounds that decay as $O(t^{-7/2} \log t)$ after t oracle calls, improving the rates $O(t^{-2})$ of the basic $\mathcal{O}_{\mathsf{cr}}$ iteration [31] and the rate $O(t^{-3})$ of an earlier accelerated method [28]. Subsequent works apply the MS framework to different oracles, obtaining improved theoretical guarantees for highly-smooth optimization [18, 6, 22, 34, 1], parallel optimization [5], logistic and ℓ_∞ regression [7, 10], minimizing functions with Hölder continuous higher derivatives [34], and distributionally-robust optimization [11, 9].

However, all the algorithms based on the MS framework share a common drawback: the iterate y_t used to produce $x_{t+1} = \mathcal{O}(y_t)$ depends on the proximal parameter $\lambda_{t+1} = \lambda_{x_{t+1}, y_t}$, which itself depends on both x_{t+1} and y_t. This circular dependence necessitates solving an implicit equation for λ_{t+1}; MS propose a bisection procedure for doing so using a number of oracle calls logarithmic in the problem parameters. From a theoretical perspective, the additional bisection complexity introduces a

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

logarithmic gap between the upper bounds due to MS-based algorithms and the best known lower bounds [2, 20] in a number of settings.

From a practical perspective, the MS bisection is undesirable since it simply discards the optimization progress made by oracle calls during each bisection. Indeed, Nesterov [30, §6] argues that the logarithmic cost of the bisection likely renders the MS scheme for accelerating $\mathcal{O}_{\mathsf{cr}}$ inferior in practice to algorithms whose error decays at the asymptotically worse rate $O(t^{-3})$ but requires no bisection. Carmon et al. [11] point out the bisection as one of the main hurdles in making their theoretical scheme practical, Song et al. [34] propose a heuristic alternative to bisection, and Doikov and Nesterov [14, §4] mention removing the logarithmic factors from the MS complexity bound as an open theoretical question. (See Appendix A for extended discussion of related work.)

1.1 Our contributions

We settle this open question, providing a variant of MS acceleration that requires no bisection (Section 2). When combined with existing MS oracles (Section 3.1), our algorithm obtains complexity bounds that are optimal up to constant factors, improving over prior art by a logarithmic term (see Table 1). In addition, our algorithm has no parameters sensitive to tuning.

We then go a step further and (in Section 3.2) develop an adaptive alternative to $\mathcal{O}_{\mathsf{cr}}$ (1). Our oracle does not require tuning the parameter M, which in theory should be proportional to the (difficult to estimate) Lipschitz constant of $\nabla^2 f$. Using our oracle, we obtain the optimal Hessian evaluation complexity $O(t^{-(4+3\nu)/2})$ for functions with order-ν Hölder Hessian (Lipschitz Hessian is the $\nu = 1$ special case), without requiring any knowledge of the Hölder constant and order ν. Our oracle is also efficient: while existing complexity bounds for computing $\mathcal{O}_{\mathsf{cr}}$ require a logarithmic number of linear system solutions per call, our oracle requires a double-logarithmic number. Moreover, when used with our acceleration method, the number of linear system solves per iteration is essentially constant.

We also provide a first-order implementation of our adaptive oracle (Section 3.3). It approximately solves linear systems via first-order operations (Hessian-vector products) using MinRes/Conjugate Residuals [35, 17] with a simple, adaptive, stopping criterion lifted directly from our analysis. Our oracle attains the optimal first-order evaluation complexity for smooth functions up to an *additive* logarithmic term, without knowledge of the gradient Lipschitz constant or any parameter tuning. Moreover, it maintains an optimal outer iteration complexity for Hölder Hessian of any order.

Finally, we report empirical results (Section 4). On logistic regression problems, combining our optimal acceleration scheme with our adaptive oracle outperforms previously proposed accelerated second-order methods. However, we also show that (while somewhat helpful for $\mathcal{O}_{\mathsf{cr}}$ with a conservative choice of M), adding momentum to well-tuned or adaptive second-order methods is *harmful* in logistic regression: simply iterating our oracle—or, better yet, applying Newton's method—dramatically outperforms all "accelerated" algorithms. This important fact seems to have gone unobserved in the literature on accelerated second-order methods, despite logistic regression appearing in many related experiments [34, 14, 26, 23]. Simply iterating our adaptive oracle outperforms the classical accelerated gradient descent, and is comparable to L-BFGS.

1.2 Limitations and outlook

While our algorithms resolve an enduring open problem in the optimal oracle complexity of convex optimization and are free of sensitive parameters that typically hinder theoretically-optimal methods, the main limitation is nevertheless practical performance. On logistic regression, Newton's method (or first-order relaxations such as L-BFGS) performs significantly better than our method, which in turn outperforms the previous second-order momentum methods in the literature. We therefore conclude that despite its appealing worst-case complexity guarantees—and despite being nearly universally helpful in first-order methods—momentum appears to be harmful for logistic regression problem. We do not fully understand why this is so, but we suspect that it has to do with additional structure in logistic regression, which Newton's method can automatically exploit but momentum cannot. We believe that future research should focus on identifying the structure that makes Newton's method so efficient, and modifying momentum schemes to leverage it.

Scalability is another important limitation. While our first-order oracle significantly improves scalability over the second-order oracle from which it is built, it still relies on exact gradient and Hessian-vector products. Therefore, it will have difficulty scaling up to very large datasets. Nevertheless, we hope that further scalability improvements may be possible by building a stochastic

Algorithm 1: Optimal MS Acceleration

Input: Initial x_0, function f, oracle $\mathcal{O}$
Parameters: Initial λ'_0, multiplicative adjustment factor $\alpha > 1$

Set $v_0 = x_0$, $A_0 = 0$
$\tilde{x}_1, \lambda_1 = \mathcal{O}(x_0; \lambda'_0)$, $\lambda'_1 = \lambda_1$
for $t = 0, 1, \ldots,$ **do**
  $a'_{t+1} = \frac{1}{2\lambda'_{t+1}}\left(1 + \sqrt{1 + 4\lambda'_{t+1}A_t}\right)$
  $A'_{t+1} = A_t + a'_{t+1}$
  $y_t = \frac{A_t}{A'_{t+1}}x_t + \frac{a'_{t+1}}{A'_{t+1}}v_t$
  if $t > 0$ **then** $\tilde{x}_{t+1}, \lambda_{t+1} = \mathcal{O}(y_t; \lambda'_{t+1})$
  if $\lambda_{t+1} \leq \lambda'_{t+1}$ **then**
    $a_{t+1} = a'_{t+1},\ A_{t+1} = A_t + a_t$
    $x_{t+1} = \tilde{x}_{t+1}$
    $\lambda'_{t+2} = \frac{1}{\alpha}\lambda'_{t+1}$
  else
    $\gamma_{t+1} = \frac{\lambda'_{t+1}}{\lambda_{t+1}}$
    $a_{t+1} = \gamma_{t+1}a'_{t+1},\ A_{t+1} = A_t + a_t$
    $x_{t+1} = \frac{(1-\gamma_{t+1})A_t}{A_{t+1}}x_t + \frac{\gamma_{t+1}A'_{t+1}}{A_{t+1}}\tilde{x}_{t+1}$
    $\lambda'_{t+2} = \alpha\lambda'_{t+1}$
  $v_{t+1} = v_t - a_{t+1}\nabla f(\tilde{x}_{t+1})$

Algorithm 0: Standard MS Acceleration

Input: Initial x_0, function f, oracle $\mathcal{O}$
Parameters: Bisection limits λ^ℓ, λ^h, and tolerance $\rho > 1$

1 Set $v_0 = x_0$, $A_0 = 0$
2 **for** $t = 0, 1, \ldots,$ **do**
3 $\lambda^\ell_{t+1}, \lambda^h_{t+1} = \lambda^\ell, \lambda^h$
4 $\lambda'_{t+1} = \frac{\lambda^\ell_{t+1} + \lambda^h_{t+1}}{2}$
5 $a'_{t+1} = \frac{1}{2\lambda'_{t+1}}\left(1 + \sqrt{1 + 4\lambda'_{t+1}A_t}\right)$
6 $A'_{t+1} = A_t + a'_{t+1}$
7 $y_t = \frac{A_t}{A'_{t+1}}x_t + \frac{a_{t+1}}{A'_{t+1}}v_t$
8 $\tilde{x}_{t+1}, \lambda_{t+1} = \mathcal{O}(y_t; \lambda'_{t+1})$
9 **if** $\lambda_{t+1} \in [\frac{1}{\rho}\lambda'_{t+1}, \lambda'_{t+1}]$ **then**
10 $a_{t+1} = a'_{t+1},\ A_{t+1} = A_t + a_t$
11 $x_{t+1} = \tilde{x}_{t+1}$
12 **else if** $\lambda_{t+1} < \frac{1}{\rho}\lambda'_{t+1}$ **then**
13 $\lambda^h_{t+1} = \lambda'_{t+1}$
14 Go to line 4
15 **else**
16 $\lambda^l_{t+1} = \lambda'_{t+1}$
17 Go to line 4
18 $v_{t+1} = v_t - a_{t+1}\nabla f(\tilde{x}_{t+1})$

oracle out of our exact-gradient first-order oracle, bringing with it the exciting prospect of a new and powerful adaptive stochastic gradient method.

2 Removing the bisection from the Monteiro-Svaiter framework

In this section we present our acceleration algorithm (Algorithm 1) which removes the bisection from the MS method (shown in stylized form as Algorithm 0) and thereby attaining optimal rates of convergence. For simplicity, in this section and the next we focus on unconstrained optimization ($\mathcal{X} = \mathbb{R}^d$) and assume that f is continuously differentiable, so that ∇f exists. In Appendix B we extend our framework to general closed and convex domains and non-differentiable convex objectives.

The key object in both the original MS algorithm and our new variant is an oracle $\mathcal{O}$ that approximately minimizes a local model of f around a query point y and satisfies the following approximation error bound, adapted from Monteiro and Svaiter [27, eq. (3.3)] (λ in [27] is $1/\lambda$ in our notation).

Definition 1 (MS oracle)**.** *An oracle $\mathcal{O} : \mathcal{X} \times \mathbb{R}_+ \to \mathcal{X} \times \mathbb{R}_+$ is a σ-MS oracle for function $f : \mathbb{R}^d \to \mathbb{R}$ if for every $y \in \mathcal{X}$ and $\lambda' > 0$, the points $(x, \lambda) = \mathcal{O}(y; \lambda')$ satisfy*

$$\left\|x - \left(y - \tfrac{1}{\lambda}\nabla f(x)\right)\right\| \leq \sigma\|x - y\|. \tag{2}$$

Definition 1 endows the oracle with an additional output λ and an additional input λ'. The value of λ has the following simple interpretation: any point x satisfying (2) approximately minimizes $F(x') = f(x') + \frac{\lambda}{2}\|x' - y\|^2$ in the sense that $\|\nabla F(x)\| \leq \lambda\sigma\|x - y\|$. In particular, computing an exact proximal point $x_\lambda = \operatorname{argmin}_{x'} F(x')$ and outputting (x, λ) implements a 0-MS oracle. The input λ' is optional: oracle implementations in prior work do not require it, but our new adaptive oracles (described in the next section) use it for improved efficiency. In Appendix B we provide a more general approximation condition for MS oracles similar to the original proposal of [27], as well as a different, stochastic condition similar to that of [3, 9].

Let us discuss the key differences between our algorithm (Algorithm 1) and the standard MS algorithm (Algorithm 0). At every iteration, the standard MS algorithm searches for a value λ'_{t+1} such that $\tilde{x}_{t+1}, \lambda_{t+1} = \mathcal{O}(y_t; \lambda'_{t+1})$ satisfies $\lambda_{t+1} \approx \lambda'_{t+1}$ (note that y_t depends on λ'_{t+1}). This is done via a

bisection procedure iteratively shrinking an interval[1] that contains a successful choice of λ'_{t+1}. This bisection process is inefficient in the sense that every time we reach lines 14 and 17 (highlighted in red) all of the optimization progress made by the last oracle call is discarded.

In contrast, even though our algorithm queries $\mathcal{O}$ in the same way (with y_t computed based on a guess λ'_{t+1}), it makes use of the oracle output even if λ_{t+1} is very far λ'_{t+1}, thus never discarding progress made by the oracle. Instead of performing a bisection, we compare λ'_{t+1} and λ_{t+1} to guide our next guess λ'_{t+2}. When λ' overshoots λ, we decrease it by a factor α (line 11, highlighted in green) and set x_t and A_t as in Algorithm 0. When it undershoots, we multiply it by α (line 16). In this case, we require an additional key algorithmic modification which we call the *momentum damping mechanism*: we scale down the growth of the parameter A_{t+1} and replace the next iterate with a convex combination of x_t and $\tilde{x}_{t+1}$ to ensure that our overly optimistic guess for λ_{t+1} does not destabilize the algorithm.[2]

Different MS oracles attain different rates of convergence when accelerated via the MS framework. In the following definition, we distill the key property that determines this rate.

Definition 2 (Movement bound)**.** *For $s \geq 1$ and $c, \lambda > 0$, and $x, y \in \mathcal{X}$ we say that (x, y, λ) satisfy a (s, c)-*movement bound *if*

$$\|x - y\| \geq \begin{cases} (\lambda/c^s)^{1/(s-1)} & s < \infty \\ 1/c & s = \infty. \end{cases} \tag{3}$$

A $(1, c)$-movement bound simply means that $\lambda \leq c$.

In the next section, we will show how to build MS oracles that, given query y, output (x, λ) such that (x, y, λ) always satisfy a (s, c)-movement bound, for certain s and c depending on the oracle type and function structure (e.g., level of smoothness). With the necessary definitions in hand, we are ready to state our main result: the iteration (and MS oracle query) complexity of Algorithm 1.

Theorem 1. *Let $f : \mathcal{X} \to \mathbb{R}$ be convex and differentiable, and consider Algorithm 1 with parameters $\alpha > 1$, $\lambda' > 0$, and a σ-MS oracle (Definition 1) for f with $\sigma \in (0, 0.99)$. Let $s \geq 1$ and $c > 0$, and suppose that for all t such that $\lambda_t > \lambda'_t$ or $t = 1$, the iterates $(\tilde{x}_t, y_{t-1}, \lambda_t)$ satisfy a (s, c)-movement bound (Definition 2). Then, for any $x_\star \in \mathcal{X}$ and $\epsilon > 0$ and $C_{\alpha,s} \coloneqq O\left(\frac{\alpha^{\frac{s+1}{3s+1}}}{\min\{1,\alpha-1\}}\right)$, when*[3]

$$T \geq \begin{cases} C_{\alpha,s}\left(\frac{c^s\|x_0 - x_\star\|^{s+1}}{\epsilon}\right)^{\frac{2}{3s+1}} & s < \infty \\ C_{\alpha,s}(c\|x_0 - x_\star\|)^{\frac{2}{3}} \log \frac{\lambda_1\|x_0 - x_\star\|^2}{\epsilon} & s = \infty, \end{cases}$$

we have $f(x_T) - f(x_\star) \leq \epsilon$.

Proof sketch. The remainder of this section is an overview of the proof of Theorem 1, which we provide in full in Appendix C. To simplify the proof sketch, we treat α, c, and $1/(1 - \sigma)$ as $O(1)$, and focus on $s < \infty$. To highlight the novel aspects of the proof, let us first briefly recall the analysis of Algorithm 0 [27, 18, 6, 22, 10]. For every $t \leq T$ let

$$E_t \coloneqq f(x_t) - f(x_\star) \ , \ D_t \coloneqq \frac{1}{2}\|v_t - x_\star\|^2 \ \text{ and } \ M_t = \frac{1}{2}\|\tilde{x}_t - y_{t-1}\|^2.$$

The key facts about the standard MS iterations are

$$E_T \leq \frac{D_0}{A_T} \ , \ \sum_{t\in[T]} \lambda_t A_t M_t \leq O(D_0) \ \text{ and } \ \sqrt{A_T} \geq \Omega(1) \sum_{t\in[T]} \frac{1}{\sqrt{\lambda'_t}}. \tag{4}$$

The first fact implies that the optimality gap at iteration T is inversely proportional to A_T, while the latter two facts imply that A_T grows rapidly. More specifically, substituting the movement bound

[1] We have simplified the bisection routine and implicitly assumed an initial interval $[\lambda^\ell, \lambda^h]$ which always contains a valid solution. One can guarantee such an interval exists by selecting very small λ^ℓ and very large λ^h. Alternatively, it is possible to adaptively construct a valid initial interval via a bracketing procedure, as we do in the empirical comparison. In either case, the cost of the bisection is logarithmic in problem parameters.

[2] It is also possible to set $x_{t+1} = \operatorname{argmin}_{x\in\{\tilde{x}_{t+1}, x_t\}} f(x)$ instead of the convex combination in line 15 and maintain our theoretical guarantees.

[3] We use the big-O notation to hide constants including polynomials of any fixed $s < \infty$.

$M_t \geq \Omega\big((\lambda_t)^{2/(s-1)}\big)$ and $\lambda'_t \geq \Omega(\lambda_t)$ (thanks to the bisection) yields $\sum_{t\in[T]} \lambda_t'^{\frac{s+1}{s-1}} A_t = O(D_0)$. Combining this with the third fact in (4) and using the reverse Hölder inequality allows one to conclude that, for $k = \frac{s+1}{3s+1}$ and $k' = \frac{s-1}{3s+1}$, we have $A_T^k \geq \Omega(D_0^{-k'}) \sum_{t\in[T]} A_i^{k'}$, which, upon further algebraic manipulation, yields $A_T \geq \Omega(T^{(3s+1)/2} D_0^{-(s-1)/2})$ and hence gives the claimed convergence rate when plugging back to the first fact in (4).

So far for the standard MS analysis.Our *first challenge* is re-establishing the facts (4). The difficult case is $\lambda_t > \lambda'_t$, where the standard cancellation that occurs in the MS analysis may fail. This is where the momentum damping mechanism (lines 14 and 15 of our algorithm) comes into play, allowing us to show that (See Proposition 1 in the appendix)

$$E_T \leq \frac{D_0}{A_T} \ , \quad \sum_{t\in\mathcal{S}_T^{>}\cup\{1\}} \lambda'_t A_t M_t \leq O(D_0) \ \text{ and } \ \sqrt{A_T} \geq \Omega(1) \sum_{t\in\mathcal{S}_T^{\leq}} \frac{1}{\sqrt{\lambda'_t}}, \tag{5}$$

where $\mathcal{S}_T^{\geq} := \{t \in [T] \mid \lambda_t \leq \lambda'_t\}$ and $\mathcal{S}_T^{>}$, $\mathcal{S}_T^{\leq}$, $\mathcal{S}_T^{<}$ and $\mathcal{S}_T^{=}$ are analogously defined.

Comparing (4) and (5), the price of removing the bisection becomes evident: at iterations t where $\lambda_t \neq \lambda'_t$ only one of the terms forcing the growth of A_t receives a contribution. The *second challenge* of our proof is establishing a lower bound on $\sqrt{A_T}$ in terms of the $1/\sqrt{\lambda'_t}$ values for $t \in \mathcal{S}_T^{>} \cup \{1\}$, where the movement bound holds for M_t. This is where the multiplicative λ' update rule (lines 11 and 16 of the algorithm) comes into play: it allows us to "credit" the contribution of every "down iterate" (in $\mathcal{S}_T^{\leq}$) to an adjacent "up" iterate ($\mathcal{S}_T^{>} \cup \{1\}$) and furthermore argue that the contribution gets an exponential bonus based on the number of iterations between the two. Consequently, we are able to identify a set $\mathcal{Q}_T \subseteq \mathcal{S}_T^{>} \cup \{1\}$ of iterates, and a sequence $\{r_t\}$ such that (see Lemma 1) $\sqrt{A_T} \geq \Omega(1) \sum_{t\in\mathcal{Q}_T} \sqrt{\frac{\alpha^{r_t-1}}{\lambda'_t}}$ and $\sum_{t\in[T]} r_t = \frac{T-1}{2}$.

Repeating the reverse Hölder argument of prior work, we obtain the recursive bound $A_T^k \geq \Omega(D_0^{-k'}) \sum_{t\in\mathcal{Q}_T} A_t^{k'} \alpha^{k r_t} \geq \Omega(D_0^{-k'}) \sum_{t\in\mathcal{Q}_T} A_t^{k'} r_t$ with $k = \frac{s+1}{3s+1}$ and $k' = \frac{s-1}{3s+1}$ as before. The *final challenge* of our proof is to show that such recursion implies sufficient growth of A_t. This is where careful algebra comes into play; we show that $A_T \geq \Omega\Big(\big(\sum_{t\in[T]} r_t\big)^{(3s+1)/2} D_0^{-(s-1)/2}\Big)$ (see Lemmas 3 and 4) which establishes our result since $\sum_{t\in[T]} r_t = \frac{T-1}{2}$.

3 MS oracle implementations

In this section we describe several oracles that satisfy both Definition 1 (the MS condition) and Definition 2 (movement bounds) and may therefore be used by Algorithm 1. Section 3.1 briefly reviews oracles that have appeared in prior work, while Section 3.2 and Section 3.3 describe our new adaptive oracle implementations. We summarize the key oracle properties and resulting complexity bounds in Table 1.

3.1 Oracles from prior work

For this subsection we consider oracles of the form $(x, \lambda) = \mathcal{O}(y)$, omitting the second argument λ' since prior work does not leverage it to improve implementation efficiency.

Gradient descent step [e.g., 29]. As gentle start, consider the oracle $\mathcal{O}_{\mathsf{gd}}(y) = (y - \eta\nabla f(y), \frac{1}{\eta})$, i.e., an oracle that returns x by taking standard gradient step with size η and $\lambda = 1/\eta$. Obviously, the oracle always satisfies a $(1, \eta^{-1})$-movement bound. Moreover, if we assume that ∇f is L-Lipschitz, then $\|x - (y - \frac{1}{\lambda}\nabla f(x))\| = \eta\|\nabla f(x) - \nabla f(y)\| \leq \eta L\|x - y\|$. Therefore, with $\eta^{-1} \geq L/\sigma$ the oracle becomes a σ-MS oracle.

Taylor descent step [4, 30, 18, 6, 22, 34]. Generalizing both $\mathcal{O}_{\mathsf{gd}}$ and the cubic-regularized Newton step oracle $\mathcal{O}_{\mathsf{cr}}$, we define for every integer $p \geq 1$ and $\nu \in [0, 1]$ the oracle $\mathcal{O}_{p,\nu\text{-}\mathsf{reg}}$, that, for parameter C and input y returns $(x, \lambda) = \mathcal{O}_{p,\nu\text{-}\mathsf{reg}}(y)$ where

$$x = \operatorname*{argmin}_{x'\in\mathbb{R}^d} \left\{ \tilde{f}_p(x'; y) + \frac{M}{p!(p+\nu)}\|x' - y\|^{p+\nu} \right\} \ , \ \lambda = \frac{M}{p!}\|x - y\|^{p+\nu-2} \tag{6}$$

and $\tilde{f}_p(x; y) = \sum_{i=0}^{p} \frac{1}{i!}\nabla^i f(y)[(x - y)^{\otimes i}]$ is the Taylor expansion of f around y evaluated at x. Oracles $\mathcal{O}_{\mathsf{gd}}$ and $\mathcal{O}_{\mathsf{cr}}$ correspond to the special cases $\mathcal{O}_{1,1\text{-}\mathsf{reg}}$ (with $\eta = M^{-1}$) and $\mathcal{O}_{2,1\text{-}\mathsf{reg}}$,

Assumption	Oracle	Complexity with Algorithm 1	Lower bound
$\nabla^p f$ is $(1,\nu)$-Hölder *	$\mathcal{O}_{p,\nu\text{-reg}}$	$O\left(\epsilon^{-\frac{2}{3(p+\nu)-2}}\right)$ evals of $\nabla^p f$	$\Omega\left(\epsilon^{-\frac{2}{3(p+\nu)-2}}\right)$ [2, 20]
-	$\mathcal{O}_{r\text{-ball}}$	$O\left(r^{-\frac{2}{3}}\log\frac{1}{\epsilon}\right)$ oracle calls	$\Omega\left(r^{-\frac{2}{3}}\right)$ [10]
Stable Hessian	$\mathcal{O}_{r\text{-BaCoN}}$	$O\left(r^{-\frac{2}{3}}\log\frac{1}{\epsilon}\right)$ Hessian evals	-
$\nabla^2 f$ is $(1,\nu)$-Hölder †	$\mathcal{O}_{\text{aMSN}}$ (Alg. 2)	$O\left(\epsilon^{-\frac{2}{4+3\nu}}\right)$ Hessian evals	$\Omega\left(\epsilon^{-\frac{2}{4+3\nu}}\right)$ [2, 20]
		$O\left(\epsilon^{-\frac{2}{4+3\nu}}\right)+\widetilde{O}(1)$ linear systems	-
∇f is ℓ-Lipschitz and $\nabla^2 f$ is $(1,\nu)$-Hölder †	$\mathcal{O}_{\text{aMSN-fo}}$ (Alg. 3)	$O\left((\frac{\epsilon}{\ell})^{-\frac{1}{2}}\right)+\widetilde{O}(1)$ first-order evals	$\Omega\left((\frac{\epsilon}{\ell})^{-\frac{1}{2}}\right)$ [29]
		$O\left(\min\left\{(\frac{\epsilon}{\ell})^{-\frac{1}{2}},\epsilon^{-\frac{2}{4+3\nu}}\right\}\right)$ iterations	-

Table 1. Complexity bounds for finding x such that $f(x)-f(x_\star)\le\epsilon$ assuming $\|x-x_\star\|\le 1$, attained by MS oracles from the literature (top 3 rows, described in Section 3.1) and oracles we develop (bottom two rows). In all cases we improve on prior work by a logarithmic factor. *We require $p+\nu\ge 2$. †Our adaptive oracles do not require knowledge of continuity constants or even the Hölder order $\nu\in[0,1]$.

respectively. Clearly, by definition, the oracle always satisfies a $(p+\nu-1,(M/p!)^{1/(p+\nu-1)})$-movement bound. Moreover, it is easy to show that

$$\left\|x-\left(y-\frac{1}{\lambda}\nabla f(x)\right)\right\|=\frac{1}{\lambda}\|\nabla f(x)-\nabla\tilde{f}_p(x;y)\|=\frac{p!}{M}\frac{\|\nabla f(x)-\nabla\tilde{f}_p(x;y)\|}{\|x-y\|^{p+\nu-2}}.$$

For any $p\ge 1$ and $\nu\in[0,1]$ we say that

$$\nabla^p f \text{ is } (H,\nu)\text{-Hölder if for all } x,y \text{ we have } \|\nabla^p f(x)-\nabla^p f(y)\|_{\mathrm{op}}\le H\|x-y\|^\nu.$$

(An $(H,1)$-Hölder derivative is H-Lipschitz.) If $\nabla^p f$ is (H,ν)-Hölder, Taylor's theorem gives $\|\nabla f(x)-\nabla\tilde{f}_p(x;y)\|\le\frac{H}{p!}\|x-y\|^{p+\nu-1}$ [34, Lemma 2.5], and so $\|x-(y-\frac{1}{\lambda}\nabla f(x))\|\le\frac{H}{M}\|x-y\|$. Therefore, with $M\ge H/\sigma$ the oracle becomes a σ-MS oracle.

Exact ball optimization oracle [10]. For a given query y, consider the exact minimizer of f constrained to a ball of radius r around y, i.e., consider an oracle $\mathcal{O}_{r\text{-ball}}$ such that $(x,\lambda)=\mathcal{O}_{r\text{-ball}}(y)$ satisfy $x\in\operatorname{argmin}_{x':\|x'-y\|\le r}f(x')$ and $\lambda=\frac{\|\nabla f(x)\|}{\|x-y\|}$. One may easily verify that $x=y-\frac{1}{\lambda}\nabla f(x)$, and therefore the oracle is a 0-MS oracle. Moreover, when f is convex, we have either $\|x-y\|=r$ or x is a global minimizer of f, and so we may assume without loss of generality that the oracle satisfies an $(\infty,1/r)$ movement bound.

Ball-Constrained Newton (BaCoN) oracle [10]. Implementing $\mathcal{O}_{r\text{-ball}}$ is generally intractable. Nevertheless, Carmon et al. [10, Alg. 3] describe a method $\mathcal{O}_{r\text{-BaCoN}}$ based on solving a sequence of trust-region problems (ball-constrained Newton steps), which we call that, for functions that are $O(1)$-Hessian stable in a ball of radius r (or $1/r$-quasi-self-concordant) and have a finite condition number, outputs (x,λ) satisfying the $\frac{1}{2}$-MS oracle and an $(\infty,O(1/r))$ movement bound. Implementing $\mathcal{O}_{r\text{-BaCoN}}$ requires only a single Hessian evaluation and a number of linear system solutions that is polylogarithmic in problem parameters.

3.2 An adaptive Monteiro-Svaiter-Newton oracle

The oracle implementations above satisfy movement bounds by design and the MS condition (2) by assumption. For example, the cubic-regularized Newton step oracle $\mathcal{O}_{\text{cr}}$ only satisfies the MS condition when the regularization parameter M is sufficiently larger than the Lipschitz constant of $\nabla^2 f$. This of course means M must be carefully tuned to ensure good performance. Prior work attempt to dynamically adjust M in order to meet certain approximation conditions [12, 19, 26, 15], but to our knowledge they do not consider the MS condition. Moreover, even computing a single cubic-regularized Newton step entails searching for λ that satisfies $\lambda\|[\nabla^2 f(y)+\lambda I]^{-1}\nabla f(y)\|=\frac{M}{2}$. Therefore, a search over M is really a redundant double search over λ.

We propose a more direct and more adaptive MS oracle recipe: *search for the smallest λ for which the regularized Newton step $x=y-[\nabla^2 f(y)+\lambda I]^{-1}\nabla f(y)$ satisfies the MS condition* (2). This

Algorithm 2: $\mathcal{O}_{\text{aMSN}}$

Input: Query $y \in \mathbb{R}^d$, $\lambda' > 0$. Flag LAZY.
Parameters: MS factor $\sigma \in (0, 1)$.

1 **if** CHECKMS$(\lambda'; y, \sigma)$ **then**
2 **if** LAZY **then return** $y - [\nabla^2 f(y) + \lambda' I]^{-1} \nabla f(y)$, λ'
3 $\lambda_{\text{vld}} \leftarrow \lambda'$, $k \leftarrow 0$
 ▷ search for λ_{invld} that fails CHECKMS
4 **while** CHECKMS$(\lambda_{\text{vld}}/2^{2^k}; y, \sigma)$ **do**
5 $\lambda_{\text{vld}} \leftarrow \lambda_{\text{vld}}/2^{2^k}$
6 $k \leftarrow k + 1$
7 $k^\star \leftarrow k$, $\lambda_{\text{invld}} \leftarrow \lambda_{\text{vld}}/2^{2^{k^\star}}$
8 **else**
9 $\lambda_{\text{invld}} \leftarrow \lambda'$, $k \leftarrow 0$
 ▷ search for λ_{vld} that passes CHECKMS
10 **while not** CHECKMS$(\lambda_{\text{vld}} 2^{2^k}; y, \sigma)$ **do**
11 $\lambda_{\text{invld}} \leftarrow \lambda_{\text{invld}} 2^{2^k}$
12 $k \leftarrow k + 1$
13 $k^\star \leftarrow k$, $\lambda_{\text{vld}} \leftarrow \lambda_{\text{invld}} 2^{2^{k^\star}}$
 ▷ the loop below executes exactly $k^\star$ times
14 **while** $\lambda_{\text{invld}} < \lambda_{\text{vld}}/2$ **do**
15 $\lambda \leftarrow \sqrt{\lambda_{\text{invld}} \lambda_{\text{vld}}}$
16 **if** CHECKMS$(\lambda; y, \sigma)$ **then** $\lambda_{\text{vld}} \leftarrow \lambda$
17 **else** $\lambda_{\text{invld}} \leftarrow \lambda$
18 **return** $y - [\nabla^2 f(y) + \lambda_{\text{vld}} I]^{-1} \nabla f(y)$, λ_{vld}
19 **function** CHECKMS$(\lambda; y, \sigma)$
20 $x = y - [\nabla^2 f(y) + \lambda I]^{-1} \nabla f(y)$
21 **if** $\left\| x - (y - \frac{1}{\lambda} \nabla f(x)) \right\| \le \sigma \|x - y\|$ **then return** True
22 **else return** False

Algorithm 3: $\mathcal{O}_{\text{aMSN-fo}}$

Input: Query $y \in \mathbb{R}^d$, $\lambda' > 0$. Flag LAZY.
Parameters: MS factor $\sigma \in (0, 1)$.

1 $\lambda \leftarrow \lambda'$, FAILEDMSONCE $\leftarrow$ False
2 **Repeat**
3 $A \leftarrow \nabla^2 f(y) + \lambda I$ and $b \leftarrow -\nabla f(y)$
 ▷ Apply Conjugate Residuals [35, 17] until obtaining $\|Aw - b\| \le \frac{\lambda\sigma}{2} \|w\|$
4 $w_0 \leftarrow 0$
5 $p_0 \leftarrow r_0 \leftarrow Aw_0 - b$
6 $s_0 \leftarrow q_0 \leftarrow Ar_0$
7 $i \leftarrow 0$
8 **while** $\|r_i\| > \frac{\lambda\sigma}{2} \|w_i\|$ **do**
9 $w_{i+1} \leftarrow w_i - \frac{\langle r_i, s_i \rangle}{\|q_i\|^2} p_i$
10 $r_{i+1} \leftarrow r_i - \frac{\langle r_i, s_i \rangle}{\|q_i\|^2} q_i$
11 $s_{i+1} \leftarrow A r_{i+1}$
12 $p_{i+1} \leftarrow \frac{\langle r_{i+1}, s_{i+1} \rangle}{\langle r_i, s_i \rangle} p_i + r_{i+1}$
13 $q_{i+1} \leftarrow \frac{\langle r_{i+1}, s_{i+1} \rangle}{\langle r_i, s_i \rangle} q_i + s_{i+1}$
14 $i \leftarrow i + 1$
 ▷ Check MS condition
15 $x \leftarrow y + w_i$
16 **if** $\left\| x - (y - \frac{1}{\lambda} \nabla f(x)) \right\| \le \sigma \|x - y\|$ **then**
17 **if** LAZY *or* FAILEDMSONCE **then**
18 **return** x, λ
 ▷ if not LAZY and λ' meets MS condition
19 **else** $\lambda \leftarrow \lambda/2$
20 **else**
21 FAILEDMSONCE $\leftarrow$ True
22 $\lambda \leftarrow 2\lambda$

yields valid MS oracle by construction, independently of any assumption. Moreover, it is simple to argue that when $\nabla^2 f$ is (H, ν)-Hölder continuous for some $\nu \in [0, 1]$, such an oracle's output would satisfy the same movement bound guaranteed for the Taylor descent step with the best choice of regularization parameter power and coefficient (see Appendix D.2)—even though our recipe requires neither of these parameters!

Exactly fulfilling our recipe, i.e., finding the ideal minimal $\lambda^\star$ that satisfies the MS condition, is difficult. Fortunately, it suffices to find a value λ such the corresponding regularized Newton step satisfies the MS condition, while the step corresponding to $\lambda/2$ does not. Algorithm 2 finds precisely such a λ. If the input λ' is invalid (i.e., its corresponding regularized Newton step does not satisfy the MS condition and CHECKMS$(\lambda'; y, \sigma)$ evaluates to False), we set $\lambda_{\text{invld}} \leftarrow \lambda'$ and test a double-exponentially increasing series of λ's, until reaching a valid λ_{invld} (line 10). If λ' is valid and the LAZY flag is set, we return it immediately. Otherwise, we set $\lambda_{\text{vld}} = \lambda'$ and decrease it in a double-exponential rate until finding an invalid λ_{invld} (line 10). In either case, we obtain (invalid,valid) pair $(\lambda_{\text{invld}}, \lambda_{\text{vld}})$ such that $\lambda_{\text{vld}}/\lambda_{\text{invld}} = 2^{2^{k^\star}}$ at the cost of $2 + k^\star$ linear system solutions. We then perform precisely $k^\star$ log-scale bisection steps in order to shrink $\lambda_{\text{vld}}/\lambda_{\text{invld}}$ down to 2 while maintaining the invariant that λ_{vld} is valid and λ_{invld} is invalid (line 14).

The following theorem bounds the complexity of Algorithm 2 in terms of linear-system solution number, and establishes a movement bound for its output assuming that $\nabla^2 f$ is locally Hölder around the query point. We defer the proof of the theorem and its following corollary to Appendix D.3.

Theorem 2. *Algorithm 2 with parameter σ is a σ-MS oracle $\mathcal{O}_{\text{aMSN}}$. For $y \in \mathbb{R}^d$ computing $(x, \lambda) = \mathcal{O}_{\text{aMSN}}(y)$ requires at most $2 + 2k^\star$ linear systems solutions, where $k^\star \le \log_2\left(1 + \left|\log_2 \frac{\lambda}{\lambda'}\right|\right)$. If*

LAZY *is False or* $\lambda > \lambda'$*, and if* $\nabla^2 f$ *is* (H, ν)*-Hölder in a ball of radius* $2\|x - y\|$ *around* y*, then* (x, y, λ) *satisfy a* $\big(1 + \nu, (2H)^{1/(1+\nu)}/\sigma\big)$*-movement bound.*

To understand the LAZY option of Algorithm 2, note that when λ' is valid we will necessarily output $\lambda \le \lambda'$. In such case Theorem 1 does not require a movement bound (except for the first iteration). Therefore, we might as well save on computation and return λ'. The following corollary gives the overall complexity bound for the combination of Algorithm 1 and $\mathcal{O}_{\mathsf{aMSN}}$, leveraging "lazy" oracle calls to show that the number of linear system solves per iteration is essentially constant.

Corollary 3. *Consider Algorithm 1 with initial point* x_0*, parameters* $\alpha = (1.1, O(1))$ *and* λ'_0*, and* σ*-MS oracle* $\mathcal{O}_{\mathsf{aMSN}}$ *(with* LAZY $=$ *True in all but the first iteration) with* $\sigma \in (0.01, 0.99)$*. For any* $H, \epsilon > 0$*,* $\nu \in [0, 1]$ *and any* $x_\star \in \mathbb{R}^d$*, if* f *is convex with* (H, ν)*-Hölder Hessian, the algorithm produces an iterate* x_T *such that* $f(x_T) \le f(x_\star) + \epsilon$ *using* $T = O\Big(\big(H\|x_0 - x_\star\|^{2+\nu}/\epsilon\big)^{2/(4+3\nu)}\Big)$ *Hessian evaluations and at most* $O\Big(T + \log\log\max\Big\{\frac{HR^\nu}{\lambda'_0}, \frac{\lambda'_0 R^2}{\epsilon}\Big\}\Big)$ *linear system solutions, where* R *is the distance between* x_0 *and* $\operatorname{argmin}_{x'} f(x')$*.*

3.3 First-order implementation via MinRes/Conjugate Residuals

We now present a first-order implementation of our adaptive oracle, $\mathcal{O}_{\mathsf{aMSN\text{-}fo}}$ (Algorithm 3), which replaces exact linear system solutions with approximations obtained via Hessian-vector products and the MinRes/Conjugate Residuals method [35, 17]. To approximate $-[\nabla^2 f(y) + \lambda I]^{-1}\nabla f(y)$, the methods generates a sequence of iterates $\{w_i\}$ such that the ith residual $r_i = [\nabla^2 f(y) + \lambda I]w_i + \nabla f(y)$ has the minimal norm in the span of observed gradients. A key component of our algorithm is the simple and tuning-free residual condition we employ for deciding when to stop the MinRes iterations (line 8). We show that once this stopping condition is met the resulting point $x = y - w_i$ is roughly as capable of satisfying the MS condition as the exact linear system solution, which in turns allows us to deduce movement bounds similar to those of Algorithm 2. Moreover, by using the method for minimizing residuals, we guarantee that the stopping condition is met fairly quickly, requiring a number of first-order operations proportional to $1/\sqrt{\lambda}$. Using a standard doubling scheme, we may relate the algorithm's overall complexity to the lowest λ it queries, obtaining the following guarantees. See proofs in Appendix D.4.

Theorem 4. *Algorithm 3 with parameter* σ *is a* σ*-MS oracle* $\mathcal{O}_{\mathsf{aMSN\text{-}fo}}$*. For* $y \in \mathbb{R}^d$ *computing* $(x, \lambda) = \mathcal{O}_{\mathsf{aMSN\text{-}fo}}(y)$ *requires at most* $O\Big(\sqrt{1 + \frac{\|\nabla^2 f(y)\|_{\mathrm{op}}}{\sigma \min\{\lambda, \lambda'\}}}\Big)$ *Hessian-vector product and* $O(\big|\log \frac{\lambda}{\lambda'}\big|)$ *gradient computations. If* LAZY *is False or* $\lambda > \lambda'$*, and if* $\nabla^2 f$ *is* (H, ν)*-Hölder, then* (x, y, λ) *satisfy a* $\big(1 + \nu, (6H)^{1/(1+\nu)}/\sigma\big)$*-movement bound.*

Corollary 5. *Consider Algorithm 1 with initial point* x_0*, parameters* $\alpha = (1.1, O(1))$ *and* λ'_0*, and* σ*-MS oracle* $\mathcal{O}_{\mathsf{aMSN\text{-}fo}}$ *with* LAZY *set in all but the first iteration and* $\sigma \in (0.01, 0.99)$*. For any* $L, H, \epsilon > 0$*,* $\nu \in [0, 1]$ *and any* $x_\star \in \mathbb{R}^d$*, if* f *is convex with* (H, ν)*-Hölder Hessian and* L*-Lipschitz gradient, the algorithm produces an iterate* x_T *such that* $f(x_T) \le f(x_\star) + \epsilon$ *within* $T = O\Big(\Big(\frac{H\|x_0 - x_\star\|^{2+\nu}}{\epsilon}\Big)^{2/(4+3\nu)}\Big)$ *iterations and at most* $O\Big(\Big(\frac{L\|x_0 - x_\star\|^2}{\epsilon}\Big)^{1/2} + \sqrt{\frac{L}{\lambda'_0}} + \log\frac{\lambda'_0}{L}\Big)$ *gradient and Hessian-vector product evaluations.*

Note that the L-Lipschitz gradient assumption is equivalent to an $(H, 0)$-Hölder Hessian assumption, giving the iteration complexity bound we state in Table 1. Moreover, by choosing a large λ'_0 (say 10^6) we may guarantee that only the logarithmic term is added to the optimal first-order evaluation complexity.

4 Experiments

We conduct three sets of experiments. First, we consider $\mathcal{O}_{\mathsf{cr}}$ with a fixed parameter M and compare previous acceleration schemes to Algorithm 1. Second, we combine Algorithm 1 with our adaptive $\mathcal{O}_{\mathsf{aMSN}}$ and test it against previous adaptive accelerated (second-order) methods and Newton's method. Finally, we compare Algorithm 1 with our first-order adaptive oracle $\mathcal{O}_{\mathsf{aMSN\text{-}fo}}$ to other first-order methods. We provide full implementation details in Appendix E.1. Figure 1 summarizes our results for logistic regression on the 'a9a' dataset [13]; see Appendix E.2 for similar results on three

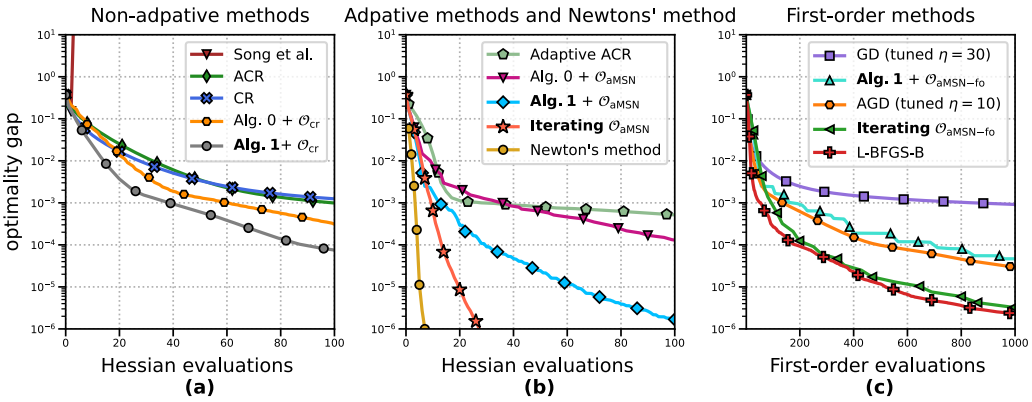

Figure 1. Empirical results on logistic regression with the "a9a" dataset. See Section 4 for description, and Appendix E.2 for additional datasets. Boldface legend entries denote methods we contribute.

additional datasets. These experiments were conducted with *no tuning* of Algorithm 1: the parameters σ and α were simply set to $\frac{1}{2}$ and 2, respectively. An additional experiment, reported in Appendix E.3, shows that the algorithm is indeed insensitive to that choice.

Non-adaptive methods. In this set of experiments we use the non-adaptive oracle $\mathcal{O}_{\mathsf{cr}}$ (1), and take M to be $\|\frac{1}{n}\sum_{i=1}^{n}\phi_i\phi_i^T\|_{\mathrm{op}}\max_{i\in[n]}\|\phi_i\|$, where $\phi_1,\ldots,\phi_n$ are the feature vectors and n is the dataset size; this is an upper bound on the Lipschitz constant of the logistic regression Hessian [see, e.g., 34]. Fixing the MS oracle allows for a controlled comparison of different acceleration schemes: Figure 1(a) shows that standard MS acceleration with a carefully-implemented bisection outperforms standard cubic regularization (CR) and its accelerated counterpart (ACR) [28, Alg. 4.8], and removing the bisection via Algorithm 1 yields the best results. We also implemented the heuristic suggested by Song et al. [34], where instead of a bisection in Algorithm 0 we select a sequence λ_t' such that $A_t = \frac{1}{2H\|x_0-x_\star\|}(t/3)^{7/2}$. This method diverges when used with our conservative choice of M. In Appendix E.4 we tune the M parameter for each method separately, finding that the optimal M is often near 0, so that $\mathcal{O}_{\mathsf{cr}}$ is nearly a Newton step (and not a valid MS oracle). When M is tuned the heuristic [34] also converges.

Adaptive methods and Newton's method. We compare the following adaptive accelerated second-order methods (which do not require an estimate of the Hessian Lipschitz constant): Adaptive ACR [20, Algorithm 4] (which adaptively sets M in $\mathcal{O}_{\mathsf{cr}}$), standard MS acceleration (Algorithm 0) with $\mathcal{O}_{\mathsf{aMSN}}$ (Algorithm 2, with $\textsc{lazy} = \text{False}$) and Algorithm 1 with $\mathcal{O}_{\mathsf{aMSN}}$ (with $\textsc{lazy} = \text{True}$ in all but the first iteration). Figure 1(b) shows that the latter converges significantly faster than the other adaptive acceleration schemes. However, the classical "unaccelerated" Newton iteration $x_{t+1} = -(\nabla^2 f(x_t))^{-1}\nabla f(x_t)$ strongly outperforms all "accelerated" methods, indicating that momentum mechanisms might actually be slowing down convergence in logistic regression problems. To test this, we consider the following simple iteration of (the non-lazy variant) of our oracle: $x_{t+1}, \lambda_{t+1} = \mathcal{O}_{\mathsf{aMSN}}(x_t;\lambda_t)$; it significantly improves over Algorithm 1.

These results beg the question: is momentum ever useful for second-order methods? In Appendix E.5 we test different schemes on the lower bound construction [2, 20]. We find momentum is helpful for $\mathcal{O}_{\mathsf{cr}}$, but not for the adaptive oracle $\mathcal{O}_{\mathsf{aMSN}}$. What makes Newton's method perform so well on logistic regression, and whether simply iterating $\mathcal{O}_{\mathsf{aMSN}}$ is worst-case optimal, are important questions for future work.

First-order methods. We compare our first-order adaptive $\mathcal{O}_{\mathsf{aMSN\text{-}fo}}$ (Algorithm 3) to the following baselines: gradient descent and accelerated gradient descent [32] with a tuned step size η, and L-BFGS-B from SciPy [8, 37, 36]. In light of the above comparison with Newton's method, we also test the following simple iteration of (the lazy variant) of our oracle: $x_{t+1}, \lambda_{t+1} = \mathcal{O}_{\mathsf{aMSN\text{-}fo}}(x_t;\lambda_t/2)$. Figure 1(c) shows that forgoing (second-order) momentum is better for the first-order oracle, too: Algorithm 1 performs comparably to tuned AGD (without tuning a single parameter), and the equally adaptive $\mathcal{O}_{\mathsf{aMSN\text{-}fo}}$ iteration performs comparably to with L-BFGS-B.

References

[1] M. M. Alves. Variants of the A-HPE and large-step A-HPE algorithms for strongly convex problems with applications to accelerated high-order tensor methods. *arXiv:2102.02045*, 2021.

[2] Y. Arjevani, O. Shamir, and R. Shiff. Oracle complexity of second-order methods for smooth convex optimization. *Mathematical Programming*, 178(1-2):327–360, 2019.

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

[4] M. Baes. Estimate sequence methods: extensions and approximations. *Optimization Online*, 2009.

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

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

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

[8] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

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

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

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

[12] C. Cartis, N. I. M. Gould, and P. L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. part II: worst-case function- and derivative-evaluation complexity. *Math. Program.*, 130(2):295–319, 2011.

[13] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[14] N. Doikov and Y. Nesterov. Inexact tensor methods with dynamic accuracies. In *International Conference on Machine Learning*, 2020.

[15] N. Doikov and Y. Nesterov. Gradient regularization of Newton method with Bregman distances. *arXiv:2112.02952*, 2021.

[16] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[17] D. C.-L. Fong and M. Saunders. CG versus MINRES: An empirical comparison. *Sultan Qaboos University Journal for Science (SQUJS)*, 17(1):44–62, 2012.

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

[19] N. I. M. Gould, M. Porcelli, and P. L. Toint. Updating the regularization parameter in the adaptive cubic regularization algorithm. *Comput. Optim. Appl.*, 53(1):1–22, 2012.

[20] G. N. Grapiglia and Y. Nesterov. Tensor methods for minimizing convex functions with Hölder continuous higher-order derivatives. *SIAM Journal on Optimization*, 30(4):2750–2779, 2020.

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

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

[23] B. Jiang, T. Lin, and S. Zhang. A unified adaptive tensor approximation scheme to accelerate composite convex optimization. *SIAM Journal on Optimization*, 30(4):2897–2926, 2020.

[24] J. Lee, C. Park, and E. Ryu. A geometric structure of acceleration and its role in making gradients small fast. *Advances in Neural Information Processing Systems*, 2021.

[25] T. Lin, M. Jordan, et al. Perseus: A simple high-order regularization method for variational inequalities. *arXiv preprint arXiv:2205.03202*, 2022.

[26] K. Mishchenko. Regularized Newton method with global $o(1/k^2)$ convergence. *arXiv:2112.02089*, 2021.

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

[28] Y. Nesterov. Accelerating the cubic regularization of Newton's method on convex problems. *Mathematical Programming*, 112(1):159–181, 2008.

[29] Y. Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2018.

[30] Y. Nesterov. Implementable tensor methods in unconstrained convex optimization. *Mathematical Programming*, 186(1):157–183, 2021.

[31] Y. Nesterov and B. T. Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[32] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

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

[34] C. Song, Y. Jiang, and Y. Ma. Unified acceleration of high-order algorithms under Hölder continuity and uniform convexity. *SIAM Journal on Optimization*, 2021.

[35] E. Stiefel. Relaxationsmethoden bester strategie zur lösung linearer gleichungssysteme. *Commentarii Mathematici Helvetici*, 29(1):157–179, 1955.

[36] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[37] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] see supplementary material
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] in the supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] full implementation details in Appendix E.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We only experimented with deterministic algorithms
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We only ran small scale experiments on CPU
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] see Appendix E.1
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] in the supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]