A SIMPLE NEARLY-OPTIMAL RESTART SCHEME FOR SPEEDING-UP FIRST ORDER METHODS

JAMES RENEGAR AND BENJAMIN GRIMMER

Abstract. We present a simple scheme for restarting first-order methods for convex optimization problems. Restarts are made based only on achieving specified decreases in objective values, the specified amounts being the same for all optimization problems. Unlike existing restart schemes, the scheme makes no attempt to learn parameter values characterizing the structure of an optimization problem, nor does it require any special information that would not be available in practice (unless the first-order method chosen to be employed in the scheme itself requires special information). As immediate corollaries to the main theorems, we show that when some well-known first-order methods are employed in the scheme, the resulting complexity bounds are nearly optimal for particular – yet quite general – classes of problems.

1. Introduction

A restart scheme is a procedure that stops an algorithm when a given criterion is satisfied, and then restarts the algorithm with new input. For certain classes of convex optimization problems, restarting can improve the convergence rate of first-order methods, an understanding that goes back decades to work of Nemirovski and Nesterov [17]. Their focus, however, was on the abstract setting in which somehow known are various scalars from which can be deduced nearly-ideal times to make restarts for the particular optimization problem being solved.

In recent years, the notion of "adaptivity" has come to play a foremost role in research on first-order methods. Here a scheme learns – when given an optimization problem to solve – good times at which to restart the first-order method by systematically trying various values for the methods's parameters and observing consequences over a number of iterations. Such a scheme, in the literature to date, is particular to a class of optimization problems, and typically is particular to the first-order method to be restarted. No simple and easily-implementable set of principles has been developed which applies to arrays of first-order methods and arrays of problem classes. We eliminate this shortcoming of the literature by introducing such a set of principles.

We present a simple restart scheme utilizing multiple instances of a general first-order method. The instances are run in parallel (or sequentially if preferred) and occasionally communicate their improvements in objective value to one another, possibly triggering restarts. In particular, a restart is triggered only by improvement in objective value, the trigger threshold being independent of problem classes and first-order methods. Nevertheless, for a variety of prominent first-order methods and important problem classes, we show nearly-optimal complexity is achieved.

We begin by outlining the key ideas, and providing an overview of ramifications for theory.

1.1. The setting.

We consider optimization problems

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in Q \,, \end{array} \tag{1.1}$$

Research supported in part by NSF grants CCF-1552518 and DMS-1812904, and by the NSF Graduate Research Fellowship under grant DGE-1650441. A portion of the initial development was done while the authors were visiting the Simons Institute for the Theory of Computing, and was partially supported by the DIMACS/Simons Collaboration on Bridging Continuous and Discrete Optimization through NSF grant CCF-1740425.

We are indebted to reviewers, whose comments and suggestions led to a wide range of substantial improvements in the presentation.

where f is a convex function, and Q is a closed convex set contained in the (effective) domain of f (i.e., where f is finite). We assume the set of optimal solutions, X^*, is nonempty. Let f^* denote the optimal value.

Let `fom` denote a first-order method capable of solving some class of convex optimization problems of the form (1.1), in the sense that for any problem in the class, when given an initial feasible point $x_0 \in Q$ and accuracy $\epsilon > 0$, the method is guaranteed to generate a feasible iterate x_k satisfying $f(x_k) \leq f^* + \epsilon$, an "$\epsilon$-optimal solution."

For example, consider the class of problems for which f is Lipschitz on an open neighborhood of Q, that is, there exists $M > 0$ such that $|f(x) - f(y)| \leq M\|x - y\|$ for all x, y in the neighborhood (where, throughout, $\|\ \ \|$ is the Euclidean norm). An appropriate algorithm is the projected subgradient method, `fom` = `subgrad`, defined iteratively by

$$x_{k+1} = P_Q\big(x_k - \tfrac{\epsilon}{\|g_k\|^2}\, g_k\big)\ , \tag{1.2}$$

where g_k is any subgradient[1] of f at x_k, and where P_Q denotes orthogonal projection onto Q, i.e., $P_Q(x)$ is the point in Q which is closest to x. Here the algorithm depends on ϵ but not on the Lipschitz constant, M.

As another example, consider the class of problems for which f is L-smooth on an open neighborhood of Q, that is, f is differentiable and satisfies $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for all x, y in the neighborhood. These are the problems for which accelerated methods were initially developed, beginning with the pioneering work of Nesterov [21]. While our framework is immediately applicable to many accelerated methods, for concreteness we refer to a particular instance of the popular algorithm FISTA [1], a direct descendent of Nesterov's original accelerated method. We signify this instance as `accel` and record it below[2]. We note that `accel` depends on L but not on ϵ. (Later we discuss "backtracking," where only an estimate $\bar{L}$ of L is required, with an algorithm automatically updating $\bar{L}$ during iterations, increasing the overall complexity by only an additive term depending on the quantity $|\log(\bar{L}/L)|$.)

The basic information required by `subgrad` and `accel` are (sub)gradients. Both methods make one call to the "oracle" per iteration, to obtain that information for the current iterate.

Our theorems are general, applying to any first-order method for which there exists an upper bound on the total number of oracle calls sufficient to compute an ϵ-optimal solution, where the upper bound is a function of ϵ and the distance from the initial iterate x_0 to the set of optimal solutions. Moreover, the theorems allow any oracle.

Thus, for example, besides immediately yielding corollaries specific to `subgrad` and `accel`, our theorems readily yield corollaries for accelerated proximal-gradient methods, which are relevant when $Q = \mathbb{R}^n$, $f = f_1 + f_2$ with f_1 being convex and smooth, and f_2 being convex and "nice" (e.g., a regularizer). Here the oracle provides an optimal solution to a particular subproblem involving a gradient of f_1 and the "proximal operator" for f_2 (c.f., [20]). For example, our results apply to `fista`, the general version of `accel`, as will be clear to readers familiar with proximal methods.[3]

---

[1] Recall that a vector g is a subgradient of f at x if for all y we have $f(y) \geq f(x) + \langle g, y - x\rangle$. If f is differentiable at x, then g is precisely the gradient.

[2] `accel`:
(0) Input: $x_0 \in Q$. Initialize: $y_0 = x_0$, $\theta_0 = 1$, $k = 0$.
(1) $x_{k+1} = P_Q(y_k - \frac{1}{L}\nabla f(y_k))$
(2) $\theta_{k+1} = \frac{1}{2}(1 + \sqrt{1 + 4\theta_k^2})$
(3) $y_{k+1} = x_{k+1} + \frac{\theta_k - 1}{\theta_{k+1}}(x_{k+1} - x_k)$

[3] We avoid digressing to discuss the principles underlying proximal methods. For readers unfamiliar with the topic, the paper introducing FISTA ([1]) is a good place to start.

1.2. **Brief motivation for the scheme.** To motivate our restart scheme, consider the subgradient method, $\texttt{fom} = \texttt{subgrad}$. If ϵ is small and $f(x_0) - f^*$ is large, the subgradient method tends to be extremely slow in reaching an ϵ-optimal solution, in part due to x_0 being far from X^* and the step sizes $\|x_{k+1} - x_k\|$ being proportional to ϵ. In this situation it would be preferable to begin by relying on a constant $2^N\epsilon$ in place of ϵ, where N is a large integer, continuing to make iterations until a $(2^N\epsilon)$-optimal solution has been reached, then rely on the value $2^{N-1}\epsilon$ instead, iterating until a $(2^{N-1}\epsilon)$-optimal solution has been reached, and so on, until finally a $(2^0\epsilon)$-optimal solution has been reached, at which time the procedure would terminate.

The catch, of course, is that generally there is no efficient way to determine whether a $(2^n\epsilon)$-optimal solution has been reached, no efficient termination criterion. To get around this conundrum, our scheme relies on the value $f(x_0) - f(x_k)$, that is, relies on the decrease in objective *that has already been achieved*, and does not attempt to measure how much objective decrease remains to be made.

1.3. **The key ideas.** For $n = -1, 0, 1, \ldots, N$, let $\texttt{fom}_n$ be a version of $\texttt{fom}$ which is known to compute an iterate which is a $(2^n\epsilon)$-optimal solution. In particular, when $\texttt{fom} = \texttt{subgrad}$, let $\texttt{fom}_n$ be the subgradient method with $2^n\epsilon$ in place of ϵ, and when $\texttt{fom} = \texttt{accel}$, simply let $\texttt{fom}_n = \texttt{accel}$ (the accelerated method is independent of ϵ).

For definiteness, assume the desired accuracy ϵ satisfies $0 < \epsilon \ll 1$, and choose N so that $2^N \approx 1/\epsilon$. Thus, $N \approx \log_2(1/\epsilon)$.

The algorithms $\texttt{fom}_n$ are run in parallel (or sequentially if preferred), with $\texttt{fom}_n$ being initiated at a point $x_{n,0}$. Each algorithm is assigned a task. The task of $\texttt{fom}_n$ is to make progress $2^n\epsilon$, that is, to obtain a feasible point x satisfying $f(x) \le f(x_{n,0}) - 2^n\epsilon$. If the task is accomplished, $\texttt{fom}_n$ notifies $\texttt{fom}_{n-1}$ of the point x (assuming $n \neq -1$), to allow for the possibility that x also serves to accomplish the task of $\texttt{fom}_{n-1}$.

If $n \neq N$, then in addition to $\texttt{fom}_{n-1}$ being notified of x, a restart for $\texttt{fom}_n$ occurs at x, that is, x becomes the (new) initial point $x_{n,0}$ for $\texttt{fom}_n$. Again the assigned task of $\texttt{fom}_n$ is to obtain a point x satisfying $f(x) \le f(x_{n,0}) - 2^n\epsilon$. (The algorithm $\texttt{fom}_N$ never restarts, for reasons particular to obtaining complexity bounds of a particular form.)

> Thus, in a nutshell, the main ideas for the scheme are simply that (1) $\texttt{fom}_n$ continues iterating either until reaching an iterate $x = x_{n,k}$ satisfying the inequality $f(x) \le f(x_{n,0}) - 2^n\epsilon$ (where $x_{n,0}$ is the most recent restart point for $\texttt{fom}_n$), or until receiving a point x from $\texttt{fom}_{n+1}$ which satisfies the inequality, and (2) when $\texttt{fom}_n$ obtains such a point x in either way, then $\texttt{fom}_n$ restarts[4] at x (i.e., $x_{n,0} \leftarrow x$), and notifies $\texttt{fom}_{n-1}$ of x.

There are a variety of ways the simple ideas can be implemented, including sequentially, or in parallel, in which case either synchronously or asynchronously. In order to provide rigorous proofs of the scheme's performance, a number of secondary details have to be specified for the scheme, although these details could be chosen in multiple ways and still similar theorems would result. Our choices for the details of the scheme when implemented sequentially, or in parallel with iterations sychronized, are given in §4. Details are given for a parallel asynchronous implementation in §7. The key ingredients to all settings, however, are precisely the simple ideas posited above.

[4] In the case of $\texttt{subgrad}_n$, restarting at x simply means computing a subgradient step at x, i.e., $x_+ = x - \frac{2^n\epsilon}{\|g_k\|^2} g_k$. For $\texttt{accel}_n$, restarting is slightly more involved, in part due to two sequences of points being computed, $\{x_{n,k}\}$ and $\{y_{n,k}\}$. The primary sequence is $\{x_{n,k}\}$. The method $\texttt{accel}_n$ restarts at a point x from a primary sequence (either from $\texttt{fom}_n$'s own primary sequence, or a point sent to $\texttt{fom}_n$ from the primary sequence for $\texttt{fom}_{n+1}$). In restarting, $\texttt{accel}_n$ begins by "reinitializing," setting $x_{n,0} = x$, $y_{n,0} = x$, $\theta_0 = 1$, $k = 0$, then begins iterating.

The simplicity of the underlying ideas gives hope for practice. In this regard we provide, in §9, numerical results displaying notable performance improvements when the scheme is employed with either `subgrad` or `accel`.

1.4. **Consequences for theory.** To theoretically demonstrate effectiveness of the scheme, we consider objective functions f possessing "Hölderian growth," in that there exist constants $\mu > 0$ and $d \geq 1$ for which

$$x \in Q \text{ and } f(x) \leq f(\mathbf{x_0}) \quad \Rightarrow \quad f(x) - f^* \geq \mu \operatorname{dist}(x, X^*)^d \,,$$

where $\mathbf{x_0}$ is a common point at which each $\texttt{fom}_n$ is first started, and where $\operatorname{dist}(x, X^*) := \min\{\|x - x^*\| \mid x^* \in X^*\}$. (Elsewhere the property is referred to as a Hölderian error bound [33], as sharpness [30], as the Łojasiewicz property [12], and the (generalized) Łojasiewicz inequality [3].) We call d the "degree of growth." The parameters μ and d are not assumed to be known to our scheme, nor is any attempt made to learn approximations to their values. Indeed, our theorems do not assume f possesses Hölderian growth.

Łojasiewicz [15, 14] showed Hölderian growth holds for generic analytic and subanalytic functions, a result which was generalized to nonsmooth subanalytic convex functions by Bolte, Daniilidis and Lewis [3]. It is easily seen that strongly convex functions have Hölderian growth with $d = 2$, but so do many other (smooth and nonsmooth) convex functions, such as the objective in the Lagrangian form of Lasso regression. Another example to keep in mind is when f is a piecewise-linear convex function and Q is polyhedral, then Hölderian growth holds with $d = 1$.

As a simple consequence (Corollary 5.2) to the theorem for our synchronous scheme (Theorem 5.1), we show that if f is M-Lipschitz on an open neighborhood of Q, and `subgrad` is employed in the scheme, the time sufficient to compute an ϵ-optimal solution (feasible x satisfying $f(x) - f^* \leq \epsilon < 1$) is at most on the order of $(M/\mu)^2 \log(1/\epsilon)$ if $d = 1$, and at most order of $M^2/(\mu^{2/d}\epsilon^{2(1-1/d)})$ if $d > 1$. These time bounds apply when the scheme is run in parallel, relying on $2 + \lceil \log_2(1/\epsilon) \rceil$ copies of the subgradient method.

As there are $O(\log(1/\epsilon))$ copies of the subgradient method being run simultaneously, the total amount of work (total number of oracle calls) is obtained by multiplying the above time bounds by $O(\log(1/\epsilon))$.

For general convex optimization problems with Lipschitz objective functions, this is the first algorithm which requires only that subgradients are computable (in particular, does not require any information about f as input, such as the optimal value f^*), *and* which has the property that if f happens to possess linear growth (i.e., happens to be Hölderian with $d = 1$), an ϵ-optimal solution is computed within a total number of subgradient evaluations depending only logarithmically on $1/\epsilon$. (We review the related literature momentarily.)

Another simple consequence of the theorem pertains to the setting where f is L-smooth on an open neighborhood of Q. (For smooth functions, the growth degree necessarily satisfies $d \geq 2$.) Here we show that if `accel` is employed in the synchronous scheme, the time required to compute an ϵ-optimal solution is at most on the order of $\sqrt{L/\mu}\log(1/\epsilon)$ if $d = 2$, and at most on the order of $L^{\frac{1}{2}}/(\mu^{\frac{1}{d}}\epsilon^{\frac{1}{2}-\frac{1}{d}})$ if $d > 2$. For the total amount of work, simply multiply by $O(\log(1/\epsilon))$. (We also observe the same bounds apply to `fista` in the more general setting of proximal methods.)

In the case $d = 2$, our bound on the total number of gradient evaluations (oracle calls) is off by a factor of only $O(\log(1/\epsilon))$ from the well-known lower bound for strongly-convex smooth functions ([18]), a small subset of the functions having Hölderian growth with $d = 2$. For $d > 2$, the bounds also are off by a factor of only $O(\log(1/\epsilon))$, according to the lower bounds stated

by Nemirovski and Nesterov [17, page 26] (for which we know of no proofs recorded in the literature).

A third straightforward corollary to the theorem for our synchronous parallel scheme applies to an algorithm which, following Nesterov[19], makes use of an accelerated method in solving a "smoothing" of f, assuming f to be Lipschitz. We denote the algorithm by `smooth`. We avoid digressing to introduce the necessary notation until §2, but here we mention a consequence when employing `smooth` in our synchronous scheme, the goal being to minimize a piecewise-linear convex function

$$f(x) = \max\{a_i^T x + b_i \mid i = 1, \dots, m\}.$$

Note that $M = \max_i \|a_i\|$ is a Lipschitz constant for f. This function has Hölderian growth with $d = 1$ (i.e., linear growth), for some $\mu > 0$. In the context of M-Lipschitz convex functions with linear growth, the ratio M/μ is sometimes called the "condition number," and is easily shown to have value at least 1.

From our previous discussion, when employing `subgrad` in our synchronous scheme to minimize f, the time bound is of order $(M/\mu)^2 \log(1/\epsilon)$, quadratic in the condition number. By contrast, employing `smooth` results in a bound of order $(M/\mu) \log(1/\epsilon) \sqrt{\log(m)}$, which generally is far superior, due to being only linearly dependent on the condition number. (The difference is observed empirically in §9.)

While in theory, smoothings exist for general Lipschitz convex functions, the set of functions known to have *tractable* smoothings is limited. Thus, in practice, `subgrad` is relevant to a wider class of problems than `smooth`. Nonetheless, when a smoothing is available, employing `smooth` in our synchronous scheme can be much preferred to employing `subgrad`.

Each iteration of `subgrad` requires the same amount of work, one call to the oracle. Likewise for `accel` and `smooth`. By contrast, the work required by adaptive methods – such as methods employing "backtracking" – typically varies among iterations, an especially important example being Nesterov's universal fast gradient method [23]. Such algorithms can be ill-suited for use in a synchronous parallel scheme in which each $\texttt{fom}_n$ makes one iteration per time period. Nonetheless, these algorithms fit well in our asynchronous scheme. As a straightforward consequence (Corollary 8.2) to the last of our theorems (Theorem 8.1), we show combining the asynchronous scheme with the universal fast gradient method results in a nearly-optimal number of gradient evaluations when the scheme is applied to the general class of problems in which f both has Hölderian growth and has "Hölder continuous gradient with exponent $0 < \nu \leq 1$." This is the first algorithm for this especially-general class of problems that both is nearly optimal in the number of oracle calls and does not require information that typically would be unavailable in practice.

1.5. **Positioning within the literature on smooth convex optimization.** As remarked above, an understanding that restarting can speed-up first-order methods goes back to work of Nemirovski and Nesterov [17], although their focus was on the abstract setting in which somehow known are various scalars from which can be deduced nearly-ideal times to make restarts for the particular optimization problem being solved.

In the last decade, adaptivity has been a primary focus in research on optimization algorithms, but far more in the context of adaptive accelerated methods than in the context of adaptive schemes (meta-algorithms).

Impetus for the development of adaptive accelerated methods was provided by Nesterov in [20], where he designed and analyzed an accelerated method in the context of f being L-smooth, the new method possessing the same convergence rate as his original (optimal) accelerated method (in particular, $O(\sqrt{L/\epsilon})$), but without needing L as input. (Instead, input meant to approximate L is needed, and during iterations, the method modifies the input value until

reaching a value which approximates L to within appropriate accuracy.) Secondary consideration, in §5.3 of that paper, was given specifically to strongly convex functions, with a proposed grid-search procedure for approximating to within appropriate accuracy the so-called strong convexity parameter (as well as L), leading overall to an adaptive accelerated method possessing up to a logarithmic factor the optimal time bound for the class of smooth and strongly-convex functions, demonstrating how designing an adaptive method aimed at a narrower class of functions can result in a dramatically improved convergence rate ($O(\log(1/\epsilon))$ vs. $O(1/\sqrt{\epsilon})$), even without having to know apriori the Lipschitz constant L or the strong convexity parameter).

A range of significant papers on adaptive accelerated methods followed, in the setting of f being smooth and either strongly convex or uniformly convex (c.f., [10], [13]), but also relaxing the strong convexity requirement to assuming Hölderian growth with degree $d = 2$ (c.f., [16]).

Far fewer have been the number of papers regarding adaptive schemes, but still, important contributions have been made, in earlier papers which focused largely on heuristics for the setting of f being L-smooth and strongly convex (c.f., [24, 8, 5]), but more recently, papers analyzing schemes with proven guarantees (c.f., [5]), including for the setting when the smooth function f is not necessarily strongly convex but instead satisfies the weaker condition of having Hölderian growth with $d = 2$ (c.f., [6]). Each of these schemes, however, is designed for a narrow family of first-order methods, and typically relies on learning appropriately-accurate approximations of the parameter values characterizing functions in a particular class (e.g., learning the Lipschitz constant L when f is assumed to be smooth). There is not a simple, general and easily-implementable meta-heuristic underlying any of the schemes.

Going beyond the setting of f being smooth and $d = 2$, and going beyond being designed to apply to a narrow family of first-order methods, the foremost contributions on restart schemes are due to Roulet and d'Aspremont [30], who consider all f possessing Hölderian growth (which they call "sharpness"), and having Hölder continuous gradient with exponent $0 < \nu \leq 1$. Their work aims, in part, to make algorithmically-concrete the earlier abstract analysis of Nemirovski and Nesterov [17].

The restart schemes of Roulet and d'Aspremont result in optimal time bounds when particular algorithms are employed in the schemes, assuming scheme parameters are set to appropriate values – values that generally would be unknown in practice. However, for smooth f (i.e., $\nu = 1$), they develop ([30, §3.2]) an adaptive grid-search procedure within the scheme, to accurately approximate the required values, leading to overall time bounds that differ from the optimal bounds only by a logarithmic factor. Moreover, for general f for which the optimal value f^* is known, they show ([30, §4]) that when an especially-simple restart scheme employs Nesterov's universal fast gradient method [23], nearly optimal time bounds result.

The restart schemes with established time bounds either rely on values that would not be known in practice (e.g., f^*), or assume the function f is of a particular form (e.g., smooth and having Hölderian growth), and adaptively approximate values characterizing the function (e.g., μ and d) in order to set scheme parameters to appropriate values. By contrast, the restart scheme we propose depends only on how much progress has been made, how much the objective value has been decreased.

Nonetheless, when Nesterov's universal fast gradient method [23] is employed in our scheme, it achieves nearly optimal complexity bounds for f having Hölderian growth and Hölder continuous gradient with exponent $0 < \nu \leq 1$ (Corollary 8.2). Here, our results break new ground in their combination of (1) generality of the class of functions f, (2) attainment of nearly-optimal complexity bounds, and (3) avoidance of relying on problem information that would not be available in practice.

1.6. **Positioning within the literature on nonsmooth convex optimization.** Employing the scheme even with the simple algorithm `subgrad` leads to new and consequential results when f is a nonsmooth convex function that is Lipschitz on an open neighborhood of Q. In this regard, we note that for such functions which in addition have growth degree $d = 1$, there was considerable interest in obtaining linear convergence even in the early years of research on subgradient methods (see [25, 26, 9, 31] for discussion and references). The goal was accomplished, however, only under substantial assumptions.

Interest in the setting continues today. In recent years, various algorithms have been developed with complexity bounds depending only logarithmically on ϵ (c.f., [4, 7, 11, 28, 34]). Nevertheless, each of the algorithms for which a logarithmic bound has been established depends on exact knowledge of values characterizing an optimization problem (e.g., f^*), or on accurate estimates of values (e.g., the distance of the initial iterate from optimality, or a "growth constant"), that generally would be unknown in practice. None of the algorithms entirely avoids the need for special information, although a few of the algorithms are capable of adaptively learning appropriately-accurate approximations to nearly all of the values they rely upon. (Most notable for us, in part due to its generality, is Johnstone and Moulin [11].)

By contrast, given that the algorithm `subgrad` does not rely on parameter values characterizing problem structure, the same is true for our synchronous scheme when `subgrad` is the method employed. It is thus consequential that the resulting time bound, presented in Corollary 5.2, is proportional to $\log(1/\epsilon)$. (The total number of subgradient evaluations is proportional to $\log(1/\epsilon)^2$.) As mentioned earlier, for general convex optimization problems with Lipschitz objective functions, this provides the first algorithm which both relies on no special information about the optimization problem being solved, and for which it known that if the objective function f happens to possess linear growth (i.e., happens to be Hölderian with $d = 1$), an ϵ-optimal solution is computed with a total number of subgradient evaluations depending only logarithmically on $1/\epsilon$. (Moreover, as we indicated, when `smooth` is employed in our synchronous scheme, not only is logarithmic dependence on $1/\epsilon$ obtained for important convex optimization problems, but also linear dependence on the condition number M/μ, an even stronger result at least in theory, than when `subgrad` is employed in the scheme.)

Perhaps the most surprising aspect of the paper is that the numerous advances are accomplished with such a simple restart scheme. We now begin detailing the scheme and the elementary theory.

2. Assumptions

Recall that we consider optimization problems of the form

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in Q \,, \end{array} \tag{2.1}$$

where f is a convex function, and Q is a closed convex set contained in the (effective) domain of f. The set of optimal solutions, X^*, is assumed to be nonempty. Recall f^* denotes the optimal value.

Recall also that `fom` denotes a first-order method capable of solving some class of convex optimization problems of the form (2.1), in the sense that for any problem in the class, when given an initial point x_0 and accuracy $\epsilon > 0$, the method can be specialized to provide an algorithm to generate a feasible iterate x_k guaranteed to be an ϵ-optimal solution. Let `fom`(ϵ) denote the specialized version of `fom`.

For `fom` = `subgrad`, we let `subgrad`(ϵ) denote the first-order method with iterates $x_{k+1} = P_Q(x_k - \frac{\epsilon}{\|g_k\|^2} g_k)$, where $g_k \in \partial f(x_k)$. For `fom` = `accel`, we let `accel`(ϵ) = `accel`, independent of ϵ.

2.1. **Assumptions on algorithms.** In considering a first order method `fom` which requires approximately the same amount of work during each iteration, we assume that the number of iterations (number of oracle calls) sufficient to achieve ϵ-optimality can be expressed as a function of two specific parameter values, as typically is the case. In particular, we assume there is a function $K_{\texttt{fom}} : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ satisfying

$$\text{dist}(x_0, X^*) \leq \delta \quad \Rightarrow \quad \min\{f(x_k) \mid 0 \leq k \leq K_{\texttt{fom}}(\delta, \epsilon)\} \;\leq\; f^* + \epsilon. \tag{2.2}$$

We also assume the function $K_{\texttt{fom}}$ satisfies the following natural condition:

$$\delta \leq \delta' \text{ and } \epsilon \geq \epsilon' \quad \Rightarrow \quad K_{\texttt{fom}}(\delta, \epsilon) \leq K_{\texttt{fom}}(\delta', \epsilon') \; . \tag{2.3}$$

For example, consider the class of problems of the form (2.1) for which all subgradients of f on Q satisfy $\|g\| \leq M$ for fixed M (we slightly abuse terminology and say that f is "M-Lipschitz on Q"). An appropriate algorithm is `subgrad`, for which it is well known that the function $K_{\texttt{fom}} = K_{\texttt{subgrad}}$ given by

$$K_{\texttt{subgrad}}(\delta, \epsilon) := (M\delta/\epsilon)^2 \tag{2.4}$$

satisfies (2.2). (A simple proof is included in Appendix A.)

As another example, for problems in which f is L-smooth on an open neighborhood of Q, the first-order method `accel` can be applied, for which (2.2) holds with $K_{\texttt{fom}} = K_{\texttt{accel}}$ given by

$$K_{\texttt{accel}}(\delta, \epsilon) := \delta\sqrt{2L/\epsilon} \; . \tag{2.5}$$

The same function bounds the number of oracle calls for `fista`, in the more general setting of proximal methods. (See [1, Thm 4.4] for `fista` – for the special case of `accel`, choose g in that theorem to be the indicator function for Q.)

A third general example we rely upon pertains to the notion of "smoothing" promoted by Nesterov[19], in which a nonsmooth convex objective function f is replaced by a smooth convex approximation f_η, depending on parameter η which controls the error in how well f_η approximates f, as well as determines the smoothness of f_η (i.e., determines the Lipschitz constant L_η for which $\|\nabla f_\eta(x) - f_\eta(y)\| \leq L_\eta \|x - y\|$). Nesterov showed that for functions f of particular structure, there is a smoothing f_η such that if η is chosen appropriately, and if an accelerated gradient method is applied with f_η in place of f, then an iteration bound growing only like $O(1/\epsilon)$ results for obtaining an iterate x_k which is an ϵ-optimal solution for f, the objective function of interest. Compare this with the worst-case $O(1/\epsilon^2)$ bound resulting from applying a subgradient method to f.

Nesterov's notion of smoothing was further developed by Beck and Teboulle in [2], where they used a slightly more general definition than the following:

> An "(α, β) smoothing of f" is a family of functions f_η parameterized by $\eta > 0$, where for each η, the function f_η is smooth on a neighborhood of Q and satisfies
> (1) $\|\nabla f_\eta(x) - \nabla f_\eta(y)\| \leq \frac{\alpha}{\eta}\|x - y\| \; , \quad \forall x, y \in Q.$
> (2) $f(x) \leq f_\eta(x) \leq f(x) + \beta\eta \; , \quad \forall x \in Q,$

(Unlike [2], we restrict attention to the Euclidean norm due to our interest in Hölderian growth.)

For example, for the piecewise-linear convex function

$$f(x) = \max\{a_i^T x - b_i \mid i = 1, \ldots, m\} \quad \text{where } a_i \in \mathbb{R}^n, b_i \in \mathbb{R} \; , \tag{2.6}$$

a $(\max_i \|a_i\|^2, \ln(m))$-smoothing[5] is given by

$$f_\eta(x) = \eta \ln\left(\sum_{i=1}^{m} \exp((a_i^T x - b_i)/\eta)\right) . \tag{2.7}$$

Similarly, for eigenvalue optimization problems involving the maximum eigenvalue function

$$\begin{array}{ll} \min & \lambda_{\max}(X) \\ \text{s.t.} & \mathcal{A}(X) = b \, , \end{array}$$

(where $\mathcal{A}$ is a linear map from $\mathbb{S}^n$ ($n \times n$ symmetric matrices) to $\mathbb{R}^m$), Nesterov[22] showed with respect to the trace inner product that a $(1, \ln(n))$-smoothing is given by

$$f_\eta(x) = \eta \ln\left(\sum_{j=1}^{n} \exp(\lambda_j(x)/\eta)\right) . \tag{2.8}$$

(This smoothing generalizes to all hyperbolic polynomials, $X \mapsto \det(X)$ being a special case ([29]).)

In Appendix B, we demonstrate that it is straightforward to develop a first-order method `smooth` (based on `accel` (or more generally, `fista`)) that makes use of a smoothing to compute an ϵ-optimal solution for f, subject to $x \in Q$, within a number of steps (gradient evaluations) not exceeding

$$K_{\texttt{smooth}}(\delta, \epsilon) := 3\delta\sqrt{2\alpha\beta}/\epsilon \, . \tag{2.9}$$

When smoothing is possible, the iteration count for obtaining an ϵ-optimal solution is typically far better than the count for the subgradient method. For example, noting that the piecewise-linear function (2.6) is Lipschitz with constant $M := \max_i \|a_i\|$, we have the upper bounds

$$K_{\texttt{subgrad}}(\delta, \epsilon) = (M\delta/\epsilon)^2 \quad \text{and} \quad K_{\texttt{smooth}}(\delta, \epsilon) = 3M\delta\sqrt{2\ln(m)}/\epsilon \, .$$

Similarly for eigenvalue optimization.

However, the cost per iteration is typically much higher for `smooth` than for `subgrad`, because rather than computing a subgradient of f, `smooth` requires a gradient of f_η (for specified η). In the case of the maximum eigenvalue function, for example, a subgradient at X is simply vv^T, where v is any unit-length eigenvector for eigenvalue $\lambda_{\max}(X)$, whereas computing the gradient $\nabla f_\eta(X)$ essentially requires a full eigendecomposition of X.

2.2. **Assumptions for theory.** Throughout the paper, theorems are stated in terms of the function $K_{\texttt{fom}}$ and values

$$D(\hat{f}) = \sup\{\text{dist}(x, X^*) \mid x \in Q \text{ and } f(x) \leq \hat{f}\}$$

(assuming $\hat{f} \geq f^*$). For the theorems to be meaningful, the values $D(\hat{f})$ must be finite.

In corollaries throughout the paper, it is assumed additionally that the convex function f has Hölderian growth, in that there exist positive constants μ and d for which

$$x \in Q \text{ and } f(x) \leq f(\mathbf{x_0}) \quad \Rightarrow \quad f(x) - f^* \geq \mu \, \text{dist}(x, X^*)^d \, ,$$

[5]This is example 4.5 in [2], but there the Euclidean norm is not used, unlike here. To verify correctness of the value $\alpha = \max_i \|a_i\|^2$, it suffices to show for each x that the largest eigenvalue of the positive-definite matrix $\nabla^2 f_\eta(x)$ does not exceed $\max_i \|a_i\|^2/\eta$. However, $\nabla^2 f_\eta(x) = -\frac{1}{\eta}\nabla f_\eta(x)\nabla f_\eta(x)^T + \frac{1}{\eta\sum_i \exp((a_i^T x + b_i)/\eta)} \sum_i \exp((a_i^T x + b_i)/\eta)\, a_i a_i^T$, and hence the largest eigenvalue of $\nabla^2 f(x)$ is no greater than $1/\eta$ times the largest eigenvalue of the matrix $\frac{1}{\sum_i \exp((a_i^T x + b_i)/\eta)} \sum_i \exp((a_i^T x + b_i)/\eta)\, a_i a_i^T$, which is a convex combination of the matrices $a_i a_i^T$. Consequently $\alpha = \max_i \|a_i\|^2$ is indeed an appropriate choice.

where $\mathbf{x_0}$ is the initial iterate. (This assumption can be slightly weakened at the expense of heavier notation.[6]) It is easy to prove that convexity of f forces d to satisfy $d \geq 1$. It also is not difficult to show that for smooth convex functions possessing Hölderian growth, it must be the case that $d \geq 2$.

For each of our theorems, the corollaries are deduced in straightforward manner from the immediate fact that if f has Hölderian growth and $\hat{f} \geq f^*$, then

$$D(\hat{f}) \leq \left(\tfrac{\hat{f}-f^*}{\mu}\right)^{1/d} \quad ,$$

and hence for any $\epsilon > 0$,

$$K_{\texttt{fom}}(D(\hat{f}), \epsilon) \leq K_{\texttt{fom}}\left(\left(\tfrac{\hat{f}-f^*}{\mu}\right)^{1/d}, \epsilon\right) \tag{2.10}$$

(making use of the natural assumption (2.3)).

3. Idealized Setting: When the Optimal Value is Known

When the optimal value is known, optimal complexity bounds can readily be established without having to run multiple copies of a first order method in parallel, as we show in the current section. This section is similar in spirit to the analysis in [30, §4], but is done in a manner which immediately applies to numerous first-order methods by focusing on the function $K_{\texttt{fom}}(\delta, \epsilon)$. Our development serves to motivate the structure of our restart scheme (presented in §4), and serves to make more transparent the proofs of results regarding our scheme.

The following scheme never terminates, and computes an ϵ-optimal solution for every $\epsilon > 0$.

"Polyak Restart Scheme"

(0) Input: f^*, $\mathbf{x_0} \in Q$ (an initial point)
  Initialize: Let $x_0 = \mathbf{x_0}$.
(1) Compute $\tilde{\epsilon} := \frac{1}{2}(f(x_0) - f^*)$.
(2) Initiate $\texttt{fom}(\tilde{\epsilon})$ at x_0,
  and continue until reaching the first iterate x_k satisfying $f(x_k) \leq f(x_0) - \tilde{\epsilon}$.
(3) Substitute $x_0 \leftarrow x_k$, and go to Step 1.

We name the scheme after B.T. Polyak due to his early promotion of the subgradient method given by $x_{k+1} = x_k - \frac{f(x_k)-f^*}{\|g_k\|^2} g_k$, where the step size $\alpha_k = \frac{f(x_k)-f^*}{\|g_k\|^2}$ is updated to be "ideal" at every iteration (c.f., [27]). Our Polyak restart scheme is a bit different, in using the value $f(x_0) - f^*$ to set a goal that when achieved, results in a restart. This restart scheme is effective even for first-order methods that do not rely on ϵ as input, such as `accel`.

The update rule based on $\tilde{\epsilon} = \frac{1}{2}(f(x_0) - f^*)$ is slightly arbitrary, in that for any constant $0 < \kappa < 1$, the rule with $\tilde{\epsilon} = \kappa\,(f(x_0) - f^*)$ would lead to iteration bounds that differ only by a constant factor from the bounds we deduce below.

---

[6] In particular, analogous results can be obtained under the weaker assumption that Hölderian growth holds on a smaller set $\{x \in Q \mid \text{dist}(x, X^*) \leq r\}$ for some $r > 0$. The key is that due to convexity of f, for $\hat{f} \geq f^*$ we would then have

$$D(\hat{f}) \leq \begin{cases} ((\hat{f} - f^*)/\mu)^{1/p} & \text{if } \hat{f} - f^* \leq \mu\, r^p \\ (\hat{f} - f^*)/(\mu\, r^{p-1}) & \text{if } \hat{f} - f^* \geq \mu\, r^p \,. \end{cases}$$

Theorem 3.1. *For each ϵ satisfying $0 < \epsilon < f(\mathbf{x_0}) - f^*$, the Polyak restart scheme computes an ϵ-optimal solution within a total number of iterations (oracle calls) for* `fom` *not exceeding*

$$\sum_{n=-1}^{\bar{N}} K_{\texttt{fom}}(D_n, 2^n\epsilon) \quad \textit{with } D_n := \min\{D(f^* + 4\cdot 2^n\epsilon), D(f(\mathbf{x_0}))\}\ , \tag{3.1}$$

where $\bar{N} = \lfloor \log_2 \left(\frac{f(\mathbf{x_0})-f^*}{\epsilon}\right) \rfloor - 1$.

Proof: Denote the sequence of values $\tilde{\epsilon}$ computed by the scheme as $\tilde{\epsilon}_1, \tilde{\epsilon}_2, \ldots$. A simple inductive argument shows the sequence is infinite, by observing that when working with $\tilde{\epsilon}_i$, the scheme employs $\texttt{fom}(\tilde{\epsilon}_i)$, which is guaranteed to compute an $\tilde{\epsilon}_i$-optimal solution, at which time $\tilde{\epsilon}_i$ is replaced by a value $\tilde{\epsilon}_{i+1} \leq \tilde{\epsilon}_i/2$.

For each i, let $y_{i,0}, \ldots, y_{i,k_i}$ denote the sequence of iterates for $\texttt{fom}(\tilde{\epsilon}_i)$ – thus, $\tilde{\epsilon}_i = \frac{1}{2}(f(y_{i,0}) - f^*)$ and y_{i,k_i} is the first of these iterates which is an $\tilde{\epsilon}_i$-optimal solution. Consequently,

$$k_i \leq K_{\texttt{fom}}(\, D(f(y_{i,0})), \tilde{\epsilon}_i\,) \ = K_{\texttt{fom}}(\, D(f^* + 2\tilde{\epsilon}_i), \tilde{\epsilon}_i\,)\ . \tag{3.2}$$

Fix any ϵ satisfying $0 < \epsilon < f(\mathbf{x_0}) - f^*$. For each i, let n_i be the largest integer satisfying $2^{n_i}\epsilon \leq \tilde{\epsilon}_i$. Of course we also have $4\cdot 2^{n_i}\epsilon > 2\tilde{\epsilon}_i$. Consequently, by (3.2),

$$k_i \leq K_{\texttt{fom}}(\, D(f^* + 4\cdot 2^{n_i}\epsilon), 2^{n_i}\epsilon\,)\ . \tag{3.3}$$

Due to the relations $\tilde{\epsilon}_{i+1} \leq \frac{1}{2}\tilde{\epsilon}_i$, the sequence $n_1, n_2, \ldots$ is strictly decreasing. Moreover, n_1 is precisely the integer $\bar{N}$ in the statement of the theorem.

Let i' be the last index i for which $\tilde{\epsilon}_i > \epsilon/2$. Then $f(y_{i'+1,0}) - f^* = 2\tilde{\epsilon}_{i'+1} \leq \epsilon$ – thus, since $y_{i'+1,0} = y_{i',k_{i'}}$, we have that $y_{i',k_{i'}}$ is an ϵ-optimal solution. Moreover, since $\tilde{\epsilon}_{i'} > \epsilon/2$, we have $n_{i'} \geq -1$.

In all, the total number of iterations of `fom` required by the Polyak scheme to compute an ϵ-optimal solution does not exceed

$$\sum_{i=1}^{i'} k_i \leq \sum_{n=-1}^{\bar{N}} K_{\texttt{fom}}\big(D(f^* + 4\cdot 2^n\epsilon), 2^n\epsilon\big)\ ,$$

establishing the theorem. □

Corollary 3.2. *Assume f has Hölderian growth. For each ϵ satisfying $0 < \epsilon < f(\mathbf{x_0}) - f^*$, the Polyak scheme computes an ϵ-optimal solution within a total number of iterations of* `fom` *not exceeding*

$$\sum_{n=-1}^{\bar{N}} K_{\texttt{fom}}\left(\left(\tfrac{4\cdot 2^n\epsilon}{\mu}\right)^{1/d}, 2^n\epsilon\right)\ , \tag{3.4}$$

where $\bar{N} = \lfloor \log_2 \left(\frac{f(\mathbf{x_0})-f^*}{\epsilon}\right) \rfloor - 1$.

Proof: Simply substitute into Theorem 3.1 using (2.10). □

Corollary 3.3. *Assume f is M-Lipschtiz on Q, and has Hölderian growth. For ϵ satisfying $0 < \epsilon < f(\mathbf{x_0}) - f^*$, the Polyak scheme with* `fom` = `subgrad` *computes an ϵ-optimal solution within a total number of* `subgrad` *iterations K, where*

$$d = 1 \ \Rightarrow\ K \leq (\bar{N}+2)(4M/\mu)^2\ , \tag{3.5}$$

$$d > 1 \ \Rightarrow\ K \leq \left(\frac{4^{1/d}M}{\mu^{1/d}\,\epsilon^{1-1/d}}\right)^2 \min\left\{\frac{16^{1-1/d}}{4^{1-1/d}-1}, \bar{N}+5\right\}\ , \tag{3.6}$$

with $\bar{N} = \lfloor \log_2 \left(\frac{f(\mathbf{x_0}) - f^*}{\epsilon} \right) \rfloor - 1$.

Proof: From $K_{\texttt{subgrad}}(\delta, \bar{\epsilon}) := (M\delta/\bar{\epsilon})^2$ follows

$$K_{\texttt{subgrad}} \left((4 \cdot 2^n \epsilon/\mu)^{1/d},\, 2^n \epsilon \right) \leq \left(\frac{M(4 \cdot 2^n \epsilon/\mu)^{1/d}}{2^n \epsilon} \right)^2 = \left(\frac{4^{1/d} M}{\mu^{1/d} \epsilon^{1-1/d}} \right)^2 \left(\frac{1}{4^{1-1/d}} \right)^n ,$$

and hence (3.4) is bounded above by

$$\left(\frac{4^{1/d} M}{\mu^{1/d} \epsilon^{1-1/d}} \right)^2 \sum_{n=-1}^{\bar{N}} \left(\frac{1}{4^{1-1/d}} \right)^n .$$

The implication for $d = 1$ is immediate by Corollary 3.2. On the other hand, since for $d > 1$,

$$\sum_{n=-1}^{\bar{N}} \left(\frac{1}{4^{1-1/d}} \right)^n < \min \left\{ \sum_{n=-1}^{\infty} \left(\frac{1}{4^{1-1/d}} \right)^n ,\ \bar{N} + 5 \right\} = \min \left\{ \frac{16^{1-1/d}}{4^{1-1/d} - 1},\ \bar{N} + 5 \right\} ,$$

the implication (3.6) is immediate, too. □

Remark: Note that when $d = 1$, as it is for the convex piecewise-linear function (2.6), the corollary shows the Polyak scheme with `fom` = `subgrad` converges linearly, i.e., the number of iterations grows only like $\log(1/\epsilon)$ as $\epsilon \to 0$.

The next corollary regards `accel`, the accelerated method, for which f is assumed to be smooth, and thus d, the degree of growth, satisfies $d \geq 2$. If the term "iterations" is replaced by "oracle calls," the corollary holds for `fista`, of which `accel` is a special case, as we mentioned in §1.1.

Corollary 3.4. *Assume f is L-smooth on a neighborhood of Q, and has Hölderian growth. For ϵ satisfying $0 < \epsilon < f(\mathbf{x_0}) - f^*$, the Polyak scheme with* `fom` = `accel` *computes an ϵ-optimal solution within a total number of* `accel` *iterations K, where*

$$d = 2 \ \Rightarrow \ K \leq 2(\bar{N} + 2)\sqrt{2L/\mu} \,, \tag{3.7}$$

$$d > 2 \ \Rightarrow \ K \leq \frac{(4/\mu)^{1/d}\sqrt{2L}}{\epsilon^{\frac{1}{2}-\frac{1}{d}}} \min \left\{ \frac{4^{\frac{1}{2}-\frac{1}{d}}}{2^{\frac{1}{2}-\frac{1}{d}} - 1},\ \bar{N} + 3 \right\} , \tag{3.8}$$

with $\bar{N} = \lfloor \log_2 \left(\frac{f(\mathbf{x_0}) - f^*}{\epsilon} \right) \rfloor - 1$.

Proof: From $K_{\texttt{accel}}(\delta, \bar{\epsilon}) := \delta\sqrt{2L/\bar{\epsilon}}$ follows

$$K_{\texttt{accel}}((4 \cdot 2^n \epsilon/\mu)^{1/d}, 2^n \epsilon) \leq \frac{\sqrt{2L}(4 \cdot 2^n \epsilon/\mu)^{1/d}}{\sqrt{2^n \epsilon}} = \frac{\sqrt{2L}(4/\mu)^{1/d}}{\epsilon^{\frac{1}{2}-\frac{1}{d}}} \left(\frac{1}{2^{\frac{1}{2}-\frac{1}{d}}} \right)^n ,$$

and hence (3.4) is bounded above by

$$\frac{\sqrt{2L}(4/\mu)^{1/d}}{\epsilon^{\frac{1}{2}-\frac{1}{d}}} \sum_{n=-1}^{\bar{N}} \left(\frac{1}{2^{\frac{1}{2}-\frac{1}{d}}} \right)^n .$$

The implication for $d = 2$ is immediate from Corollary 3.2. On the other hand, since for $d > 2$,

$$\sum_{n=-1}^{\bar{N}} \left(\frac{1}{2^{\frac{1}{2}-\frac{1}{d}}} \right)^n < \min \left\{ \sum_{n=-1}^{\infty} \left(\frac{1}{2^{\frac{1}{2}-\frac{1}{d}}} \right)^n ,\ \bar{N} + 3 \right\} = \min \left\{ \frac{4^{\frac{1}{2}-\frac{1}{d}}}{2^{\frac{1}{2}-\frac{1}{d}} - 1},\ \bar{N} + 3 \right\} ,$$

the implication (3.8) is immediate, too. □

Remark: Note that when $d = 2$ – a class of functions in which the strongly convex functions form a small subset – the corollary shows the Polyak scheme with `fom` = `accel` converges linearly.

The final corollary of this section regards the Polyak scheme with `fom` = `smooth`, appropriate when f is a nonsmooth convex function for which a smoothing is known.

Corollary 3.5. *Assume f has an (α, β)-smoothing, and assume f has Hölderian growth. For ϵ satisfying $0 < \epsilon < f(\mathbf{x_0}) - f^*$, the Polyak scheme with* `fom` = `smooth` *computes an ϵ-optimal solution for f within a total number of* `smooth` *iterations K, where*

$$d = 1 \;\Rightarrow\; K \le 12(\bar{N}+2)\sqrt{2\alpha\beta}/\mu\,, \tag{3.9}$$

$$d > 1 \;\Rightarrow\; K \le \frac{3\sqrt{2\alpha\beta}(4/\mu)^{1/d}}{\epsilon^{1-1/d}} \min\left\{\frac{4^{1-1/d}}{2^{1-1/d}-1},\, \bar{N}+3\right\}\,, \tag{3.10}$$

with $\bar{N} = \lfloor \log_2\left(\frac{f(\mathbf{x_0})-f^*}{\epsilon}\right)\rfloor - 1$.

Proof: From (2.9) follows

$$K_{\texttt{smooth}}((4\cdot 2^n\epsilon/\mu)^{1/d}, 2^n\epsilon) \le \frac{3\sqrt{2\alpha\beta}(4\cdot 2^n\epsilon/\mu)^{1/d}}{2^n\epsilon} = \frac{3\sqrt{2\alpha\beta}(4/\mu)^{1/d}}{\epsilon^{1-1/d}}\left(\frac{1}{2^{1-1/d}}\right)^n\,,$$

and hence (3.4) is bounded above by

$$\frac{3\sqrt{2\alpha\beta}(4/\mu)^{1/d}}{\epsilon^{1-1/d}} \sum_{n=-1}^{\bar{N}} \left(\frac{1}{2^{1-1/d}}\right)^n\,.$$

The implication for $d = 1$ is immediate from Corollary 3.2. On the other hand, since for $d > 1$,

$$\sum_{n=-1}^{\bar{N}} \left(\frac{1}{2^{1-1/d}}\right)^n < \min\left\{\sum_{n=-1}^{\infty}\left(\frac{1}{2^{1-1/d}}\right)^n,\, \bar{N}+3\right\} = \min\left\{\frac{4^{1-1/d}}{2^{1-1/d}-1},\, \bar{N}+3\right\}\,,$$

the implication (3.10) is immediate, too. □

To understand the relevance of the corollary, consider the piecewise-linear convex function f defined by (2.6), which has $(\max_i \|a_i\|^2, \ln(m))$-smoothing given by (2.7). The function f has linear growth (i.e., $d = 1$), for some $\mu > 0$. Consequently, according to Corollary 3.5, the Polyak scheme with `fom` = `smooth` obtains an ϵ-optimal solution within

$$12(\bar{N}+2)\sqrt{2\ln(m)}\max_i \|a_i\|/\mu \text{ iterations, where } \bar{N} = \lfloor \log_2\left(\tfrac{f(\mathbf{x_0})-f^*}{\epsilon}\right)\rfloor - 1\,.$$

On the other hand, f is clearly Lipschitz with constant $M = \max_i \|a_i\|$, and thus from Corollary 3.3, the Polyak scheme with `fom` = `subgrad` obtains an ϵ-optimal solution within

$$16(\bar{N}+2)(\max_i \|a_i\|/\mu)^2 \text{ iterations.}$$

For convex functions which are Lipschitz and have linear growth, the ratio M/μ is sometimes referred to as the "condition number" of f, and is easily shown to have value at least 1. While the Polyak scheme with either `smooth` or `subgrad` results in linear convergence, the bound for `subgrad` has a factor depending quadratically on the condition number, whereas the dependence for `smooth` is linear.

Similarly, for an eigenvalue optimization problem, the dependence on ϵ for the Polyak scheme with `smooth` is always at least as good as for the scheme with `subgrad` (and is better when $d > 1$), and for `smooth` the dependence on the condition number $1/\mu$ is linear whereas for `subgrad` it is quadratic.

Nonetheless, while applying `smooth` typically results in improved iteration bounds, the cost per iteration of `smooth` can be considerably higher than the cost per iteration of `subgrad`, as was discussed at the end of §2.1 for the eigenvalue optimization problem.

In the following sections where f^* is not assumed to be known, the computational scheme we develop has nearly the same characteristics as the Polyak scheme. In particular, the analogous iteration bounds for the various choices of `fom` are only worse by a logarithmic factor, to account for multiple instances of `fom` making iterations simultaneously.

4. Synchronous Parallel Scheme (Sync‖FOM)

Henceforth, we assume f^* is unknown, and hence the Polyak Scheme is unavailable due to its reliance on the value $\tilde{\epsilon} := \frac{1}{2}(f(x_0) - f^*)$. Our approach relies on running several copies of a first-order method in parallel, that is, running algorithms $\texttt{fom}(\epsilon_n)$ in parallel, with various values for ϵ_n. Similar to the Polyak Scheme, each of these parallel algorithms is searching for an iterate x_k satisfying $f(x_k) \leq f(x_0) - \epsilon_n$.

We assume the user specifies an integer $N \geq 0$, with $N+2$ being the number of copies of `fom` that will be run in parallel. We also assume the user specifies a value $\epsilon > 0$, the goal being to obtain an ϵ-optimal solution. Ideally the user would choose $N \approx \log_2(\frac{f(\mathbf{x_0})-f^*}{\epsilon})$ where $\mathbf{x_0}$ is a feasible point at which the algorithms are initiated. But since f^* is not assumed to be known, our analysis must apply to various choices of N. Our results yield interesting iteration bounds even for the easily computable choice $N = \max\{0, \lceil \log_2(1/\epsilon) \rceil\}$.

The algorithms to be run in parallel are $\texttt{fom}(\epsilon_n)$ with $\epsilon_n = 2^n\epsilon$ $(n = -1, 0, \ldots, N)$.[7] Notice that if $N > \log_2(\frac{f(x_0)-f^*}{\epsilon})$, one of these parallel algorithms must have ϵ_n within a factor of 2 of the idealized choice used by the Polyak Restarting Scheme of $\tilde{\epsilon} := \frac{1}{2}(f(x_0) - f^*)$. Using the key ideas posited in §1.3, we carefully manage when these parallel algorithms restart, thereby matching the performance of the Polyak Restarting Scheme (up to a small constant factor), even without knowing f^*. The remaining details for specifying this restarting scheme given below are secondary to the key ideas from §1.3. It is possible to specify details differently and still obtain similar theoretical results.

To ease notation, we write $\texttt{fom}_n$ rather than $\texttt{fom}(2^n\epsilon)$ or $\texttt{fom}(\epsilon_n)$. Thus, $\texttt{fom}_n$ is a version of `fom` designed to compute a $2^n\epsilon$-optimal solution, and it does so within $K_{\texttt{fom}}(\delta, 2^n\epsilon)$ iterations, where $\delta = \text{dist}(x_0, X^*)$, with x_0 being the initial point for $\texttt{fom}_n$.

In this section we specify details for a scheme in which all of the algorithms $\texttt{fom}_n$ make an iteration simultaneously. We dub this the "synchronous parallel first-order method," and denote it as Sync‖FOM. (We will note the simple manner in which Sync‖FOM can also be implemented sequentially, with the algorithms $\texttt{fom}_n$ taking turns in making an iteration, in the cyclic order $\texttt{fom}_N, \texttt{fom}_{N-1}, \ldots, \texttt{fom}_{-1}, \texttt{fom}_N, \texttt{fom}_{N-1}, \ldots$.)

4.1. Details of Sync‖FOM.

We assume each $\texttt{fom}_n$ has its own oracle, in the sense that given x, $\texttt{fom}_n$ is capable of computing – or can readily access – the information it needs for performing an iteration at x, without having to wait in a queue with other copies $\texttt{fom}_m$ in need of information.

We speak of "time periods," each having the same length, one unit of time. The primary effort for $\texttt{fom}_n$ in a time period is to make one iteration[8], the amount of "work" required being measured as the number of calls to the oracle.

[7] The choice of values $2^n\epsilon$ $(n = -1, 0, \ldots, N)$ is slightly arbitrary. Indeed, given any scalar $\gamma > 1$, everywhere replacing $2^n\epsilon$ by $\gamma^n\epsilon$ leads to similar theoretical results. However, our analysis shows relying on powers of 2 suffices to give nearly optimal guarantees, although a more refined analysis, in the spirit of [30], might reveal desirability of relying on values of γ other than 2, depending on the setting.

[8] The synchronous parallel scheme is inappropriate for first order methods that have wide variation in the amount of work done in iterations. Instead, the asynchronous parallel scheme (§7) is appropriate.

At the outset, each of the algorithms $\texttt{fom}_n$ ($n = -1, 0, \ldots, N$) is initiated at the same feasible point, $\mathbf{x_0} \in Q$.

The algorithm $\texttt{fom}_N$ differs from the others in that it is never restarted. It also differs in that it has no "inbox" for receiving messages. For $n < N$, there is an inbox in which $\texttt{fom}_n$ can receive messages from $\texttt{fom}_{n+1}$.

At any time, each algorithm has a "task." Restarts occur only when tasks are accomplished. When a task is accomplished, the task is updated.

For all n, the initial task of $\texttt{fom}_n$ is to obtain a point x satisfying $f(x) \leq f(\mathbf{x_0}) - 2^n\epsilon$. Generally for $n < N$, at any time, $\texttt{fom}_n$ is pursuing the task of obtaining x satisfying $f(x) \leq f(x_{n,0}) - 2^n\epsilon$, where $x_{n,0}$ is the most recent (re)start point for $\texttt{fom}_n$. Likewise for $\texttt{fom}_N$ (the algorithm that never restarts), except that $x_{n,0}$ is replaced by $\bar{x}_N$, the most recent "designated point" in $\texttt{fom}_N$'s sequence of iterates. (The first designated point is $\bar{x}_N = \mathbf{x_0}$.)

In a time period, the following steps are made by $\texttt{fom}_N$: If the current iterate fails to satisfy the inequality in the task for $\texttt{fom}_N$, then $\texttt{fom}_N$ makes one iteration, and does nothing else in the time period. On the other hand, if the current iterate, say $x_{N,k}$, does satisfy the inequality, then (1) $x_{N,k}$ becomes the new designated point ($\bar{x}_N \leftarrow x_{N,k}$) and $\texttt{fom}_N$'s task is updated accordingly, (2) the new designated point is sent to the inbox of $\texttt{fom}_{N-1}$ to be available for $\texttt{fom}_{N-1}$ at the beginning of the next time period, and (3) $\texttt{fom}_N$ computes the next iterate $x_{N,k+1}$ (without having restarted).

The steps made by $\texttt{fom}_n$ ($n < N$) are as follows: First the objective value for the current iterate of $\texttt{fom}_n$ is examined, and so is the objective value of the point in the inbox (if the inbox is not empty). If the smallest of these function values – either one or two values, depending on whether the inbox is empty – does not satisfy the inequality in the task for $\texttt{fom}_n$, then $\texttt{fom}_n$ completes its effort in the time period by simply clearing its inbox (if it is not already empty) and making one iteration, without restarting. On the other hand, if the smallest of the function values does satisfy the inequality, then: (1) $\texttt{fom}_n$ is restarted at the point with the smallest function value – this point becomes the new $x_{n,0}$ in the updated task for $\texttt{fom}_n$ of obtaining x satisfying $f(x) \leq f(x_{n,0}) - 2^n\epsilon$ – and makes one iteration, (2) the new $x_{n,0}$ is sent to the inbox of $\texttt{fom}_{n-1}$ (assuming $n > -1$) to be available for $\texttt{fom}_{n-1}$ at the beginning of the next time period, and (3) the inbox of $\texttt{fom}_n$ is cleared.

This concludes the description of $\texttt{Sync}\|\texttt{FOM}$.

4.2. **Remarks.**

- Ideally for each $n > -1$, there is apparatus dedicated solely to the sending of messages from $\texttt{fom}_n$ to $\texttt{fom}_{n-1}$. If messages from all $\texttt{fom}_n$ are handled by a single server, then the length of periods might be increased to more than one unit of time, to reflect possible congestion.
- In light of $\texttt{fom}_n$ sending messages only to $\texttt{fom}_{n-1}$, the scheme can naturally be made sequential, with $\texttt{fom}_n$ performing an iteration, followed by $\texttt{fom}_{n-1}$ (and with $\texttt{fom}_N$ following $\texttt{fom}_{-1}$).
- If instead of $\texttt{fom}_n$ sending messages only to $\texttt{fom}_{n-1}$, it is allowed that after each iteration, the best iterate among all of the copies $\texttt{fom}_n$ is sent to the mailboxes of all of the copies, the empirical results of applying $\texttt{Sync}\|\texttt{FOM}$ can be noticeably improved (see § 9), although our proof techniques for worst-case iteration bounds can then be improved by only a constant factor. Consequently, in developing theory, we consider only the restrictive situation in which $\texttt{fom}_n$ sends messages only to $\texttt{fom}_{n-1}$, i.e., our theory considers only the situation in which communication is minimized. Even so, our simple theory breaks new ground for iteration bounds, in multiple regards, as will be noted.

5. Theory for Sync‖FOM

Here we state the main theorem regarding Sync‖FOM, and present corollaries for subgrad, accel and smooth. The theorem is proven in §6.

Our theorem states bounds on the complexity of Sync‖FOM depending on whether the positive integer N is chosen to satisfy a certain inequality involving the difference $f(\mathbf{x_0}) - f^*$. Since we are not assuming f^* is known, we cannot assume N is chosen to satisfy the inequality.

Theorem 5.1. *If the positive integer N happens to satisfy $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$, then* Sync‖FOM *computes an ϵ-optimal solution within time*

$$\bar{N} + 1 + 3 \sum_{n=-1}^{\bar{N}} K_{\mathtt{fom}}\left(D_n, 2^n \epsilon\right) \qquad \text{with } D_n := \min\{D(f^* + 5 \cdot 2^n \epsilon), D(f(\mathbf{x_0}))\}\,, \tag{5.1}$$

where $\bar{N}$ is the smallest integer satisfying both $f(\mathbf{x_0}) - f^* < 5 \cdot 2^{\bar{N}} \epsilon$ and $\bar{N} \geq -1$.

In any case, Sync‖FOM *computes an ϵ-optimal solution within time*

$$\mathbf{T}_N + K_{\mathtt{fom}}\big(\mathrm{dist}(\mathbf{x_0}, X^*), 2^N \epsilon\big)\,, \tag{5.2}$$

where $\mathbf{T}_N$ is the quantity obtained by substituting N for $\bar{N}$ in (5.1).

The theorem is proven in §6.

The theorem gives insight in to how the two user-specified parameters N and ϵ affect performance. If these choices happen to satisfy $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$, the time bound (5.1) applies, reflecting the fact (revealed in the proof) that the copies $\mathtt{fom}_{\bar{N}+1}, \ldots, \mathtt{fom}_N$ play no essential role in computing an ϵ-optimal solution.

On the other hand, the time bound (5.2) always holds, even if the inequality $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$ is not satisfied. By choosing N to be the easily computable value $N = \max\{-1, \lceil \log_2(1/\epsilon) \rceil\}$, the additive term $K_{\mathtt{fom}}(\mathrm{dist}(\mathbf{x_0}, X^*), 2^N \epsilon)$ becomes bounded by the constant $K_{\mathtt{fom}}(\mathrm{dist}(\mathbf{x_0}, X^*), 1)$ regardless of the value ϵ. Thus, with this choice of N, understanding the dependence of the iteration bound on ϵ shifts to focus on the value $\mathbf{T}_N$. (The point of Sync‖FOM never restarting $\mathtt{fom}_N$ is precisely to ensure that the additive term can be made independent of ϵ by using an easily-computable choice for N which grows only like $\log(1/\epsilon)$ as $\epsilon \to 0$.)

As there are $N + 2$ copies of fom being run in parallel, the total amount of work (total number of oracle calls) is proportional to the time bound multiplied by $N + 2$. Of course the same bound on the total amount of work applies if the scheme is performed sequentially rather than in parallel (i.e., $\mathtt{fom}_n$ performs an iteration, followed by $\mathtt{fom}_{n-1}$, with $\mathtt{fom}_N$ following $\mathtt{fom}_{-1}$).

The bound (5.1) of Theorem 5.1 is of the same form as the bound (3.1) of Theorem 3.1. It is thus no surprise that with proofs exactly similar to the ones for Corollaries 3.3, 3.4 and 3.5, specific time bounds can be obtained for when Sync‖FOM is implemented with subgrad, accel and smooth, respectively.

Corollary 5.2. *Assume* fom *is chosen as* subgrad, accel *or* smooth, *and make the same assumptions as in Corollary 3.3, 3.4 or 3.5, respectively (except do not assume f^* is known).*

If Sync‖FOM *is applied with* fom, *and if ϵ and N happen to satisfy $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$, then letting T denote the time sufficient for obtaining an ϵ-optimal solution, we have*

$$\begin{array}{lll}
\texttt{subgrad} & d=1 \;\Rightarrow\; T \le \bar{N}+1+3(\bar{N}+2)(5M/\mu)^2\,, \\
& d>1 \;\Rightarrow\; T \le \bar{N}+1+3\left(\frac{5^{1/d}M}{\mu^{1/d}\epsilon^{1-1/d}}\right)^2 \min\left\{\frac{16^{1-1/d}}{4^{1-1/d}-1},\,\bar{N}+5\right\}\,, \\
\texttt{accel} & d=2 \;\Rightarrow\; T \le \bar{N}+1+3(\bar{N}+2)\sqrt{10L/\mu}\,, \\
& d>2 \;\Rightarrow\; T \le \bar{N}+1+\frac{3(5/\mu)^{1/d}\sqrt{2L}}{\epsilon^{\frac{1}{2}-\frac{1}{d}}} \min\left\{\frac{4^{\frac{1}{2}-\frac{1}{d}}}{2^{\frac{1}{2}-\frac{1}{d}}-1},\,\bar{N}+3\right\}\,, \\
\texttt{smooth} & d=1 \;\Rightarrow\; T \le \bar{N}+1+45(\bar{N}+2)\sqrt{2\alpha\beta}/\mu\,, \\
& d>1 \;\Rightarrow\; T \le \bar{N}+1+\frac{9\sqrt{2\alpha\beta}(5/\mu)^{1/d}}{\epsilon^{1-1/d}} \min\left\{\frac{4^{1-1/d}}{2^{1-1/d}-1},\,\bar{N}+3\right\}\,.
\end{array}$$

In any case, a time bound is obtained by substituting N for $\bar{N}$ above, and adding

$$\begin{array}{rl}
(M\,\mathrm{dist}(\mathbf{x_0},X^*)/(2^N\epsilon))^2 & \textit{for } \texttt{subgrad}\,, \\
\mathrm{dist}(\mathbf{x_0},X^*)\sqrt{L/(2^{N-1}\epsilon)} & \textit{for } \texttt{accel}\,, \\
3\,\mathrm{dist}(\mathbf{x_0},X^*)\sqrt{2\alpha\beta}/(2^N\epsilon) & \textit{for } \texttt{smooth}\,.
\end{array} \tag{5.3}$$

The bound for `subgrad` when $d=1$ and $N=\max\{0,\lceil\log_2(1/\epsilon)\rceil\}$ establishes `Sync`$\|$`FOM` as the first algorithm for nonsmooth convex optimization which both has total work proven to depend only logarithmically on $1/\epsilon$, and which relies on nothing other than being able to compute subgradients (in particular, does not rely on special knowledge about f (such as knowing the optimal value, or knowing that f is Hölderian with $d=1$, etc.)).

Similarly, but now for $d=2$, the easily computable choice $N=\max\{0,\lceil\log_2(1/\epsilon)\rceil\}$ results for `accel` in a time bound that grows only like $\sqrt{L/\mu}\log(1/\epsilon)$ as $\epsilon\to 0$, although the total amount of work grows like $\sqrt{L/\mu}\log(1/\epsilon)^2$. This upper bound on the total amount of work (number of gradient evaluations) is within a factor of $\log(1/\epsilon)$ of the well-known lower bound for strongly-convex smooth functions ([18]), a small subset of the functions having Hölderian growth with $d=2$. For $d>2$, the work bound also is within a factor of $\log(1/\epsilon)$ of being optimal, according to the lower bounds stated by Nemirovski and Nesterov [17, page 26].

Finally, `Sync`$\|$`FOM` with `fom` = `smooth` and $N=\max\{0,\lceil\log_2(1/\epsilon)\rceil\}$ shares all of the strengths described at the end of §3 for the Polyak Restart Scheme with `fom` = `smooth` (and has the additional property that f^* need not be known.) Particularly notable is, when $d=1$, the combination of linear dependence on the condition number (for the examples considered in §3, among others), and work bound depending only logarithmically on $1/\epsilon$. For no other smoothing algorithm has logarithmic dependence on $1/\epsilon$ been established in general when $d=1$ (other than our Polyak scheme with `fom` = `smooth`). Likewise, when $d>1$, no other smoothing algorithm has been shown to have as good of dependence on ϵ.

Proof of Corollary 5.2: The proof is exactly similar to the proofs of Corollaries 3.3, 3.4 and 3.5. We focus on `subgrad` to show the minor changes needing to be made to the proof of Corollary 3.3.

Just as Theorem 3.1 immediately yielded the iteration bound (3.4) used in the proofs of Corollaries 3.3 , 3.4 and 3.5, so does Theorem 5.1 yield the time bound

$$\bar{N}+1+3\sum_{n=-1}^{\bar{N}} K_{\texttt{fom}}\left(\left(\frac{5\cdot 2^n\epsilon}{\mu}\right)^{1/d}, 2^n\epsilon\right)\,, \tag{5.4}$$

Now consider `Sync`$\|$`FOM` with `fom` = `subgrad`.

From $K_{\texttt{subgrad}}(\delta, \bar{\epsilon}) := (M\delta/\bar{\epsilon})^2$ follows

$$K_{\texttt{subgrad}}\left((5 \cdot 2^n\epsilon/\mu)^{1/d},\, 2^n\epsilon\right) \leq \left(\frac{M(5 \cdot 2^n\epsilon/\mu)^{1/d}}{2^n\epsilon}\right)^2 = \left(\frac{5^{1/d}M}{\mu^{1/d}\epsilon^{1-1/d}}\right)^2 \left(\frac{1}{4^{1-1/d}}\right)^n ,$$

and hence (5.4) is bounded above by

$$\bar{N} + 1 + 3\left(\frac{5^{1/d}M}{\mu^{1/d}\epsilon^{1-1/d}}\right)^2 \sum_{n=-1}^{\bar{N}} \left(\frac{1}{4^{1-1/d}}\right)^n .$$

The implication for $d = 1$ is immediate. On the other hand, since for $d > 1$,

$$\sum_{n=-1}^{\bar{N}} \left(\frac{1}{4^{1-1/d}}\right)^n < \min\left\{\sum_{n=-1}^{\infty} \left(\frac{1}{4^{1-1/d}}\right)^n,\ \bar{N} + 5\right\} = \min\left\{\frac{16^{1-1/d}}{4^{1-1/d} - 1},\, \bar{N} + 5\right\} ,$$

the implication for $d > 1$ is immediate, too.

To obtain upper bounds that hold regardless of whether the inequality $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N\epsilon$ is satisifed, then according to Theorem 5.1, simply substitute N for $\bar{N}$, and add (5.3), completing the proof for `subgrad`.

In the same manner that above proof for `subgrad` is nearly identical to proof of Corollary 3.3, the proofs for `accel` and `smooth` are nearly identical to the proofs of Corollaries 3.4 and 3.5, respectively. The proofs for `accel` and `smooth` are thus left to the reader. □

6. Proof of Theorem 5.1

The theorem is obtained through a sequence of results in which we speak of $\texttt{fom}_n$ "updating" at a point x. The starting point $\mathbf{x_0}$ is considered to be the first update point for every $\texttt{fom}_n$. After $\texttt{Sync}\|\texttt{FOM}$ has started, then for $n < N$, updating at x means restarting at x. For $\texttt{fom}_N$, updating at x is the same as having computed x satisfying the current task of $\texttt{fom}_N$, in which case x becomes the new "designated point" and is sent to the inbox of $\texttt{fom}_{N-1}$.

Lemma 6.1. *Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* \geq 2 \cdot 2^n\epsilon$. Then no later than time $t + K_{\texttt{fom}}(D(f(x)), 2^n\epsilon)$, $\texttt{fom}_n$ updates again.*

Proof: Indeed, if $\texttt{fom}_n$ has not updated by the specified time, then at that time, it has computed a point y satisfying $f(y) - f^* \leq 2^n\epsilon$, simply by definition of the function $K_{\texttt{fom}}$, the assumption that $\texttt{fom}_n$ performs one iteration in each time period, and the assumption that time periods are of unit length. Since

$$f(y) = f(x) + (f(y) - f^*) + (f^* - f(x)) \leq f(x) + 2^n\epsilon - 2 \cdot 2^n\epsilon = f(x) - 2^n\epsilon ,$$

in that case an update happens precisely at the specified time. □

Proposition 6.2. *Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* \geq 2 \cdot 2^n\epsilon$. Let $\mathbf{j} := \lfloor \frac{f(x) - f^*}{2^n\epsilon} \rfloor - 2$. Then $\texttt{fom}_n$ updates at a point $\bar{x}$ satisfying $f(\bar{x}) - f^* < 2 \cdot 2^n\epsilon$ no later than time*

$$t + \sum_{j=0}^{\mathbf{j}} K_{\texttt{fom}}(D(f(x) - j \cdot 2^n\epsilon), 2^n\epsilon) . \tag{6.1}$$

Proof: Note that $\mathbf{j}$ is the integer satisfying

$$f(x) - (\mathbf{j} + 1) \cdot 2^n\epsilon < f^* + 2 \cdot 2^n\epsilon \leq f(x) - \mathbf{j} \cdot 2^n\epsilon . \tag{6.2}$$

Lemma 6.1 implies $\texttt{fom}_n$ has a sequence of update points $x_0, x_1, \ldots, x_J, x_{J+1}$, where $x_0 = x$,

$$f(x_{j+1}) \leq f(x_j) - 2^n\epsilon \quad \text{for all } j = 0, \ldots J\ , \tag{6.3}$$

$$f(x_{J+1}) < f^* + 2 \cdot 2^n\epsilon \leq f(x_J)\ , \tag{6.4}$$

and where x_{J+1} is obtained no later than time

$$t + \sum_{j=0}^{J} K_{\texttt{fom}}(D(f(x_j)), 2^n\epsilon)\ . \tag{6.5}$$

By induction, (6.3) implies $f(x_j) \leq f(x) - j \cdot 2^n\epsilon$ $(j = 0, 1, \ldots, J+1)$, which has as a consequence that

$$K_{\texttt{fom}}(D(f(x_j)), 2^n\epsilon) \leq K_{\texttt{fom}}(D(f(x) - j \cdot 2^n\epsilon), 2^n\epsilon)\ ,$$

and also has as a consequence – due to the left inequality in (6.2) and the right inequality in (6.4) – that $J \leq \mathbf{j}$. Hence, the quantity (6.5) is bounded above by (6.1), completing the proof for the choice $\bar{x} = x_{J+1}$ (an appropriate choice for $\bar{x}$ due to the left inequality in (6.4)). □

The following corollary replaces the upper bound (6.1) with a bound which depends on quantities $f^* + i \cdot 2^n\epsilon$ instead of $f(x) - j \cdot 2^n\epsilon$. This is perhaps the key conceptual step in the proof of Theorem 5.1. From this modified bound, Corollary 6.4 shows that not long after $\texttt{fom}_n$ satisfies the condition $f(x) - f^* < 5 \cdot 2^n\epsilon$, $\texttt{fom}_{n-1}$ updates at a point x' satisfying the similar condition $f(x') - f^* < 5 \cdot 2^{n-1}\epsilon$. Inductively applying this result, Corollary 6.5 bounds how long it takes for $\texttt{fom}_{-1}$ to find a $(2 \cdot 2^{-1}\epsilon)$-optimal solution. This sequence of results is the heart of our analysis and positions us to prove the full runtime bound claimed by Theorem 5.1.

Corollary 6.3. *Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* < \mathbf{i} \cdot 2^n\epsilon$, where $\mathbf{i}$ is an integer and $\mathbf{i} \geq 3$. Then $\texttt{fom}_n$ updates at a point $\bar{x}$ satisfying $f(\bar{x}) - f^* < 2 \cdot 2^n\epsilon$ no later than time*

$$t + \sum_{i=3}^{\mathbf{i}} K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \tag{6.6}$$

$$\text{where } D_{n,i} = \min\{D(f^* + i \cdot 2^n\epsilon), D(f(\mathbf{x_0}))\}\ . \tag{6.7}$$

Proof: Clearly, we may assume $f(x) \geq f^* + 2 \cdot 2^n\epsilon$. Let $\mathbf{j}$ be as in Proposition 6.2, and hence the time bound (6.1) applies for updating at some $\bar{x}$ satisfying $f(\bar{x}) - f^* < 2 \cdot 2^n\epsilon$.

Note that for all non-negative integers j,

$$f(x) - j \cdot 2^n\epsilon < f^* + (\mathbf{i} - j) \cdot 2^n\epsilon\ . \tag{6.8}$$

Consequently, since $\hat{f} \mapsto K_{\texttt{fom}}(D(\hat{f}), 2^n\epsilon)$ is an increasing function, and since $f(x) \leq f(\mathbf{x_0})$ (due to x being an update point), the quantity (6.1) is bounded above by

$$t + \sum_{i=(\mathbf{i}-\mathbf{j})}^{\mathbf{i}} K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon)\ . \tag{6.9}$$

Finally, as $\mathbf{j}$ is the integer satisfying the inequalities (6.2), the rightmost of those inequalities, and (6.8) for $j = \mathbf{j}$, imply $\mathbf{i} - \mathbf{j} \geq 3$, and thus (6.9) is bounded from above by (6.6). □

Corollary 6.4. *Let $n > -1$. Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* < 5 \cdot 2^n\epsilon$. Then no later than time*

$$t + 1 + \sum_{i=3}^{5} K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon)$$

(where $D_{n,i}$ is given by (6.7)), $\texttt{fom}_{n-1}$ *updates at x' satisfying $f(x') - f^* < 5 \cdot 2^{n-1}\epsilon$.*

Proof: By Corollary 6.3, no later than time

$$t + \sum_{i=3}^{5} K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ ,$$

$\texttt{fom}_n$ updates at $\bar{x}$ satisfying $f(\bar{x}) < f^* + 2 \cdot 2^n\epsilon$. When $\texttt{fom}_n$ updates at $\bar{x}$, the point is sent to the inbox of $\texttt{fom}_{n-1}$, where it is available to $\texttt{fom}_{n-1}$ at the beginning of the next period.

Either $\texttt{fom}_{n-1}$ restarts at $\bar{x}$, or its most recent restart point $\hat{x}$ satisfies $f(\bar{x}) > f(\hat{x}) - 2^{n-1}\epsilon$. In the former case, $\texttt{fom}_{n-1}$ restarts at a point $x' = \bar{x}$ satisfying $f(x') < f^* + 2 \cdot 2^n\epsilon = f^* + 4 \cdot 2^{n-1}\epsilon$, whereas in the latter case it has already restarted at a point $x' = \hat{x}$ satisfying $f(x') < f(\bar{x}) + 2^{n-1}\epsilon < f^* + 5 \cdot 2^{n-1}\epsilon$. □

Corollary 6.5. *For any $\bar{N} \in \{-1, \ldots, N\}$, assume that at time t,* $\texttt{fom}_{\bar{N}}$ *updates at x satisfying $f(x) - f^* < 5 \cdot 2^{\bar{N}}\epsilon$. Then no later than time*

$$t + \bar{N} + 1 + \sum_{n=-1}^{\bar{N}} \sum_{i=3}^{5} K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ ,$$

$\texttt{Sync}\|\texttt{FOM}$ *has computed an ϵ-optimal solution.*

Proof: If $\bar{N} = -1$, Corollary 6.3 implies the additional time required by $\texttt{fom}_{-1}$ to compute a $(2 \cdot 2^{-1}\epsilon)$-optimal solution $\bar{x}$ is bounded from above by

$$\sum_{i=3}^{5} K_{\texttt{fom}}(D_{-1,i},\ 2^{-1}\epsilon) \ . \tag{6.10}$$

The present corollary thus is established for the case $\bar{N} = -1$.

On the other hand, if $\bar{N} > -1$, induction using Corollary 6.4 shows that no later than time

$$t + \bar{N} + 1 + \sum_{n=0}^{\bar{N}} \sum_{i=3}^{5} K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ ,$$

$\texttt{fom}_{-1}$ has restarted at x satisfying $f(x) - f^* < 5 \cdot 2^{-1}\epsilon$. Then, as above, the additional time required by $\texttt{fom}_{-1}$ to compute an ϵ-optimal solution does not exceed (6.10). □

We are now in position to prove Theorem 5.1, which we restate as a corollary for the reader's convenience. Recall that $\mathbf{x_0}$ is the point at which each $\texttt{fom}_n$ is started at time $t = 0$.

Corollary 6.6. *If $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N\epsilon$, then* $\texttt{Sync}\|\texttt{FOM}$ *computes an ϵ-optimal solution within time*

$$\bar{N} + 1 + 3 \sum_{n=-1}^{\bar{N}} K_{\texttt{fom}}\,(D_n, 2^n\epsilon) \qquad \text{with} \quad D_n := \min\{D(f^* + 5 \cdot 2^n\epsilon), D(f(\mathbf{x_0}))\} \ , \tag{6.11}$$

where $\bar{N}$ is the smallest integer satisfying both $f(\mathbf{x_0}) - f^* < 5 \cdot 2^{\bar{N}}\epsilon$ and $\bar{N} \geq -1$.

In any case, $\texttt{Sync}\|\texttt{FOM}$ *computes an ϵ-optimal solution within time*

$$\mathbf{T}_N + K_{\texttt{fom}}\big(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon\big) \ ,$$

where $\mathbf{T}_N$ is the quantity obtained by substituting N for $\bar{N}$ in (6.11).

Proof: For the case that $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N\epsilon$, the time bound (6.11) is immediate from Corollary 6.5 and the fact that $K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \leq K_{\texttt{fom}}(D_{n,5}, 2^n\epsilon)$ for $i = 3, 4, 5$.

In any case, because $\texttt{fom}_N$ never restarts, within time

$$K_{\texttt{fom}}(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon)\ , \tag{6.12}$$

$\texttt{fom}_N$ computes a point x satisfying $f(x) - f^* \leq 2^N\epsilon$. If x is not an update point for $\texttt{fom}_N$, then the most recent update point $\hat{x}$ satisfies $f(\hat{x}) < f(x) + 2^N\epsilon$. Hence, irrespective of whether x is an update point, by the time $\texttt{fom}_N$ has computed x, it has obtained an update point x' satisfying $f(x') - f^* < 2 \cdot 2^N\epsilon < 5 \cdot 2^N\epsilon$. Consequently, relying on Corollary 6.5 for $\bar{N} = N$, as well as on the relations $K_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \leq K_{\texttt{fom}}(D_{n,5}, 2^n\epsilon)$ for $i = 3, 4, 5$, the time required for $\texttt{Sync}\|\texttt{FOM}$ to compute an ϵ-optimal solution does not exceed the value (6.12) plus the value (6.11), where in (6.11), N is substituted for $\bar{N}$. □

7. Asynchronous Parallel Scheme ($\texttt{Async}\|\texttt{FOM}$)

Motivation for developing an asynchronous scheme arises in two ways. One consideration is that some first order methods require a number of oracle calls that varies significantly among iterations (e.g., adaptive methods utilizing backtracking). If, as in $\texttt{Sync}\|\texttt{FOM}$, the algorithms $\texttt{fom}_n$ are made to wait until each of them has finished an iteration, the time required to obtain an ϵ-optimal solution might be greatly affected.

The other source of motivation comes from an inspection of the analysis given for $\texttt{Sync}\|\texttt{FOM}$ in §6. The analysis reveals there potentially is much to be gained from an asynchronous scheme, even for methods which, like `subgrad` and `accel`, require the same number of oracle calls at every iteration. In particular, the gist of Corollary 6.4 is that the role $\texttt{fom}_n$ plays in guiding $\texttt{fom}_{n-1}$ is critical only after $\texttt{fom}_n$ has first obtained an update point x satisfying $f(x) < f^* + 5 \cdot 2^n\epsilon$. Moreover, after $\texttt{fom}_n$ has such an update point x, $\texttt{fom}_n$ will send $\texttt{fom}_{n-1}$ at most four points (and possibly x), simply because $\texttt{fom}_n$ sends a point only when the objective has been decreased by at least $2^n\epsilon$, and such a decrease can happen at most four times (due to $f(x) < f^* + 5 \cdot 2^n\epsilon$). Thus, it is only points among the last five that $\texttt{fom}_n$ sends to $\texttt{fom}_{n-1}$ which play an essential role in the efficiency of $\texttt{Sync}\|\texttt{FOM}$.

However, during the time that $\texttt{fom}_n$ is performing iterations to arrive at the last few points to be sent to $\texttt{fom}_{n-1}$, an algorithm $\texttt{fom}_m$, for $m \ll n$, can be sending scads of messages to $\texttt{fom}_{m-1}$. It is thus advantageous if $\texttt{fom}_n$ is disengaged, to the extent possible, from the timeline being followed by $\texttt{fom}_m$, especially if all messages go through a single server. Our asynchronous scheme keeps $\texttt{fom}_n$ appropriately disengaged.

7.1. Details of $\texttt{Async}\|\texttt{FOM}$.

We now turn to precisely specifying the asynchronous scheme. Particular attention has to be given to the delay between when a message is sent by $\texttt{fom}_n$ and when it is received by $\texttt{fom}_{n-1}$, and what is to be done if $\texttt{fom}_{n-1}$ happens to be in the middle of a long iteration when the message arrives. A number of other details also have to be specified in order to prove results regarding the scheme's performance. Specifications different than the ones we describe can lead to similar theoretical results.

We refer to "epochs" rather than time periods to denote the duration that $\texttt{fom}_n$ spends on each of its tasks. Each $\texttt{fom}_n$ has its own epochs.

The first epoch for every $\texttt{fom}_n$ begins at time zero, when $\texttt{fom}_n$ is started at $x_{n,0} = \mathbf{x_0} \in Q$, the same point for every $\texttt{fom}_n$.

As for the synchronous scheme, $\texttt{fom}_N$ never restarts, nor receives messages. Its only epoch is of infinite length. During the epoch, $\texttt{fom}_N$ proceeds as before, focused on accomplishing the task of computing an iterate satisfying $f(x_{N,k}) \leq f(\bar{x}_N) - 2^N\epsilon$ where $\bar{x}_N$ is the most recent "designated point." If iterate $x_{N,k}$ satisfies the inequality, it becomes the new designated point,

and is sent to the inbox of $\texttt{fom}_{N-1}$, the only difference with $\texttt{Sync}\|\texttt{FOM}$ being that the point arrives in the inbox no more than t_{transit} units of time after being sent, where $t_{\text{transit}} > 0$. The same transit time holds for messages sent by any $\texttt{fom}_n$ to $\texttt{fom}_{n-1}$. (After fully specifying the scheme, we explain that the assumption of a uniform time bound on message delivery is not overly restrictive.)

For $n < N$, the beginning of a new epoch coincides with $\texttt{fom}_n$ obtaining a point x satisfying the inequality of its task, that is, $f(x) \leq f(x_{n,0}) - 2^n\epsilon$ where $x_{n,0}$ is the most recent (re)start point for $\texttt{fom}_n$.

As for the synchronous scheme, for $n < N$, $\texttt{fom}_n$ can receive messages from $\texttt{fom}_{n+1}$. Now, however, it is assumed that when a message arrives in the inbox, a "pause" for $\texttt{fom}_n$ begins, lasting a positive amount of time, but no more than t_{pause} time units. The pause is meant to capture various delays that can happen upon receipt of a new message, including time for $\texttt{fom}_n$ to come to a good stopping place in its calculations before actually going to the inbox.

If another message from $\texttt{fom}_{n+1}$ arrives in the inbox during a pause, then immediately the old message is overwritten by the new one, the old pause is cancelled and a new pause begins.

If during an epoch for $\texttt{fom}_n$, a message arrives after $\texttt{fom}_n$ has restarted (i.e., after $\texttt{fom}_n$ has begun making iterations), then during the pause, $\texttt{fom}_n$ determines whether the point x' sent by $\texttt{fom}_{n+1}$ satisfies the inequality of $\texttt{fom}_n$'s current task. If during the pause, another message from $\texttt{fom}_{n+1}$ arrives, overwriting x' with a new point x'', then in the new pause, $\texttt{fom}_n$ turns attention to determining if x'' satisfies the inequality (indeed, x'' is a better candidate than x' because $f(x'') \leq f(x') - 2^{n+1}\epsilon$). And so on, until the sequence of contiguous pauses ends.[9] Let x be the final point.

If x satisfies the inequality of the task for $\texttt{fom}_n$, then immediately at the end of the (contiguous) pause, a new epoch begins for $\texttt{fom}_n$. On the other hand, if x fails to satisfy the inequality, then the current epoch continues, with $\texttt{fom}_n$ returning to making iterations.

The other manner an epoch for $\texttt{fom}_n$ can end is by $\texttt{fom}_n$ itself computing an iterate that satisfies the inequality. At the instant that $\texttt{fom}_n$ computes such an iterate, a new epoch begins.

It remains to describe what occurs at the beginning of an epoch for $\texttt{fom}_n$.

As already mentioned, at time zero – the beginning of the first epoch for all $\texttt{fom}_n$ – every $\texttt{fom}_n$ is started at $x_{n,0} = \mathbf{x_0}$. No messages are sent at time zero.

For $n < N$, as stated above, the beginning of a new epoch occurs precisely when $\texttt{fom}_n$ has obtained a point x satisfying the inequality of its task.

If no message from $\texttt{fom}_{n+1}$ arrives simultaneously with the beginning of a new epoch for $\texttt{fom}_n$, then $\texttt{fom}_n$ instantly[10] (1) relabels x as $x_{n,0}$ and updates its task accordingly, (2) restarts at $x_{n,0}$, and (3) sends a copy of $x_{n,0}$ to the inbox of $\texttt{fom}_{n-1}$ (assuming $n > -1$), where it will arrive no later than time $t + t_{\text{transit}}$, with t being the time the new epoch for $\texttt{fom}_n$ has begun.

On the other hand, if a message from $\texttt{fom}_{n+1}$ arrives in the inbox exactly at the beginning of the new epoch, then $\texttt{fom}_n$ immediately pauses. Here, $\texttt{fom}_n$ makes different use of the pause than what occurs for pauses happening after $\texttt{fom}_n$ has restarted. Specifically, letting x' be the point in the inbox, then during the pause, $\texttt{fom}_n$ determines whether $f(x') < f(x)$ – if so, x' is preferred to x as the restart point. If during the pause, another message arrives, overwriting x' with x'', then during the new pause, $\texttt{fom}_n$ determines whether x'' is preferred to x (x'' is definitely preferred to x' – indeed, $f(x'') \leq f(x') - 2^{n+1}\epsilon$). And so on, until the sequence of contiguous pauses ends, at which time $\texttt{fom}_n$ instantly (1) labels the most preferred point as $x_{n,0}$

[9]The sequence of contiguous pauses is finite, because (1) $\texttt{fom}_{n+1}$ sends a message only when it has obtained x satisfying $f(x) \leq f(x_{n+1,0}) - 2^{n+1}\epsilon$, and (2) we assume f^* is finite.

[10]We assume the listed chores are accomplished instantly by $\texttt{fom}_n$. Assuming positive time is required would have negligible effect on the complexity results, but would make notation more cumbersome.

and updates its task accordingly, (2) restarts at $x_{n,0}$, and (3) sends a copy of $x_{n,0}$ to the inbox of $\texttt{fom}_{n-1}$ (assuming $n > -1$).

The description of Async‖FOM is now complete.

7.2. **Remarks.**

- Regarding the transit time, a larger value for t_{transit} might be chosen to reflect the possibility of greater congestion in message passing, as could occur if all messages go through a queue on a single server. We emphasize, however, that a long queue of unsent messages can be avoided, because (1) if two messages from $\texttt{fom}_n$ are queued, the later message contains a point that has better objective value than the earlier message, and (2) as indicated above, only the last few messages (at most four, perhaps none) sent by $\texttt{fom}_n$ play an essential role. Thus, to avoid congestion, when a second message from $\texttt{fom}_n$ arrives in the queue, delete the earlier message. So long as this policy of deletion is also applied to all of the other copies $\texttt{fom}_m$, the fact that only the last few messages from $\texttt{fom}_n$ are significant then implies there is not even a need to move the second message from $\texttt{fom}_n$ forward into the place that had been occupied by the deleted message – any truly essential message from $\texttt{fom}_n$ will naturally move from the back to the front of the queue within time proportional to $N + 1$ after its arrival. Managing the queue in this manner – deleting a message from $\texttt{fom}_n$ as soon as another message from $\texttt{fom}_n$ arrives in the queue – ensures that t_{transit} can be chosen proportional to $N+1$ in the worst case of there being only a single server devoted to message passing.
- As for the synchronous scheme, the ideal arrangement is, of course, for each $n > -1$, to have apparatus dedicated solely to the sending of messages from $\texttt{fom}_n$ to $\texttt{fom}_{n-1}$. Then t_{transit} can be chosen as a constant independent of the size of N.
- For many first-order methods, it is natural to choose t_{pause} to be the maximum possible number of oracle calls in an iteration, in which case a pause for $\texttt{fom}_n$ can be interpreted as the time needed for $\texttt{fom}_n$ to complete its current iteration before checking whether a message is in its inbox.

The synchronous scheme can be viewed as a special case of the asynchronous scheme by choosing $t_{\text{transit}} = 1$ and $t_{\text{pause}} = 0$ (rather, can be viewed as a limiting case, because for Async‖FOM, the length of a pause is assumed to be positive). Indeed, choices in designing the asynchronous scheme were made with this in mind, the intent being that the main result for the synchronous scheme would be a corollary of a theorem for the asynchronous scheme. However, as the foremost ideas for both schemes are the same, and since a proof for the synchronous scheme happens to have greater transparency, in the end we chose to first present the theorem and proof for the synchronous scheme.

8. Theory for Async‖FOM

Here we state the main theorem regarding Async‖FOM, and deduce from it a corollary for Nesterov's universal fast gradient method [23].

8.1. **Main theorem for Async‖FOM.** So as to account for the possibility of variability in the amount of work among iterations, we now rely on a function $T_{\texttt{fom}}(\delta, \bar{\epsilon})$ assumed to provide an upper bound on the amount of time (often proportional to the total number of oracle calls) required by fom to compute an $\bar{\epsilon}$-optimal solution when fom is initiated at an arbitrary point $x_0 \in Q$ satisfying $\text{dist}(x_0, X^*) \leq \delta$.

Recall that for scalars $\hat{f} \geq f^*$,

$$D(\hat{f}) := \sup\{\text{dist}(x, X^*) \mid x \in Q \text{ and } f(x) \leq \hat{f}\} .$$

Theorem 8.1. *If $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$, then* `Async‖FOM` *computes an ϵ-optimal solution within time*

$$(\bar{N}+1)t_{\text{transit}} + 2(\bar{N}+2)t_{\text{pause}} + 3\sum_{n=-1}^{\bar{N}} T_{\texttt{fom}}\left(D_n, 2^n\epsilon\right) \tag{8.1}$$
$$\text{with } D_n := \min\{D(f^* + 5\cdot 2^n\epsilon), D(f(\mathbf{x_0}))\}\ ,$$

where $\bar{N}$ is the smallest integer satisfying both $f(\mathbf{x_0}) - f^* < 5\cdot 2^{\bar{N}}\epsilon$ and $\bar{N} \geq -1$.

In any case, `Async‖FOM` *computes an ϵ-optimal solution within time*

$$\mathbf{T}_N + T_{\texttt{fom}}\big(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon\big)\ ,$$

where $\mathbf{T}_N$ is the quantity obtained by substituting N for $\bar{N}$ in (8.1).

The proof is deferred to Appendix C, due to its similarities with the proof of Theorem 5.1.

Remarks:

- As observed in §7, a larger value for t_{transit} might be chosen to reflect the possibility of greater congestion in message passing, although by appropriately managing communication, in the worst case (in which all communication goes through a single server), a message from $\texttt{fom}_n$ would reach $\texttt{fom}_{n-1}$ within time proportional to $N+1$. For t_{transit} being proportional to $N+1$, we see from (8.1) that the impact of congestion on `Async‖FOM` is modest, adding an amount of time proportional to $(N+1)^2$. Compare this with the synchronous scheme, where to incorporate congestion in message passing, every time period would be lengthened, in the worst case to length $1+\kappa\cdot(N+1)$ for some postive constant κ. The time bound (5.1) would then be multiplied by $1+\kappa\cdot(N+1)$, quite different than adding only a single term proportional to $(N+1)^2$.
- Unlike the synchronous scheme, for the asynchronous scheme we know of no sequential analogue that in general is truly natural.

8.2. **A corollary.** To provide a representative application of Theorem 8.1, we consider Nesterov's universal fast gradient method [23, §4]– denoted `univ` – which applies whenever f has Hölder continous gradient with exponent ν $(0 \leq \nu \leq 1)$, meaning

$$M_\nu := \sup\left\{\frac{\|\nabla f(x) - \nabla f(y)\|}{\|x-y\|^\nu} \mid x, y \in Q,\ x \neq y\right\} < \infty \quad \text{(i.e., is finite)}\ .$$

(If $\nu = 0$ then f is M_0-Lipschitz on Q, whereas if $\nu = 1$, f is M_1-smooth on Q.)

The input to `univ` consists of the desired accuracy $\bar{\epsilon}$, an initial point $x_0 \in Q$, and a value $L_0 > 0$ meant, roughly speaking, as a guess of M_ν. Nesterov [23, (4.5)] showed the function

$$K_{\texttt{univ}}(\delta, \bar{\epsilon}) = 4\left(M_\nu \delta^{1+\nu}/\bar{\epsilon}\right)^{\frac{2}{1+3\nu}} \tag{8.2}$$

provides an upper bound on the number of iterations sufficient to compute an $\bar{\epsilon}$-optimal solution if $\text{dist}(x_0, X^*) \leq \delta$ (where in [23, (4.5)] we have substituted $\xi(x_0, x^*) = \frac{1}{2}\|x_0 - x^*\|^2$ and used $2^{\frac{2(1+\nu)}{1+3\nu}} \leq 4$ when $0 \leq \nu \leq 1$).

The number of oracle calls in some iterations, however, can significantly exceed the number in other iterations. Indeed, the proofs in [23] leave open the possibility that the number of oracle calls made only in the k^{th} iteration might exceed k. Thus, the scheme `Async‖FOM` is highly preferred to `Sync‖FOM` when `fom` is chosen to be `univ`.

For the universal fast gradient method, the upper bound established in [23] on the number of oracle calls in the first k iterations is

$$4(k+1) + \log_2\left(\delta^{\frac{2(1-\nu)}{1+3\nu}}(1/\bar{\epsilon})^{\frac{3(1-\nu)}{1+3\nu}} M_\nu^{\frac{4}{1+3\nu}}\right) - 2\log_2 L_0\ , \tag{8.3}$$

assuming $\operatorname{dist}(x_0, X^*) \le \delta$. This upper bound depends on an assumption such as

$$L_0 \le \left(\tfrac{1-\nu}{1+\nu} \cdot \tfrac{1}{\bar\epsilon}\right)^{\frac{1-\nu}{1+\nu}} M_\nu^{\frac{2}{1+\nu}} \tag{8.4}$$

($L_0 \le M_1$ when $\nu = 1$). Presumably the assumption can be removed by a slight extension of the analysis, resulting in a time bound that differs insignificantly, just as the assumption can readily be removed for the first universal method introduced by Nesterov in [20] (essentially the same algorithm as `univ` when $\nu = 1$).[11] However, as the focus of the present paper is on the simplicity of $\|$`FOM` and not on `univ` per se, we do not digress to attempt removing the assumption, but instead make assumptions that ensure the results from [23] apply.

In particular, we assume L_0 is a positive constant satisfying (8.4) for $\bar\epsilon = 2^{-1}\epsilon$, and thus satisfies (8.4) when $\bar\epsilon = 2^n\epsilon$ for any $n \ge -1$. Moreover, we assume that whenever $\texttt{univ}_n$ ($n = -1, 0, \dots N$) is (re)started, the input consists of $\bar\epsilon = 2^n\epsilon$, $x_{n,0}$ (the (re)start point), and L_0 (the same value at every (re)start).

Relying on the same value L_0 at every (re)start likely results in complexity bounds that are slightly worse in some cases (specifically, when $d = 1 + \nu$ and $0 \le \nu < 1$), but still not far from being optimal, as we will see.

In view of (8.3), and assuming (8.4) holds, it is natural to define

$$T_{\texttt{univ}}(\delta, \bar\epsilon) = 4(K_{\texttt{univ}}(\delta, \bar\epsilon) + 1) + \log_2 \left(\delta^{\frac{2(1-\nu)}{1+3\nu}} (1/\bar\epsilon)^{\frac{3(1-\nu)}{1+3\nu}} M_\nu^{\frac{4}{1+3\nu}}\right) - 2\log_2 L_0 \,, \tag{8.5}$$

an upper bound on the time (total number of oracle calls) sufficient for `univ` to compute an $\bar\epsilon$-optimal solution when initiated at an arbitrary point $x_0 \in Q$ satisfying $\operatorname{dist}(x_0, X^*) \le \delta$. In order to reduce notation, for $\epsilon > 0$ let

$$\mathcal{C}(\delta, \epsilon) := 4 + \log_2 \left(\delta^{\frac{2(1-\nu)}{1+3\nu}} (2/\epsilon)^{\frac{3(1-\nu)}{1+3\nu}} M_\nu^{\frac{4}{1+3\nu}}\right) - 2\log_2 L_0 \,,$$

in which case for $n \ge -1$, from (8.5) and (8.2) follows

$$\begin{aligned} T_{\texttt{univ}}(\delta, 2^n\epsilon) &\le 4K_{\texttt{univ}}(\delta, 2^n\epsilon) + \mathcal{C}(\delta, \epsilon) \\ &\le 16\left(M_\nu \delta^{1+\nu}/(2^n\epsilon)\right)^{\frac{2}{1+3\nu}} + \mathcal{C}(\delta, \epsilon) \,. \end{aligned} \tag{8.6}$$

Note that $\mathcal{C}(\delta, \epsilon)$ is a constant independent of ϵ if $\nu = 1$, and otherwise grows like $\log(1/\epsilon)$ as $\epsilon \to 0$.

We assume f has Hölderian growth, that is, there exist constants $\mu > 0$ and $d \ge 1$ for which

$$x \in Q \text{ and } f(x) \le f(\mathbf{x_0}) \quad \Rightarrow \quad f(x) - f^* \ge \mu \operatorname{dist}(x, X^*)^d \,.$$

Consequently, the values D_n in Theorem 8.1 satisfy

$$D_n \le \min\{(5 \cdot 2^n\epsilon/\mu)^{1/d}, D(f(\mathbf{x_0}))\} \,. \tag{8.7}$$

For a function f which has Hölderian growth and has Hölder continuous gradient, necessarily the values d and ν satisfy $d \ge 1 + \nu$ (see [30, §1.3]).

[11] For that setting, see [29, Appendix A] for a slight extension to Nesterov's arguments that suffice to remove the assumption.

Corollary 8.2. *Consider* `Async‖FOM` *with* `fom = univ`.
If $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$, *then* `Async‖FOM` *computes an* ϵ*-optimal solution in time* T *for which*

$$d = 1 + \nu \;\Rightarrow\; T \leq (\bar{N}+1)t_{\text{transit}} + 2(\bar{N}+2)t_{\text{pause}} + 3(\bar{N}+2)\,\mathcal{C}(D(f(\mathbf{x_0})),\epsilon) + 48(\bar{N}+2)\,(5M_\nu/\mu)^{\frac{2}{1+3\nu}}\;, \tag{8.8}$$

$$d > 1 + \nu \;\Rightarrow\; T \leq (\bar{N}+1)t_{\text{transit}} + 2(\bar{N}+2)t_{\text{pause}} + 3(\bar{N}+2)\,\mathcal{C}(D(f(\mathbf{x_0})),\epsilon) + 48\left(\frac{M_\nu(5/\mu)^{\frac{1+\nu}{d}}}{\epsilon^{1-\frac{1+\nu}{d}}}\right)^{\frac{2}{1+3\nu}} \min\left\{\frac{4^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}}{2^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}-1}, \bar{N}+5\right\}, \tag{8.9}$$

where $\bar{N}$ *is the smallest integer satisfying both* $f(\mathbf{x_0}) < f^* + 5 \cdot 2^{\bar{N}}\epsilon$ *and* $\bar{N} \geq -1$.

In any case, a time bound is obtained by substituting N *for* $\bar{N}$ *above, and adding*

$$16\left(M_\nu \text{dist}(\mathbf{x_0}, X^*)^{1+\nu}/(2^N\epsilon)\right)^{\frac{2}{1+3\nu}} + \mathcal{C}(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon)\;. \tag{8.10}$$

Remarks: According to Nemirovski and Nesterov [17, page 6] (who state lower bounds when $0 < \nu \leq 1$), for the easily computable choice $N = \max\{0, \lceil \log_2(1/\epsilon)\rceil\}$, the time bounds of the corollary would be optimal – with regards to ϵ, μ, d and M_ν – for a sequential algorithm, except in the cases when both $d = 1+\nu$ and $0 < \nu < 1$, where due to the term "$3(\bar{N}+2)\,\mathcal{C}(D(f(\mathbf{x_0})),\epsilon)$," the bound (6.8) grows like $\log(1/\epsilon)^2$ as $\epsilon \to 0$, rather than growing like $\log(1/\epsilon)$. Consequently, in those cases, the total amount of work is within a multiple of $\log(1/\epsilon)^2$ of being optimal, whereas in the other cases for which they state lower bounds, the total amount of work is within a multiple of $\log(1/\epsilon)$ of being optimal.

Proof of Corollary 8.2: Define D_n as in Theorem 8.1, that is, $D_n = \min\{D(f^* + 5 \cdot 2^n\epsilon), D(f(\mathbf{x_0}))\}$. Let $\mathcal{C} = \mathcal{C}(D(f(\mathbf{x_0})),\epsilon)$.

Assume $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N\epsilon$. By (8.6) and (8.7), for $n \geq -1$ we have

$$\begin{aligned} T_{\texttt{univ}}(D_n, 2^n\epsilon) &\leq \mathcal{C} + 16\left(\frac{M_\nu(5\cdot 2^n\epsilon/\mu)^{\frac{1+\nu}{d}}}{2^n\epsilon}\right)^{\frac{2}{1+3\nu}} \\ &= \mathcal{C} + 16\left(\frac{M_\nu(5/\mu)^{\frac{1+\nu}{d}}}{\epsilon^{1-\frac{1+\nu}{d}}}\right)^{\frac{2}{1+3\nu}} \left(\frac{1}{2^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}}\right)^n\;. \end{aligned} \tag{8.11}$$

Substituting this into the bound (8.1) of Theorem 8.1 establishes the implication (8.8)

For $d > 1 + \nu$, observe

$$\begin{aligned} \sum_{n=-1}^{\bar{N}} \left(\frac{1}{2^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}}\right)^n &< \min\left\{\sum_{n=-1}^{\infty}\left(\frac{1}{2^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}}\right)^n, \bar{N}+5\right\} \\ &= \min\left\{\frac{4^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}}{2^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}}-1}, \bar{N}+5\right\}\;, \end{aligned} \tag{8.12}$$

where for the inequality we have used $1 \leq 2^{(1-\frac{1+\nu}{d})\frac{2}{1+3\nu}} \leq 4$. Substituting (8.11) into the bound (8.1) of Theorem 8.1, and then substituting (8.12) for the resulting summation, establishes the implication (8.9), concluding the proof in the case that $f(\mathbf{x_0}) - f^* < 5 \cdot 2^n\epsilon$.

To obtain time bounds when $f(\mathbf{x_0}) - f^* \geq 5 \cdot 2^N\epsilon$, then according to Theorem 8.1, simply substitute N for $\bar{N}$ in the bounds above, and add

$$T_{\texttt{univ}}\big(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon\big)\;,$$

which due to (8.6), is bounded above by (8.10). □

9. Numerical Experiments

In this section, we present numerical experiments giving insight into the behavior of the proposed parallel restarting schemes (`Sync` ‖ `FOM` and `Async` ‖ `FOM`) utilizing either `subgrad`, `accel`, or `smooth`. Implementations of both of these methods are available at [12] in a Jupyter notebook (implemented in Julia).

9.1. Restarting `subgrad` and `smooth` for piecewise linear optimization.

Our first experiments consider minimizing a generic piecewise linear convex function

$$\min_{x\in\mathbb{R}^n} \max\{a_i^T x - b_i \mid i = 1,\dots,m\} \quad \text{where } a_i \in \mathbb{R}^n, b_i \in \mathbb{R}\,. \tag{9.1}$$

We use a synthetic problem instance with $m = 2000$ and $n = 100$ constructed by sampling each $a_i \in \mathbb{R}^n$ from a standard Gaussian distribution and each $b_i \in \mathbb{R}$ from a unit Poisson distribution. In the following subsections, we examine the performance of restarting `subgrad`, `smooth`, and `accel`. We set a target accuracy of $\epsilon = 0.002$, $x_0 = (1,\dots,1)$, and $N = 14$, which results in using 16 instances of the given first-order method.

First we apply the subgradient method `subgrad` to this piecewise linear problem. Notice that the definition of `subgrad` in (1.2) is only parameterized by ϵ. Hence applying `Sync`‖`FOM` here does not require any form of parameter guessing or tuning to run.

We apply two variations of `Sync`‖`FOM`: first as defined in Section 4 where each instance sends messages with its current iterate to consider for restarting to the next instance whenever it restarts, and second where each instance sends messages to all other instances each iteration. Figure 1 shows the performance of both variation of `Sync`‖`FOM` (in the top left and top right plots respectively) over the course of 800 iterations as well as the performance (in the bottom plot) of `subgrad` without restarting applied with the same values of ϵ used throughout `Sync`‖`FOM`.

We see that each `subgrad` instance (whether run with `Sync`‖`FOM` or alone) flattens out at an objective gap less than its target objective gap of 0.001×2^k. Each instance of the subgradient method converges to its target accuracy faster when applied in the restarting scheme, most notably when ϵ is small. Hence applying our restarting scheme to the subgradient method gives better results than any tuning of the stepsize (controlled by choosing ϵ) could. Moreover, the `Sync`‖`FOM` variation having every instance message every instance further improves the restarting schemes accuracy reached by another order of magnitude.

This piecewise linear problem can be solved more efficiently via the smoothing discussed in Section 2.1. For any $\eta > 0$, consider

$$\min_{x\in\mathbb{R}^n} \eta \ln\left(\sum_{i=1}^{m} \exp((a_i^T x - b_i)/\eta)\right)\,. \tag{9.2}$$

This a $(\alpha,\beta) = (\max_{i,j}\{a_{ij}^2\}, \ln(m))$-smoothing of (9.1) (see [2]). Then `smooth`(ϵ) solves (9.1) by applying the accelerated method to (9.2) with $\eta = \epsilon/3\beta$, which has a α/η-smooth objective. Since α and β are both known, no parameter guessing or tuning is required to run `Sync`‖`FOM` with `smooth`.

Figure 2 shows the objective gap of each `smooth` instance employed by `Sync`‖`FOM` (as defined in Section 4 in the top left plot and with additional message passing in the top right) and the objective gap of each `smooth` instance without restarting (in the bottom plot). Much like the subgradient method, each `smooth` instance in our scheme strictly dominates the performance of

[12] https://github.com/bgrimmer/Parallel-Restarting-Scheme

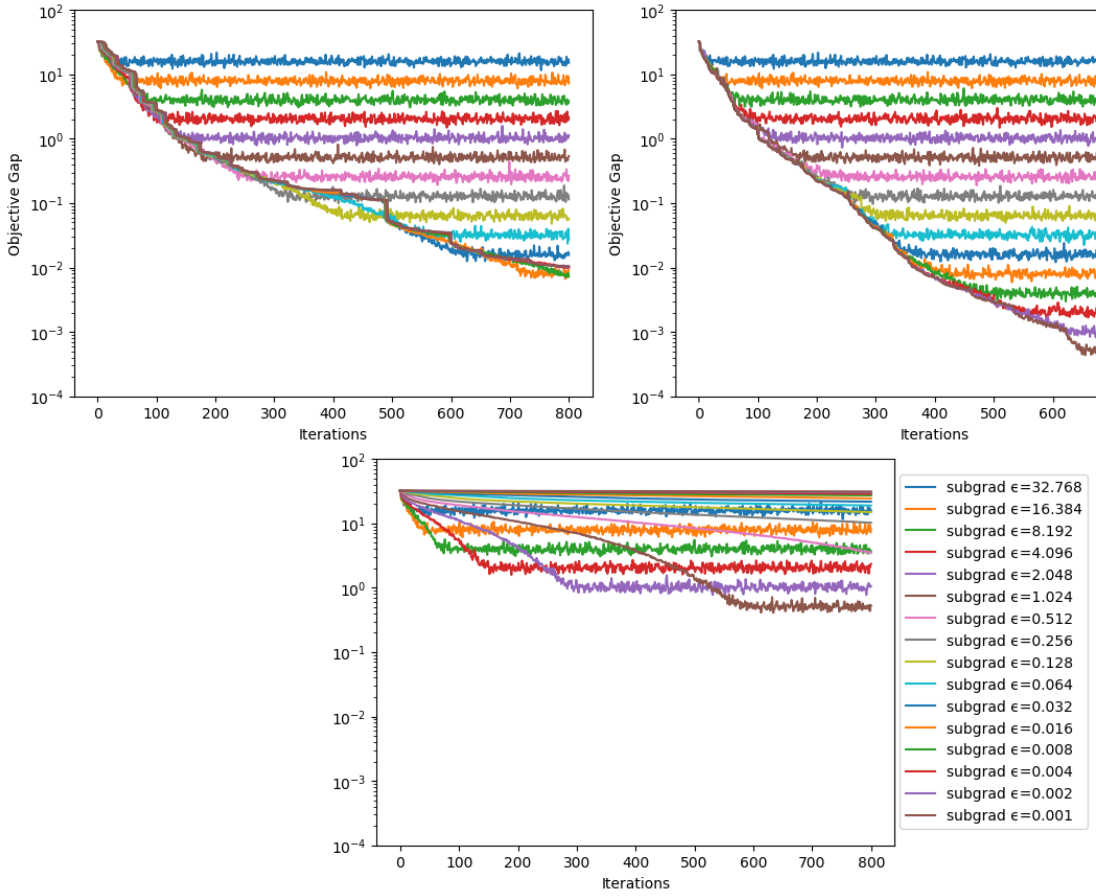

FIGURE 1. The top left plot shows the minimum objective gap of (9.1) seen by each `subgrad` instance in `Sync`‖`FOM` with target accuracies $\epsilon = 0.001 \times 2^k$. The top right plot shows improved convergence of `Sync`‖`FOM` when $\texttt{fom}_n$ broadcasts its restarts to everyone, not just to $\texttt{fom}_{n-1}$. The bottom plot shows the slower convergence of these methods without restarting.

its counterpart without restarting and adding additional message passing to `Sync`‖`FOM` notably speeds up the convergence. (That .0001 accuracy is achieved – whereas the target is .002 – is due to the choice $\eta = \epsilon/3\beta$, which always guarantees an ϵ-optimal solution will be reached, but for this numerical example happened to lead to even greater accuracy.)

9.2. **Restarting `accel` for least squares optimization.** Now consider solving a standard least squares regression problem defined by

$$\min_{x \in \mathbb{R}^n} \frac{1}{2m} \|Ax - b\|_2^2 \tag{9.3}$$

for some $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. We sample the A and $x^* \in \mathbb{R}^n$ from standard Gaussian distributions with $m = 2000$ and $n = 1000$ and then set $b = Ax^*$. We initialize our experiments with $x_0 = (0, \ldots, 0)$, $N = 30$ and a target accuracy of $\epsilon = 10^{-9}$.

We solve this problem with both `Sync` ‖ `FOM` and `Async` ‖ `FOM` using 32 parallel versions of `accel` (and without any additional message passing). This problem is known to be smooth with a Lipschitz constant L given by the maximum eigenvalue of $A^T A$. Moreover, this problem

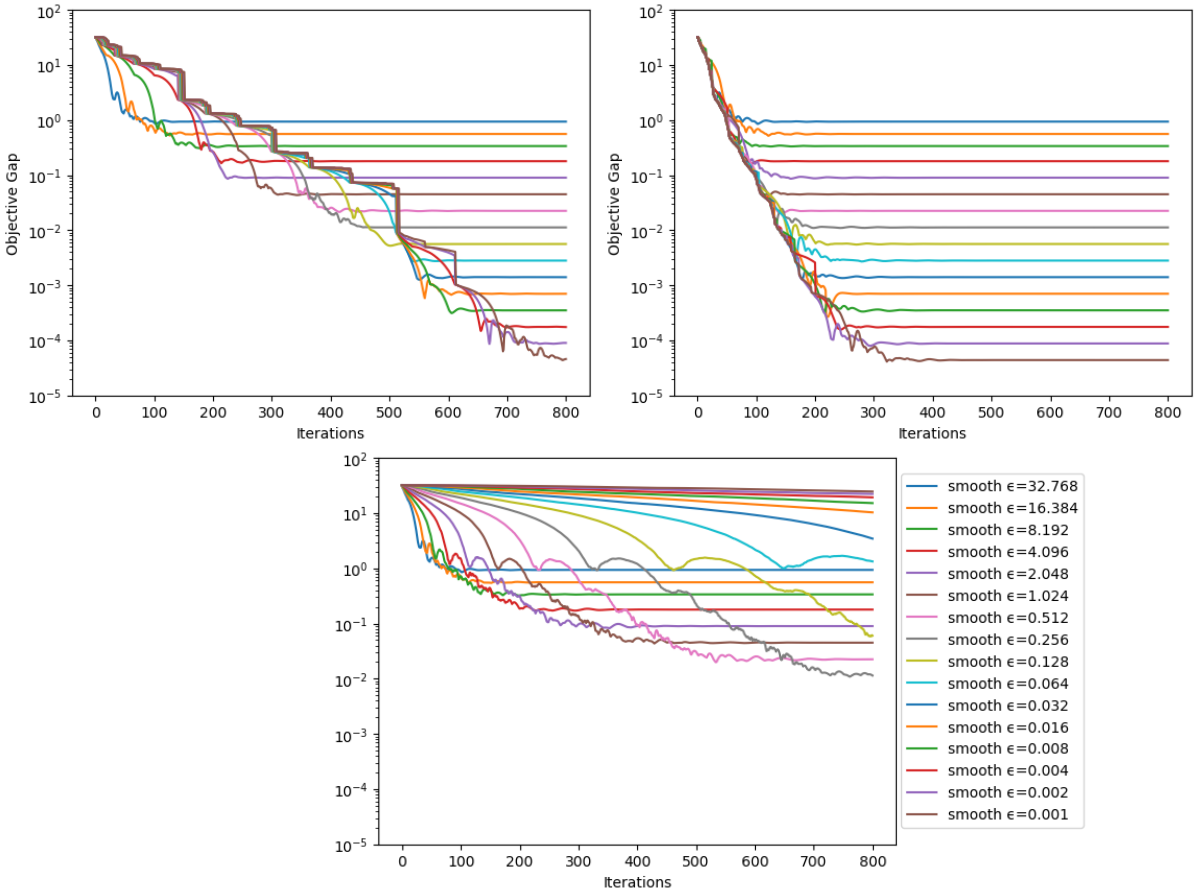

FIGURE 2. The top left plot shows the minimum objective gap of (9.1) seen by each `smooth` instance in `Sync`‖`FOM` with target accuracies $\epsilon = 0.001 \times 2^k$. The top right plot shows improved convergence of `Sync`‖`FOM` when $\texttt{fom}_n$ broadcasts its restarts to everyone, not just to $\texttt{fom}_{n-1}$. The bottom plot shows the slower convergence of these methods without restarting.

satisfies quadratic growth (Hölder growth with $d = 2$) with coefficient μ given by the minimum eigenvalue of $A^T A$. Hence this setting is amenable to applying our restarting scheme and should have the convergence rate improved from the sublinear convergence rate of $O(\sqrt{L/\epsilon})$ to the linear convergence rate of $O(\sqrt{L/\mu}\log(1/\epsilon))$.

The results of applying the both the synchronous and asynchronous schemes are shown in Figure 3. Since the 32 plotted curves heavily overlap and use similar colors, we omit the legend from our plot. Our implementation of `Async`‖`FOM` launches 32 threads using the multi-core, distributed processing libraries in Julia (v0.7.0), which each execute one of the schemes 32 parallel versions of the accelerated method. This code is run on an eight-core Intel i7-6700 CPU, and so multiple $\texttt{accel}_n$ are indeed able to execute and send messages concurrently. Message passing between processes is done using the RemoteChannel object provided by Julia.

This experiment shows both restarting schemes converge linearly to the target accuracy of $\epsilon = 10^{-9}$ within the first 2000 iterations. Since the first algorithm $\texttt{accel}_{30}$ never restarts, we see that the accelerated method without restarting only reaches an accuracy of approximately 10^{-3} by iteration 2000. Observe that each $\texttt{accel}_n$ slows down to a sublinear rate after it reaches its

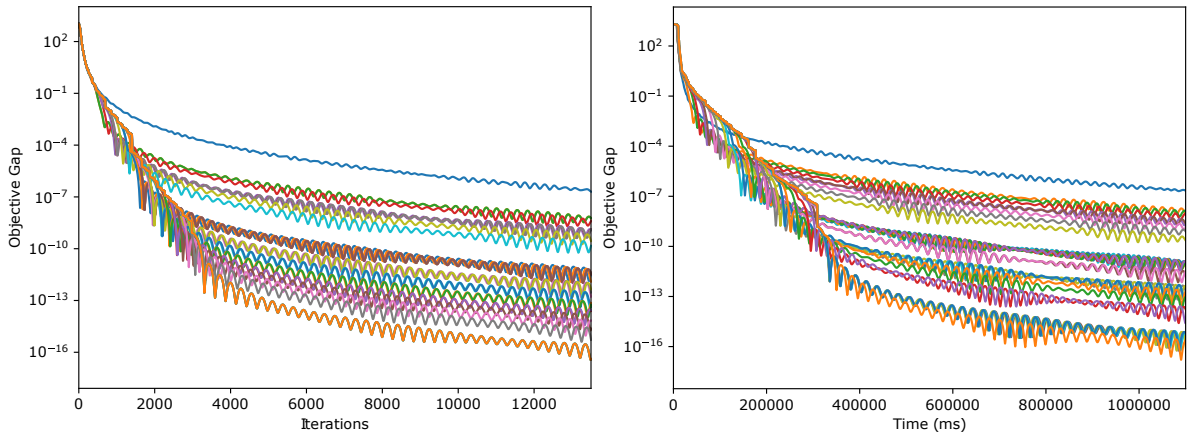

FIGURE 3. The left plot shows the objective gap of (9.3) seen by each `accel` instance used by `Sync||FOM` over 13000 iterations. The right plot shows `Async||FOM` in real time as each `accel` instances completes approximately 13000 iterations on an 8-core machine. The highest curve in both plots corresponds to $\texttt{accel}_{30}$ and the lowest curve corresponds to $\texttt{accel}_{-1}$ (and everything in between roughly follows this ordering).

target objective gap of $2^n\epsilon$, which corresponds to when that algorithm has restarted for the last time (and will hence not longer benefit from any messages it receives). This behavior matches what our convergence theory predicts (namely, linear convergence to an accuracy of $2^n\epsilon$ followed by the accelerated method's standard $O(\sqrt{L/\epsilon})$ rate).

The results from the asynchronous scheme are fairly similar to those of the synchronous scheme and agree with our theory's predictions. We remark that `Async||FOM` is fundamentally nondeterministic since there is a race condition in when each first-order method will complete its assigned task and when messages will be received. Hence, the results in the right side of Figure 3 only show one possible outcome of running `Async||FOM`, although we found that the general shape of the plot is consistent across many applications of the method.

REFERENCES

[1] 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.

[2] Amir Beck and Marc Teboulle. Smoothing and first order methods: A unified framework. *SIAM Journal on Optimization*, 22(2):557–580, 2012.

[3] Jérôme Bolte, Aris Daniilidis, and Adrian Lewis. The Łojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems. *SIAM Journal on Optimization*, 17(4):1205–1223, 2007.

[4] Jérôme Bolte, Trong Phong Nguyen, Juan Peypouquet, and Bruce W Suter. From error bounds to the complexity of first-order descent methods for convex functions. *Mathematical Programming*, 165(2):471–507, 2017.

[5] Olivier Fercoq and Zheng Qu. Restarting accelerated gradient methods with a rough strong convexity estimate. *arXiv preprint arXiv:1609.07358*, 2016.

[6] Olivier Fercoq and Zheng Qu. Adaptive restart of accelerated gradient methods under local quadratic growth condition. *IMA Journal of Numerical Analysis*, 39(4):2069–2095, 2019.

[7] Andrew Gilpin, Javier Pena, and Tuomas Sandholm. First-order algorithm with $\mathcal{O}(\ln(1/\epsilon))$ convergence for ϵ-equilibrium in two-person zero-sum games. *Mathematical Programming*, 133:279–298, 2010.

[8] Pontus Giselsson and Stephen Boyd. Monotonicity and restart in fast gradient methods. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 5058–5063. IEEE, 2014.
[9] Jean-Louis Goffin. On convergence rates of subgradient optimization methods. *Mathematical Programming*, 13(1):329–347, 1977.
[10] Anatoli Iouditski and Yurii Nesterov. Primal-dual subgradient methods for minimizing uniformly convex functions. *arXiv:1401.1792*, 2014.
[11] Patrick R Johnstone and Pierre Moulin. Faster subgradient methods for functions with hölderian growth. *Mathematical Programming*, 180(1):417–450, 2020.
[12] Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer, 2016.
[13] Qihang Lin and Lin Xiao. An adaptive accelerated proximal gradient method and its homotopy continuation for sparse optimization. In *International Conference on Machine Learning*, pages 73–81, 2014.
[14] Stanislas Łojasiewicz. Sur la géométrie semi-et sous-analytique. In *Annales de l'institut Fourier*, volume 43, pages 1575–1595, 1993.
[15] Stanislaw Lojasiewicz. Une propriété topologique des sous-ensembles analytiques réels. *Les équations aux dérivées partielles*, 117:87–89, 1963.
[16] Ion Necoara, Yurii Nesterov, and Francois Glineur. Linear convergence of first order methods for non-strongly convex optimization. *Mathematical Programming*, pages 1–39, 2016.
[17] Arkadi Nemirovski and Yurii Nesterov. Optimal methods of smooth convex minimization. *U.S.S.R. Comput. Math. Math. Phys.*, 25(2):21–30, 1985.
[18] Arkadi Nemirovski and David Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley, 1983.
[19] Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152, 2005.
[20] Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.
[21] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.
[22] Yurii Nesterov. Smoothing technique and its applications in semidefinite optimization. *Mathematical Programming*, 110(2):245–259, 2007.
[23] Yurii Nesterov. Universal gradient methods for convex optimization problems. *Mathematical Programming*, 152(1-2):381–404, 2015.
[24] Brendan ODonoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3):715–732, 2015.
[25] Boris T Polyak. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, 1963.
[26] Boris T Polyak. Subgradient methods: a survey of Soviet research. In *Nonsmooth optimization: Proceedings of the IIASA workshop*, pages 5–30, 1977.
[27] Boris T Polyak. Introduction to optimization. translations series in mathematics and engineering. *Optimization Software*, 1987.
[28] James Renegar. "Efficient" subgradient methods for general convex optimization. *SIAM Journal on Optimization*, 26:2649–2676, 2016.
[29] James Renegar. Accelerated first-order methods for hyperbolic programming. *Mathematical Programming*, pages 1–35, 2017.
[30] Vincent Roulet and Alexandre d'Aspremont. Sharpness, restart and acceleration. In *Advances in Neural Information Processing Systems*, pages 1119–1129, 2017.
[31] Naum Z Shor. *Minimization Methods for Non-Differentiable Functions*. Springer, 1985.
[32] Paul Tseng. On accelerated proximal gradient methods for convex-concave optimization. *submitted to SIAM Journal on Optimization*, 2:3, 2008.
[33] Tianbao Yang. Adaptive accelerated gradient converging methods under Hölderian error bound condition. In *31st Conference on Neural Information Processing System*, 2017.
[34] Tianbao Yang and Qihang Lin. RSG: Beating subgradient method without smoothness and strong convexity. *The Journal of Machine Learning Research*, 19(1):236–268, 2018.

Appendix A. **Complexity of Subgradient Method**

Here we record the simple proof of the complexity bound for `subgrad` introduced in §2.1, namely, that if f is M-Lipschitz on Q, then for the function

$$K_{\texttt{subgrad}}(\delta, \epsilon) := (M\delta/\epsilon)^2 \,, \tag{A.1}$$

the iterates $x_{k+1} = P_Q(x_k - \frac{\epsilon}{\|g_k\|^2} g_k)$ satisfy

$$\text{dist}(x_0, X^*) \leq \delta \quad \Rightarrow \quad \min\{f(x_k) \mid 0 \leq k \leq K_{\texttt{subgrad}}(\delta, \epsilon)\} \;\leq\; f^* + \epsilon \,. \tag{A.2}$$

Indeed, letting x^* be the closest point in X^* to x_k, and defining $\alpha_k = \epsilon/\|g_k\|^2$, we have

$$\begin{aligned}
\text{dist}(x_{k+1}, X^*)^2 &\leq \|x_{k+1} - x^*\|^2 = \|P(x_k - \alpha_k g_k) - x^*\|^2 \leq \|(x_k - \alpha_k g_k) - x^*\|^2 \\
&= \|x_k - x^*\|^2 - 2\alpha_k \langle g_k, x_k - x^* \rangle + \alpha_k^2 \|g_k\|^2 \\
&\leq \|x_k - x^*\|^2 - 2\alpha_k (f(x_k) - f^*) + \alpha_k^2 \|g_k\|^2 \quad \text{(because } g_k \text{ is a subgradient at } x_k\text{)} \\
&\leq \text{dist}(x_k, X^*)^2 - (\, 2(f(x_k) - f^*) - \epsilon \,)\, \epsilon/M^2 \,,
\end{aligned}$$

the final inequality due to $\alpha_k = \epsilon/\|g_k\|^2$, $\|g_k\| \leq M$ and the definition of x^*. Thus,

$$f(x_k) - f^* > \epsilon \;\Rightarrow\; \text{dist}(x_{k+1}, X^*)^2 < \text{dist}(x_k, X^*)^2 - (\epsilon/M)^2 \,.$$

Consequently, by induction, if x_{k+1} is the first iterate satisfying $f(x_{k+1}) - f^* \leq \epsilon$, then

$$0 \leq \text{dist}(x_{k+1}, X^*)^2 < \text{dist}(x_0, X^*)^2 - (k+1)(\epsilon/M)^2 \,,$$

implying the function $K_{\texttt{subgrad}}$ defined by (A.1) satisfies (A.2).

Appendix B. **Smoothing**

Here we present the first-order method `smooth` which is applicable when the objective function has a smoothing. We also justify the claim that $K_{\texttt{smooth}}(\delta, \epsilon)$, as defined by (2.9), provides an appropriate bound for the number of iterations.

We begin with an observation regarding the function $K_{\texttt{fom}}(\delta, \epsilon)$ for bounding the number of iterations of `fom` in terms of the desired accuracy ϵ and the distance from the initial iterate to X^*, the set of optimal solutions: Occasionally in analyses of first-order methods, the value $K_{\texttt{fom}}(\delta_\epsilon, \epsilon)$ is shown to be an upper bound, where δ_ϵ is the distance from the initial iterate to $X(\epsilon) := \{x \in Q \mid f(x) \leq \epsilon\}$. Since $\delta_\epsilon < \delta$, the latter upper bound is tighter. Particularly notable in focusing on $\text{dist}(x_0, X(\epsilon))$ rather than X^* is the masterpiece [32] of Paul Tseng.[13]

In many other cases, even though the analysis is done in terms of $\text{dist}(x_0, X^*)$, easy modifications can easily be made to obtain an iteration bound in terms of ϵ (the desired accuracy) and $\text{dist}(x_0, X(\epsilon'))$ for ϵ' satisfying $0 < \epsilon' < \epsilon$. This happens to be the case for the analysis of `fista` in [1] (a special case being `accel`, which we rely upon for `smooth`.

In particular, small modifications to the proof of [1, Thm 4.4] show that `fista` computes an ϵ-optimal solution within a number of iterations not exceeding

$$2\,\text{dist}(x_0, X(\epsilon/2))\sqrt{L/\epsilon} \,. \tag{B.1}$$

We provide specifics in a footnote.[14]

Recall the following definition:

[13] This work can easily be found online, although it was never published (presumably a consequence of Tseng's disappearance).

[14] In the proof of [1, Thm 4.4], x^* designates any optimal solution. However, no properties of optimality are relied upon other than $f(x^*) = f^*$ (including not being relied upon in [1, Lemma 4.1], to which the proof refers). Instead choose x^* to be the point in $X(\epsilon/2)$ which is nearest to x_0, and everywhere replace f^* by $f^* + \epsilon/2$.

An "(α, β) smoothing of f" is a family of functions f_η parameterized by $\eta > 0$, where for each η, the function f_η is smooth on a neighborhood of Q and satisfies
(1) $\|\nabla f_\eta(x) - \nabla f_\eta(y)\| \le \frac{\alpha}{\eta}\|x - y\|$, $\quad \forall x, y \in Q$.
(2) $f(x) \le f_\eta(x) \le f(x) + \beta\eta$, $\quad \forall x \in Q$,

To make use of an (α, β)-smoothing of a nonsmooth objective f when the goal is to obtain an ϵ-optimal solution (for f), we rely on `accel` (or more generally, `fista`) applied to f_η with η chosen appropriately. In particular, let the algorithm `smooth`(ϵ) for obtaining an ϵ-optimal solution of f be precisely the algorithm `accel`$(2\epsilon/3)$, applied to obtaining an $(2\epsilon/3)$-optimal solution of $f_{\epsilon/3\beta}$ (i.e., $\eta = \epsilon/3\beta$). Note that the Lipschitz constant $L = \alpha/\eta$ is available for `accel` to use in computing iterates (assuming α is known, as it is for our examples in §2.1).

To understand these choices of parameters, first observe that by definition, `accel`$(2\epsilon/3)$ is guaranteed – for whatever smooth function to which it is applied – to compute an iterate which is an $(2\epsilon/3)$-optimal solution. Let x_k be such an iterate when the function is $f_{\epsilon/3\beta}$, that is

$$f_{\epsilon/3\beta}(x_k) - f^*_{\epsilon/3\beta} \le 2\epsilon/3 \,. \tag{B.2}$$

Now observe that letting x^* be a minimizer of f, we have

$$f^*_{\epsilon/3\beta} \le f_{\epsilon/3\beta}(x^*) \le f(x^*) + \beta(\epsilon/3\beta) = f^* + \epsilon/3 \,.$$

Substituting this and $f(x_k) \le f_{\epsilon/3\beta}(x_k)$ into (B.2) gives, after rearranging, $f(x_k) - f^* \le \epsilon$, that is, `smooth`$(\epsilon)$ will have computed an ϵ-optimal solution for f.

Consequently, the number of iterations required by `smooth` to obtain an ϵ-optimal solution for f is no greater than the number of iterations for `accel` to compute an $(2\epsilon/3)$-optimal solution of $f_{\epsilon/3\beta}$, and hence per the discussion giving the iteration bound (B.1), does not exceed

$$2\,\mathrm{dist}(x_0, X_{\epsilon/3\beta}(\epsilon/3)\sqrt{(3\alpha\beta/\epsilon)(2\epsilon/3)} \,, \tag{B.3}$$

where $X_{\epsilon/3\beta}(\epsilon/3) = \{x \in Q \mid f_{\epsilon/3\beta}(x) - f^*_{\epsilon/3\beta} \le \epsilon/3\}$, and where we have used the fact that the Lipschitz constant of $\nabla f_{\epsilon/3\beta}$ is $\alpha/(\epsilon/3\beta)$. We claim, however, $X^* \subseteq X_{\epsilon/3\beta}(\epsilon/3)$, which with (B.3) results in the iteration bound

$$K_{\texttt{smooth}}(\delta, \epsilon) = 3\delta\sqrt{2\alpha\beta}/\epsilon$$

for `smooth` to compute an ϵ-optimal solution for f, where δ is the distance from the initial iterate to X^*, the set of optimal solutions for f.

Finally, to verify the claim, letting x^* be any optimal point for f, we have

$$f_{\epsilon/3\beta}(x^*) \le f(x^*) + \beta(\epsilon/3\beta) = f^* + \epsilon/3 \le f^*_{\epsilon/2\beta} + \epsilon/3 \,,$$

as desired.

Also make the trivial observation that the conclusion to [1, Lemma 4.2] remains valid even if the scalars α_i are not sign restricted, so long as all of the scalars β_i are non-negative (whereas the lemma as stated in the paper starts with the assumption that all of these scalars are positive).

With these changes, the proof of [1, Thm 4.4] shows that the iterates x_k of `fista` satisfy

$$f(x_k) - (f^* + \epsilon/2) \le \frac{2L\mathrm{dist}(x_0, X(\epsilon/2))^2}{(k+1)^2} \,.$$

Thus, to obtain an ϵ-optimal solution, (B.1) iterations suffice.

Appendix C. **Proof of Theorem 8.1**

The proof proceeds through a sequence of results analogous to the sequence in the proof of Theorem 5.1, although bookkeeping is now more pronounced. As for the proof of Theorem 5.1, we refer to $\texttt{fom}_n$ "updating" at a point x. The starting point $\mathbf{x_0}$ is considered to be the first update point for every $\texttt{fom}_n$. After $\texttt{Async}\|\texttt{FOM}$ has started, then for $n < N$, updating at x means the same as restarting at x, and for $\texttt{fom}_N$, updating at x is the same as having computed an iterate x satisfying the current task of $\texttt{fom}_N$ (in which case the point is sent to $\texttt{fom}_{N-1}$, even though $\texttt{fom}_N$ does not restart).

Keep in mind that when a new epoch for $\texttt{fom}_n$ occurs, if a message from $\texttt{fom}_{n+1}$ arrives in the inbox exactly when the epoch begins, then the restart point will not be decided immediately, due to $\texttt{fom}_n$ being made to pause, possibly contiguously. In any case, the restart point is decided after only a finite amount of time, due to $\texttt{fom}_{n+1}$ sending at most finitely many messages in total.[15]

Proposition C.1. *Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* \geq 2 \cdot 2^n\epsilon$. Then $\texttt{fom}_n$ updates again, and does so no later than time*

$$t + m\, t_{\text{pause}} + T_{\texttt{fom}}(D(f(x)), 2^n\epsilon) \,, \tag{C.1}$$

where m is the number of messages received by $\texttt{fom}_n$ between the two updates.

Proof: Let $\widetilde{m}$ be the total number of messages received by $\texttt{fom}_n$ from time zero onward. We know $\widetilde{m}$ is finite. Consequently, at time

$$t + \widetilde{m}\, t_{\text{pause}} + T_{\texttt{fom}}(D(f(x)), 2^n\epsilon) \,, \tag{C.2}$$

$\texttt{fom}_n$ will have devoted at least $T_{\texttt{fom}}(D(f(x)), 2^n\epsilon)$ units of time to computing iterations. Thus, if $\texttt{fom}_n$ has not updated before time (C.2), then at that instant, $\texttt{fom}_n$ will not be in a pause, and will have obtained $\bar{x}$ satisfying

$$f(\bar{x}) - f^* \leq 2^n\epsilon \quad \text{– thus, } f(\bar{x}) \leq f(x) - 2^n\epsilon \text{ (because } f(x) - f^* \geq 2 \cdot 2^n\epsilon) \,. \tag{C.3}$$

Consequently, if $\texttt{fom}_n$ has not updated by time (C.2), it will update at that time, at $\bar{x}$.

Having proven that after updating at x, $\texttt{fom}_n$ will update again, it remains to prove the next update will occur no later than time (C.1). Of course the next update occurs at (or soon after) the start of a new epoch. Let m_1 be the number of messages received by $\texttt{fom}_n$ after updating at x and before the new epoch begins, and let m_2 be the number of messages received in the new epoch and before $\texttt{fom}_n$ updates (i.e., before the restart point is decided). (Thus, from the beginning of the new epoch until the update, $\texttt{fom}_n$ is contiguously paused for an amount of time not exceeding $m_2\, t_{\text{pause}}$.) Clearly, $m_1 + m_2 = m$.

In view of the preceding observations, to establish the time bound (C.1) it suffices to show the new epoch begins no later than time

$$t + m_1\, t_{\text{pause}} + T_{\texttt{fom}}(D(f(x)), 2^n\epsilon) \,. \tag{C.4}$$

However, if the new epoch has not begun prior to time (C.4) then at that time, $\texttt{fom}_n$ is not in a pause, and has spent enough time computing iterations so as to have a point $\bar{x}$ satisfying (C.3), causing a new epoch to begin instantly. □

[15] The number of messages sent by $\texttt{fom}_{n+1}$ is finite because (1) $\texttt{fom}_{n+1}$ sends a message only when it has obtained x satisfying $f(x) \leq f(x_{n+1,0}) - 2^{n+1}\epsilon$, where $x_{n+1,0}$ is the most recent (re)start point, and (2) we assume f^* is finite.

Proposition C.2. *Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* \geq 2 \cdot 2^n\epsilon$. Let $\mathbf{j} := \lfloor \frac{f(x)-f^*}{2^n\epsilon} \rfloor - 2$. Then $\texttt{fom}_n$ updates at a point $\bar{x}$ satisfying $f(\bar{x}) - f^* < 2 \cdot 2^n\epsilon$ no later than time*

$$t + m\, t_{\text{pause}} + \sum_{j=0}^{\mathbf{j}} T_{\texttt{fom}}(D(f(x) - j \cdot 2^n\epsilon), 2^n\epsilon) \ ,$$

where m is the total number of messages received by $\texttt{fom}_n$ between the updates at x and $\bar{x}$.

Proof: The proof is essentially identical to the proof of Proposition 6.2, but relying on Proposition C.1 rather than on Lemma 6.1. □

Corollary C.3. *Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* < \mathbf{i} \cdot 2^n\epsilon$, where $\mathbf{i}$ is an integer and $\mathbf{i} \geq 3$. Then $\texttt{fom}_n$ updates at a point $\bar{x}$ satisfying $f(\bar{x}) - f^* < 2 \cdot 2^n\epsilon$ no later than time*

$$t + m\, t_{\text{pause}} + \sum_{i=3}^{\mathbf{i}} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon)$$
$$\text{where } D_{n,i} = \min\{D(f^* + i \cdot 2^n\epsilon), D(f(x))\} \ , \tag{C.5}$$

and where m is the number of messages received by $\texttt{fom}_n$ between the updates at x and $\bar{x}$.

Moreover, if $n > -1$, then after sending the message to $\texttt{fom}_{n-1}$ containing the point $\bar{x}$, $\texttt{fom}_n$ will send at most one further message.

Proof: Except for the final assertion, the proof is essentially identical to the proof of Corollary 6.3, but relying on Proposition C.2 rather than on Proposition 6.2.

For the final assertion, note that if $\texttt{fom}_n$ sent two points after sending $\bar{x}$ - say, first x' and then x'' – we would have

$$f(x'') \leq f(x') - 2^n\epsilon \leq (f(\bar{x}) - 2^n\epsilon) - 2^n\epsilon < f^* \ ,$$

a contradiction. □

Corollary C.4. *Let $n > -1$. Assume that at time t, $\texttt{fom}_n$ updates at x satisfying $f(x) - f^* < 5 \cdot 2^n\epsilon$, and assume from time t onward, $\texttt{fom}_n$ receives at most $\hat{m}$ messages. Then for either $\hat{m}' = 0$ or $\hat{m}' = 1$, $\texttt{fom}_{n-1}$ updates at x' satisfying $f(x') - f^* < 5 \cdot 2^{n-1}\epsilon$ no later than time*

$$t + t_{\text{transit}} + (\hat{m} + 2 - \hat{m}')t_{\text{pause}} + \sum_{i=3}^{5} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon)$$

(where $D_{n,i}$ is given by (C.5)), and from that time onward, $\texttt{fom}_{n-1}$ receives at most $\hat{m}'$ messages.

Proof: By Corollary C.3, no later than time

$$t + \hat{m}t_{\text{pause}} + \sum_{i=3}^{5} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ ,$$

$\texttt{fom}_n$ updates at $\bar{x}$ satisfying $f(\bar{x}) - f^* < 2 \cdot 2^n\epsilon$. When $\texttt{fom}_n$ updates at $\bar{x}$, the point is sent to the inbox of $\texttt{fom}_{n-1}$, where it arrives no more than t_{transit} time units later, causing a pause of $\texttt{fom}_{n-1}$ to begin immediately. Moreover, following the arrival of $\bar{x}$, at most one additional message will be received by $\texttt{fom}_{n-1}$ (per the last assertion of Corollary C.3).

If during the pause, $\texttt{fom}_{n-1}$ does not receive an additional message, then at the end of the pause, $\texttt{fom}_{n-1}$ either updates at $\bar{x}$ or continues its current epoch. The decision on whether to

update at $\bar{x}$ is then enacted no later than time

$$t + t_{\text{transit}} + (\hat{m}+1)t_{\text{pause}} + \sum_{i=3}^{5} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ , \tag{C.6}$$

after which $\texttt{fom}_{n-1}$ receives at most one message.

On the other hand, if during the pause, an additional message is received, then a second pause immediately begins, and $\bar{x}$ is overwritten by a point $\bar{\bar{x}}$ satisfying $f(\bar{\bar{x}}) \leq f(\bar{x}) - 2^n\epsilon < f^* + 2^n\epsilon$. Here, the decision on whether to update at $\bar{\bar{x}}$ is enacted no later than time

$$t + t_{\text{transit}} + (\hat{m}+2)t_{\text{pause}} + \sum_{i=3}^{5} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ , \tag{C.7}$$

after which $\texttt{fom}_{n-1}$ receives no messages.

If $\texttt{fom}_{n-1}$ chooses to update at $\bar{x}$ (or $\bar{\bar{x}}$), the corollary follows immediately from (C.6) and (C.7), as the value of f at the update point is then less than $f^* + 4 \cdot 2^{n-1}\epsilon$. On the other hand, if $\texttt{fom}_{n-1}$ chooses not to update, it is only because its most recent update point satisfies $f(x_{n-1,0}) < f(\bar{x}) + 2^{n-1}\epsilon$ (resp., $f(x_{n-1,0}) < f(\bar{\bar{x}}) + 2^{n-1}\epsilon$), and hence $f(x_{n-1,0}) < f^* + 5 \cdot 2^{n-1}\epsilon$. Thus, here as well, the corollary is established. □

Corollary C.5. *For any $\bar{N} \in \{-1, \ldots, N\}$, assume that at time t, $\texttt{fom}_{\bar{N}}$ updates at x satisfying $f(x) - f^* < 5 \cdot 2^{\bar{N}}\epsilon$, and assume from time t onward, $\texttt{fom}_{\bar{N}}$ receives at most $\hat{m}$ messages. Then no later than time*

$$t + (\bar{N}+1)t_{\text{transit}} + (\hat{m} + 2\bar{N} + 2)t_{\text{pause}} + \sum_{n=-1}^{\bar{N}} \sum_{i=3}^{5} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ ,$$

$\texttt{Async}\|\texttt{FOM}$ *has computed an ϵ-optimal solution.*

Proof: If $\bar{N} = -1$, Corollary C.3 implies that after updating at x, the additional time required by $\texttt{fom}_{-1}$ to compute a $(2 \cdot 2^{-1}\epsilon)$-optimal solution $\bar{x}$ is bounded from above by

$$\hat{m}\, t_{\text{pause}} + \sum_{i=3}^{5} T_{\texttt{fom}}(D_{-1,i}, 2^{-1}\epsilon) \ . \tag{C.8}$$

The present corollary is thus immediate for the case $\bar{N} = -1$.

On the other hand, if $\bar{N} > -1$, induction using Corollary C.4 shows that for either $\hat{m}' = 0$ or $\hat{m}' = 1$, no later than time

$$t + (\bar{N}+1)t_{\text{transit}} + (\hat{m} + 2\bar{N} + 2 - \hat{m}')t_{\text{pause}} + \sum_{n=0}^{\bar{N}} \sum_{i=3}^{5} T_{\texttt{fom}}(D_{n,i}, 2^n\epsilon) \ ,$$

$\texttt{fom}_{-1}$ has restarted at x satisfying $f(x) - f^* < 5 \cdot 2^{-1}\epsilon$, and from then onwards, $\texttt{fom}_{-1}$ receives at most $\hat{m}'$ messages. Then by Corollary C.3, the additional time required by $\texttt{fom}_{-1}$ to compute a $(2 \cdot 2^{-1}\epsilon)$-optimal solution does not exceed (C.8) with $\hat{m}$ replaced by $\hat{m}'$. The present corollary follows. □

We are now in position to prove Theorem 8.1, which we restate as a corollary.

Corollary C.6. *If $f(\mathbf{x_0}) - f^* < 5 \cdot 2^N \epsilon$, then* `Async‖FOM` *computes an ϵ-optimal solution within time*

$$(\bar{N}+1)t_{\text{transit}} + 2(\bar{N}+2)t_{\text{pause}} + 3\sum_{n=-1}^{\bar{N}} T_{\texttt{fom}}\left(D_n, 2^n\epsilon\right) \tag{C.9}$$
$$\text{with } D_n := \min\{D(f^* + 5\cdot 2^n\epsilon), D(f(\mathbf{x_0}))\}\ ,$$

where $\bar{N}$ is the smallest integer satisfying both $f(\mathbf{x_0}) - f^* < 5\cdot 2^{\bar{N}}\epsilon$ and $\bar{N} \geq -1$.

In any case, `Async‖FOM` *computes an ϵ-optimal solution within time*

$$\mathbf{T}_N + T_{\texttt{fom}}\big(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon\big)\ ,$$

where $\mathbf{T}_N$ is the quantity obtained by substituting N for $\bar{N}$ in (C.9).

Proof: Consider the case $f(\mathbf{x_0}) - f^* < 5\cdot 2^N\epsilon$, and let $\bar{N}$ be as in the statement of the theorem, that is, the smallest integer satisfying both $f(\mathbf{x_0}) - f^* < 5\cdot 2^{\bar{N}}\epsilon$ and $\bar{N} \geq -1$.

If $\bar{N} = N$, then $\texttt{fom}_{\bar{N}}$ never receives messages, in which case the time bound (C.9) is immediate from Corollary C.5 with $t = 0$ and $\hat{m} = 0$. On the other hand, if $\bar{N} < N$, then since $f(\mathbf{x_0}) - f^* < \frac{5}{2}\cdot 2^{\bar{N}+1}\epsilon$, $\texttt{fom}_{\bar{N}+1}$ sends at most two messages from time zero onward. Again (C.9) is immediate from Corollary C.5.

Regardless of whether $f(\mathbf{x_0}) - f^* < 5\cdot 2^N\epsilon$, we know $\texttt{fom}_N$ – which never restarts – computes x satisfying $f(x) - f^* \leq 2^N\epsilon$ within time

$$T_{\texttt{fom}}(\text{dist}(\mathbf{x_0}, X^*), 2^N\epsilon)\ . \tag{C.10}$$

If x is not an update point for $\texttt{fom}_N$, then the most recent update point $\hat{x}$ satisfies $f(\hat{x}) < f(x) + 2^N\epsilon \leq f^* + 2\cdot 2^N\epsilon$. Hence, irrespective of whether x is an update point, by the time $\texttt{fom}_N$ has computed x, it has obtained an update point x' ($x' = \hat{x}$ if not $x' = x$) satisfying $f(x') - f^* < 5\cdot 2^N\epsilon$. Consequently, the time required for `Async‖FOM` to compute an ϵ-optimal solution does not exceed (C.10) plus the value (C.9), where in (C.9), N is substituted for $\bar{N}$. □

School of Operations Research and Information Engineering, Cornell University, Ithaca, NY, U.S.