Anderson Accelerated Asynchronous Method for Distributed Optimization

Anonymous authors
Paper under double-blind review

Abstract

Anderson acceleration (AA) is an effective technique for accelerating fixed-point iterations, but it is rarely applied to distributed optimization or distributed machine learning. In this paper, we apply AA to accelerate an asynchronous distributed gradient method over the master-worker architecture, resulting in the Asynchronous Distributed Gradient Method with Anderson Acceleration (ADGM-AA). In particular, we first transform the asynchronous gradient method into a fixed-point iteration, and then incorporate it with AA. To ensure the global convergence of ADGM-AA, we equip it with a novel reference-path-based safeguard scheme. We prove that under mild conditions, ADGM-AA converges with fixed step-sizes that are independent of the delays. Compared with the delay-dependent step-size in most existing works, our delay-free step-size is easier to determine and often leads to faster convergence. To emphasize, numerical experiments demonstrate that by incorporating the AA scheme, the proposed ADGM-AA significantly outperforms the vanilla asynchronous distributed gradient method.

1 Introduction

This paper focuses on consensus optimization over the master-worker architecture, where one master and n workers collaborate to solve

$$\min_{x\in\mathbb{R}^d} f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x). \tag{1}$$

In problem (1), each $f_i : \mathbb{R}^d \to \mathbb{R}$ is a local objective function and is known only by worker i. Problem (1) is popular in distributed learning, where each f_i represents a training loss, each worker is a set of GPUs, and multiple GPUs collaborate to train a model.

To scalably solve problem (1), numerous distributed optimization methods are proposed, in which the workers are responsible for most computations, and the master aggregates the computation results from the workers to update the iterate. A critical design choice in distributed methods is the coordination mechanism, leading to two categories of algorithms: synchronous and asynchronous. In synchronous settings, the master and workers must coordinate at each iteration to ensure all workers compute based on a consistent global iterate, which, however, often leads to significant idle time as faster workers wait for stragglers. In contrast, asynchronous algorithms allow the master and workers to proceed independently and update parameters using potentially stale information, which enhances scalability and resource efficiency but usually complicates convergence analysis.

1.1 Related work

In this work, we adapt Anderson acceleration to accelerate an asynchronous distributed optimization method. Consequently, our literature survey focuses on works on Anderson acceleration and asynchronous distributed optimization methods.

1.1.1 Anderson acceleration

Anderson acceleration (AA) is a technique for accelerating fixed-point iterations with historical information, and was first proposed by Donald G. Anderson to solve nonlinear integral equations in 1965 (Anderson, 1965). Later, AA is shown to have a quasi-Newton explanation and reduces to the GMRES method (Saad & Schultz, 1986) for solving linear equations (Walker & Ni, 2011). Due to its effectiveness, AA is applied to many research areas such as computational physics (Eyert, 1996), material sciences (Pulay, 1980), and computational chemistry (Pulay, 1982). Although most optimization methods can be formulated into a fixed-point iteration, the application of AA to optimization has occurred only in recent years (Bertrand & Massias, 2021; Li & Li, 2020; Liu et al., 2024; Mai & Johansson, 2020; Ouyang et al., 2020; Scieur et al., 2016). Specifically, the work (Bertrand & Massias, 2021) adapts AA to the coordinate descent method, the works (Li & Li, 2020; Mai & Johansson, 2020; Liu et al., 2024) apply AA to the proximal gradient method, the work (Li & Li, 2020) first incorporates AA with Chebyshev polynomials and then applies the resulting technique to the gradient descent method, the work (Liu et al., 2024) uses AA to accelerate an energy-adaptive gradient method, and the work (Ouyang et al., 2020) applies AA to the alternating direction method of multipliers (ADMM).

Despite the rich literature on AA, existing works only focus on centralized optimization, and the application of AA to distributed optimization remains unexplored.

1.1.2 Asynchronous distributed optimization

Due to the excellent scalability and high resource usage efficiency, asynchronous distributed optimization methods have attracted considerable interest, especially in the machine learning community (Aytekin et al., 2016; Cederberg et al., 2025; Chraibi et al., 2024; Liu et al., 2014; Mishchenko et al., 2018; Wu et al., 2023; 2022; Zhang & Kwok, 2014). These methods can be further categorized based on their communication topology: they either follow a master-worker architecture or operate over a decentralized network. Below, we survey works that consider problem (1) over the master-worker architecture, which is more closely related to this paper.

For asynchronous optimization methods over the master-worker architecture, a large body of works consider deterministic-gradient algorithms such as the asynchronous (proximal) incremental aggregated gradient (Aytekin et al., 2016; Feyzmahdavian & Johansson, 2023; Sun et al., 2019; Vanli et al., 2016), the asynchronous coordinate descent method (Liu et al., 2014; Wu et al., 2023; 2022), the asynchronous bundle methods (Cederberg et al., 2025), and the asynchronous alternating direction method of multipliers (Zhang & Kwok, 2014). In parallel, substantial progress has been made in stochastic optimization, such as the asynchronous stochastic gradient descent method (Leblond et al., 2017; Ma et al., 2023; Recht et al., 2011; Zhao & Li, 2016). Due to the asynchronous implementation, information used in the updates is usually delayed. To guarantee the convergence, most existing asynchronous distributed optimization methods (Aytekin et al., 2016; Cederberg et al., 2025; Feyzmahdavian et al., 2014; Feyzmahdavian & Johansson, 2023; Liu et al., 2014; Vanli et al., 2016; Sun et al., 2019; Zhang & Kwok, 2014) assume that delays are bounded and require the algorithm step-size to rely on and decrease with the delay bound, which often leads to overly small step-sizes and slow convergence due to the usually large upper bound of delays. For example, the work (Mishchenko et al., 2018) implements an asynchronous distributed gradient method over a master-worker architecture with 40 nodes and reports the maximum delay of over 1200. Exceptions include some works (Chraibi et al., 2024; Mishchenko et al., 2018; Wu et al., 2023; Cederberg et al., 2025), which converge with fixed step-sizes independent of the delays. More precisely, the seminal work (Mishchenko et al., 2018) proposes an asynchronous proximal gradient algorithm called DAve-RPG, which inspires the remaining methods, including an asynchronous proximal bundle method (Cederberg et al., 2025), a Bregman variant of the DAve-RPG method (Chraibi et al., 2024), and an asynchronous coordinate update method (Wu et al., 2023).

1.2 Contribution

We aim to apply the effective AA technique to distributed optimization. In particular, since DAve-RPG converges with fixed delay-independent step-sizes and inspires a series of subsequent works, we choose to accelerate its variant for smooth problems (DAve-G) with AA. The algorithm development faces two primary

challenges: first, the straightforward application of AA to DAve-G cannot be implemented in a distributed way; second, to guarantee the global convergence, we need to design a safe-guard scheme, while it is particularly challenging in the asynchronous and distributed setting.

Despite the aforementioned challenges, we incorporate AA into DAve-G in a novel way, which gives an Asynchronous Distributed Gradient Method with Anderson Acceleration (ADGM-AA). Our main contributions include

1) We rewrite DAve-G as a fixed-point iteration in a novel way and follow the idea of AA (rather than directly using it) to accelerate the fixed-point iteration, which allows the resulting algorithm to be implemented in an asynchronous and distributed way.
2) We propose a novel reference-path-based safe-guard scheme that are distributively implementable.
3) We theoretically prove the convergence of ADGM-AA with the safe-guard scheme and fixed step-sizes independent of delays.

The effectiveness of ADGM-AA is demonstrated numerically via classification tasks with real-world datasets.

1.3 Outline

The remaining part of this paper is organized as follows. Section 2 introduces Anderson acceleration and the asynchronous distributed method DAve-G. Section 3 develops ADGM-AA by integrating Anderson acceleration with DAve-G, and designs a reference-path-based safe-guard scheme to guarantee the global convergence of the proposed method. Section 4 analyses the convergence of ADGM-AA, and Section 5 presents our numerical experimental results. Finally, Section 6 concludes the paper.

1.4 Notation

Throughout this paper, we use $\|\cdot\|$ to denote the Euclidean norm for vectors and the spectral norm for matrices. For any closed convex set C, $P_C(\cdot)$ is the projection operator onto C. We use $\mathbf{1}$ to represent the all-one vector and $\mathbf{0}$ the all-zero vector. For any positive integer $m \geq 0$, we define

$$[m] = \{0, 1, \ldots, m-1\}.$$

2 Preliminaries: Anderson acceleration and DAve-G

In this section, we introduce Anderson acceleration (AA) and the asynchronous method DAve-G for solving problem (1), which inspires our algorithm development in Section 3.

2.1 Anderson acceleration

AA is a technique for accelerating fixed-point iterations of the following form

$$x_{k+1} = g(x_k) \tag{2}$$

where $g : \mathbb{R}^d \to \mathbb{R}^d$ is an operator. The update (2) solves the fixed point of g and describes many algorithms in optimization, such as steepest descent (Li & Li, 2020), and ADMM (Ouyang et al., 2020).

The vanilla AA consists of three steps. First, it considers an intermediate iterate of the following form

$$x_{k+\frac{1}{2}} = \sum_{t=0}^{m-1} \gamma_{t,k} x_{k-t},$$

where $m \geq 1$ is a positive integer, $t \in [m]$, and $\gamma_{t,k}$ are real numbers satisfying

$$\sum_{t=0}^{m-1} \gamma_{t,k} = 1.$$

Second, AA sets $\{\gamma_{t,k}\}_{t\in[m]}$ as the one that minimizes an approximation of the residual $\|g(x_{k+\frac{1}{2}}) - x_{k+\frac{1}{2}}\|$. In particular, for any $k \geq 0$, define $r_k = g(x_k) - x_k$,

$$R_k = (r_k, r_{k-1}, \ldots, r_{k-m+1}) \in \mathbb{R}^{d\times m},\ \Gamma_k = (\gamma_{0,k}, \gamma_{1,k}, \ldots, \gamma_{(m-1),k})^T \in \mathbb{R}^m. \tag{3}$$

The residual at $x_{k+\frac{1}{2}}$ can be approximated as

$$g(x_{k+\frac{1}{2}}) - x_{k+\frac{1}{2}} \approx \sum_{t=0}^{m-1} \gamma_{t,k}(g(x_{k-t}) - x_{k-t}) = R_k\Gamma_k, \tag{4}$$

and AA sets Γ_k as the one that minimizes the approximate residual

$$\min_{\Gamma:\Gamma^T\mathbf{1}=1} \|R_k\Gamma\|. \tag{5}$$

Note that when g is affine, the approximation (4) holds equal. Finally, AA approximates $g(x_{k+\frac{1}{2}})$ to improve the quality of $x_{k+\frac{1}{2}}$:

$$x_{k+1} = \sum_{t=0}^{m-1} \gamma_{t,k} g(x_{k-t}), \tag{6}$$

which uses the approximation

$$g(x_{k+\frac{1}{2}}) = g(\sum_{t=0}^{m-1} \gamma_{t,k} x_{k-t}) \approx \sum_{t=0}^{m-1} \gamma_{t,k} g(x_{k-t}). \tag{7}$$

Similar to (4), the approximation in (7) holds with equality when g is affine.

For affine g, AA could converge globally from an arbitrary initial point (Potra & Engler, 2013; Toth & Kelley, 2015; Walker & Ni, 2011). However, for non-affine g, due to the error arising in the approximations (4) and (7), AA could diverge unless the initial point is sufficiently close to the fixed point (Mai & Johansson, 2020). To guarantee global convergence of AA, additional safe-guard schemes are often required.

2.2 DAve-G

It is an asynchronous distributed gradient method for solving problem (1) over the master-worker architecture (Mishchenko et al., 2018) and the corresponding synchronous algorithm takes the form of

$$x_{k+1} = x_k - \frac{\alpha}{n}\sum_{i=1}^{n} \nabla f_i(x_k), \tag{8}$$

which is mathematically equivalent to the centralized steepest descent method[1]. To perform (8), at each iteration k, each worker i computes the gradient $\nabla f_i(x_k)$ and submits it to the master, and the master aggregates the gradients from all workers to update x_k. In this synchronous implementation, fast workers need to wait for slow workers, which causes low efficiency of computing resource usage.

As an asynchronous variant of (8), DAve-G updates as

$$x_{k+1} = \frac{1}{n}\sum_{i=1}^{n}(\hat{x}_k^i - \alpha g_k^i), \tag{9}$$

where $\hat{x}_k^i = x_{k-d_k^i}$ and $g_k^i = \nabla f_i(\hat{x}_k^i)$ for a non-negative integer $d_k^i \in [0, k]$ (referred to as delays). In the implementation of DAve-G, the master keeps the current iterate x (we ignore the iteration index for simplicity). Each worker i consecutively receives x from the master, and computes $y^i = x - \alpha\nabla f_i(x)$ and sends it to the master. After the master receives y^i from some (one or more) workers i, it performs $x_+ = \frac{1}{n}\sum_{i=1}^{n} y^i$, where, for each worker i, y^i is the most recent y^i the master received from it. This asynchronous update way causes higher computation resource usage than the synchronous method (8), but causes delays in the update. In particular, at the kth update, each y^i in the update equals to $x_{k-d_k^i} - \alpha\nabla f_i(x_{k-d_k^i})$, and the corresponding update formula is (9).

[1]DAve-G allows for multiple inner-loop iterations, and here we only consider its single-inner-loop version.

3 Algorithm development

This section applies AA to accelerate DAve-G. To this end, we first adapt the vanilla AA to DAve-G, and then design a novel reference-path-based safe-guard scheme to guarantee its global convergence.

3.1 Asynchronous distributed gradient method with Anderson acceleration

To apply AA to DAve-G, we need to rewrite (9) into a fixed-point iteration, which is not straightforward since the left-hand side is of d-dimension and the right-hand side involves n iterates of dimension d. To address this issue, we rewrite problem (1) as

$$\begin{aligned} \min \;\; & F(\mathbf{x}) = \sum_{i=1}^{n} f_i(x^i) \\ \text{s.t.} \;\; & \mathbf{x} \in \mathcal{C} \end{aligned} \tag{10}$$

where each $x^i \in \mathbb{R}^d$, $\mathbf{x} = ((x^1)^T, \ldots, (x^n)^T)^T \in \mathbb{R}^{nd}$ and $\mathcal{C} = \{\mathbf{x} \mid x^1 = x^2 = ... = x^n\}$. Then, (9) can be viewed as a projected steepest descent step for solving problem (10): letting $\mathbf{x}_{k+1} = ((x_{k+1})^T, \ldots, (x_{k+1})^T)^T$ where x_{k+1} is computed from (9), we have

$$\mathbf{x}_{k+1} = P_{\mathcal{C}}(\hat{\mathbf{x}}_k - \alpha \nabla F(\hat{\mathbf{x}}_k)), \tag{11}$$

where $\mathbf{x}_{k+1} = ((\hat{x}_k^1)^T, \ldots, (\hat{x}_k^n)^T)^T$, $\hat{x}_k^i = x_{k-d_k^i}$ $(i = 1, 2, \ldots, n)$ and $P_C(\cdot)$ represents the projection operation onto $\mathcal{C}$.

The update (11) is equivalent to the asynchronous fixed-point iteration

$$\mathbf{x}_{k+1} = \mathrm{T}(\hat{\mathbf{x}}_k), \tag{12}$$

where the operator $\mathrm{T}(\mathbf{x}) = P_{\mathcal{C}}(\mathbf{x} - \alpha \nabla F(\mathbf{x}))$ and it is easy to verify that for any fixed point $\mathbf{x}^\star$, it is a fixed point of T if and only if it is optimal to problem (10). Compared to the centralized update (2), (12) incorporates delayed information due to the asynchronous implementation. Moreover, for any fixed point $\mathbf{x}^\star = ((x^\star)^T, \ldots, (x^\star)^T)^T$ of (12), $x^\star$ is an optimal solution of problem (9).

Given the asynchronous fixed-point update (12), we are ready to incorporate AA with DAve-G. Fix $m > 0$ as an integer. For any $k \geq 0$, define

$$\mathbf{r}_k = \mathrm{T}(\hat{\mathbf{x}}_k) - \hat{\mathbf{x}}_k, \;\; \mathbf{R}_k = (\mathbf{r}_k, \ldots, \mathbf{r}_{k-m+1}), \tag{13}$$

where each $\mathbf{r}_k$ measures the optimality residual at $\hat{\mathbf{x}}_k$. Following (4) – (6), we consider a combination of $\{\hat{\mathbf{x}}_t\}_{t=k-m+1}^{k}$:

$$\mathbf{x}_{k+\frac{1}{2}} = \sum_{t=0}^{m-1} \gamma_{t,k} \hat{\mathbf{x}}_{k-t}, \tag{14}$$

and minimize an approximate optimality residual at $\mathbf{x}_{k+\frac{1}{2}}$ to determine $\gamma_{t,k}$: define $\Gamma_k = (\gamma_{0,k}, \ldots, \gamma_{m-1,k})^T$ and solve

$$\Gamma_k \in \arg \; \min\nolimits_{\Gamma \in \mathbb{R}^m : \Gamma^T \mathbf{1} = 1} \; \|\mathbf{R}_k \Gamma\|. \tag{15}$$

Finally, we set

$$\mathbf{x}_{k+1}^{\mathrm{AA}} = \sum_{t=0}^{m-1} \gamma_{t,k} \, \mathrm{T}(\hat{\mathbf{x}}_{k-t}), \tag{16}$$

which approximates $T(\mathbf{x}_{k+\frac{1}{2}})$. Although the resulting iterate $\mathbf{x}_{k+1}^{\mathrm{AA}}$ is of the dimension nd, we can reduce its dimension to d. To see this, note that $\mathbf{x}_{k+1}^{\mathrm{AA}}$ belongs to the set $\mathcal{C}$ so that all its elements are identical. Then, we set

$$x_{k+1}^{\mathrm{AA}} = \frac{1}{n} \sum_{t=0}^{m-1} \gamma_{t,k} \sum_{i=1}^{n} (\hat{x}_{k-t}^i - \alpha g_{k-t}^i), \tag{17}$$

which equals to any element of $\mathbf{x}_{k+1}^{\mathrm{AA}}$. In addition, m is often replaced with $m_k = \{m, k\}$ in AA.

Setting $x_{k+1} = x_{k+1}^{\mathrm{AA}}$ applies AA to DAve-G, and we refer to the resulting algorithm as the Asynchronous Distributed Gradient Method with Anderson Acceleration (ADGM-AA). Moreover, note that different from the centralized AA, we do not have the gradient ∇f_i of different i at the same point in the asynchronous setting, and the derivations (13)–(16) are based on information at the delayed iterates $\{\hat{\mathbf{x}}_t\}_{t=k-m+1}^{k}$.

3.2 A reference-path-based safe-guard scheme

Even in the centralized setting, steepest descent with AA does not converge globally for non-linear operators, where a counterexample is provided in Mai & Johansson (2020). To guarantee global convergence of ADGM-AA for non-affine operators, we equip it with a safe-guard scheme. To do so, a straightforward idea is to check whether the AA step yields a decreasing objective value, i.e.,

$$f(x_{k+1}) - f(x_k) \leq -C_k \tag{18}$$

for some $C_k \geq 0$. If it holds, then perform the AA step $x_{k+1} = x_{k+1}^{\mathrm{AA}}$, otherwise, execute the DAve-G step $x_{k+1} = x_{k+1}^{\mathrm{DAve}}$ where x_{k+1}^{DAve} is the left-hand side of (9). However, the condition (18) cannot be checked in a distributed way since the master in general does not have f_i's.

Alternatively, we come up with the following idea: since DAve-G is guaranteed to converge without any safe-guard schemes, as long as the outcomes of the AA-step and the DAve-G step are sufficiently close, the convergence of ADGM-AA may be guaranteed. Inspired by this, we treat the sequence generated by DAve-G as a *reference path*, and restrict the iterate to be close enough to the reference path. Specifically, at each iteration, we check the following condition:

$$\|x_{k+1}^{\mathrm{AA}} - x_{k+1}^{\mathrm{DAve}}\| \leq \eta_k \tag{19}$$

where $\eta_k \geq 0$. If (19) holds, then $x_{k+1} = x_{k+1}^{\mathrm{AA}}$. Otherwise, we set $x_{k+1} = x_{k+1}^{\mathrm{DAve}}$.

Define

$$\tilde{\eta}_k = \begin{cases} \eta_k, & x_{k+1} = x_{k+1}^{\mathrm{AA}}, \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

It follows that

$$\|x_{k+1} - x_{k+1}^{\mathrm{DAve}}\| \leq \tilde{\eta}_k, \tag{21}$$

and we require $\tilde{\eta}_k$ to be summable to restrict the distance between x_{k+1} and the reference path:

$$\sum_{k=0}^{\infty} \tilde{\eta}_k < +\infty. \tag{22}$$

The condition (22) can be easily satisfied by, e.g.,

$$\eta_k = c(k+1)^{-(1+\epsilon)}, \tag{23}$$

where $c > 0$ and $\epsilon > 0$.

Compared to (18), the safe-guard conditon (19) with η_k in (23) is much simpler and can be verified in a distributed way. A detailed implementation of ADGM-AA with the safe-guard scheme is presented in Algorithms 1–2.

4 Convergence analysis

This section analyses the convergence of the proposed ADGM-AA. To this end, we assume that each f_i is smooth, problem (1) has at least one optimal solution, and the delays $\{d_k^i\}$ are bounded.

Algorithm 1 Master process in ADGM-AA

Input: storage size $m \geq 1$, initial iterate x_0, safe-guard parameters $c > 0$, $\epsilon > 0$.
Output: x_k.

Send $(x_0, 0)$ to all workers.
Receive $(\nabla f_i(x_0), 0)$ from each worker i and set $\hat{x}_0^i = x_0, g_0^i = \nabla f_i(x_0)$.
Compute x_1 by (9), and then send $(x_1, 1)$ to all workers.
Set $k = 1$ and $\mathbf{r}_0 = ((x_1 - x_0)^T, \ldots, (x_1 - x_0)^T)^T$.
while not converge **do**
    Wait until receives $\{(\nabla f_i(x_{t_i}), t_i)\}_{i \in S_k}$ from a set S_k of workers.
    for $i = 1, \ldots, n$ **do**
        if $i \in S_k$ **then**
            Set $(\hat{x}_k^i, g_k^i) = (x_{t_i}, \nabla f_i(x_{t_i}^i))$.
        else
            Set $(\hat{x}_k^i, g_k^i) = (\hat{x}_{k-1}^i, g_{k-1}^i)$.
        end if
    end for
    Compute x_{k+1}^{DAve} by (9).
    Set $\mathbf{r}_k = ((x_{k+1}^{\mathrm{DAve}} - \hat{x}_k^1)^T, \ldots, (x_{k+1}^{\mathrm{DAve}} - \hat{x}_k^n)^T)^T$ and $m_k \leftarrow \min\{m, k\}$.
    Compute Γ_k by (15).
    Compute x_{k+1}^{AA} by (17).
    if the safe-guard condition (19) holds **then**
        $x_{k+1} \leftarrow x_{k+1}^{\mathrm{AA}}$.
    else
        $x_{k+1} \leftarrow x_{k+1}^{\mathrm{DAve}}$.
    end if
    $k \leftarrow k + 1$.
    Send (x_k, k) to workers $i \in S_k$.
end while

Algorithm 2 Worker i process in ADGM-AA

repeat
    Receive (x, k) from the master.
    Compute gradient $\nabla f_i(x)$.
    Send $(\nabla f_i(x), k)$ to the master.
until terminated by the master.

Assumption 4.1 (smoothness)**.** *Each function f_i is L-smooth, i.e., it is differentiable, and there exists a constant $L \geq 0$ such that for any $x, y \in \mathbb{R}^d$,*

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L\|x - y\|.$$

Assumption 4.2 (optimal solution existence)**.** *Problem (1) has at least one optimal $x^\star$.*

Assumption 4.3 (bounded delay)**.** *It holds that $D \stackrel{\triangle}{=} \max\limits_{k \geq 0} \max\limits_{i \in \{1, \ldots, n\}} d_k^i < +\infty$.*

Assumptions 4.1–4.3 are standard in asynchronous optimization (Aytekin et al., 2016; Wu et al., 2023) to guarantee convergence. Moreover, it is clear that for any optimal solution $x^\star$ of problem (1), $\mathbf{x}^\star = ((x^\star)^T, \ldots, (x^\star)^T)^T$ is a fixed point to T. Under Assumptions 4.1 – 4.2, we derive the following lemma, which will be used in subsequent analysis.

Lemma 4.1. *Suppose that Assumptions 4.1 – 4.2 hold. Let*

$$\mathbf{y}_k = P_{\mathcal{C}}(\hat{\mathbf{x}}_k - \alpha \nabla F(\hat{\mathbf{x}}_k)).$$

If $\alpha \in (0, 1/L]$*, then*

$$\|\mathbf{y}_k - \mathbf{x}^\star\|^2 \leq \|\hat{\mathbf{x}}_k - \mathbf{x}^\star\|^2 - 2\alpha(F(\mathbf{y}_k) - F(\mathbf{x}^\star)). \tag{24}$$

Proof. See Appendix A. □

With Lemma 4.1, we first show the boundedness of the sequence $\{x_k\}$.

Lemma 4.2. *Suppose that Assumptions 4.1–4.3 hold. Let* $\{x_k\}$ *be generated by ADGM-AA with the safe-guard scheme. If* $\alpha \in (0, \frac{1}{L})$*, then for any* $k \geq 0$*,*

$$\|x_k - x^\star\| \leq \|x_0 - x^\star\| + \eta_{\text{sum}}$$

where $\eta_{\text{sum}} = \sum_{t=0}^{\infty} \tilde{\eta}_t$*.*

Proof. See Appendix B. □

Theorem 4.1. *Suppose that Assumptions 4.1–4.3 hold. Let* $\{x_k\}$ *be generated by ADGM-AA with the safe-guard scheme. If* $\alpha \in (0, \frac{1}{L})$*, then for any* $k \geq 1$*,*

$$\min_{\ell \leq k} f(x_\ell) - f(x^\star) \leq \frac{3(\alpha L + 1)(\|x_0 - x^\star\| + \eta_{\text{sum}})^2}{2\alpha k/(D+1)}. \tag{25}$$

Proof. See Appendix C. □

Next, we present the convergence rate under the more restrictive strongly convex assumption.

Assumption 4.4 (strong convexity)**.** *The function* f *is* μ*-strongly convex, i.e., for all* $x, y \in \mathbb{R}^d$*,*

$$f(y) - f(x) \geq \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2.$$

Note that Assumption 4.4 assumes the sum of the f_i's rather than all f_i's to be strongly convex.

Theorem 4.2. *Suppose that Assumption 4.4 and all the conditions in Theorem 4.1 hold. Let* $\{x_k\}$ *be generated by ADGM-AA with the safe-guard scheme. Then, for any* $k \geq 0$*, it holds that*

$$\|x_k - x^\star\| \leq \rho^{t_k}\|x_0 - x^\star\| + \sum_{t=0}^{t_k - 1} \rho^{t_k - t}\theta_t, \tag{26}$$

where $\rho = \frac{1}{1+2\alpha\mu}$*,* $t_k = \lceil (k-1)/(D+1) \rceil$*,* $\theta_0 = 2\eta_0(\|x_0 - x^\star\| + \eta_{\text{sum}})(1 + \alpha L) + (\eta_0)^2$*, and* $\theta_t = \sum_{\ell=(t-1)(D+1)}^{t(D+1)-1} (2\eta_\ell(\|x_0 - x^\star\| + \eta_{\text{sum}})(1 + \alpha L) + (\eta_\ell)^2)$ *for all* $t \geq 1$*. Moreover, the equation (26) implies* $\lim_{k \to +\infty} x_k = x^\star$*.*

Proof. See Appendix D. □

Corollary 4.1. *Suppose that all the conditions in Theorem 4.2 hold. If* $\eta_k \leq c(k+1)^{-b}$ *for some* $c > 0$ *and* $b > 1$*, then*

$$\sum_{t=0}^{t_k - 1} \rho^{t_k - t}\theta_t \leq O\left(\frac{1}{(t_k - 1)^b}\right). \tag{27}$$

It also holds that

$$\|x_k - x^\star\| \leq O\left(\frac{1}{(t_k - 1)^b}\right) \tag{28}$$

Proof. See Appendix E. □

5 Numerical experiments

In this section, we test the practical performance of ADGM-AA on distributed training of classification models using logistic regression and least squares. Our experiments are implemented on a machine with 48 cores in Python, where the master and each worker are a core. In addition, we set the number $n = 8$ of workers in problem (1). We use the mpi4py package to construct communication between the master and workers, and all the delays are generated from real interactions between the cores, rather than artificially set.

In our experiments, we compare ADGM-AA with two popular asynchronous methods for solving problem (1), including PIAG (Aytekin et al., 2016) and DAve-G (Mishchenko et al., 2018). For ADGM-AA, we set different values $m = 6, 11, 16$ to test the effect of the memory size m (i.e., using information from the previous 5, 10, and 15 iterations, excluding the current iterate), and apply the safe-guard scheme with η_k in (20) where $c = 10^8$ and $\epsilon = 10^{-8}$. All methods use tuned step-sizes and are initialized at $x_0 = \mathbf{1}$. We use a few real-world datasets[2] summarized in Table 1, where the data in each dataset is evenly partitioned across the workers.

Table 1: Datasets

Dataset	A3A	A6A	W1A	W4A	Rcv1	Cifar10	Covtype	Madelon	Gisette
Samples	3185	11220	2477	7366	20242	50000	581012	2000	6000
Features	123	123	300	300	47236	3072	54	500	5000

5.1 Logistic regression

The logistic regression corresponds to problem (1) with

$$f_i(x) = \sum_{j=1}^{M_i} \ln\left(1 + \exp\left(-y_{ij} a_{ij}^T x\right)\right) + \lambda \|x\|^2,$$

where $a_{ij} \in \mathbb{R}^d$ is a training sample, y_{ij} is the corresponding label, M_i is the number of training samples in worker i, and $\lambda = 10^{-3}$. For the logistic regression problem, its gradient function is non-affine, so the operator T is non-affine in Section 3.

We plot the gradient norms, defined as $\|\nabla f(x_k)\|$, versus the number of iterations and display the results in Figure 1. From Figure 1, we observe that 1) ADGM-AA significantly converges faster than the two alternative methods DAve-G and PIAG; 2) As m increases from 6 to 16, the convergence of ADGM-AA becomes faster for most datasets. The experimental results demonstrate the competitive performance of the proposed methods.

5.2 Least squares

The least squares corresponds to problem (1) with

$$f_i(x) = \|A_i x - b_i\|^2,$$

where each row of matrix $A_i \in \mathbb{R}^{p_i \times d}$ is a training sample, each element of $b_i \in \mathbb{R}^{p_i}$ is the corresponding label, and p_i is the number of training samples in worker i. Here, the operator T is affine in Section 3, because the projection operator P_C is linear and the gradient function of the problem is affine.

The experimental results are exhibited in Figure 2. Similar to the logistic regression case, ADGM-AA converges significantly faster than PIAG and DAve-G. Moreover, increasing m from 6 to 16 enhances the convergence speed of ADGM-AA.

[2]The datasets A3A, A6A, W1A, W4A, Rcv1, Cifar10, and Covtype are downloaded from https://www.csie.ntu.edu.tw/∽cjlin/libsvmtools/datasets/. The datasets Madelon and Gisette are downloaded from http://archive.ics.uci.edu/ml/datasets.

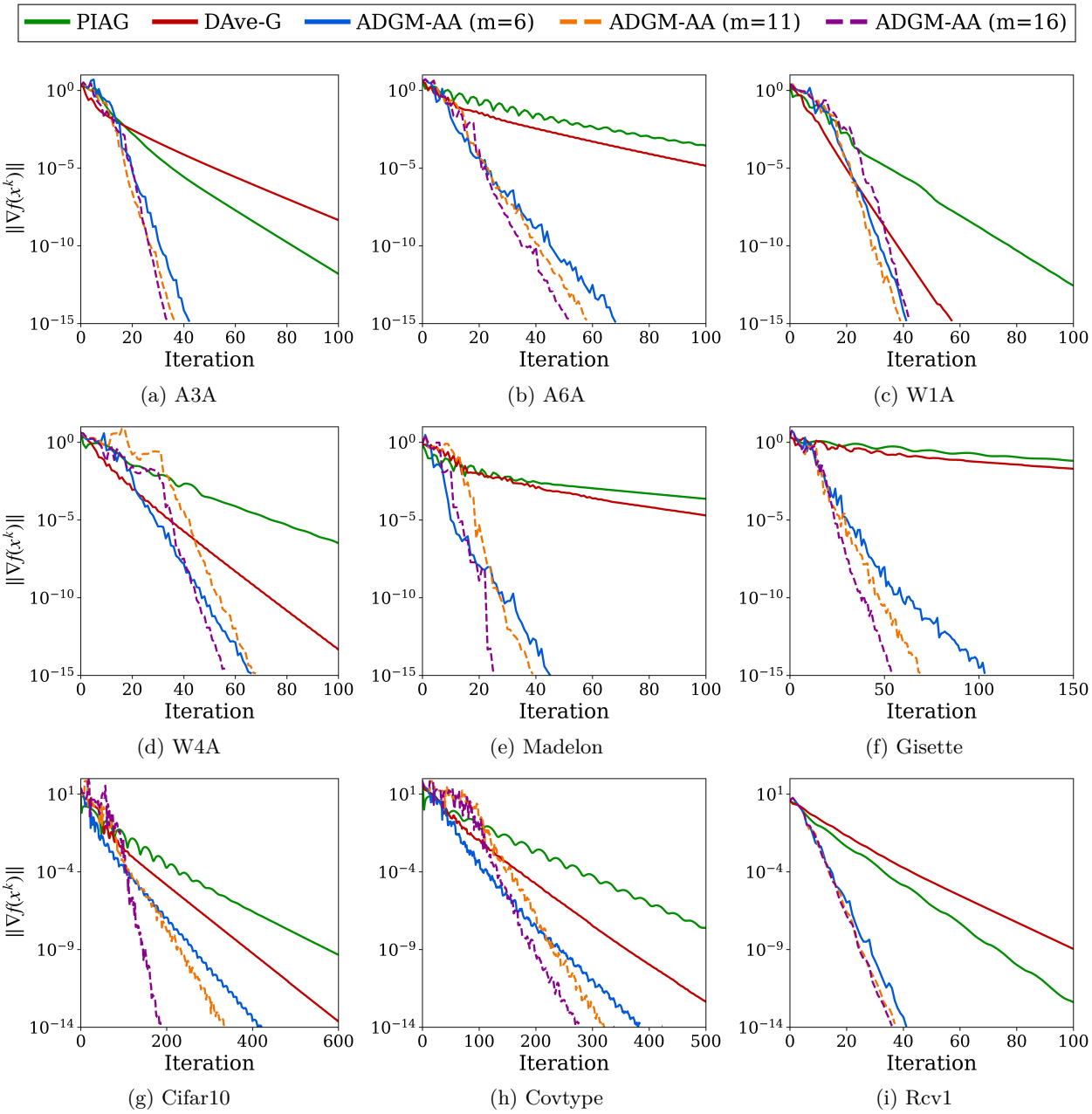
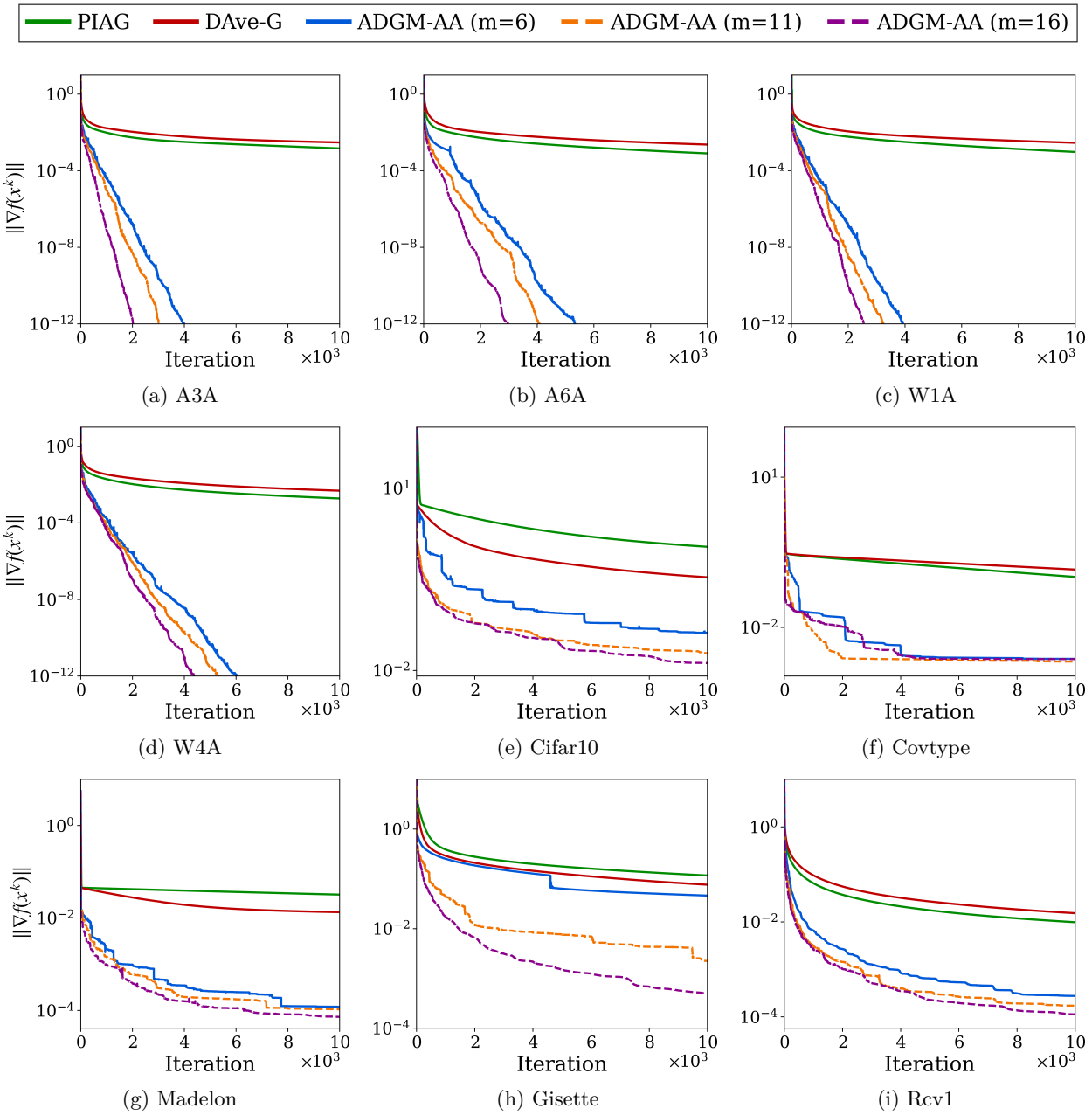


Figure 1: Comparison of PIAG, DAve-G, and ADGM-AA in solving logistic regression.

6 Conclusion

We have extended Anderson acceleration (AA) to asynchronous optimization. In particular, we have applied AA to an asynchronous method DAve-G, which faces two key challenges: First, the asynchronous update in DAve-G is not a fixed-point update; Second, it is difficult to design a safe-guard scheme that can be verified in a distributed way. To address these challenges, we first equivalently rewrote DAve-G as a fixed-point iteration by copying variables, and then designed a novel reference-path-based safe-guard condition. We derived the convergence of our method under standard assumptions and a delay-free step-size condition. The experimental results demonstrate that our algorithm significantly outperforms alternative methods DAve-G and PIAG.



(a) A3A (b) A6A (c) W1A

(d) W4A (e) Cifar10 (f) Covtype

(g) Madelon (h) Gisette (i) Rcv1

Figure 2: Comparison of PIAG, DAve-G, and ADGM-AA in solving least squares.

References

Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12 (4):547–560, 1965.

Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson. Analysis and implementation of an asynchronous optimization algorithm for the parameter server. *arXiv preprint arXiv:1610.05507*, 2016.

Quentin Bertrand and Mathurin Massias. Anderson acceleration of coordinate descent. In *International Conference on Artificial Intelligence and Statistics*, pp. 1288–1296. PMLR, 2021.

Daniel Cederberg, Xuyang Wu, Stephen P Boyd, and Mikael Johansson. An asynchronous bundle method for distributed learning problems. In *The Thirteenth International Conference on Learning Representations*, 2025.

Sélim Chraibi, Franck Iutzeler, Jérôme Malick, and Alexander Rogozin. Delay-tolerant distributed Bregman proximal algorithms. *Optimization Methods and Software*, pp. 1–17, 2024.

Volker Eyert. A comparative study on methods for convergence acceleration of iterative vector sequences. *Journal of Computational Physics*, 124(2):271–285, 1996.

Hamid Reza Feyzmahdavian and Mikael Johansson. Asynchronous iterations in optimization: New sequence results and sharper algorithmic guarantees. *Journal of Machine Learning Research*, 24(158):1–75, 2023.

Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. A delayed proximal gradient method with linear convergence rate. In *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2014.

Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. ASAGA: Asynchronous parallel SAGA. In *Artificial Intelligence and Statistics*, pp. 46–54. PMLR, 2017.

Zhize Li and Jian Li. A fast Anderson-Chebyshev acceleration for nonlinear optimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 1047–1057. PMLR, 2020.

Hailiang Liu, Jia-Hao He, and Xuping Tian. Anderson acceleration of gradient methods with energy for optimization problems. *Communications on Applied Mathematics and Computation*, 6(2):1299–1318, 2024.

Ji Liu, Steve Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *International Conference on Machine Learning*, pp. 469–477. PMLR, 2014.

Guodong Ma, Jiachen Jin, Jinbao Jian, Jianghua Yin, and Daolan Han. A modified inertial three-term conjugate gradient projection method for constrained nonlinear equations with applications in compressed sensing. *Numerical Algorithms*, 92(3):1621–1653, 2023.

Vien Mai and Mikael Johansson. Anderson acceleration of proximal gradient methods. In *International Conference on Machine Learning*, pp. 6620–6629. PMLR, 2020.

Konstantin Mishchenko, Franck Iutzeler, Jérôme Malick, and Massih-Reza Amini. A delay-tolerant proximal-gradient algorithm for distributed learning. In *International Conference on Machine Learning*, pp. 3587–3595. PMLR, 2018.

Wenqing Ouyang, Yue Peng, Yuxin Yao, Juyong Zhang, and Bailin Deng. Anderson acceleration for nonconvex ADMM based on Douglas-Rachford splitting. In *Computer Graphics Forum*, volume 39, pp. 221–239. Wiley Online Library, 2020.

B.T. Polyak. *Introduction to optimization*. Optimization Software, 1987.

Florian A Potra and Hans Engler. A characterization of the behavior of the Anderson acceleration on linear problems. *Linear Algebra and its Applications*, 438(3):1002–1011, 2013.

Péter Pulay. Convergence acceleration of iterative sequences. The case of SCF iteration. *Chemical Physics Letters*, 73(2):393–398, 1980.

Peter Pulay. Improved SCF convergence acceleration. *Journal of Computational Chemistry*, 3(4):556–560, 1982.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 24, 2011.

Youcef Saad and Martin H Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

Damien Scieur, Alexandre d'Aspremont, and Francis Bach. Regularized nonlinear acceleration. *Advances in Neural Information Processing Systems*, 29, 2016.

Tao Sun, Yuejiao Sun, Dongsheng Li, and Qing Liao. General proximal incremental aggregated gradient algorithms: Better and novel results under general scheme. *Advances in Neural Information Processing Systems*, 32, 2019.

Alex Toth and Carl T Kelley. Convergence analysis for Anderson acceleration. *SIAM Journal on Numerical Analysis*, 53(2):805–819, 2015.

Nuri Denizcan Vanli, Mert Gurbuzbalaban, and Asu Ozdaglar. A stronger convergence result on the proximal incremental aggregated gradient method. *arXiv preprint arXiv:1611.08022*, 2016.

Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

Stephen J Wright and Benjamin Recht. *Optimization for data analysis*. Cambridge University Press, 2022.

Xuyang Wu, Sindri Magnusson, Hamid Reza Feyzmahdavian, and Mikael Johansson. Delay-adaptive step-sizes for asynchronous learning. In *International Conference on Machine Learning*, pp. 24093–24113. PMLR, 2022.

Xuyang Wu, Changxin Liu, Sindri Magnússon, and Mikael Johansson. Delay-agnostic asynchronous coordinate update algorithm. In *International Conference on Machine Learning*, pp. 37582–37606. PMLR, 2023.

Ruiliang Zhang and James Kwok. Asynchronous distributed ADMM for consensus optimization. In *International Conference on Machine Learning*, pp. 1701–1709. PMLR, 2014.

Shenyi Zhao and Wujun Li. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

A Proof of Lemma 4.1

Note that $\mathbf{y}_k$ is the result of a projected steepest descent step at $\hat{\mathbf{x}}_k$, and the projected steepest descent method is a special case of the proximal gradient method. Then, applying Wright & Recht, 2022, (9.19) on the proximal gradient method, we have (24).

B Proof of Lemma 4.2

Define $\beta_k = \max_{t\le k} \|x_t - x^\star\|$ for all $k \ge 0$. Since

$$\mathbf{y}_k = ((x_{k+1}^{\mathrm{DAve}})^T, \ldots . (x_{k+1}^{\mathrm{DAve}})^T)^T$$

in Lemma 4.1, we have by (24) that

$$\begin{aligned} \|x_{k+1}^{\mathrm{DAve}} - x^\star\|^2 \le & \frac{1}{n}\sum_{i=1}^n \|\hat{x}_k^i - x^\star\|^2 - 2\alpha(f(x_{k+1}^{\mathrm{DAve}}) - f(x^\star)) \\ \le & \frac{1}{n}\sum_{i=1}^n \|x_{k-d_k^i} - x^\star\|^2 \le (\beta_k)^2. \end{aligned} \tag{29}$$

If $x_{k+1} = x_{k+1}^{\mathrm{AA}}$, then by (19) and (21), we have

$$\|x_{k+1} - x^\star\| \le \|x_{k+1}^{\mathrm{DAve}} - x^\star\| + \|x_{k+1} - x_{k+1}^{\mathrm{DAve}}\| \le \beta_k + \tilde{\eta}_k. \tag{30}$$

Otherwise,

$$\|x_{k+1} - x^\star\| \le \|x_{k+1}^{\mathrm{DAve}} - x^\star\| \le \beta_k.$$

As a result,

$$\beta_{k+1} \le \beta_k + \tilde{\eta}_k \le \beta_0 + \sum_{t=0}^{\infty} \tilde{\eta}_t = \|x_0 - x^\star\| + \eta_{\mathrm{sum}}.$$

C Proof of Theorem 4.1

The proof consists of two steps. Step 1 proves the following inequality

$$\begin{aligned} \|x_{k+1} - x^\star\|^2 \le & \frac{1}{n}\sum_{i=1}^n \|\hat{x}_k^i - x^\star\|^2 - 2\alpha(f(x_{k+1}) - f(x^\star)) \\ & + \tilde{\eta}_k(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})(2 + 2\alpha L) + (\tilde{\eta}_k)^2, \end{aligned} \tag{31}$$

and Step 2 uses (31) to derive (25).

Step 1: If $x_{k+1} = x_{k+1}^{\mathrm{DAve}}$, then by (29), the equation (31) naturally holds. Otherwise, by (29) and Lemma 4.2, we have

$$\begin{aligned} & \|x_{k+1} - x^\star\|^2 \\ = & \|x_{k+1}^{\mathrm{DAve}} - x^\star\|^2 + 2\langle x_{k+1}^{\mathrm{AA}} - x_{k+1}^{\mathrm{DAve}}, x_{k+1}^{\mathrm{DAve}} - x^\star\rangle + \|x_{k+1}^{\mathrm{AA}} - x_{k+1}^{\mathrm{DAve}}\|^2 \\ \le & \|x_{k+1}^{\mathrm{DAve}} - x^\star\|^2 + 2\tilde{\eta}_k(\|x_0 - x^\star\| + \eta_{\mathrm{sum}}) + (\tilde{\eta}_k)^2 \\ \le & \frac{1}{n}\sum_{i=1}^n \|\hat{x}_k^i - x^\star\|^2 - 2\alpha(f(x_{k+1}^{\mathrm{DAve}}) - f(x^\star)) + 2\tilde{\eta}_k(\|x_0 - x^\star\| + \eta_{\mathrm{sum}}) + (\tilde{\eta}_k)^2. \end{aligned} \tag{32}$$

Moreover, by the convexity and the L-smoothness of f,

$$\begin{aligned} f(x_{k+1}) - f(x_{k+1}^{\mathrm{DAve}}) & \le \langle \nabla f(x_{k+1}), x_{k+1} - x_{k+1}^{\mathrm{DAve}}\rangle \\ & = \langle \nabla f(x_{k+1}) - \nabla f(x^\star), x_{k+1} - x_{k+1}^{\mathrm{DAve}}\rangle \\ & \le L\|x_{k+1} - x^\star\| \cdot \|x_{k+1} - x_{k+1}^{\mathrm{DAve}}\| \\ & \le L(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})\tilde{\eta}_k, \end{aligned} \tag{33}$$

where the last step uses Lemma 4.2 and (19). Substituting (33) into (32) gives (31).

Step 2: Define $\mathcal{I}_0 = \{0\}$ and $\mathcal{I}_t = \{(t-1)(D+1)+1, \ldots, t(D+1)\}$ for each $t \geq 1$. Also let $a_t = \max_{k \in \mathcal{I}_t} \|x_k - x^\star\|^2$. First, by induction we prove that for any $k+1 \in \mathcal{I}_{t+1}$,

$$\|x_{k+1} - x^\star\|^2 \leq a_t - 2\alpha(f(x_{k+1}) - f(x^\star)) + \sum_{\ell=t(D+1)}^{k} b_\ell \tag{34}$$

where $b_\ell = 2\tilde{\eta}_\ell(\|x_0 - x^\star\| + \eta_{\text{sum}})(1+\alpha L) + (\tilde{\eta}_\ell)^2$. Fix $k = t(D+1)$. By Assumption 4.3, $d_k^i \leq D$ and $k - d_k^i \in \mathcal{I}_t$. Then, by (31) and $\hat{x}_k^i = x_{k-d_k^i}$, we have (34).

Suppose that for some $K+1 \in \mathcal{I}_{t+1}$, the equation (34) holds for all $k+1 \leq K$ and $k+1 \in \mathcal{I}_{t+1}$. Then, by (31) and $\hat{x}_K^i = x_{K-d_K^i}$,

$$\|x_{K+1} - x^\star\|^2 \leq \max_{1 \leq i \leq n} \|x_{K-d_K^i} - x^\star\|^2 - 2\alpha(f(x_{K+1}) - f(x^\star)) + b_K. \tag{35}$$

Moreover, $K - d_K^i \in \mathcal{I}_t \cup \{t(D+1)+1, \ldots, K\}$. Then, if $K - d_K^i \geq t(D+1)+1$, by (34), we have that

$$\|x_{K-d_K^i} - x^\star\|^2 \leq a_t + \sum_{\ell=t(D+1)}^{K-d_K^i-1} b_\ell \leq a_t + \sum_{\ell=t(D+1)}^{K-1} b_\ell.$$

If $K - d_K^i \in \mathcal{I}_t$, we obtain that

$$\|x_{K-d_K^i} - x^\star\|^2 \leq a_t \leq a_t + \sum_{\ell=t(D+1)}^{K-1} b_\ell.$$

Thus, we have that

$$\|x_{K-d_K^i} - x^\star\|^2 \leq a_t + \sum_{\ell=t(D+1)}^{K-1} b_\ell. \tag{36}$$

Substituting the above equation into (35) yields (34) with $k = K$. Concluding all the above, we obtain (34) for all $K+1 \in \mathcal{I}_{t+1}$.

Next, by (34), we have that for all $t \geq 0$,

$$\begin{aligned} a_{t+1} &\leq a_t - 2\alpha \min_{k' \in \mathcal{I}_{t+1}} (f(x_{k'}) - f(x^\star)) + \sum_{\ell=t(D+1)}^{(t+1)(D+1)-1} b_\ell \\ &\leq a_0 - \sum_{\ell=0}^{t} 2\alpha \min_{k' \in \mathcal{I}_{\ell+1}} (f(x_{k'}) - f(x^\star)) + \sum_{\ell=0}^{(t+1)(D+1)-1} b_\ell, \end{aligned} \tag{37}$$

where the second step uses telescoping cancellation. Moreover, for any $k \geq 0$, we have

$$k \in \mathcal{I}_{t_k}, \tag{38}$$

where $t_k = \lceil k/(D+1) \rceil$. Letting $t = t_k - 1$ in (34) and using (37) gives

$$\begin{aligned} \|x_k - x^\star\|^2 &\leq a_t - 2\alpha(f(x_k) - f(x^\star)) + \sum_{\ell=t(D+1)}^{k-1} b_\ell \\ &\leq a_0 - 2\alpha\Big(\min_{k' \in \mathcal{I}_{t_k}} (f(x_{k'}) - f(x^\star)) + \sum_{\ell=0}^{t-1} \min_{k' \in \mathcal{I}_{\ell+1}} (f(x_{k'}) - f(x^\star))\Big) + \sum_{\ell=0}^{k-1} b_\ell. \end{aligned} \tag{39}$$

Since $k_\ell \le k$ for all $\ell = 0, \dots, t-1$, the above equation implies

$$\begin{aligned}\min_{\ell \le k} f(x_\ell) - f(x^\star) &\le \frac{1}{t+1}\Big(\min_{k' \in \mathcal{I}_{t_k}} (f(x_{k'}) - f(x^\star)) + \sum_{\ell=0}^{t-1} \min_{k' \in \mathcal{I}_{\ell+1}} (f(x_{k'}) - f(x^\star))\Big) \\ &\le \frac{a_0 + \sum_{\ell=0}^{\infty} b_\ell}{2\alpha t_k} \\ &\le \frac{a_0 + \sum_{\ell=0}^{\infty} b_\ell}{2\alpha k/(D+1)}\end{aligned} \tag{40}$$

Moreover, since

$$\sum_{\ell=0}^{\infty} (\tilde{\eta}_\ell)^2 \le \Big(\sum_{\ell=0}^{\infty} \tilde{\eta}_\ell\Big)^2 = \eta_{\text{sum}}^2,$$

we have

$$\sum_{\ell=0}^{\infty} b_\ell \le 2\eta_{\text{sum}}(\|x_0 - x^\star\| + \eta_{\text{sum}})(1 + \alpha L) + \eta_{\text{sum}}^2. \tag{41}$$

Therefore,

$$\begin{aligned}a_0 + \sum_{\ell=0}^{\infty} b_\ell &\le \|x_0 - x^\star\|^2 + 2\eta_{\text{sum}}(\|x_0 - x^\star\| + \eta_{\text{sum}})(1 + \alpha L) + \eta_{\text{sum}}^2 \\ &\le (3\alpha L + 3)(\|x_0 - x^\star\| + \eta_{\text{sum}})^2,\end{aligned}$$

substituting which into (40) gives (25).

D Proof of Theorem 4.2

Under the strong convexity assumption,

$$f(x_{k+1}) - f(x^\star) \ge \frac{\mu}{2}\|x_{k+1} - x^\star\|^2, \tag{42}$$

substituting which into (34) yields

$$(1 + 2\alpha\mu)\|x_{k+1} - x^\star\|^2 \le a_t + \sum_{\ell=t(D+1)}^{k} b_\ell \le a_t + \sum_{\ell=t(D+1)}^{(t+1)(D+1)-1} b_\ell \tag{43}$$

for all $k + 1 \in \mathcal{I}_{t+1}$. By the above inequality,

$$a_{t+1} \le \rho a_t + \rho\theta_t \le \rho^{t+1} a_0 + \sum_{\ell=0}^{t} \rho^{t+1-\ell}\theta_\ell. \tag{44}$$

Moreover, for any $k \ge 0$, by (38), we have

$$\|x_k - x^\star\|^2 \le a_{t_k},$$

which, together with (44), gives (26).

Next, due to (26) and (41) and according to Polyak, 1987, Lemma 3, Sec 2.2, we have $\lim_{k \to +\infty} x_k = x^\star$.

E Proof of Corollary 4.1

Because $\tilde{\eta}_k \leq \eta_k$ and $\eta_k \leq c(k+1)^{-b}$, we have for $t \geq 1$ that

$$\begin{aligned}\theta_t =& \sum_{\ell=(t-1)(D+1)}^{t(D+1)-1} (2\tilde{\eta}_\ell(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})(1+\alpha L) + (\tilde{\eta}_\ell)^2) \\ \leq& \sum_{\ell=(t-1)(D+1)}^{t(D+1)-1} \frac{c}{(\ell+1)^b}(2(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})(1+\alpha L) + \frac{c}{(\ell+1)^b}) \\ \leq& \frac{c(D+1)}{((t-1)(D+1)+1)^b}(2(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})(1+\alpha L) + c).\end{aligned}$$

By the above inequality, we obtain that

$$\begin{aligned}\sum_{t=0}^{t_k-1} \rho^{t_k - t}\theta_t \leq& (2(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})(1+\alpha L) + c) \sum_{t=1}^{t_k-1} \rho^{t_k-t} \frac{c(D+1)}{((t-1)(D+1)+1)^b} \\ &+ \rho^{t_k}\theta_0 \end{aligned} \tag{45}$$

Let $p = t_k - t$. The right-hand side is rewritten as

$$(2(\|x_0 - x^\star\| + \eta_{\mathrm{sum}})(1+\alpha L) + c) \sum_{p=1}^{t_k-1} \rho^p \frac{c(D+1)}{((t_k - p - 1)(D+1)+1)^b} + \rho^{t_k}\theta_0$$

Firstly, when $0 \leq p \leq \lfloor (t_k - 1)/2 \rfloor$, we have that $t_k - 1 - p \geq (t_k - 1)/2$, thereby having that

$$\frac{1}{(t_k - 1 - p)^b} \leq \frac{2^b}{(t_k - 1)^b}.$$

Hence, we have that

$$\begin{aligned}\sum_{p=1}^{\lfloor (t_k-1)/2 \rfloor} \rho^p \frac{c(D+1)}{((t_k - p - 1)(D+1)+1)^b} \leq& \frac{2^b c}{(t_k-1)^b (D+1)^{b-1}} \sum_{p=1}^{\infty} \rho^p \\ \leq& \frac{2^b c\rho}{(t_k-1)^b(D+1)^{b-1}(1-\rho)}.\end{aligned} \tag{46}$$

Secondly, when $\lfloor (t_k - 1)/2 \rfloor < p \leq t_k - 1$, we have that $\rho^p \leq \rho^{(t_k-1)/2}$. Here, we obtain that

$$\sum_{p=\lfloor (t_k-1)/2 \rfloor + 1}^{t_k-1} \rho^p \frac{c(D+1)}{((t_k - p - 1)(D+1)+1)^b} \leq c(t_k - 1)\rho^{(t_k-1)/2}. \tag{47}$$

Finally, we know that

$$\begin{aligned}&\lim_{t_k \to \infty} \frac{2^b c (t_k-1)^b \rho}{(t_k-1)^b(D+1)^{b-1}(1-\rho)} = \frac{2^b c\rho}{(D+1)^{b-1}(1-\rho)}, \\ &\lim_{t_k \to \infty} c(t_k-1)^{b+1}\rho^{(t_k-1)/2} = 0, \quad \text{and} \quad \lim_{t_k \to \infty} \theta_0 (t_k-1)^b \rho^{t_k} = 0.\end{aligned}$$

Therefore, by using inequalities (45), (46), and (47), we obtain (27). On the other hand, due to $\lim_{t_k \to \infty} \rho^{t_k}\|x_0 - x^\star\|(t_k-1)^b = 0$, and by using the inequality (26), we have (28).