Iterative Pre-Conditioning to Expedite the Gradient-Descent Method

Kushal Chakrabarti[⋆], Nirupam Gupta[†], and Nikhil Chopra[⋆]

Abstract— This paper considers the problem of multi-agent distributed optimization. In this problem, there are multiple agents in the system, and each agent only knows its local cost function. The objective for the agents is to collectively compute a common minimum of the aggregate of all their local cost functions. In principle, this problem is solvable using a distributed variant of the traditional gradient-descent method, which is an iterative method. However, the speed of convergence of the traditional gradient-descent method is highly influenced by the *conditioning* of the optimization problem being solved. Specifically, the method requires a large number of iterations to converge to a solution if the optimization problem is *ill-conditioned*.

In this paper, we propose an *iterative pre-conditioning* approach that can significantly attenuate the influence of the problem's conditioning on the convergence-speed of the gradient-descent method. The proposed pre-conditioning approach can be easily implemented in distributed systems and has minimal computation and communication overhead. For now, we only consider a specific distributed optimization problem wherein the individual local cost functions of the agents are *quadratic*. Besides the theoretical guarantees, the improved convergence speed of our approach is demonstrated through experiments on a real data-set.

I. INTRODUCTION

We consider a synchronous distributed system that comprises of a server and m agents in a server-based architecture, as shown in Fig. 1. The server-based architecture can be emulated easily on a rooted peer-to-peer network using the well-known routing and message authentication primitives [1], [2]. Each agent $i \in \{1, \ldots, m\}$ holds a pair (A^i, b^i), where A^i is a matrix of size $n_i \times n$ and b^i is a column-vector of size n. Let $\mathbb{R}^n$ denote the set of real-valued vectors of size n. For a vector $v \in \mathbb{R}^n$, let $\|v\|$ denote its 2-norm. If $(\cdot)^T$ denotes the transpose then $\|v\|^2 = v^T v$. The objective of the agents is to solve for the least-squares problem:

$$\underset{x \in \mathbb{R}^n}{minimize} \ \sum_{i=1}^{m} \frac{1}{2} \left\| A^i x - b^i \right\|^2. \tag{1}$$

Applications of such problems include linear regression, state estimation, hypothesis testing.

The server collaborates with the agents to solve the problem. The server-based architecture is also commonly known as the *federated model* [3], where the agents collaboratively solve an optimization problem, such as (1), using a central server without ever sharing their local data (A^i, b^i) with anyone in the system. In this
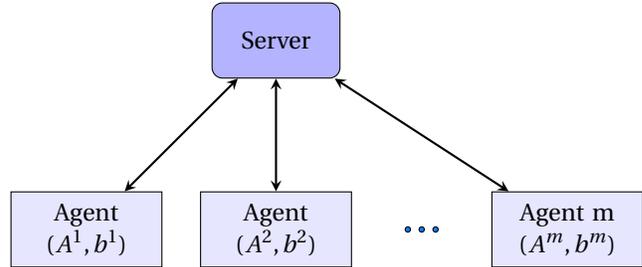

Fig. 1: The system architecture.

architecture, the server maintains an estimate of the global solution of (1), and each agent downloads the estimated global solution from the server. Each agent updates the estimated global solution locally using the data it possesses and uploads its local updated estimate to the server. The server accumulates the locally updated solutions from all the agents to improve the global solution.

The most common and most straightforward algorithm for the above distributed minimization problem is the distributed variant of the vanilla gradient-descent method, called the distributed gradient-descent (DGD) method [4]. Recently, Azizan-Ruhi et al., 2019 [5] have proposed an *accelerated* projection method, which is built upon the seminal work on accelerated methods by Nesterov [6]. However, Azizan-Ruhi et al. do not provide any theoretical guarantees on the improved convergence of their method over the traditional DGD method [5].

In this paper, we propose a different approach than the accelerated methods to improve the convergence speed of the DGD method. Instead of using *momentum*, we use a *pre-conditioning* scheme wherein each iteration the update vector, used by the server to update its estimate of the global solution of (1) is multiplied by a *pre-conditioner matrix* that we provide. The pre-conditioner matrix in itself is updated in each iteration; hence we call our approach the *iterative pre-conditioning* method.

The DGD method is an iterative algorithm, in which the server maintains an estimate for a point of minimum of (1), denoted by x^*, and updates it iteratively by collaborating with the agents as follows. For each iteration $t = 0, 1, \ldots$, let $x(t)$ denote the estimate of x^* at the beginning of iteration t. The initial value $x(0)$ is chosen arbitrarily. In each iteration t, the server broadcasts $x(t)$

[⋆] University of Maryland, College Park, Maryland 20742, U.S.A.
[†] Georgetown University, Washington, DC 20057, U.S.A.
Emails: *kchakrabarti0@gmail.com, nirupam.gupta@georgetown.edu* and *nchopra@umd.edu*

to all the agents. Each agent i computes the gradient of the function $(1/2) \| A^i x - b^i \|^2$ at $x = x(t)$, denoted by $g^i(t)$, and sends it to the server. Note that

$$g^i(t) = (A^i)^T \left(A^i x(t) - b^i \right), \quad \forall i, t. \tag{2}$$

Upon receiving the gradients $\{ g^i(t) \,|\, i = 1, \dots, m \}$ from all the agents, the server updates $x(t)$ to $x(t+1)$ using step-size of constant value δ as follows:

$$x(t+1) = x(t) - \delta \sum_{i=1}^{m} g^i(t), \quad \forall t. \tag{3}$$

To be able to present the contribution of this paper, we first briefly review the convergence of the DGD method described above.

We define the following notations. Let $A = [(A^1)^T, \dots, (A^m)^T]^T$ denote the matrix obtained by stacking matrices A^i's vertically. So, matrix A is of size $N \times n$ where $N = (\sum_{i=1}^{m} n_i)$. We assume that A is a tall matrix, i.e., $N \geq n$. Similarly, we concatenates the b^i's to get $b = [(b^1)^T, \dots, (b^m)^T]^T \in \mathbb{R}^N$.

A. Convergence of DGD

If matrix A is full-column rank then we know that there exists step-size δ for which there is a positive value $\rho < 1$ such that [7],

$$\| x(t) - x^* \| \leq \rho^t \| x(0) - x^* \|, \quad t = 0, 1, \dots$$

The value ρ is commonly referred as the *convergence rate*. Smaller ρ implies higher *convergence speed*, and vice-versa. If we let λ and γ denote the largest and smallest eigenvalues of $A^T A$, then it is known that

$$\rho \geq \rho_{GD} = \frac{(\lambda/\gamma) - 1}{(\lambda/\gamma) + 1}. \tag{4}$$

The ratio λ/γ is also commonly referred as the *condition number* of matrix $A^T A$ which we denote by $\kappa(A^T A)$.

B. Pre-Conditioning of DGD

Our objective is to improve the convergence rate ρ of DGD beyond ρ_{GD} by a suitable *pre-conditioning* proposed in this paper. Let K, referred as the *pre-conditioner*, be a square matrix of size $n \times n$. The server now updates its estimate as follows:

$$x(t+1) = x(t) - \delta K \sum_{i=1}^{m} g^i(t), \quad \forall t. \tag{5}$$

If the matrix product $K A^T A$ is positive definite then convergence of (5) can be made linear by choosing δ appropriately, and the smallest possible convergence rate for (5) is given by (ref. Chapter 11.3.3 of [7])

$$\rho_K^* = \frac{\kappa(K A^T A) - 1}{\kappa(K A^T A) + 1}. \tag{6}$$

However, most of the existing pre-conditioning techniques are not applicable for the distributed framework considered in this paper. The incomplete LU factorization algorithms [8], accelerated iterative methods [9], symmetric successive over-relaxation method [9], for computing such a matrix K such that $\kappa(K A^T A)$ is provably smaller than $\kappa(A^T A)$ require the server to have direct access to the matrices A^i's. Some other pre-conditioning methods [10] require A to be a symmetric positive definite matrix. The recently proposed distributed pre-conditioning scheme for the D-Heavy Ball method in [5] has the same convergence rate as APC. The experimental results in Section III suggest that our proposed scheme converges faster than APC, and hence faster than the said pre-conditioning scheme.

C. Summary of Our Contributions

We propose an *iterative pre-conditioner* matrix $K(t)$, instead of a constant pre-conditioner matrix K. That is, the server updates its estimate as follows:

$$x(t+1) = x(t) - \delta K(t) \sum_{i=1}^{m} g^i(t), \quad \forall t. \tag{7}$$

The pre-conditioner $K(t)$ can be computed in a distributed manner in the federated architecture for each iteration t, as is presented in Section II. We show that the iterative process (7) converges provably faster to the optimum point x^* than the original DGD algorithm (3). In the experiments, we have also observed that the convergence speed of our proposed algorithm is faster than the accelerated projection-based consensus (APC) method proposed in [5].

Moreover, the computational complexity of the proposed method is the same as the DGD algorithm, which is the smallest amongst the existing distributed algorithms for solving (1). We have formally shown the proposed algorithm to converge faster than the distributed gradient method. In contrast, the conditions where APC is guaranteed to converge faster than DGD have not been provided in [5]. APC, as well as the pre-conditioning scheme for D-HBM in [5], have additional computational overhead before the iterations begin.

II. PROPOSED ALGORITHM

In this section, we present our proposed algorithm and theoretically guarantee that the proposed method converges faster than the distributed gradient descent method (3) for solving the distributed least-squares problem (1).

To be able to present our algorithm, we introduce the following notation.

- For a positive integer m, let

$$[m] := \{1, \dots, m\}.$$

- Let e_j denote the j-th column of the $(n \times n)$-dimensional identity matrix I.

- Let $K(t) \in \mathbb{R}^{n \times n}$ be the pre-conditioner matrix for iteration t, and let $k_j(t) \in \mathbb{R}^n$ denote the j-th column of $K(t)$.

- For $\beta > 0$, define

$$R_j^i(t) := \left((A^i)^T A^i + \frac{\beta}{m} I \right) k_j(t) - \frac{1}{m} e_j, \quad j = 1, \dots, n, \; i \in [m].$$
(8)

The proposed method is summarized in Algorithm 1. Note that α, δ, β are positive valued parameters in the algorithm.

Algorithm 1

1: Initialize $x(0) \in \mathbb{R}^n$, $K(-1) \in \mathbb{R}^{n \times n}$
2: **for** $t = 0, 1, \dots$ **do**
3:     The server transmits $x(t)$ and $K(t-1)$ to all the agents $i \in [m]$
4:     **for** each agent $i \in [m]$ **do**
5:         Compute $g^i(t)$ using (2)
6:         **for** each $j \in 1, \dots, n$ **do**
7:             Compute $R_j^i(t-1)$ given by (8)
8:         **end for**
9:         Transmit $g^i(t)$ and $\{R_j^i(t-1)\}_{j=1}^n$ to the server
10:     **end for**
11:     **for** each $j \in 1, \dots, n$ **do**
12:         The server updates each column of $K(t)$

$$k_j(t) = k_j(t-1) - \alpha \sum_{i=1}^m R_j^i(t-1)$$
(9)

13:     **end for**
14:     The server updates the estimate

$$x(t+1) = x(t) - \delta K(t) \sum_{i=1}^m g^i(t)$$
(10)

15: **end for**

For each iteration $t = 0, 1, \dots$, the server maintains a pre-conditioner matrix $K(t-1)$ and an estimate $x(t)$ of the point of optimum x^*. The initial pre-conditioner matrix $K(-1)$ and estimate $x(0)$ are chosen arbitrarily. The server sends $x(t)$ and $K(t-1)$ to the agents. Each agent $i \in [m]$ computes $R_j^i(t-1)$ for $j = 1, \dots, n$ as given in (8), the local gradient $g^i(t)$ as given in (2), and sends these to the server. Then, the server computes the updated pre-conditioner $K(t)$ as given in (9), and uses this updated pre-conditioner $K(t)$ to compute the updated estimate $x(t+1)$ as given by (10).

A. Computational Complexity

We count the number of floating-point multiplications needed per-iteration of Algorithm 1. For each agent i, the computation of $\{\mathbb{R}_j^i(t) | j = 1, \dots, n\}$ in (8) are independent of each other. So, each agent i computes them in parallel. Now, (8) can be rewritten as

$$R_j^i(t) := (A^i)^T A^i k_j(t) + \frac{\beta}{m} k_j(t) - \frac{1}{m} e_j.$$

Two matrix-vector multiplications are required for computing $R_j^i(t)$: $A^i k_j(t)$ and $(A^i)^T (A^i k_j(t))$, in that order.

Here A is an $(n_i \times n)$-dimensional matrix and $k_j(t)$, $A^i k_j(t)$ respectively are column vectors of dimension n, N. So, a total of $(2n_i n + n)$ flops are required for computing $R_j(t)$.

Two matrix-vector multiplications are required for computing $g^i(t)$: $A^i x(t)$, $(A^i)^T (A^i x(t) - b^i)$, in that order. By a similar argument as for $R_j^i(t)$, computing $g^i(t)$ needs $\mathcal{O}(n_i n)$ flops. One matrix-vector multiplication is required for computing $x(t+1)$: $K(t) \sum_{i=1}^m g^i(t)$, which needs $\mathcal{O}(n_i n)$ flops. So, per-iteration computational cost of Algorithm 1 is $\mathcal{O}(n_i n)$.

B. Convergence Analysis

To be able to present the convergence of Algorithm 1, we introduce the following notations.

- Let,

$$K^* = \left(A^T A + \beta I \right)^{-1}.$$

The matrix $A^T A$ is positive semi-definite. Thus, $(A^T A + \beta I)$ is positive definite for $\beta > 0$. So, K^* is well-defined.

- Let k_j^* be the j-th column of K^*, $j = 1, \dots, n$.

- Let, λ and γ denote the smallest and largest eigenvalue of $A^T A$, respectively.

- Let,

$$\rho_K^* := \frac{\lambda - \gamma}{\lambda + \gamma + 2\beta}, \; \rho_\beta^* := \frac{(\lambda - \gamma)\beta}{(\lambda + \gamma)\beta + 2\lambda\gamma},$$
$$\sigma_0 := \delta\lambda \left\| K(-1) - K^* \right\|_F.$$

We make the following assumption.

Assumption 1: Assume that the matrix $A^T A$ is full rank.

The above lemma shows that each column of the pre-conditioner matrix $K(t)$ asymptotically converges to the corresponding column of K^*. In other words, the matrix $K(t)$ asymptotically converges to K^*.

Lemma 1: Consider the iterative process (9), and let $\beta > 0$. Under Assumption 1, there exists $\alpha > 0$ for which there is a positive value $\rho_K < 1$ such that for each $j = 1, \dots, n$,

$$\left\| k_j(t) - k_j^* \right\| \le \rho_K \left\| k_j(t-1) - k_j^* \right\|, \; t = 0, 1, 2, \dots,$$

where $\rho_K \ge \rho_K^*$.

The smallest and the largest Eigenvalues of $K^* A^T A$ have a direct influence on the convergence speed of Algorithm 1. The following lemma finds these Eigenvalues.

Lemma 2: If Assumption 1 holds and $\beta > 0$, then $K^* A^T A$ is positive definite, and the largest and the smallest eigenvalues of $K^* A^T A$ are $\frac{\lambda}{\lambda + \beta}$ and $\frac{\gamma}{\gamma + \beta}$, respectively.

The following proposition tells us that Algorithm 1 converges to the point of minima at a linear rate.

Proposition 1: Consider Algorithm 1 with $\beta > 0$. Under Assumption 1, there exists $\delta > 0$ for which there is a

positive value $\rho_\beta < 1$ such that

$$\|x(t+1) - x^*\| \leq \left(\rho_\beta + \sigma_0(\rho_K^*)^{t+1}\right)\|x(t) - x^*\|, \forall t,$$

where $\rho_\beta \geq \rho_\beta^*$.

From Proposition 1, the rate of convergence of Algorithm 1 depends on the choice of initial value $K(-1)$ through σ_0. The closer $K(-1)$ is to K^*, the faster is the rate.

C. Comparison with DGD

Now we are ready to present our key result, which is a formal comparison between the convergence speed of Algorithm 1 and the original DGD method in (3). We define a few necessary notations.

- Let, the estimate of the optimum point computed by Algorithm 1 and DGD be denoted by $x_1(t)$ and $x_2(t)$, respectively, after t iterations.
- Let, $z_i(t) = x_i(t) - x^*$ for $i = 1,2$.
- Let, the known upper bound on $\|z_i(t)\|$ be $E_i(t)$.

Theorem 1: Consider Algorithm 1 and the DGD algorithm (3) with identical initial estimate $x(0) \in \mathbb{R}^n$. If Assumption 1 holds, then there exists $t_{sw} < \infty$ such that $E_1(t) < E_2(t)$ for all $t > t_{sw}$.

Theorem 1 implies that if the server executes both Algorithm 1 and the original DGD algorithm using the same initial estimate $x(0)$ of x^*, then after a certain number of iterations the upper bound on the error-norm generated by Algorithm 1 is smaller than the original DGD.

III. EXPERIMENTS

In this section, we present the experimental results to verify the obtained theoretical convergence guarantees of Algorithm 1. The matrix A is loaded from one of the real data-sets in SuiteSparse Matrix Collection[1]. Our primary focus here is obtaining faster convergence on a problem (1) where the condition number of $A^T A$ is large enough, specifically $5.8 * 10^7$, while keeping the computational load minimum.

Example 1. The matrix A is a real symmetric positive definite matrix from the benchmark dataset *"bcsstm07"* which is part of a structural problem. The dimension of A is 420×420. We generate the vector b by setting $b = Ax^*$ where x^* is $n = 420$ dimensional vector of one's. Since A is positive definite, (1) has a unique solution x^*. The data (A, b) is split among $m = 10$ machines, so that $n^i = 42$, $i = 1, \ldots, 10$.

We apply Algorithm 1 to solve the aforementioned optimization problem. It can be seen that, the speed

---

[1]https://sparse.tamu.edu

---

of the algorithm depends on the parametric choices of β, α, δ (ref. Fig. 2a). Also, the algorithm converges to x^* irrespective of the initial choice of the entries in $x(0)$ and $K(-1)$ (ref. Fig. 2b).

The experimental results have been compared with the conventional distributed gradient descent (DGD) method and the accelerated projection-based consensus (APC) [5] method (ref. Fig. 3). The pre-conditioning technique for the distributed heavy-ball method in [5] has the same theoretical rate as APC. Besides, it has an additional computational overhead of $\mathcal{O}(p^2 n + p^3)$, with $p \leq n_i$ depending on the given matrix A_i, which is larger than the cost of Algorithm 1 in general.

We compare the number of iterations by DGD, APC and Algorithm 1 to reach a relative estimation error (defined as $\frac{\|x(t) - x^*\|}{\|x^*\|}$) of 10^{-4} (ref. Table I). The tuning parameter β has been set at 5 for Algorithm 1. The rest of the algorithm parameters have been set such that the respective algorithms will have their smallest possible convergence rates. Specifically, $(\alpha^* = 3.17 * 10^{-7}, \delta^* = 1.95)$ for Algorithm 1, $(\gamma^* = 1.08, \eta^* = 12.03)$ for APC and $\delta^* = 3.17 * 10^{-7}$ for DGD. We found that Algorithm 1 performs the fastest among these algorithms. Note that evaluating the optimal tuning parameters for any of these algorithms requires knowledge about the smallest and largest Eigenvalues of $A^T A$.

We notice that, the error norm $\|x(t) - x^*\|$ for Algorithm 1 is less than that of DGD from an approximate iteration index of 300 onward (ref. Fig. 3b). Since $x(0)$ is same for both of the algorithms, this observation is in agreement with the claim in Theorem 1. Similarly, Algorithm 1 is faster than APC after 1114 iterations (ref. Fig. 3b).

IV. SUMMARY

In this paper, we have proposed an algorithm for solving distributed linear least-squares minimization problems over a federated architecture with minimum computational load. However, the algorithm can be emulated in a rooted peer-to-peer network. The vital contribution lies in mitigating the detrimental impact of ill-conditioning on the convergence of the traditional gradient descent method. The computation of the pre-conditioner is done at the server level without requiring any access to the data. In practice, we test the algorithm on a real dataset with a significant condition number ($cond(A^T A) = 5.8 * 10^7$) and get much better performance compared to the classical distributed gradient algorithm as well as the recently proposed accelerated projection-consensus algorithm (APC), regarding the number of iterations needed for convergence to the true solution. We have formally shown the proposed algorithm to converge faster than the distributed gradient method, while the APC method only
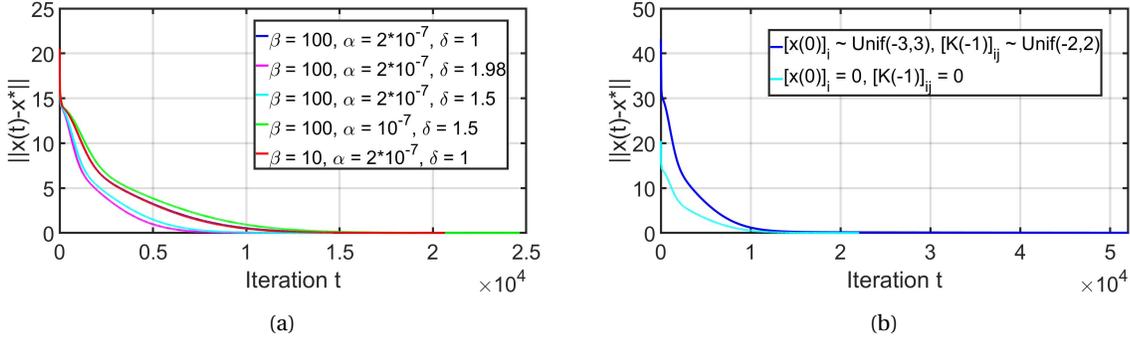
Fig. 2: Temporal evolution of error norm for estimate $\|x(t) - x^*\|$ for Example 1, under Algorithm 1 with different parameter choices and initialization. (a) $x(0) = [0,\ldots,0]^T$; (b) $\beta = 100, \alpha = 2*10^{-7}, \delta = 1$.
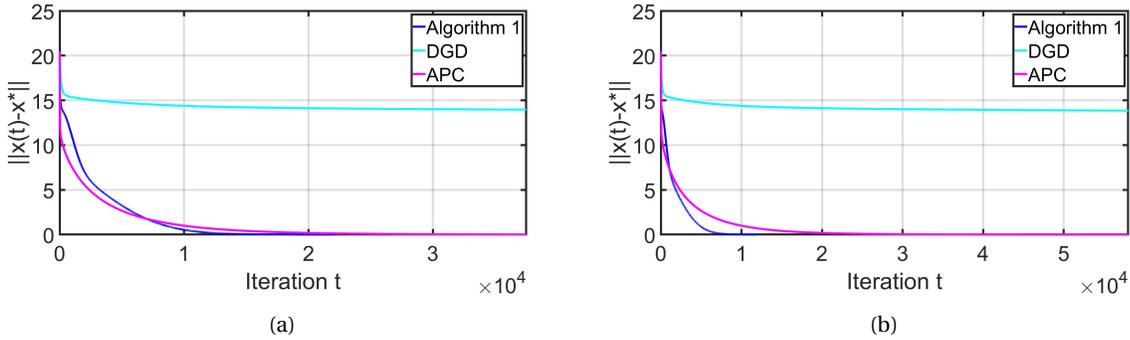

Fig. 3: Temporal evolution of error norm for estimate $\|x(t) - x^*\|$ against number of iterations for Example 1, under Algorithm 1, DGD and APC [5]; with (a) arbitrary parameter choices (b) optimal parameter choices. Initialization for (a) and (b) both: (Algorithm 1) $x(0) = [0,\ldots,0]^T$, $K(-1) = O_{n \times n}$; (DGD) $x(0) = [0,\ldots,0]^T$; (APC) according to the algorithm. In (a): (Algorithm 1) $\beta = 100, \alpha = 2*10^{-7}, \delta = 1$; (DGD) $\delta = 10^{-7}$; (APC) $\gamma = \eta = 1$. In (b): (Algorithm 1) $\beta = 5, \alpha = 3.17*10^{-7}, \delta = 1.95$; (DGD) $\delta = 3.17*10^{-7}$; (APC) $\gamma^* = 1.08, \eta^* = 12.03$.

TABLE I

Comparison Between DGD, APC and Algorithm 1 with Optimal Parameters on "bcsstm07"; regarding (a) Number of Iterations to Attain a Relative Estimation Error 10^{-4}, and (b) Decay rate of the Instantaneous Error-norm. Here, $cond(A^T A) = 5.8*10^7$.			
Algorithm	DGD	APC	Algo. 1 ($\beta = 5, K(-1) = O_{n \times n}$)
(a) Iterations needed	$> 10^5$	$4.85*10^4$	$2.11*10^4$
(b) Rate of Decrease	0.9999	0.9672	$0.9583 + 4.3*10^8*(0.9999)^{t+1}$

speculates it to be faster.

REFERENCES

[1] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.
[2] Andrew S Tanenbaum. Network protocols. *ACM Computing Surveys (CSUR)*, 13(4):453–489, 1981.
[3] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.
[4] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[5] Navid Azizan-Ruhi, Farshad Lahouti, Amir Salman Avestimehr, and Babak Hassibi. Distributed solution of large-scale linear systems via accelerated projection-based consensus. *IEEE Transactions on Signal Processing*, 67(14):3806–3817, 2019.

[6] Y Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Sov. Math. Doklady*, 27(2):372–376, 1983.

[7] Jeffrey A Fessler. Image reconstruction: Algorithms and analysis. http://web.eecs.umich.edu/~fessler/book/c-opt.pdf. [Online book draft; accessed 17-February-2020].

[8] J Andvandervorst Meijerink and Henk A Van Der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of computation*, 31(137):148–162, 1977.

[9] Owe Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25(1):165–187, 1985.

[10] Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

A. Proof of Lemma 1

From (8) and noticing that $\sum_{i=1}^{m} R_j^i(t-1) = (A^T A + \beta I) k_j(t-1) - e_j$, dynamics (9) can be rewritten as

$$k_j(t) = k_j(t-1) - \alpha \left[(A^T A + \beta I) k_j(t-1) - e_j \right]. \quad (11)$$

Define $\tilde{k}_j(t) := k_j(t) - k_j^*$. From the definition of K^* we have,

$$(A^T A + \beta I) K^* = I \implies (A^T A + \beta I) k_j^* = e_j, \ j = 1, ..., n.$$

From (11),

$$\tilde{k}_j(t) = \left[I - \alpha (A^T A + \beta I) \right] \tilde{k}_j(t-1). \quad (12)$$

Since $(A^T A + \beta I)$ is positive definite for $\beta > 0$, $\exists \alpha$ for which there is a positive $\rho_K < 1$ such that $\| \tilde{k}_j(t) \| \le \rho_K \| \tilde{k}_j(t-1) \|$, $t = 0, 1, 2, ...$, where the smallest value of ρ_K is $\frac{\kappa(A^T A + \beta I) - 1}{\kappa(A^T A + \beta I) + 1}$ (ref. Corollary 11.3.3 and Chapter 11.3.3 of [7]). As $\kappa(A^T A + \beta I) = \frac{\lambda + \beta}{\gamma + \beta}$, the claim follows.

B. Proof of Lemma 2

Consider the Eigen decomposition $A^T A = U \Sigma U^T$, where U is the Eigenvector matrix of $A^T A$ and Σ is a diagonal matrix with the Eigenvalues of $A^T A$ in the diagonal. Since $K^* := (A^T A + \beta I)^{-1}$ and the identity matrix can be written as $I = U U^T$, we have

$$K^* = U diag\{ \frac{1}{\lambda_k + \beta} \}_{k=1}^{n} U^T$$

$$\implies K^* A^T A = U diag\{ \frac{1}{\lambda_k + \beta} \}_{k=1}^{n} U^T U diag\{ \lambda_k \}_{k=1}^{n} U^T$$

$$\implies K^* A^T A = U diag\{ \frac{\lambda_k}{\lambda_k + \beta} \}_{k=1}^{n} U^T,$$

where $\lambda_k > 0$ are the Eigenvalues of the positive definite matrix $A^T A$. Since $\frac{d}{d\lambda_k} \left(\frac{\lambda_k}{\lambda_k + \beta} \right) > 0$, the claim follows.

C. Proof of Proposition 1

Using (2) and observing that $A^T A = \sum_{i=1}^{m} (A^i)^T A^i$, dynamics (10) can be rewritten as

$$x(t+1) = x(t) - \delta K(t) A^T (Ax(t) - b). \quad (13)$$

Define $z(t) := x(t) - x^*$. The objective cost in (1) can be rewritten as $\frac{1}{2} \| Ax - b \|^2$, gradient of which is given by $A^T Ax - A^T b$. Thus, x^* satisfies $A^T A x^* = A^T b$. From (13) and $A^T A x^* = A^T b$, we get

$$z(t+1) = z(t) - \delta K(t) A^T A (x(t) - x^*)$$
$$= (I - \delta K(t) A^T A) z(t)$$
$$= (I - \delta K^* A^T A) z(t) - \delta \tilde{K}(t) A^T A z(t)$$
$$= (I - \delta K^* A^T A) z(t) - \delta u(t), \quad (14)$$

where $\tilde{K}(t)$ comprises of the columns $\tilde{k}_j(t) := k_j(t) - k_j^*$, $j = 1, ..., n$ and $u(t) := \tilde{K}(t) A^T A z(t)$. From Lemma 1, it follows that

$$\left\| \tilde{k}^j(t) \right\|^2 \le (\rho_K^*)^{2t+2} \left\| \tilde{k}^j(-1) \right\|^2$$
$$\implies \| \tilde{K}(t) \|_F^2 \le (\rho_K^*)^{2t+2} \| \tilde{K}(-1) \|_F^2. \quad (15)$$

Now,

$$\| u(t) \| \le \| \tilde{K}(t) \| \| A^T A \| \| z(t) \|$$
$$\le \| \tilde{K}(t) \|_F \| A^T A \| \| z(t) \|$$
$$\le \lambda \| \tilde{K}(-1) \|_F (\rho_K^*)^{t+1} \| z(t) \|, \quad (16)$$

where the last inequality follows from (15). From (14) and (16),

$$\| z(t+1) \| \le \left(\| I - \delta K^* A^T A \| + \sigma_0 (\rho_K^*)^{t+1} \right) \| z(t) \|, \ \forall t. \quad (17)$$

Since $K^* A^T A$ is positive definite for $\beta > 0$ (from Lemma 2), $\exists \delta$ for which $\rho_\beta := \| I - \delta K^* A^T A \| < 1$ and (17) holds. The smallest value of ρ_β is $\frac{\kappa(K^* A^T A) - 1}{\kappa(K^* A^T A) + 1}$. Then with the largest and the smallest Eigenvalues of $K^* A^T A$ obtained in Lemma 2, the claim follows.

D. Proof of Theorem 1

Define, $s_t := \left(\rho_\beta^* + \sigma_0 (\rho_K^*)^{t+1} \right)$. It can be easily checked that $\rho_\beta^* < \rho_{GD}$ for $\beta > 0$. Since $\rho_K^* < 1$ and $\rho_\beta^* < \rho_{GD}$, there exists $\tau < \infty$ such that $s_t < \rho_{GD}, \forall t > \tau$. Define, $r_t := s_t / \rho_{GD}$. Then $r_t < 1 \forall t > \tau$. From the recursion in Proposition 1, we have

$$\| z_1(t+1) \| \le \left(\Pi_{k=\tau+1}^{t} s_k \right) \| z_1(\tau+1) \|, \ \forall t > \tau. \quad (18)$$

Since $\{ s_t > 0 \}_{t \ge 0}$ is a strictly decreasing sequence, from the same recursion we also have

$$\| z_1(\tau+1) \| \le s_\tau \| z_1(\tau) \| \le s_0 \| z_1(\tau) \|$$
$$\implies \| z_1(\tau+1) \| \le s_0^{\tau+1} \| z(0) \|. \quad (19)$$

Combining (18) and (19),

$$\| z_1(\tau+1) \| \le \left(\Pi_{k=\tau+1}^{t} s_k \right) s_0^{\tau+1} \| z(0) \| \le (s_{\tau+1})^{t-\tau} s_0^{\tau+1} \| z(0) \|$$
$$= (r_{\tau+1} \rho_{GD})^{t-\tau} s_0^{\tau+1} \| z(0) \|$$
$$= c_1 (r_{\tau+1} \rho_{GD})^{t+1} \| z(0) \|, \ \forall t > \tau$$

where $c_1 = \left(\frac{s_0}{r_{\tau+1} \rho_{GD}} \right)^{\tau+1}$ is a constant. Thus, we have

$$E_1(t+1) = c_1 (r_{\tau+1} \rho_{GD})^{t+1} \| z(0) \|$$
$$= c_1 (r_{\tau+1})^{t+1} E_2(t+1), \ \forall t > \tau.$$

Since $r_{\tau+1} < 1$ and c_1 is a constant, $\exists t_{sw} < \infty$ such that $c_1 (r_{\tau+1})^{t+1} < 1 \forall t > t_{sw}$. This completes the proof.