

STRAGGLER ROBUST DISTRIBUTED MATRIX INVERSE APPROXIMATION

Neophytos Charalambides[†], Mert Pilanci[‡], and Alfred O. Hero III[†],

[†]EECS Department University of Michigan, [‡]EE Department Stanford University

ABSTRACT

A cumbersome operation in numerical analysis and linear algebra, optimization, machine learning and engineering algorithms; is inverting large full-rank matrices which appears in various processes and applications [1]. This has both numerical stability and complexity issues, as well as high expected time to compute. We address the latter issue, by proposing an algorithm which uses a black-box least squares optimization solver as a subroutine, to give an estimate of the inverse (and pseudoinverse) of real nonsingular matrices; by estimating its columns. This also gives it the flexibility to be performed in a distributed manner, thus the estimate can be obtained a lot faster, and can be made robust to *stragglers*. Furthermore, we assume a centralized network with no message passing between the computing nodes, and do not require a matrix factorization; e.g. LU, SVD or QR decomposition beforehand.

Index Terms— Numerical linear algebra, numerical analysis, straggler mitigation, approximation algorithms.

1. INTRODUCTION AND RELATED WORK

In numerous applications in domains such as social networks, numerical analysis and integration, machine learning, scientific computing, etc.; “there *are* situations in which a matrix inverse must be computed” [2]. It is one of the most important operations, as it reverses a system. Unfortunately, this is a cumbersome and intricate process. It is equivalent to performing Gaussian elimination, which in general takes $O(n^3)$ for square matrices of order n . There are more sophisticated algorithms which are similar to matrix multiplication algorithms; of lower complexity. In chronological order, the more popular of these algorithms with respective complexity are the ones by Strassen with $O(n^{2.807})$ [3], Coppersmith-Winograd with $O(n^{2.375477})$ [4], and Le Gall with $O(n^{2.3728639})$ [5]. Due to practical issues, the Strassen algorithm is used more often. A lot of other inversion algorithms assume some structure on the matrix, require a matrix-matrix product, or use a matrix factorization; e.g. LU, SVD or QR decomposition [6].

Due to the large amount of datasets which are used in modern machine learning, a method for speeding up the computations is to perform them in a distributed manner; where a network of workers perform certain subtasks in parallel. We point out a few such algorithms [7–14]. Some drawbacks

in these algorithms are that they either make an assumption on the matrix, assume distributed memory, not suitable for centralized computations, are specific for distributed and parallel computing platforms (e.g. Apache Spark and Hadoop, MapReduce, CUDA), require a matrix factorization (e.g. LU, QR) or require heavy and multiple communication instances; which makes them unsuitable for iterative methods. Some of our ideas are similar to [15] and [16].

In this paper, we propose a centralized distributed algorithm which *approximates* the inverse of a nonsingular $\mathbf{A} \in \mathbb{R}^{n \times n}$, which makes none of the aforementioned assumptions. The main idea behind the algorithm is that the worker nodes use a least squares solver to approximate a column of \mathbf{A}^{-1} (or multiple), which the central server will then concatenate to obtain the approximation $\widehat{\mathbf{A}}^{-1}$. For simplicity and conciseness in presentation, we will use steepest descent (SD) for our least squares solver, and also present simulation results with conjugate gradients (CG). The algorithm is simple and short, and can be made robust to *stragglers* [17], nodes with longer response time comparing to other nodes. We also extend this idea to distributed approximation of the pseudoinverse $\widehat{\mathbf{A}}^\dagger$ for \mathbf{A} full-rank, which may be of more interest to the machine learning community.

The paper is organized as follows. In section 2 we recall basics of matrix inversion, least squares approximation and SD. In section 3 we propose our matrix inverse and pseudoinverse algorithms. In section 4 we discuss how in a distributed setting these algorithms can be made robust to stragglers. Finally, in section 5 we present some numerical simulations on randomly generated matrices. The main contributions are:

- A new matrix inverse approximation algorithm, which is extended to a pseudoinverse algorithm as well.
- Theoretical guarantee on approximation error for symmetric positive definite \mathbf{A} .
- Discussion on how in a distributed setting, it can be made robust to stragglers.
- Presentation of experiments that corroborate our theoretical results.

2. PRELIMINARY BACKGROUND

Recall a nonsingular matrix is a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ of full-rank, which has a unique inverse \mathbf{A}^{-1} such that

$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$. The simplest way of computing \mathbf{A}^{-1} is by performing Gaussian elimination on $[\mathbf{A}|\mathbf{I}_n]$; which gives $[\mathbf{I}_n|\mathbf{A}^{-1}]$. Another common method is to decompose $\mathbf{A} = \mathbf{L}\mathbf{U}$ where \mathbf{L}, \mathbf{U} are respectively lower and upper triangular square matrices, which are then inverted by using back and forward substitution, thus $\mathbf{A} = \mathbf{U}^{-1}\mathbf{L}^{-1}$. This decomposition itself requires $O(n^3)$ operations.

For full-rank rectangular matrices $\mathbf{A}^{n \times m}$ where $n > m$, one resorts to the left MoorePenrose pseudoinverse $\mathbf{A}^\dagger \in \mathbb{R}^{m \times n}$, for which $\mathbf{A}^\dagger \mathbf{A} = \mathbf{I}_m$. In algorithm 2 we present how to approximate the left pseudoinverse of \mathbf{A} , where we use the fact that for \mathbf{A} full-ranked; $\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$. The right pseudoinverse of a matrix $\mathbf{A}^{m \times n}$ can be obtained similarly; $\mathbf{A}^\dagger = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1}$.

Another well-studied problem is that of estimating the least squares solution

$$\theta_{ls}^* = \arg \min_{\theta \in \mathbb{R}^m} \{ \|\mathbf{A}\theta - \mathbf{y}\|_2^2 \} \quad (1)$$

for $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{y} \in \mathbb{R}^n$, for which $n \gg m$ in many applications; where the rows represent the feature vectors of a dataset. This has the closed-form solution $\theta_{ls}^* = \mathbf{A}^\dagger \mathbf{y}$. Note that when \mathbf{A} has a null-space; $\hat{\theta}_{ls}$ is not unique, since for any $\mathbf{z} \in \text{null}(\mathbf{A})$ we have $\mathbf{A}(\theta_{ls}^* + \mathbf{z}) = \mathbf{A}\theta_{ls}^* + \mathbf{0} = \mathbf{A}\theta_{ls}^*$.

The main idea behind the least squares problem is to find the orthogonal projection of \mathbf{y} in $\text{col-span}(\mathbf{A})$. By the rank-nullity theorem and the assumption that $n \gg m$ it follows that $\text{null}(\mathbf{A}) \neq \emptyset$, hence there exist infinitely-many solutions to (1). Under the assumption that $\text{rank}(\mathbf{A}) = m$, for the objective function of (1)

$$f(\theta) := \theta^T (\mathbf{A}^T \mathbf{A}) \theta + \mathbf{y}^T \mathbf{y} - 2\theta^T \mathbf{A}^T \mathbf{y}$$

we have $\nabla_\theta^2 f(\theta) = 2\mathbf{A}^T \mathbf{A} \succ 0$, hence its Hessian is nonsingular; and $f(\theta)$ is strictly convex. Furthermore, by setting the gradient to 0 we get

$$\nabla_\theta f(\theta) = 2\mathbf{A}^T (\mathbf{A}\theta - \mathbf{y}) = 0 \implies \hat{\theta}_{ls} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y}$$

which is precisely $\mathbf{A}^\dagger \mathbf{y}$.

Computing \mathbf{A}^\dagger in order to solve (1) directly is itself intractable for large m , as it requires computing the inverse of $\mathbf{A}^T \mathbf{A}$. We address this in subsection 3.3. Instead, we use gradient methods to get *approximate* solutions; e.g. SD or CG, which require less operations, and can be done distributively. One could of course use other methods, e.g. Newton's.

We briefly recall SD. When considering a minimization problem with a convex differentiable objective function $\phi : \Theta \rightarrow \mathbb{R}$ over an open constrained set $\Theta \subseteq \mathbb{R}^m$, we select an initial $\theta^{(0)} \in \Theta$ and repeat:

$$\theta^{(k+1)} = \theta^{(k)} - t_k \cdot \nabla_\theta \phi(\theta^{(k)}), \quad \text{for } k = 1, 2, 3, \dots$$

until a termination criterion is met, which criterion may depend on the problem we are looking at. The parameter t_k is the step-size, which may be adaptive or fixed. In our experiments, we use backtracking line search to determine t_k .

3. APPROXIMATION ALGORITHMS

3.1. Proposed Inverse Algorithm

Our goal is to estimate $\mathbf{A}^{-1} = [\mathbf{b}_1 \cdots \mathbf{b}_n]$, for \mathbf{A} a square matrix of order n . A key property to note is

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}[\mathbf{b}_1 \cdots \mathbf{b}_n] = [\mathbf{A}\mathbf{b}_1 \cdots \mathbf{A}\mathbf{b}_n] = \mathbf{I}_n$$

which implies that $\mathbf{A}\mathbf{b}_i = \mathbf{e}_i$ for all $i \in \mathbb{N}_n := \{1, \dots, n\}$, where \mathbf{e}_i are the standard basis column vectors. Assume for now that we use any black-box least squares solver to estimate

$$\hat{\mathbf{b}}_i = \arg \min_{\mathbf{b} \in \mathbb{R}^n} \overbrace{\{\|\mathbf{A}\mathbf{b} - \mathbf{e}_i\|_2^2\}}^{f_i(\mathbf{b})} \quad (2)$$

which we call n times, to estimate $\widehat{\mathbf{A}}^{-1} = [\hat{\mathbf{b}}_1 \cdots \hat{\mathbf{b}}_n]$. This approach may be viewed as solving

$$\widehat{\mathbf{A}}^{-1} = \arg \min_{\mathbf{B} \in \mathbb{R}^{n \times n}} \{ \|\mathbf{A}\mathbf{B} - \mathbf{I}_n\|_F^2 \}.$$

Alternatively, one could estimate the rows of \mathbf{A}^{-1} . The algorithm performed by a single server is shown in 1.

Algorithm 1: Estimating \mathbf{A}^{-1}

Input: nonsingular $\mathbf{A} \in \mathbb{R}^{n \times n}$

Output: estimate $\widehat{\mathbf{A}}^{-1}$ of \mathbf{A} 's inverse

for $i=1$ to n **do**

 solve $\hat{\mathbf{b}}_i = \arg \min_{\mathbf{b} \in \mathbb{R}^n} \{ \|\mathbf{A}\mathbf{b} - \mathbf{e}_i\|_2^2 \}$

end

$\widehat{\mathbf{A}}^{-1} \leftarrow [\hat{\mathbf{b}}_1 \cdots \hat{\mathbf{b}}_n]$

return $\widehat{\mathbf{A}}^{-1}$

In the case where SD is used to estimate each column; the overall operation count is $O(nTn^2)$, where T is the average number of iterations used per column estimation. The overall error of our estimate may be quantified as

- $\text{err}_{\ell_2}(\widehat{\mathbf{A}}^{-1}) := \|\widehat{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\|_2^2$

- $\text{err}_F(\widehat{\mathbf{A}}^{-1}) := \|\mathbf{A}^{-1} - \widehat{\mathbf{A}}^{-1}\|_F^2$

- $\text{err}_{rF}(\widehat{\mathbf{A}}^{-1}) := \frac{\|\mathbf{A}^{-1} - \widehat{\mathbf{A}}^{-1}\|_F^2}{\|\mathbf{A}^{-1}\|_F^2} = \frac{\sum_{i=1}^n \|\mathbf{A}\hat{\mathbf{b}}_i - \mathbf{e}_i\|_2^2}{\|\mathbf{A}^{-1}\|_F^2}$

which we refer to as the ℓ_2 -error, *Frobenius-error* and *relative Frobenius-error* respectively. The corresponding pseudoinverse approximation errors are defined accordingly.

It is clear that \mathbf{A}^{-1} could be estimated in a distributed manner, where we have n servers which each estimate one column; and then a central server aggregates the columns to form $\widehat{\mathbf{A}}^{-1}$. The expected runtime assuming no delays in this case is therefore $O(T_{\max} n^2)$, for T_{\max} the maximum number

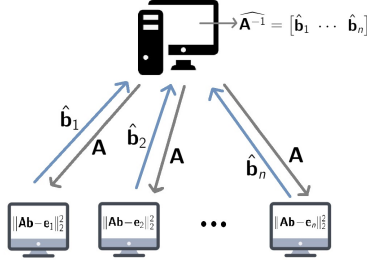


Fig. 1. Communication schematic, with the recovery of $\widehat{\mathbf{A}}^{-1}$.

of iterations of gradient descent used by the workers; to estimate their respective column. This is depicted in figure 3.1.

In contrast to many of the algorithms referenced in the introduction, this algorithm requires the central and worker servers to communicate only once; assuming the workers have knowledge of \mathbf{A} , and does not deal with submatrices of \mathbf{A} or \mathbf{A}^{-1} , making it simpler. Furthermore, the communication load is optimal; in the sense that the workers send the same amount of information once. There is also flexibility for the workers to estimate more than one columns if there are less than n workers.

3.2. Frobenius Error

In order to bound $\text{err}_{rF}(\widehat{\mathbf{A}}^{-1})$, we first upper bound the numerator and then lower bound the denominator. Since $\|\mathbf{A}^{-1} - \widehat{\mathbf{A}}^{-1}\|_F^2 = \sum_{i=1}^n \|\mathbf{A}^{-1}\mathbf{e}_i - \hat{\mathbf{b}}_i\|_2^2$, bounding the numerator reduces to bounding $\|\mathbf{A}^{-1}\mathbf{e}_i - \hat{\mathbf{b}}_i\|_2^2$ for all $i \in \mathbb{N}_n$. This calculation is straightforward

$$\begin{aligned} \|\mathbf{A}^{-1}\mathbf{e}_i - \hat{\mathbf{b}}_i\|_2^2 &\leq \|\mathbf{A}^{-1}\mathbf{e}_i\|_2^2 + \|\hat{\mathbf{b}}_i\|_2^2 \\ &\leq \|\mathbf{A}^{-1}\|_2^2 + \|\mathbf{e}_i\|_2^2 + \|\hat{\mathbf{b}}_i\|_2^2 \\ &= 1/\sigma_{\min}(\mathbf{A})^2 + 1 + \|\hat{\mathbf{b}}_i\|_2^2 \end{aligned}$$

where both inequalities follow from the triangle inequality. For the denominator; by the definition of the Frobenius norm

$$\|\mathbf{A}^{-1}\|_F^2 = \sum_{i=1}^n \frac{1}{\sigma_i(\mathbf{A})^2} \geq \frac{n}{\sigma_{\max}(\mathbf{A})^2}.$$

By combining the two we get

$$\begin{aligned} \text{err}_{rF}(\widehat{\mathbf{A}}^{-1}) &\leq \frac{n(1 + 1/\sigma_{\min}(\mathbf{A})^2) + \sum_{i=1}^n \|\hat{\mathbf{b}}_i\|_2^2}{n/\sigma_{\max}(\mathbf{A})^2} \\ &= \kappa_2(\mathbf{A})^2 + \sigma_{\max}(\mathbf{A})^2 \cdot \left(1 + \frac{1}{n} \sum_{i=1}^n \|\hat{\mathbf{b}}_i\|_2^2\right) \end{aligned}$$

for $\kappa_2(\mathbf{A}) = \|\mathbf{A}\|_2 \|\mathbf{A}^{-1}\|_2$ the condition number of \mathbf{A} . This shows a dependency on the estimates, and is an additive error bound in terms of the condition number of the problem. The following proposition gives an error bound when using SD for our subroutine.

Proposition 1. For $\mathbf{A} \succ 0$, we have $\text{err}_F(\widehat{\mathbf{A}}^{-1}) \leq \frac{n\epsilon^2}{2}$ and $\text{err}_{rF}(\widehat{\mathbf{A}}^{-1}) \leq \frac{n\epsilon^2/2}{\sigma_{\min}(\mathbf{A})^2}$, when using SD to solve (2) with termination criterion $\|\nabla f_i(\mathbf{b}^{(k)})\|_2 \leq \epsilon$, for all $i \in \mathbb{N}_n$.

In section section 5 we observe this for general random matrices, not just positive definite matrices. The dependence on $1/\sigma_{\min}(\mathbf{A})$ may be an artifact of using use gradient methods, since the error will be multiplied by \mathbf{A}^{-1} . In theory, this can be annihilated if one runs the algorithm on $d\mathbf{A}$ for $d \simeq 1/\sigma_{\min}(\mathbf{A})$, and then multiply the result by d . The scalar d should not be selected a relatively large (w.r.t. $1/\sigma_{\min}(\mathbf{A})$), as it would most likely result in $\widehat{\mathbf{A}}^{-1} \simeq \mathbf{0}_{n \times n}$.

Proof. We know that for $f(\mathbf{b})$ strongly convex (equivalently in our case; \mathbf{A} positive definite) that

$$\|\nabla f_i(\mathbf{b})\|_2 \leq \sqrt{2\sigma_{\min}(\mathbf{A})^2\alpha} \implies f_i(\mathbf{b}) - f_i(\mathbf{b}_{i_s}^*) \leq \alpha.$$

For us $\epsilon = \sqrt{2\sigma_{\min}(\mathbf{A})^2\alpha}$, thus $\alpha = \frac{1}{2} \cdot \left(\frac{\epsilon}{\sigma_{\min}(\mathbf{A})}\right)^2$, so when solving (2) we have

$$f_i(\mathbf{b}) - f_i(\mathbf{b}_{i_s}^*) = f_i(\mathbf{b}) - 0 = \|\mathbf{A}\hat{\mathbf{b}}_i - \mathbf{e}_i\|_2^2$$

hence

$$\|\mathbf{A}\hat{\mathbf{b}}_i - \mathbf{e}_i\|_2^2 \leq \frac{1}{2} \cdot \left(\frac{\epsilon}{\sigma_{\min}(\mathbf{A})}\right)^2 \quad (3)$$

for all $i \in \mathbb{N}_n$. We want an upper bound for each summand $\|\mathbf{A}^{-1}\mathbf{e}_i - \hat{\mathbf{b}}_i\|_2^2$ of the numerator of $\text{err}_{rF}(\widehat{\mathbf{A}}^{-1})$

$$\begin{aligned} \|\mathbf{A}^{-1}\mathbf{e}_i - \hat{\mathbf{b}}_i\|_2^2 &= \|\mathbf{A}^{-1}(\mathbf{e}_i - \mathbf{A}\hat{\mathbf{b}}_i)\|_2^2 \\ &\leq \|\mathbf{A}^{-1}\|_2^2 \cdot \|\mathbf{e}_i - \mathbf{A}\hat{\mathbf{b}}_i\|_2^2 \\ &\stackrel{\#}{\leq} \|\mathbf{A}^{-1}\|_2^2 \cdot \frac{1}{2} \cdot \left(\frac{\epsilon}{\sigma_{\min}(\mathbf{A})}\right)^2 \\ &= \frac{\epsilon^2}{2} \end{aligned}$$

where $\#$ follows from (3), thus $\text{err}_F(\widehat{\mathbf{A}}^{-1}) \leq \frac{n\epsilon^2}{2}$. Plugging this into the definition of $\text{err}_{rF}(\widehat{\mathbf{A}}^{-1})$ we get

$$\text{err}_{rF}(\widehat{\mathbf{A}}^{-1}) \leq \frac{\|\mathbf{A}^{-1}\|_2^2}{\|\mathbf{A}^{-1}\|_F^2} \cdot \frac{n}{2} \cdot \left(\frac{\epsilon}{\sigma_{\min}(\mathbf{A})}\right)^2 \stackrel{\ddagger}{\leq} \frac{n\epsilon^2/2}{\sigma_{\min}(\mathbf{A})^2}$$

where \ddagger follows from the fact that $\|\mathbf{A}^{-1}\|_2^2 \leq \|\mathbf{A}^{-1}\|_F^2$. \square

3.3. Proposed Pseudoinverse Algorithm

Just like the inverse, the pseudoinverse of a matrix also appears in many applications and computations; throughout a variety of fields. This is even more cumbersome, as itself requires computing an inverse. For this subsection, we consider $\mathbf{A} \in \mathbb{R}^{n \times m}$ for $n > m$.

One could naively attempt to modify algorithm 1 in order to retrieve \mathbf{A}^\dagger such that $\mathbf{A}^\dagger \mathbf{A} = \mathbf{I}_m$, by approximating its

rows. This would *not* work, as the underlying optimizations problems are no longer strictly convex. Fortunately, we can use algorithm 1 to estimate $\mathbf{B}^{-1} := (\mathbf{A}^T \mathbf{A})^{-1}$, and then multiply by \mathbf{A}^T . The multiplication may be done by the workers (who will estimate \mathbf{A}^\dagger 's rows) or the central server. For coherence and applicability to section 4, we present the former.

The additional operation count compared to algorithm 1, is $O(m^2 n)$ for computing \mathbf{B} , and $O(mn)$ per worker for computing $\hat{\mathbf{b}}_i$.

Algorithm 2: Estimating \mathbf{A}^\dagger

Input: full-rank $\mathbf{A} \in \mathbb{R}^{n \times m}$ where $n > m$

Output: estimate $\widehat{\mathbf{A}}^\dagger$ of \mathbf{A} 's pseudoinverse

$\mathbf{B} \leftarrow \mathbf{A}^T \mathbf{A}$

for $i=1$ to m **do**

$$\begin{aligned} & \text{solve } \hat{\mathbf{c}}_i = \arg \min_{\mathbf{c} \in \mathbb{R}^{1 \times m}} \overbrace{\{\|\mathbf{c}_i \mathbf{B} - \mathbf{e}_i^T\|_2^2\}}^{g_i(\mathbf{c})} \\ & \hat{\mathbf{b}}_i \leftarrow \hat{\mathbf{c}}_i \cdot \mathbf{A}^T \end{aligned}$$

end

$$\widehat{\mathbf{A}}^\dagger \leftarrow [\hat{\mathbf{b}}_1^T \dots \hat{\mathbf{b}}_m^T]^T$$

return $\widehat{\mathbf{A}}^\dagger$

Corollary 2. For full-rank $\mathbf{A} \in \mathbb{R}^{n \times m}$ with $n > m$, we have $\text{err}_F(\widehat{\mathbf{A}}^\dagger) \leq m \cdot \left(\frac{\epsilon \cdot \kappa_2(\mathbf{A})}{\sqrt{2} \sigma_{\min}(\mathbf{A})}\right)^2$ and $\text{err}_{rF}(\widehat{\mathbf{A}}^\dagger) \leq \frac{m \epsilon^2 \cdot \kappa_2(\mathbf{A})^2}{2}$ when using SD to solve the subroutine optimization problems of algorithm 2; with termination criterion $\|\nabla g_i(\mathbf{c}^{(k)})\|_2 \leq \epsilon$.

Proof. The idea resembles the proof of 3.1, so we omit some of the details. Similarly to the derivation of (3), we get

$$\|\mathbf{B}^{-1} \mathbf{e}_i - \hat{\mathbf{c}}_i^T\|_2^2 \leq \left(\frac{\epsilon/\sqrt{2}}{\sigma_{\min}(\mathbf{B})}\right)^2 = \left(\frac{\epsilon/\sqrt{2}}{\sigma_{\min}(\mathbf{A})}\right)^2 =: \delta$$

which implies for each summand of $\text{err}_F(\widehat{\mathbf{A}}^\dagger)$; $\|\hat{\mathbf{b}}_i - \mathbf{A}_{i*}^\dagger\|_2^2 = \|\hat{\mathbf{c}}_i \mathbf{A}^T - \mathbf{e}_i \cdot \mathbf{B}^{-1} \mathbf{A}^T\|_2^2$, we have $\|\hat{\mathbf{b}}_i - \mathbf{A}_{i*}^\dagger\|_2^2 \leq \delta \|\mathbf{A}^T\|_2^2$. Summing the right hand side m times and using the fact that $1/\sigma_{\min}(\mathbf{A})^2 = \|\mathbf{A}^\dagger\|_2^2 \leq \|\mathbf{A}^\dagger\|_F^2$, completes the proof. \square

4. ROBUST TO STRAGGLERS

One further motivation for this algorithm, is that it can also be made robust to stragglers. There is extensive literature on matrix-matrix; matrix-vector multiplication and computing the gradient with the presence of stragglers, though there does not seem to be anything on computing the inverse of a matrix with the presence of straggler nodes.

The encoding-decoding scheme $(\mathbf{B}, \mathbf{a}_T)$ from [18] can be adapted to this setting, when the proposed algorithms are deployed in a distributed manner. The correspondence between

(i) the gradient coding scheme and (ii) distributed matrix inversion, is that the partial gradients in (i) will correspond to the set of columns a worker in (ii) is assigned to compute. It is crucial that there is no overlap between the assigned columns of congruent workers [18]. This gradient coding scheme sets a framework for other straggler tolerant distributed ‘‘linear’’ computations; e.g. matrix multiplication. The main idea behind this scheme relies on the pigeonhole principle. In such a scenario, the communication load per worker will scale according to the number of rows it is assigned.

Additionally, since the encoding can be viewed as a task assignment, the columns can be sent in a sequential manner. The overall communication instances will be more, though the overall communication load will be less on average.

Furthermore, this suggests a *distributed Newton’s method* which is robust to stragglers, similar in spirit to [19]. Parallel to gradient coding [17]; where the crux is to recover the gradient of the objective function in order to perform SD, in Newton’s method it is to recover the inverse of the Hessian.

5. EXPERIMENTS

The accuracy of the proposed algorithms was tested on randomly generated matrices, using both SD and CG [6] for the subroutine optimization problems. Some of the results are summarized below. The depicted results are averages of 20 runs, with termination criteria $\|\nabla f_i(\mathbf{b}^{(k)})\|_2 \leq \epsilon$ for SD and $\|\mathbf{b}_i^{(k)} - \mathbf{b}_i^{(k-1)}\|_2 \leq \epsilon$ for CG; for the presented ϵ accuracy parameters. The criteria for $\widehat{\mathbf{A}}^\dagger$ were analogous. We considered $\mathbf{A} \in \mathbb{R}^{100 \times 100}$ and $\mathbf{A} \in \mathbb{R}^{100 \times 50}$. The error subscripts represent $\mathcal{A} = \{\ell_2, F, rF\}$, $\mathcal{N} = \{\ell_2, F\}$, $\mathcal{F} = \{F, rF\}$.

Average $\widehat{\mathbf{A}}^{-1}$ errors for $\mathbf{A} \sim 50 \cdot \mathcal{N}(0, 1)$ — SD					
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
$\text{err}_{\mathcal{A}}$	$\mathcal{O}(10^{-2})$	$\mathcal{O}(10^{-5})$	$\mathcal{O}(10^{-7})$	$\mathcal{O}(10^{-9})$	$\mathcal{O}(10^{-12})$

Mean $\widehat{\mathbf{A}}^{-1}$ errors, $\mathbf{A} = \mathbf{M} + \mathbf{M}^T$; $\mathbf{M} \sim 25 \cdot \mathcal{N}(0, 1)$ — CG					
	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
$\text{err}_{\mathcal{N}}$	$\mathcal{O}(10^{-3})$	$\mathcal{O}(10^{-5})$	$\mathcal{O}(10^{-8})$	$\mathcal{O}(10^{-11})$	$\mathcal{O}(10^{-12})$
err_{rF}	$\mathcal{O}(10^{-3})$	$\mathcal{O}(10^{-5})$	$\mathcal{O}(10^{-7})$	$\mathcal{O}(10^{-10})$	$\mathcal{O}(10^{-12})$

Average $\widehat{\mathbf{A}}^\dagger$ errors for $\mathbf{A} \sim \mathcal{N}(0, 1)$ — SD					
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
err_{ℓ_2}	$\mathcal{O}(10^{-4})$	$\mathcal{O}(10^{-6})$	$\mathcal{O}(10^{-8})$	$\mathcal{O}(10^{-10})$	$\mathcal{O}(10^{-12})$
$\text{err}_{\mathcal{F}}$	$\mathcal{O}(10^{-5})$	$\mathcal{O}(10^{-7})$	$\mathcal{O}(10^{-9})$	$\mathcal{O}(10^{-11})$	$\mathcal{O}(10^{-13})$

Average $\widehat{\mathbf{A}}^\dagger$ errors for $\mathbf{A} \sim \mathcal{N}(0, 1)$ — CG					
	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
err_{ℓ_2}	$\mathcal{O}(10^{-4})$	$\mathcal{O}(10^{-6})$	$\mathcal{O}(10^{-8})$	$\mathcal{O}(10^{-10})$	$\mathcal{O}(10^{-12})$
$\text{err}_{\mathcal{F}}$	$\mathcal{O}(10^{-2})$	$\mathcal{O}(10^{-3})$	$\mathcal{O}(10^{-8})$	$\mathcal{O}(10^{-10})$	$\mathcal{O}(10^{-12})$

6. ACKNOWLEDGEMENTS

This work was partially supported by grant ARO W911NF-15-1-0479. We would also like to thank Nima Mohseni for a suggestion in 3.3.

7. REFERENCES

- [1] Bernard G. Greenberg and Ahmed E. Sarhan. Matrix inversion, its interest and application in analysis of data. 1959.
- [2] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, USA, 2nd edition, 2002.
- [3] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.
- [4] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.
- [5] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.
- [6] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.
- [7] Chandan Misra, Sourangshu Bhattacharya, and Soumya K. Ghosh. SPIN: A fast and scalable matrix inversion method in apache spark. *CoRR*, abs/1801.04723, 2018.
- [8] Jun Liu, Yang Liang, and Nirwan Ansari. Spark-based large-scale matrix inversion for big data processing. *IEEE Access*, 4:2166–2176, 2016.
- [9] Jingen Xiang, Huangdong Meng, and Ashraf Aboul-naga. Scalable matrix inversion using mapreduce. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 177–190, 2014.
- [10] Enrique S Quintana, Gregorio Quintana, Xiaobai Sun, and Robert van de Geijn. A note on parallel matrix inversion. *SIAM Journal on Scientific Computing*, 22(5):1762–1771, 2001.
- [11] Enrique S. Quintana, Gregorio Quintana, Xiaobai Sun, and Robert van de Geijn. Efficient matrix inversion via Gauss-Jordan elimination and its parallelization. Technical report, USA, 1998.
- [12] Kaiqi Yang, Yubai Li, and Yijia Xia. A parallel method for matrix inversion based on Gauss-Jordan algorithm. *Journal of Computational Information Systems*, 9(14):5561–5567, 2013.
- [13] KK Lau, MJ Kumar, and R Venkatesh. Parallel matrix inversion techniques. In *Proceedings of 1996 IEEE Second International Conference on Algorithms and Architectures for Parallel Processing, ICA/sup 3/PP’96*, pages 515–521. IEEE, 1996.
- [14] David H Bailey and HRP Gerguson. A Strassen-Newton algorithm for high-speed parallelizable matrix inversion. In *Proceedings of the 1988 ACM/IEEE conference on Supercomputing*, pages 419–424. IEEE Computer Society Press, 1988.
- [15] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yan-nis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77, 2011.
- [16] Zhaojun Bai, Mark Fahey, and Gene Golub. Some large scale matrix computation problems. *J. Comput. Appl. Math.*, 74:71–89.
- [17] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.
- [18] Neophytos Charalambides, Hessam Mahdavi-far, and Alfred O. Hero. Numerically stable binary gradient coding. *arXiv preprint arXiv:2001.11449*, 2020.
- [19] Vipul Gupta, Swanand Kadhe, Thomas Courtade, Michael W Mahoney, and Kannan Ramchandran. Oversketching newton: Fast convex optimization for serverless systems. *arXiv preprint arXiv:1903.08857*, 2019.