axioms

MDPI

Article

A Randomized Distributed Kaczmarz Algorithm and Anomaly Detection

Fritz Keinert and Eric S. Weber *

Department of Mathematics, Iowa State University, 396 Carver Hall, Ames, IA 50011, USA; keinert@iastate.edu
* Correspondence: esweber@iastate.edu

Abstract: The Kaczmarz algorithm is an iterative method for solving systems of linear equations. We introduce a randomized Kaczmarz algorithm for solving systems of linear equations in a distributed environment, i.e., the equations within the system are distributed over multiple nodes within a network. The modification we introduce is designed for a network with a tree structure that allows for passage of solution estimates between the nodes in the network. We demonstrate that the algorithm converges to the solution, or the solution of minimal norm, when the system is consistent. We also prove convergence rates of the randomized algorithm that depend on the spectral data of the coefficient matrix and the random control probability distribution. In addition, we demonstrate that the randomized algorithm can be used to identify anomalies in the system of equations when the measurements are perturbed by large, sparse noise.

Keywords: Kaczmarz method; linear equations; random control; distributed optimization; stochastic gradient descent; sparse noise; anomaly detection

MSC: 65F10; 15A06; 68W15; 41A65

Citation: Keinert, F.; Weber, E.S. A Randomized Distributed Kaczmarz Algorithm and Anomaly Detection. *Axioms* **2022**, *11*, 106. https://doi.org/10.3390/axioms11030106

Academic Editor: Luigi Brugnano

Received: 10 December 2021
Accepted: 23 February 2022
Published: 26 February 2022

1. Introduction

The Kaczmarz method [1] is an iterative algorithm for solving a system of linear equations $A\vec{x} = \vec{b}$, where A is an $m \times k$ matrix. Written out, the equations are $\vec{a}_i^* \vec{x} = b_i$ for $i = 1, \ldots, m$, where $\vec{a}_i^*$ is the ith row of the matrix A. Given a solution guess $\vec{x}^{(n-1)}$ and an equation number i, we calculate $r_i = b_i - \vec{a}_i^* \vec{x}^{(n-1)}$ (the residual for equation i), and define

$$\vec{x}^{(n)} = \vec{x}^{(n-1)} + \frac{r_i}{\|\vec{a}_i\|^2} \vec{a}_i. \tag{1}$$

This makes the residual of $\vec{x}^{(n)}$ in equation i equal to 0. Here and elsewhere, $\|\cdot\|$ is the usual Euclidean (ℓ^2) norm. We iterate repeatedly through all equations (i.e., we consider $\lim_{n\to\infty} \vec{x}^{(n)}$ where $n \equiv i \mod m$, so the equations are repeated cyclically). Kaczmarz proved that if the system of equations has a unique solution, then $\vec{x}^{(n)}$ converges to that solution. Later, it was proved in [2] that if the system is consistent (but the solution is not unique), then the sequence converges to the solution of minimal norm. Likewise, it was proved in [3,4] that if inconsistent, a relaxed version of the algorithm can provide approximations to a weighted least-squares solution.

A protocol was introduced in [5] to utilize the Kaczmarz algorithm to solve a system of equations that are distributed across a network; each node on the network has one equation, and the equations are indexed by the nodes of the network. We consider the network to be a graph and select from the graph a minimal spanning tree. The iteration begins with a single estimate of the solution at the root of the tree. The root updates this estimate using the Kaczmarz update as in Equation (1) according to its equation then passes that updated estimate to its neighbors. Each of these nodes in turn updates the estimate it receives using its equation then passes that updated estimate to it neighbors (except its

predecessor). This recursion continues until the estimates reach the leaves of the tree. The multiple estimates are then aggregated by each leaf passing its estimate to its predecessor; each of these nodes then take weighted averages of all of its inputs. This second recursion continues until reaching the root; the single estimate at the root then becomes the input for the next iteration. To formalize this, we first introduce some notation.

1.1. Notation

A tree is a connected graph with no cycles. We denote arbitrary nodes (vertices) of a tree by v, u. Let V denote the collection of all nodes in the tree. Our tree will be rooted; the root of the tree is denoted by r. Following the notation from MATLAB, when v is on the path from r to u, we will say that v is a *predecessor* of u and write $u \prec v$. Conversely, u is a *successor* of v. By *immediate successor* of v we mean a successor u such that there is an edge between v and u (this is referred to as a *child* in graph theory parlance [6]); similarly, v is an *immediate predecessor* (i.e., *parent*) if v is a predecessor of u. We denote the set of all immediate successors of node v by $\mathcal{C}(v)$; we will also use $\mathcal{P}(u)$ to denote the parent (i.e., immediate predecessor) of node u. A node without a successor is called a *leaf*; leaves of the tree are denoted by ℓ. We will denote the set of all leaves by $\mathcal{L}$. Often we will have need to enumerate the leaves as $\ell_1, \ldots, \ell_t$, hence t denotes the number of leaves.

A *weight* w is a nonnegative function on the edges of the tree; we denote this by $w(u, v)$, where u and v are nodes that have an edge between them. We assume $w(u, v) = w(v, u)$, though we will typically write $w(u, v)$ when $u \prec v$. When $u \prec v$, but u is not a immediate successor, we write

$$w(u, v) := \prod_{j=1}^{J-1} w(u_{j+1}, u_j) \tag{2}$$

where $v = u_1, \ldots, u_J = u$ is a path from v to u.

We let Π denote the affine orthogonal projection onto the solution space of the matrix equation $A\vec{x} = \vec{b}$. For a positive semidefinite matrix C, $\lambda_{max}(C)$ denotes the largest eigenvalue, and $\lambda_{min}^{nz}(C)$ denotes the smallest nonzero eigenvalue.

1.2. The Distributed Kaczmarz Algorithm

The iteration begins with an estimate, say $\vec{x}^{(n)}$ at the root of the tree (we denote this by $\vec{x}_r^{(n)}$). Each node u receives from its immediate predecessor v an input estimate $\vec{x}_v^{(n)}$ and generates a new estimate via the Kaczmarz update:

$$\vec{x}_u^{(n)} = \vec{x}_v^{(n)} + \frac{r_u(\vec{x}_v^{(n)})}{\|\vec{a}_u\|^2}\vec{a}_u, \tag{3}$$

where the residual is given by

$$r_u(\vec{x}_v^{(n)}) := b_u - \vec{a}_u^* \vec{x}_v^{(n)}. \tag{4}$$

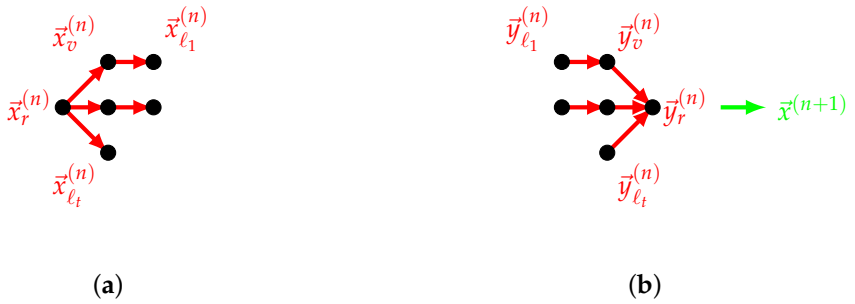
Node u then passes this estimate to all of its immediate successors, and the process is repeated recursively. We refer to this as the *dispersion stage*. Once this process has finished, each leaf ℓ of the tree now possesses an estimate: $\vec{x}_\ell^{(n)}$.

The next stage, which we refer to as the *pooling stage*, proceeds as follows. For each leaf, set $\vec{y}_\ell^{(n)} = \vec{x}_\ell^{(n)}$. Each node v calculates an updated estimate as:

$$\vec{y}_v^{(n)} = \sum_{u \in \mathcal{C}(v)} w(u, v)\vec{y}_u^{(n)}, \tag{5}$$

subject to the constraints that $w(u, v) > 0$ when $u \in \mathcal{C}(v)$ (the set of all immediate successors of v) and $\sum_{u \in \mathcal{C}(v)} w(u, v) = 1$. This process continues until reaching the root of the tree, resulting in the estimate $\vec{y}_r^{(n)}$.

We set $\vec{x}^{(n+1)} = \vec{y}_r^{(n)}$, and repeat the iteration. The updates in the dispersion stage (Equation (3)) and pooling stage (Equation (5)) are illustrated in Figure 1.



(**a**) (**b**)

Figure 1. Illustration of updates in the distributed Kaczmarz algorithm with measurements indexed by nodes of the tree. (**a**) Updates disperse through nodes, (**b**) updates pool and pass to next iteration.

1.3. Related Work

Recent variations on the Kaczmarz method allowed for relaxation parameters [2], reordering equations to speed up convergence [7], or considering block versions of the Kaczmarz method with relaxation matrices Ω_i [3]. Relatively recently, choosing the next equation randomly has been shown to improve the rate of convergence of the algorithm [8–10]. Moreover, this randomized version of the Kaczmarz algorithm has been shown to be comparable to the gradient descent method [11]. The randomized version we present in the present article is similar to the Cimmino method [12], which was extended in [13] and is most similar to the greedy method given in [14]. Both of these methods involve averaging estimates, in addition to applying the Kaczmarz update, as we do here. In addition, a special case of our randomized variant is found in [15]. There, the authors analyze a randomized block Kaczmarz algorithm with averaging, as we do; however, they assume that the size of the block is the same for every iteration, which we do not, and they also assume that the weights associated to the averaging and the probabilities for selecting the blocks are proportional to the inverse of the norms of the row vectors of the coefficient matrix, which we also do not do. See Remark 2 for further details.

The situation we consider in the present article can be considered a distributed estimation problem. Such problems have a long history in applied mathematics, control theory, and machine learning. At a high level, similar to our approach, they all involve averaging local copies of the unknown parameter vector interleaved with update steps [16–25]. Recently, a number of protocols for gossip methods, including a variation of the Kaczmarz method, was analyzed in [26].

However, our version of the Kaczmarz method differs from previous work in a few aspects: (i) we assume an *a priori* fixed tree topology (which is more restrictive than typical gossip algorithms); (ii) there is no master node as in parallel algorithms, and no shared memory architecture; (iii) as we will emphasize in Theorem 2, we make no strong convexity assumptions (which is typically needed for distributed optimization algorithms, but see [27,28] for a relaxation of this requirement); and (iv) we make no assumptions on the matrix A, in particular we do not assume that it is nonnegative.

On the other end of the spectrum are algorithms that distribute a computational task over many processors arranged in a fixed network. These algorithms are usually considered in the context of parallel processing, where the nodes of the graph represent CPUs in a highly parallelized computer. See [29] for an overview.

The algorithm we are considering does not really fit either of those categories. It requires more structure than the gossip algorithms, but each node depends on results from other nodes, more than the usual distributed algorithms.

This was pointed out in [29]. For iteratively solving a system of linear equations, a Successive Over-Relaxation (SOR) variant of the Jacobi method is easy to parallelize; standard SOR, which is a variation on Gauss-Seidel, is not. The authors also consider what

they call the *Reynolds method*, which is similar to a Kaczmarz method with all equations being updated simultaneously. Again, this method is easy to parallelize. A sequential version called Reynolds Gauss–Seidel (RGS) can only be parallelized in certain settings, such as the numerical solution of PDEs.

A distributed version of the Kaczmarz algorithm was introduced in [30]. The main ideas presented there are very similar to ours: updated estimates are obtained from prior estimates using the Kaczmarz update with the equations that are available at the node, and distributed estimates are averaged together at a single node (which the authors refer to as a fusion center; for us, it is the root of the tree). In [30], the convergence analysis is limited to the case of consistent systems of equations, and inconsistent systems are handled by Tikhonov regularization [31,32]. Another distributed version was proposed in [33], which has a shared memory architecture. Finally, the Kaczmarz algorithm has been proposed for online processing of data in [34,35]. In these studies, the processing is online and so is neither distributed nor parallel.

The Kaczmarz algorithm, at its most basic, is an alternating projection method, consisting of iterations of (affine) projections. Our distributed Kaczmarz algorithm (whether randomized or not) consists of iterations of averages of these projections. When consistent, the iterations converge to an element of the common fixed point set of these operators. This is a special case of the *common fixed point problem*, and our algorithm is a special case of the *string-averaging methods* for finding a common fixed point set. The string-averaging method has been extensively studied, for example [36–39] in the cyclic control regime, and [40,41] in the random (dynamic) control regime. An application of string averaging methods to medical imaging is found in [42]. The recent study [43] provides an overview of the method and extensive bibliography. While our situation is a special case of these methods, our analysis of the algorithm provides stronger conclusions, since our main results (Theorems 2 and 3) provide explicit estimates on the convergence rates of our algorithm, rather than the qualitative convergence results found in the literature on string-averaging methods. In addition, Theorem 3 provides a convergence analysis even in the inconsistent case, which is typically not available for string-averaging methods, as the standard convergence analysis requires that the common fixed point set is nonempty (though ref. [44] proves that for certain inconsistent cases, convergence is still guaranteed).

1.4. Main Contributions

Our main contributions in this study concern quantitative convergence results of the distributed Kaczmarz algorithm for systems of linear equations that are distributed over a tree. Just as in the case of cyclic control of the classical Kaczmarz algorithm, in our distributed setting, we are able to prove these quantitative results by introducing randomness into the control law of the method. We prove that the random control as described in Algorithm 1 converges at a rate that depends on the parameters of the probability distribution as well as the spectral data of the coefficient matrix. This is in contrast to typical distributed estimation algorithms for which convergence is guaranteed but the convergence rate is unknown.

As a result of this quantitative convergence analysis, we are able to utilize Algorithm 1 to handle the context of (unknown) corrupted equations. Again, this is in contrast to distributed estimation methods or string-averaging methods, which are known to not converge when the system of equations is inconsistent. We note that Algorithm 1 also will not necessarily converge when the system is inconsistent. We suppose that the number of corrupted equations is small in comparison to the total number of equations, and the remaining equations are in fact consistent. If we have an estimate on the number of corrupted equations, by utilizing Algorithms 2 and 3, with high probability we can identify those equations and successfully remove them, thereby finding the solution to the remaining equations. Likewise, in contrast to the string averaging methods, we are are able to prove convergence rates when the solution set (i.e., the fixed point set in the literature on string-averaging methods) is nonempty, as well as handle the case of when the fixed point set is empty provided we have an estimate on the number of outliers (i.e., the number

of equations that can be removed so that the solution set of the remaining equations is nonempty). Our Algorithms 2 and 3 are nearly verbatim those found in [45], as are the theorems (i.e., Theorems 4 and 5) supporting the efficacy of those algorithms. We prove a necessary result (Lemma 1), with the remaining analysis as in [45], which also has an extensive analysis of numerical experiments that we do not reproduce here.

Algorithm 1 Randomized Tree Kaczmarz (RTK) algorithm.

Input $A, \vec{b}, \vec{x}^{(0)}, \mathcal{D}, K$
for $k \leq K$ **do**
  Draw sample $Z \sim \mathcal{D}$
  if $r \in Z$ **then**
    $\vec{x}_r = \vec{x}^{(k-1)} + \dfrac{b_r - \vec{a}_r^* \vec{x}^{(k-1)}}{\|\vec{a}_r\|^2} \vec{a}_r$
  else
    $\vec{x}_r = \vec{x}^{(k-1)}$
  end if
  for $q = 1, \ldots, Depth$ **do**
    for v with $d(v, r) = q$ **do**
      if $v \in Z$ **then**
        $\vec{x}_v = \vec{x}_{\mathcal{P}(v)} + \dfrac{b_v - \vec{a}_v^* \vec{x}_{\mathcal{P}(v)}}{\|\vec{a}_v\|^2} \vec{a}_v$
      else
        $\vec{x}_v = \vec{x}_{\mathcal{P}(v)}$
      end if
    end for
  end for
  for $\ell \in \mathcal{L}$ **do**
    $\vec{y}_\ell = \vec{x}_\ell$
  end for
  for $q = Depth - 1, \ldots, 0$ **do**
    for u with $d(u, r) = q$ **do**
      $T(u) = \{v \in \mathcal{C}(u) : \vec{y}_v \neq \vec{x}_u\}$
      if $T(u) \neq \emptyset$ **then**
        $\vec{y}_u = \dfrac{1}{|T(u)|} \sum_{v \in T(u)} \vec{y}_v$
      else
        $\vec{y}_u = \vec{x}_u$
      end if
    end for
  end for
  $\vec{x}^{(k)} = \vec{y}_r$
end for
return $\vec{x}^{(K)}$

Algorithm 2 Multiple Round Randomized Tree Kaczmarz (MRRTK) algorithm.

Input $A, \vec{b}, \vec{x}^{(0)}, \mathcal{D}, K, W, d$
$S = \emptyset$
for $i \leq W$ **do**
  $\vec{x}^{(K,i)} = RTK(A, \vec{b}, \vec{x}^0, \mathcal{D}, K)$
  $D = argmax_{D \subset [A], |D| = d} \sum_{j \in D} |A\vec{x}^{(K,i)} - \vec{b}|_j$
  $S = S \cup D$
end for
return $\vec{x}$, where $A_{S^C} \vec{x} = \vec{b}_{S^C}$

Algorithm 3 Multiple Round Randomized Tree Kaczmarz (MRRTKUS) algorithm with Unique Selection.

Input $A, \vec{b}, \vec{x}^{(0)}, \mathcal{D}, K, W, d$
$S = \emptyset$
for $i \leq W$ **do**
  $\vec{x}^{(K,i)} = RTK(A, \vec{b}, \vec{x}^{0}, \mathcal{D}, K)$
  $D = argmax_{D \subset [A] \setminus S, |D|=d} \sum_{j \in D} |A\vec{x}^{(K,i)} - \vec{b}|_j$
  $S = S \cup D$
end for
return $\vec{x}$, where $A_{S^C}\vec{x} = \vec{b}_{S^C}$

2. Randomization of the Distributed Kaczmarz Algorithm

The Distributed Kaczmarz Algorithm (DKA) described in Section 1.2 was introduced in [5]. The main results there concern qualitative convergence guarantees of the DKA: Theorems 2 and 4 prove that the DKA converges to the solution (solution of minimal norm) when the system has a unique solution (is consistent, respectively). Theorem 14 proves that when the system of equations is inconsistent, the relaxed version of the DKA converges for every appropriate relaxation parameter, and the limits approximate a weighted least-squares solution. No quantitative estimates of the convergence rate are given in [5], and in fact it is observed in [46] that the convergence rate is dependent upon the topology of the tree as well as the distribution of the equations across the nodes.

2.1. Randomized Variants

In this subsection, we consider randomized variants of the protocol introduced in Section 1.2 (see Algorithm 1). This will allow us to provide quantitative estimates on the rate of convergence in terms of the spectral data of A. We will be using the analysis of the randomized Kaczmarz algorithm as presented in [8] and the analysis of the randomized block Kaczmarz algorithm as presented in [14].

We will have two randomized variants, but both can be thought of in a similar manner. During the dispersion stage of the iteration, one or more of the nodes will be *active*, meaning that the estimate they receive will be updated according to Equation (3) then passed on to its successor nodes (or held if the node is a leaf). The remaining nodes will be *passive*, meaning that the estimate they receive will be passed on to successive nodes without updating. In the first variant, exactly one node will be chosen randomly to be active for the current iteration, and the remaining nodes will be passive. In the second variant, several of the nodes will be chosen randomly to be active subject to the constraint that no two active nodes are in a predecessor-successor relationship. The pooling stage proceeds with the following variation. When a node receives estimates from its successors, it averages only those estimates that differ from its estimate during the dispersion stage. If a node receives estimates from all of its successors that are the same as its own estimate during the dispersion stage, it passes this estimate to its predecessor.

For these random choices, we will require that the root node know the full topology of the network. For each iteration, the root node will select the active nodes for that iteration according to some probability distribution; the nodes that are selected for activation will be notified by the root node during the iteration.

In both of our random variants, we make the assumption that the system of equations is consistent. This assumption is required for the results that we use from [8,14]. As such, no relaxation parameter is needed in our randomized variants, though in Section 2.3 we observe that convergence can be accelerated by over-relaxation. See [9,47] for results concerning the randomized Kaczmarz algorithm in the context of inconsistent systems of equations.

See Algorithm 1 for pseudocode description of the randomized variants; we refer to this algorithm as the Randomized Tree Kaczmarz (RTK) algorithm.

2.1.1. Single Active Nodes

Let Y denote a random variable whose values are in V. In our first randomized variant, the root node selects $v \in V$ according to the probability distribution

$$P(Y = v) = \frac{\|\vec{a}_v\|^2}{\|A\|_F^2};$$

denote this distribution by $\mathcal{D}_0$. Note that this requires the root node to have access to the norms $\{\|\vec{a}_v\|\}_{v \in V}$. During the dispersion stage of iteration n, the node that is selected, denoted by Y_n, is notified by the root node as estimate $\vec{x}^{(n)}$ traverses the tree.

Proposition 1. *Suppose that the sequence of approximations $\vec{x}^{(n)}$ are obtained by Algorithm 1 with distribution $\mathcal{D}_0$. Then, the approximations have the form*

$$\vec{x}^{(n)} = \vec{x}^{(n-1)} + \frac{b_{Y_n} - \vec{a}_{Y_n}^* \vec{x}^{(n-1)}}{\|\vec{a}_{Y_n}\|^2} \vec{a}_{Y_n}.$$

Proof. At the end of the dispersion stage of the iteration, we have that the leaves $\ell \prec Y_n$ possess an estimate that has been updated; all other leaves have estimates that are not updated. Thus, we have $\vec{x}_\ell^{(n-1)} = \vec{x}_{Y_n}^{(n-1)} = \vec{x}^{(n-1)} + \frac{b_{Y_n} - \vec{a}_{Y_n}^* \vec{x}^{(n-1)}}{\|\vec{a}_{Y_n}\|^2} \vec{a}_{Y_n}$ for those $\ell \prec Y_n$, and $\vec{x}_\ell^{(n-1)} = \vec{x}^{(n-1)}$ otherwise.

Then, during the pooling stage, the only nodes that receive an estimate that is different from their estimate during the dispersion stage are $u \succ Y_n$, and those estimates are all $\vec{x}_{Y_n}^{(n-1)}$. Thus, for all such nodes, $\vec{y}_u = \vec{x}_{Y_n}^{(n-1)}$. Every other node has $\vec{y}_u = \vec{x}^{(n-1)}$. Since the root $r \succ Y_n$, we obtain that

$$\vec{x}^{(n)} = \vec{y}_r^{(n-1)} = \vec{x}_{Y_n}^{(n-1)} = \vec{x}^{(n-1)} + \frac{b_{Y_n} - \vec{a}_{Y_n}^* \vec{x}^{(n-1)}}{\|\vec{a}_{Y_n}\|^2} \vec{a}_{Y_n}.$$

□

Corollary 1. *Suppose the sequence of approximations $\vec{x}^{(n)}$ are obtained by Algorithm 1 with distribution $\mathcal{D}_0$. Then, the following linear convergence rate in expectation holds:*

$$\mathbb{E}\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\|^2 \leq \left(1 - \frac{\lambda_{min}^{nz}(A^T A)}{\|A\|_F^2}\right)^n \|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\|^2. \tag{6}$$

Proof. When the blocks are singletons, by Proposition 1, the update is identical to the Randomized Kaczmarz algorithm of [8]. The estimate in Equation (6) is given in [14]. □

2.1.2. Multiple Active Nodes

We now consider blocks of nodes, meaning multiple nodes, that are active during each iteration. For our analysis, we require that the nodes that are active during any iteration are independent of each other in terms of the topology of the tree. Let $\mathbb{P}(V)$ denote the power set of the set of nodes V. Let Z be a random variable whose values are in $\mathbb{P}(V)$ with probability distribution $\mathcal{D}$.

Definition 1. *For $I \in \mathbb{P}(V)$, we say that I satisfies the* incomparable condition *whenever the following holds: for every distinct pair $u, v \in I$, neither $v \prec u$ nor $u \prec v$. We say that the probability distribution $\mathcal{D}$ satisfies the* incomparable condition *whenever the following implication holds: if $I \in \mathbb{P}(V)$ is such that $P(Z = I) > 0$, then I satisfies the incomparable condition.*

Following [14], we define the expectation for each node $v \in V$:

$$p_v = P(v \in Z).$$

We then define the matrix

$$W = \sum_{v \in V} p_v \frac{a_v a_v^*}{\|a_v\|^2}. \tag{7}$$

For $I \in \mathbb{P}(V)$ and $u \in V$, we define

$$T(u, I) = \{w \in \mathcal{C}(u) | \exists v \in I \text{ s.t. } v \preceq w\}$$

We then define for $v \in I$:

$$\gamma(v, I) = \prod_{u \succ v} \frac{1}{|T(u, I)|}. \tag{8}$$

These quantities reflect how estimates travel from the leaves of the tree back to the root. As multiple estimates are averaged at a node in the tree, the node needs to know how many of its descendants have estimates that have been updated, which (essentially) corresponds to how many descendants have been chosen to be active during that iteration. (Note that it is possible that a node could be active, but its estimate is NOT updated because it may be the case that the estimate that is passed to it is already a solution it its equation, but to simplify the analysis, we suppose that this does not occur). The weights $\gamma(v, I)$ are the final weights used in the update when the estimates ultimately return to the root. Note that these quantities depend on the choice of $I \in P(\mathbb{V})$ as well as the topology of the tree itself.

Proposition 2. *Suppose that the sequence of approximations $\vec{x}^{(n)}$ are generated by Algorithm 1, where the probability distribution $\mathcal{D}$ satisfies the incomparable condition. Let Z_n be the n-th sample of the random variable Z. Then, the approximations have the form*

$$\vec{x}^{(n)} = \vec{x}^{(n-1)} + \sum_{v \in Z_n} \gamma(v, Z_n) \frac{b_v - \vec{a}_v^* \vec{x}^{(n-1)}}{\|\vec{a}_v\|^2} \vec{a}_v. \tag{9}$$

Proof. For any node w such that there is no $v \in Z_n$ with $w \prec v$, then $\vec{x}_w^{(n-1)} = \vec{x}_r^{(n-1)} = \vec{x}^{(n-1)}$. However, if there is a $v \in Z_n$ with $w \prec v$, then by the incomparable condition,

$$\vec{x}_w^{(n-1)} = \vec{x}_v^{(n-1)} = \vec{x}^{(n-1)} + \frac{b_v - \vec{a}_v^* \vec{x}^{(n-1)}}{\|\vec{a}_v\|^2} \vec{a}_v. \tag{10}$$

We have in the pooling stage, if $v \in Z_n$ then $\vec{y}_v = \vec{x}_v$. Moreover, for $u = \mathcal{P}(v)$,

$$\vec{y}_u^{(n-1)} = \frac{1}{|T(u, Z_n)|} \sum_{w \in T(u, Z_n)} \vec{y}_w^{(n-1)} = \frac{1}{|T(u, Z_n)|} \sum_{\substack{w \in Z_n \\ w \prec u}} \vec{y}_w^{(n-1)} = \frac{1}{|T(u, Z_n)|} \sum_{\substack{w \in Z_n \\ w \prec u}} \vec{x}_w^{(n-1)}$$

where the second equation follows from the incomparable condition. It now follows by induction that

$$\vec{y}_r^{(n-1)} = \sum_{w \in Z_n} \left(\prod_{w \prec u} \frac{1}{|T(u, Z_n)|} \right) \vec{x}_w^{(n-1)}.$$

We now obtain Equation (9) from Equation (8). □

For $I \in \mathbb{P}(V)$ that satisfies the incomparable condition, we define

$$B_I = \sum_{v \in I} \gamma(v, I) \frac{a_v a_v^*}{\|a_v\|^2} \tag{11}$$

where $\gamma(v, I)$ are as in Equation (8).

Let $\mathcal{D}$ be a probability distribution on $\mathbb{P}(V)$ that satisfies the incomparable condition, and let Z be a $\mathbb{P}(V)$-valued random variable with distribution $\mathcal{D}$. For each $I \in \mathbb{P}(V)$ such that $P(Z = I) > 0$, let

$$\gamma_{min}(I) = \min\{\gamma(v, I) | v \in I\}; \quad \gamma_{max}(I) = \max\{\gamma(v, I) | v \in I\};$$

where $\gamma(v, I)$ are as in Equation (8). Then, let

$$\Gamma(A, \mathcal{D}) = \min\{2\gamma_{min}(I) - \gamma_{max}(I)\lambda_{max}(B_I) | P(Z = I) > 0\}. \tag{12}$$

For notational brevity, we define the quantity

$$\Sigma(A, \mathcal{D}) := 1 - \Gamma(A, \mathcal{D})\lambda_{min}^{nz}(W). \tag{13}$$

We note that *a priori* there is no reason that $\Sigma(A, \mathcal{D}) < 1$. However, we will see in Section 2.2 examples for which Σ is less than 1 as well as conditions which guarantee this inequality.

We will use Theorem 4.1 in [14]. We alter the statement somewhat and include the proof for completeness.

Theorem 1. *Let Z_1 be a sample of the random variable Z with distribution $\mathcal{D}$. Let Π be the projection onto the space of solutions to the system of equations, and let $\vec{x}^{(0)}$ be an initial estimate of a solution that is in the range of A^T. Let*

$$\vec{x}^{(1)} = \vec{x}^{(0)} + \left(\sum_{v \in Z_1} \gamma(v, Z_1) \frac{b_v - \vec{a}_v^* \vec{x}^{(0)}}{\|\vec{a}_v\|^2} \vec{a}_v \right).$$

Then, the following estimate holds in expectation:

$$\mathbb{E}\|\vec{x}^{(1)} - \Pi\vec{x}^{(1)}\|^2 \leq \Sigma(A, \mathcal{D})\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\|^2. \tag{14}$$

Proof. As derived in Theorem 4.1 in [14], we have

$$\|\vec{x}^{(1)} - \Pi\vec{x}^{(1)}\|^2 \leq \left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right)^* \left(I - 2B_{Z_1} + B_{Z_1}^2\right) \left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right). \tag{15}$$

We make the following estimates:

$$B_{Z_1} \succeq \gamma_{min}(Z_1) \left(\sum_{v \in Z_1} \frac{\vec{a}_v \vec{a}_v^*}{\|\vec{a}_v\|^2} \right),$$

and

$$\begin{aligned} B_{Z_1}^2 &\preceq \lambda_{max}(B_{Z_1}) B_{Z_1} \\ &\preceq \lambda_{max}(B_{Z_1}) \gamma_{max}(Z_1) \left(\sum_{v \in Z_1} \frac{\vec{a}_v \vec{a}_v^*}{\|\vec{a}_v\|^2} \right). \end{aligned}$$

We thus obtain the estimate

$$\begin{aligned} I - 2B_{Z_1} + B_{Z_1}^2 &\preceq I - 2\gamma_{min}(Z_1) \left(\sum_{v \in Z_1} \frac{\vec{a}_v \vec{a}_v^*}{\|\vec{a}_v\|^2} \right) + \gamma_{max}(Z_1)\lambda_{max}(B_{Z_1}) \left(\sum_{v \in Z_1} \frac{\vec{a}_v \vec{a}_v^*}{\|\vec{a}_v\|^2} \right) \\ &\preceq I - \Gamma(A, \mathcal{D}) \left(\sum_{v \in Z_1} \frac{\vec{a}_v \vec{a}_v^*}{\|\vec{a}_v\|^2} \right), \end{aligned}$$

from which Equation (15) becomes

$$\|\vec{x}^{(1)} - \Pi\vec{x}^{(1)}\|^2 \leq \left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right)^* \left(I - \Gamma(A,\mathcal{D})\left(\sum_{v\in Z_1} \frac{\vec{a}_v\vec{a}_v^*}{\|\vec{a}_v\|^2}\right)\right)\left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right). \quad (16)$$

Taking the expectation of the left side, we obtain

$$\mathbb{E}\|\vec{x}^{(1)} - \Pi\vec{x}^{(1)}\|^2 \leq \left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right)^* (I - \Gamma(A,\mathcal{D})W)\left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right) \quad (17)$$

$$\leq \Sigma(A,\mathcal{D})\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\|^2. \quad (18)$$

The last inequality follows from the Courant–Fischer inequality applied to the matrix $D^{1/2}A$: for the matrix $W = A^TDA$, with $D = diag\left(\frac{p_v}{\|\vec{a}_v\|^2}; v \in V\right)$, we obtain

$$\left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right)^* W\left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right) = \|D^{1/2}A\left(\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\right)\|^2 \geq \lambda_{min}^{nz}(W)\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\|^2.$$

□

Theorem 2. *Suppose the sequence of approximations $\vec{x}^{(n)}$ are obtained by Algorithm 1 with the distribution $\mathcal{D}$ satisfying the incomparable condition and initialized with $\vec{x}^{(0)} \in \mathcal{R}(A^T)$. Then, the following linear convergence rate in expectation holds:*

$$\mathbb{E}\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\|^2 \leq (\Sigma(A,\mathcal{D}))^n\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\|^2. \quad (19)$$

Proof. We take the expected value of $\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\|^2$ conditioned on the history $Z_1, \ldots, Z_{n-1}$. By Theorem 1, we have the estimate

$$\begin{aligned}
&\mathbb{E}\left\{\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\|^2 : Z_1, \ldots, Z_{n-1}\right\} \leq \mathbb{E}\left\{\|\vec{x}^{(n)} - \Pi\vec{x}^{(n-1)}\|^2 : Z_1, \ldots, Z_{n-1}\right\} \\
&\leq \mathbb{E}\left\{\Sigma(A,\mathcal{D})\|\vec{x}^{(n-1)} - \Pi\vec{x}^{(n-1)}\|^2 : Z_1, \ldots, Z_{n-1}\right\} \\
&\leq \Sigma(A,\mathcal{D})\|\vec{x}^{(n-1)} - \Pi\vec{x}^{(n-1)}\|^2.
\end{aligned}$$

We now take the expectation over the entire history to obtain

$$\mathbb{E}\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\|^2 \leq \Sigma(A,\mathcal{D})\mathbb{E}\|\vec{x}^{(n-1)} - \Pi\vec{x}^{(n-1)}\|^2.$$

The result now follows by iterating. □

Remark 1. *We note here that Theorem 2 recovers Corollary 1 as follows: if $\mathcal{D}$ selects only singletons from $\mathbb{P}(V)$, and selects singleton $\{v\}$ with probability $\frac{\|\vec{a}_v\|^2}{\|A\|_F^2}$, then we have*

$$\gamma_{min}(\{v\}) = \gamma_{max}(\{v\}) = \lambda_{max}(B_{\{v\}}) = \Gamma(A,\mathcal{D}) = 1$$

and $W = \frac{A^TA}{\|A\|_F^2}$. Thus, $\Sigma(A,\mathcal{D}) = 1 - \frac{\lambda_{min}^{nz}(A^TA)}{\|A\|_F^2}$.

Remark 2. *A similar result to Theorem 2 is obtained in [15]. There, the authors make additional assumptions that we do not. First, in [15], each block that is selected always contains the same number of rows; our analysis works when the blocks have different sizes, which is necessary as we will consider in several of our sampling schemes different block sizes. Second, in [15], their analysis requires the assumption that the expectation p_v and the weights $\gamma(v,Z)$ satisfy the following constraint: there exists a $\alpha > 0$ such that for every v, $p_v\gamma(v,Z)\|\vec{a}_v\|^{-2} = \alpha$. We do not make this*

assumption, and in fact this need not hold, since the weights $\gamma(v, Z)$ are determined both by Z and the tree structure. Thus, we do not have control over this quantity.

2.2. Sampling Schemes

We propose here several possible sampling schemes for Algorithm 1 and illustrate their asymptotics in the special case of a binary tree. Recall that m is the number of equations (and hence the number of nodes in the tree), and t is the number of leaves (nodes with no successors) in the tree. To simplify our analysis, we assume that each $\|\vec{a}_v\| = 1$. We note that for any set $I \in \mathbb{P}(V)$, we have the estimate

$$\lambda_{max}(B_I) \leq 1 \tag{20}$$

since $\sum_{v \in I} \gamma(v, I) = 1$. In addition, if we assume that the distribution $\mathcal{D}$ satisfies the condition that the set $\mathcal{I} = \{I \in \mathbb{P}(V) : P(Z = I) > 0\}$ is a partition of V, then

$$\lambda^{nz}_{min}(W) \leq 1 \tag{21}$$

since we have that $\sum_{v \in V} p_v = 1$. As a consequence of these estimates, we obtain the following guarantee that $\Sigma(A, \mathcal{D}) < 1$.

Proposition 3. *Suppose $\mathcal{D}$ satisfies the incomparable condition and that Equation (21) is satisfied. In addition, suppose that for every $I \in \mathcal{I}$ has the property that $2\gamma_{min}(I) - \gamma_{max}(I) > 0$. Then, $\Sigma(A, \mathcal{D}) < 1$.*

Generations. We block the nodes by their distance from the root: $G_k = \{v : d(r, v) = k\}$. If the depth of the tree is K, then we draw from $\{G_0, \ldots, G_{K-1}\}$ uniformly. Here, the probabilities $p_v = \frac{1}{K}$, so we have $W = \frac{1}{K} A^T A$ since we are also assuming that $\|\vec{a}_v\| = 1$. Thus, the spectral data $\lambda^{nz}_{min}(W)$ reduces to $\frac{1}{K}\lambda^{nz}_{min}(A^T A)$. Thus, our convergence rate is

$$\Sigma(A, \mathcal{D}) = 1 - \Gamma(A, \mathcal{D}) \frac{\lambda^{nz}_{min}(A^T A)}{K}.$$

For arbitrary trees, the quantities $\gamma_{min}(G_k)$ and $\gamma_{max}(G_k)$ depend on the topology, but for p-regular trees (meaning all nodes that are not leaves have p successors), we have

$$\gamma_{min}(G_k) = \gamma_{max}(G_k) = \frac{1}{p^k}.$$

Thus, from Equation (20) we obtain the estimate

$$\Gamma(A, \mathcal{D}) \geq \frac{1}{p^{K-1}}$$

and our convergence rate is bounded by

$$1 - \Gamma(A, \mathcal{D})\lambda^{nz}_{min}(W) \leq 1 - \frac{\lambda^{nz}_{min}(A^T A)}{K p^{K-1}}.$$

For our regular p-tree, $K = O(\log m)$ and $p^{K-1} = O(m)$, so asymptotically the convergence rate is at worst $1 - O((m \log m)^{-1})\lambda^{nz}_{min}(A^T A)$.

Families. Here, blocks consist of all immediate successors (children) of a common node, i.e., $\mathcal{C}(u)$ for u not a leaf. The singleton $\{r\}$ also is a block. We select each block uniformly, so $p_v = \dfrac{1}{m-t}$. In this case, for each family (block) F,

$$\gamma_{min}(F) = \gamma_{max}(F) = \frac{1}{|F|}.$$

Thus, we obtain the estimate

$$\Gamma(A, \mathcal{D}) \geq \frac{1}{c},$$

where c denotes the largest family. We obtain a convergence rate of

$$\Sigma(A, \mathcal{D}) = 1 - \frac{\lambda^{nz}_{min}(A^T A)}{c(m-t)}.$$

In the case of a binary tree, $c = 2$, and $m - t = O(m)$, so asymptotically the convergence rate is at worst $1 - O(m^{-1})\lambda^{nz}_{min}(A^T A)$.

2.3. Accelerating the Convergence via Over-Relaxation

We can accelerate the convergence, i.e., lower the convergence factor, by considering larger stepsizes. In the classical cyclic Kaczmarz algorithm, a relaxation parameter $\omega \in (0, 2)$ is allowed, and the update is given by

$$\vec{x}^{(n)} = \vec{x}^{(n-1)} + \omega \frac{b_n - \vec{a}_n^* \vec{x}^{(n-1)}}{\|\vec{a}_n\|^2} \vec{a}_n.$$

Experimentally, convergence is faster with $\omega > 1$ [4,14,46]. We consider here such a relaxation parameter in Algorithm 1. This alters the analysis of Theorem 1 only slightly. Indeed, the update in Proposition 2 becomes:

$$\vec{x}^{(n)} = \vec{x}^{(n-1)} + \omega \sum_{v \in Z_n} \gamma(v, Z_n) \frac{b_v - \vec{a}_v^* \vec{x}^{(n-1)}}{\|\vec{a}_v\|^2} \vec{a}_v. \tag{22}$$

Thus, Equation (15) in the proof of Theorem 1 becomes:

$$\|\vec{x}^{(1)} - \Pi \vec{x}^{(1)}\|^2 \leq \left(\vec{x}^{(0)} - \Pi \vec{x}^{(0)}\right)^* \left(I - \omega 2 B_{Z_1} + \omega^2 B^2_{Z_1}\right) \left(\vec{x}^{(0)} - \Pi \vec{x}^{(0)}\right). \tag{23}$$

The remainder of the calculation follows through, with the final estimate

$$\mathbb{E}\|\vec{x}^{(1)} - \Pi \vec{x}^{(1)}\|^2 \leq (1 - \Gamma_\omega(A, \mathcal{D}) \lambda^{nz}_{min}(W)) \|\vec{x}^{(0)} - \Pi \vec{x}^{(0)}\|^2, \tag{24}$$

where

$$\Gamma_\omega(A, \mathcal{D}) = \min\Big\{2\omega\gamma_{min}(I) - \omega^2 \gamma_{max}(I) \lambda_{max}(B_I) | P(Z = I) > 0\Big\}.$$

If we assume that

$$\lambda^{block}_{max} := max\{\lambda_{max}(B_I) | P(Z = I) > 0\} < 1,$$

then we can maximize a lower bound on $\Gamma_\omega(A, \mathcal{D})$ as a function of ω. Indeed, if we assume that for each I, $\gamma_{min}(I) = \gamma_{max}(I) =: \gamma$, then the maximum occurs at

$$\omega_0 = \frac{1}{\lambda^{block}_{max}},$$

and we then obtain the estimate

$$\Gamma_{\omega_0}(A, \mathcal{D}) \geq \frac{\gamma}{\lambda_{max}^{block}} \tag{25}$$

This suggests that the rate of convergence can be accelerated by choosing the stepsize ω_0, since the estimate from Equation (25) is better than the estimate

$$\Gamma(A, \mathcal{D}) \geq 2(1 - \lambda_{max}^{block}).$$

Our numerical experiments presented in Section 4 empirically support acceleration through over-relaxation and in fact suggest further improvement than what we prove here.

3. The RTK in the Presence of Noise

We now consider the performance of the Randomized Tree Kaczmarz algorithm in the presence of noise. That is to say, we consider the system of equations $A\vec{x} = \vec{b} + \vec{\epsilon}$, where $\vec{\epsilon}$ represents noise within the observed measurements. We assume that the noiseless matrix equation $A\vec{x} = \vec{b}$ is consistent, and its solution is the solution we want to estimate. We suppose that the equations $\vec{a}_v^* \vec{x} = b_v + \epsilon_v$ are distributed across a network that is a tree, as before. We will consider two aspects of the noisy case: first, we establish the convergence rate of the RTK in the presence of noise and estimate the errors in the approximations due to the noise; second, we will consider methods for mitigating the noise, meaning that if the noise vector $\vec{\epsilon}$ satisfies a certain sparsity constraint, then we can estimate which nodes are corrupted by noise (i.e., anomalous) and ignore them in the RTK.

3.1. Convergence Rate in the Presence of Noise

We now consider the convergence rate of Algorithm 1 in the presence of noise. The randomized Kaczmarz method in the presence of noise was investigated in [47]. The main result of that study is that when the measurement vector $\vec{b}$ is corrupted by noise (likely causing the system to be inconsistent), the randomized Kacmarz algorithm displays the same convergence rate as the randomized Kaczmarz algorithm does in the consistent case up to an error term that is proportional to the variance of the noise, and the constant of proportionality is given by spectral data of the coefficient matrix.

To formalize, we consider the case that the system of equations $A\vec{x} = \vec{b}$ is consistent but that the measurement vector $\vec{b}$ is corrupted by noise, yielding the observed system of equations $A\vec{x} = \vec{b} + \vec{\epsilon}$. The update in Algorithm 1 then becomes:

$$\begin{aligned}\vec{x}^{(n)} &= \vec{x}^{(n-1)} + \sum_{v \in I} \gamma(v, I) \frac{b_v + \epsilon_v - \vec{a}_v^* \vec{x}^{(n-1)}}{\|\vec{a}_v\|^2} \vec{a}_v \\ &= \vec{x}^{(n-1)} + \sum_{v \in I} \gamma(v, I) \frac{b_v - \vec{a}_v^* \vec{x}^{(n-1)}}{\|\vec{a}_v\|^2} \vec{a}_v + \sum_{v \in I} \gamma(v, I) \frac{\epsilon_v}{\|\vec{a}_v\|^2} \vec{a}_v.\end{aligned}$$

We denote the error in the update by

$$\mathcal{E}_I = \sum_{v \in I} \gamma(v, I) \frac{\epsilon_v}{\|\vec{a}_v\|^2} \vec{a}_v.$$

Note that

$$\|\mathcal{E}_I\| \leq \sum_{v \in I} \gamma(v, I) \frac{|\epsilon_v|}{\|\vec{a}_v\|} \leq \max_{v \in V} \left\{ \frac{|\epsilon_v|}{\|\vec{a}_v\|} \right\}. \tag{26}$$

For $d \times d$ matrices $A_1, \ldots, A_N$, we denote the product $A_N A_{N-1} \cdots A_1 = \prod_{n=1}^{N} A_n$, so that the product notation is indexed from right to left. For a product where the beginning index is greater than the ending index, e.g., $\prod_{n=k+1}^{k} A_n$, we define the product to be the identity I. As before, Z is a $\mathbb{P}(V)$ valued random variable with distribution $\mathcal{D}$, and B_Z is as given in Equation (11).

Theorem 3. *Suppose the system of equations $A\vec{x} = \vec{b}$ is consistent, and let Π denote the projection onto the solution space. Let $\vec{x}^{(n)}$ be the n-th iterate of Algorithm 1 run with the noisy measurements $A\vec{x} = \vec{b} + \vec{\epsilon}$, distribution $\mathcal{D}$, and initialization $\vec{x}^{(0)} \in \mathcal{R}(A^T)$. Then, the following estimate holds in expectation:*

$$\mathbb{E}\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\| \leq \Sigma(A,\mathcal{D})^{n/2}\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\| + \left[1 - \Sigma(A,\mathcal{D})^{1/2}\right]^{-1} \max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\} \quad (27)$$

where $\Sigma(A,\mathcal{D})$ is as given in Equation (13).

Proof. Let $\vec{x}^S$ be any solution to the system of equations $A\vec{x} = \vec{b}$. We have by induction that

$$\begin{aligned}
\vec{x}^S - \vec{x}^{(n)} &= (I - B_{Z_n})(\vec{x}^S - \vec{x}^{(n-1)}) + \mathcal{E}_{Z_n} \\
&= (I - B_{Z_n})(I - B_{Z_{n-1}})(\vec{x}^S - \vec{x}^{(n-2)}) + (I - B_{Z_n})(\mathcal{E}_{Z_{n-1}}) + \mathcal{E}_{Z_n} \\
&= \dots \\
&= \prod_{j=1}^{n}(I - B_{Z_j})(\vec{x}^S - \vec{x}^{(0)}) + \sum_{j=1}^{n}\prod_{k=j+1}^{n}(I - B_{Z_k})\mathcal{E}_{Z_j}.
\end{aligned}$$

Note that the first term is precisely the estimates obtained from Algorithm 1 with $\vec{\epsilon} = 0$, so we can utilize Theorem 2 to obtain the following estimate

$$\begin{aligned}
\mathbb{E}\|\vec{x}^S - \vec{x}^{(n)}\| &\leq \mathbb{E}\left\|\prod_{j=1}^{n}(I - B_{Z_j})(\vec{x}^S - \vec{x}^{(0)})\right\| + \mathbb{E}\left\|\sum_{j=1}^{n}\prod_{k=j+1}^{n}(I - B_{Z_k})\mathcal{E}_{Z_j}\right\| \\
&\leq (\Sigma(A,\mathcal{D}))^{n/2}\|\vec{x}^S - \vec{x}^{(0)}\| + \mathbb{E}\sum_{j=1}^{n}\left\|\prod_{k=j+1}^{n}(I - B_{Z_k})\mathcal{E}_{Z_j}\right\|.
\end{aligned}$$

Thus, we need to estimate the terms in the sum.

As in the proof of Theorem 2, we can estimate the expectation conditioned on $Z_1, \dots, Z_{n-1}$ as

$$\mathbb{E}\left\{\left\|\prod_{k=j+1}^{n}(I - B_{Z_k})\mathcal{E}_{Z_j}\right\| : Z_1, \dots, Z_{n-1}\right\} \leq (\Sigma(A,\mathcal{D}))^{1/2}\left\|\prod_{k=j+1}^{n-1}(I - B_{Z_k})\mathcal{E}_{Z_j}\right\|.$$

Iterating this estimate $n - j$ times using the tower property of conditional expectation yields

$$\mathbb{E}\left\{\left\|\prod_{k=j+1}^{n}(I - B_{Z_k})\mathcal{E}_{Z_j}\right\| : Z_1, \dots, Z_j\right\} \leq (\Sigma(A,\mathcal{D}))^{(n-j)/2}\left\|\mathcal{E}_{Z_j}\right\|.$$

We have by Equation (26) that

$$\mathbb{E}\left\|\mathcal{E}_{Z_j}\right\| \leq \max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\}.$$

Thus, taking the full expectation over the entire history yields

$$\mathbb{E}\left\|\prod_{k=j+1}^{n}(I - B_{Z_k})\mathcal{E}_{Z_j}\right\| \leq (\Sigma(A,\mathcal{D}))^{(n-j)/2}\max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\}. \quad (28)$$

We thus obtain

$$\mathbb{E}\sum_{j=1}^{n}\left\|\prod_{k=j+1}^{n}(I-B_{Z_k})\mathcal{E}_{Z_j}\right\| \leq \sum_{j=1}^{n}(\Sigma(A,\mathcal{D}))^{(n-j)/2}\max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\}$$
$$\leq \sum_{j=0}^{\infty}(\Sigma(A,\mathcal{D}))^{j/2}\max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\}$$

from which the estimate in Equation (27) now follows. □

Remark 3. *We note here that Theorem 3 recovers a similar, but coarser, estimate as the one obtained in Theorem 2.1 in [47]. Indeed, as in Remark 1, suppose that the distribution $\mathcal{D}$ only selects singletons from $\mathbb{P}(V)$, and each singleton $\{v\}$ is selected with probability $\frac{\|a_v\|^2}{\|A\|_F^2}$. Then, we have $\Sigma(A,\mathcal{D}) = 1 - \frac{\lambda_{min}^{nz}(A^TA)}{\|A\|_F^2}$, and so Theorem 3 becomes*

$$\mathbb{E}\|\vec{x}^{(n)} - \Pi\vec{x}^{(n)}\| \leq \Sigma(A,\mathcal{D})^{n/2}\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\| + \left[1-\Sigma(A,\mathcal{D})^{1/2}\right]^{-1}\max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\}$$
$$= \left(1-\frac{1}{R}\right)^{n/2}\|\vec{x}^{(0)} - \Pi\vec{x}^{(0)}\| + \left(1-\sqrt{1-\frac{1}{R}}\right)^{-1}\max_{v\in V}\left\{\frac{|\epsilon_v|}{\|\vec{a}_v\|}\right\}.$$

Here, $R = \frac{\|A\|_F^2}{\lambda_{min}^{nz}(A^TA)}$ in the notation of Theorem 2.1 in [47]. The estimate is similar for $R \approx 1$, but for $R >> 1$, our estimate is worse. This is because the proof of Theorem 2.1 [47] utilizes orthogonality at a crucial step, which is not valid for our situation—the error $\mathcal{E}_I$ is not orthogonal to the solution space for the affected equations.

3.2. Anomaly Detection in Distributed Systems of Noisy Equations

We again consider the case of noisy measurements, again denoted by $A\vec{x} = \vec{b} + \vec{e}$, where now the error vector $\vec{e}$ is assumed to be sparse, and the nonzero entries are large. This situation is considered in [45]. In that study, the authors propose multiple methods of using the Randomized Kaczmarz algorithm to estimate which equations are corrupted by the noise, i.e., which equations correspond to the nonzero entries of $\vec{e}$. Once those equations are detected, they are removed from the system. The assumption is that the subsystem of uncorrupted equations is consistent; thus, once the corrupted equations are removed, the Randomized Kaczmarz algorithm can be used to estimate a solution. Moreover, the Randomized Kaczmarz algorithm can be used on the full (corrupted) system of equations to obtain an estimate of the solution with positive probability. We demonstrate here that the methods proposed in [45] can be utilized in our context of distributed systems of equations to identify corrupted equations and estimate a solution to the uncorrupted equations.

Indeed, we utilize without any alteration the methods of [45] to detect corrupted equations; we provide the Algorithms 2 and 3 for completeness. To prove that the algorithms are effective in our distributed context, we follow the proofs in [45] with virtually no change. Once we establish an initial lemma, the proofs of the main results (Theorems 4 and 5) are identical. The lemma we require is an adaptation of Lemma 2 in [45] to our Distributed Randomized Kaczmarz algorithm. Our proof proceeds similarly to that in [45]; we include it here for completeness.

We establish some notation first. For an arbitrary $U \subset V$, we use $A_{V\setminus U}$ to denote the submatrix of A obtained by removing the rows indexed by U. Similarly, for the vector $\vec{b}$, $\vec{b}_{V\setminus U}$ consists of the components whose indices are in $V \setminus U$. For the probability distribution $\mathcal{D}$ and $U \subset V$, we denote by $\widetilde{\mathcal{D}}$ the conditional probability distribution on $\mathbb{P}(V \setminus U)$ conditioned on $I \cap U = \emptyset$ for $I \subset V \setminus U$.

For the remainder of this subsection, let $U \subset V$ denote the support of the noise $\vec{\epsilon}$, with $|U| = s$; let $\varepsilon^* = \min\{|\vec{\epsilon}_j| : j \in U\}$. We assume that $A_{V\setminus U}\vec{x} = \vec{b}_{V\setminus U}$ has a *unique* solution; let $\vec{x}^S$ be that solution to this restricted system. Let

$$\mathscr{P}(\mathcal{D}, U) = \sum_{I \cap U = \emptyset} P_{\mathcal{D}}(I). \tag{29}$$

Recall that k is the number of variables in the system of equations.

Lemma 1. *Let $0 < \delta < 1$. Define*

$$n^* = max\left(0, \left\lceil \frac{\log\left(\frac{\delta(\varepsilon^*)^2}{4\|\vec{x}^S\|^2}\right)}{\log \Sigma(A_{V\setminus U}, \widetilde{\mathcal{D}})} \right\rceil\right) \tag{30}$$

Then, in round i of Algorithm 2 or Algorithm 3, the iterate $\vec{x}^{(n^*,i)}$ produced by n^* iterations of the RTK satisfies

$$P\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\| \leq \frac{\varepsilon^*}{2}\right] \geq (1-\delta)(\mathscr{P}(\mathcal{D}, U))^{n^*}. \tag{31}$$

Proof. Let E be the event that the blocks $Z_1, \ldots, Z_{n^*}$ chosen according to distribution $\mathcal{D}$ are all uncorrupted, i.e., $Z_j \cap U = \emptyset$ for $j = 1, 2, \ldots, n^*$. This is equivalent to applying the RTK algorithm to the system $A_{V\setminus U}\vec{x} = \vec{b}_{V\setminus U}$ with distribution $\widetilde{\mathcal{D}}$. Thus, by Theorem 2, we have the conditional expectation:

$$\begin{aligned}\mathbb{E}\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 | E\right] &\leq \mathbb{E}_{A_{V\setminus U}, \vec{b}_{V\setminus U}}\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 | E\right] \\ &\leq \Sigma(A_{V\setminus U}, \widetilde{\mathcal{D}})^{n^*}\|\vec{x}^S\|^2.\end{aligned}$$

From Equation (30), we obtain

$$\mathbb{E}\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 | E\right] \leq \frac{\delta(\varepsilon^*)^2}{4}.$$

Thus,

$$P\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 \geq \frac{(\varepsilon^*)^2}{4}\right] \leq \frac{\mathbb{E}\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 | E\right]}{\frac{(\varepsilon^*)^2}{4}} \leq \delta.$$

Hence,

$$P\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 \leq \frac{(\varepsilon^*)^2}{4} | E\right] \geq 1 - \delta$$

and

$$P\left[\|\vec{x}^{(n^*,i)} - \vec{x}^S\|^2 \leq \frac{(\varepsilon^*)^2}{4}\right] \geq (1-\delta)P(E) \geq (1-\delta)(\mathscr{P}(\mathcal{D}, U))^{n^*}.$$

□

The following are Theorems 2 and 3 in [45], respectively, restated to our situation; the proofs are identical using our Lemma 1 and are omitted.

Theorem 4. *Let $0 < \delta < 1$. Fix $d \geq s$, $W \leq \left\lfloor \frac{|V| - k}{d} \right\rfloor$, and let n^* be as in Equation (30). Then, the MRRTK Algorithm (Algorithm 2) applied to $A, \vec{b} + \vec{e}$ will detect the corrupted equations with probability at least*

$$1 - \left[1 - (1-\delta)(\mathscr{P}(\mathcal{D}, U))^{n^*}\right]^W.$$

Theorem 5. *Let* $0 < \delta < 1$. *Fix* $d \geq 1$, $W \leq \left\lfloor \frac{|V| - k}{d} \right\rfloor$, *and let* n^* *be as in Equation (30). Then, the MRRTKUS Algorithm (Algorithm 3) applied to* $A, \vec{b} + \vec{e}$ *will detect the corrupted equations and the remaining equations will have solution* $\vec{x}^S$ *with probability at least*

$$1 - \sum_{j=0}^{\lceil s/d \rceil - 1} \binom{W}{j} p^j (1-p)^{W-j}$$

where $p = (1-\delta)(\mathscr{P}(\mathcal{D}, U))^{n^*}$.

See [45] for an extensive analysis of numerical experiments of these algorithms, which we do not reproduce here.

4. Numerical Experiments

4.1. The Test Equations

We randomly generated several types of test equations, full and sparse, of various sizes. The results for different types of matrices were very similar, so we just present some results for full matrices with entries generated from a standard normal distribution.

We generated the matrices once, made sure they had full rank, and stored them. Thus, all algorithms are working on the same matrices. However, the sequence of equations used in the random algorithms is generated at runtime.

There are two types of problems we considered. In the *underdetermined* case, illustrated here with a 255×1023 matrix, all algorithms converge to the solution of minimal norm. In the *consistent overdetermined* case, illustrated here with a 1023×255 matrix, all algorithms converge to the standard solution. The matrix dimensions are of the form $2^d - 1$, for easier experimentation with binary trees.

In the inconsistent overdetermined case, deterministic Kaczmarz algorithms will converge to a weighted least-squares solution, depending on the type of algorithm and on the relaxation parameter ω. However, random Kaczmarz algorithms do not converge in this case but do accumulate around a weighted least-squares solution, e.g., Theorem 1 in [15], and Theorem 3.

4.2. The Algorithms

We included several types of deterministic Kaczmarz algorithms in the numerical experiments:

- Standard Kaczmarz.
- Sequential block Kaczmarz, with several different numbers of blocks. The equations are divided into a small number of blocks, and the updates are performed as an orthogonal projection onto the solution space of each block, rather than each individual equation.
- Distributed Kaczmarz based on a binary tree as in [5].
- Distributed block Kaczmarz. This is distributed Kaczmarz based on a tree of depth 2 with a small number of leaves, where each leaf contains a block of equations.

In sequential block Kaczmarz we work on each block in sequence. In distributed block Kaczmarz we work on each block in parallel and average the results.

The block Kaczmarz case, whether sequential or distributed, is not actually covered by our theory. However, we believe that our results could be extended to this case fairly easily, as long as the equations in each block are underdetermined and have full rank, following the approach in [4].

These deterministic algorithms are compared to corresponding types of random Kaczmarz algorithms:

- Random standard Kaczmarz; one equation at a time is randomly chosen.

- Random block Kaczmarz; one block at a time is randomly chosen. There is no difference between sequential and parallel random block Kaczmarz.
- Random distributed Kaczmarz based on a binary tree, for several kinds of random choices:
  - Generations, that is, we use all nodes at a randomly chosen distance from the root.
  - Families, that is, using the children of a randomly chosen node; for a binary tree, these are pairs.
  - Subtrees, that is, using the subtree rooted at a randomly chosen node. This is not an incomparable choice, so it is not covered by our theory.

For a matrix of size $m \times k$, in the deterministic cases one iteration consists of applying m updates, using each equation once. A block of size $n \times k$ counts as n updates. In the random cases, different random choices may involve different numbers of equations; we apply updates until the number of equations used reaches or slightly exceeds m.

After each iteration, we compute the 2-norm of the error. The *convergence factor* at each step is the factor by which the error has gone down at the last step. These factors often vary considerably in the first few steps and then settle down. The empirical convergence factors given in the tables below are calculated as the geometric average of the convergence factors over the second half of the iterations.

4.3. Numerical Results

The empirical convergence factors shown in Tables 1 and 2 are based on 10 iterations. Each convergence factor is computed as the fifth root of the ratio of errors between iterations 5 and 10. For the random algorithms, each experiment (of 10 iterations) was run 20 times and the resulting convergence factors averaged.

Table 1. Convergence factors for various algorithms, for a random underdetermined equation of size 255×1023. Numbers in parentheses represent the estimates from Equation (13).

	Relaxation Parameter ω						
	0.5	**1**	**1.5**	**2**	**2.5**	**3**	**3.5**
deterministic							
standard	0.8138	0.5428	0.5579				
sequential blocks							
4 blocks	0.7963	0.4926	0.5227				
16 blocks	0.8101	0.5362	0.5531				
64 blocks	0.8135	0.5448	0.5628				
parallel blocks							
4 blocks	0.9346	0.8947	0.8604	0.8276	0.7948	0.7616	0.7273
16 blocks	0.9800	0.9636	0.9499	0.9379	0.9271	0.9173	0.9080
64 blocks	0.9947	0.9897	0.9849	0.9804	0.9761	0.9720	0.9681
255 blocks	0.9986	0.9973	0.9960	0.9947	0.9934	0.9922	0.9909
binary tree	0.9941	0.9903	0.9870	0.9841			
random							
standard	0.8440	0.7472	0.7039				
blocks							
4 blocks	0.8013	0.6736	0.6724				
16 blocks	0.8146	0.7162	0.7001				
64 blocks	0.8252	0.7393	0.7136				
binary tree							
family	0.9055 (0.9099)	0.8510 (0.8817)	0.8133 (0.9099)	0.7742	0.7692	0.7528	0.8178
generation	0.9940 (0.9985)	0.9903 (0.9980)	0.9874 (0.9985)	0.9849			
subtree	0.9352	0.9017	0.8617	0.8735	0.9078		

For binary trees, with a random choice of generations or families, we also calculated the estimated convergence factors described in Equation (13). These estimates are for one random step. For a binary tree of K levels, it takes on average K choices of generation to use the entire tree once. For families, it is 2^{K-1} choices of families (pairs, in this case). To estimate the convergence factor for one iteration, we took a corresponding power of the estimates.

Table 1 shows the convergence factors for the underdetermined case. An empty entry means that the algorithm did not converge for this value of ω.

Here are some observations about Table 1:

- Sequential block Kaczmarz and random block Kaczmarz for 255 blocks are identical to their standard Kaczmarz counterparts and are not shown.
- Sequential methods, including standard Kacmarz and sequential block Kaczmarz, converge faster than parallel methods, such as binary trees or distributed block Kaczmarz. This is not surprising: in sequential methods, each step uses the results of the preceding step; in parallel methods, each step uses older data.
- The same reasoning explains why the Family selection is faster than Generations. Consider level 3, as an example. With Generations, we do 8 equations in parallel. With Families, we do four sets of 2 equations each, but each pair uses the result from the previous step.
- The block algorithm for a single block with $\omega = 1$ converges in a single step, so the convergence factor is 0. At the other end of the spectrum, with 255 blocks of one equation each, the block algorithm becomes standard Kaczmarz. As the number of blocks increases, the convergence factor is observed to increase and approach the standard Kaczmarz value.
- Standard Kaczmarz, deterministic or random, converges precisely for $\omega \in (0, 2)$. By the results in [4], this is also true for sequential block Kaczmarz.
  Distributed Kaczmarz methods are guaranteed to converge for the same range of ω, but in practice they often converge for larger ω as well, sometimes up to ω near 4. Random distributed methods appear to have similar behavior.
- The observed convergence factors for random algorithms are comparable to those for their deterministic counterparts but slightly worse. We attribute this to the fact that in the underdetermined case, all equations are important; random algorithms on an $m \times k$ matrix do not usually include all equations in a set of m updates, while deterministic algorithms do.
  As pointed out in [8], there are types of equations where random algorithms are significantly faster than deterministic algorithms, but our sample equations are obviously not in that category.

Table 2 shows the convergence factors for the consistent overdetermined case.

Table 2. Convergence factors for various algorithms, for a random overdetermined consistent equation of size 1023 × 255. Numbers in parentheses represent the estimates from Equation (13).

	Relaxation Parameter ω						
	0.5	**1**	**1.5**	**2**	**2.5**	**3**	**3.5**
deterministic							
standard	0.4923	0.2564	0.2424				
sequential blocks							
4 blocks	0.0625	0.0000	0.0625				
16 blocks	0.4311	0.1863	0.1894				
64 blocks	0.4759	0.2131	0.2263				

Table 2. *Cont.*

	Relaxation Parameter ω						
	0.5	**1**	**1.5**	**2**	**2.5**	**3**	**3.5**
255 blocks	0.4930	0.2588	0.2409				
parallel blocks							
4 blocks	0.5000	0.0000	0.5000				
16 blocks	0.9125	0.8696	0.8303	0.7919	0.7550	0.7195	0.6795
64 blocks	0.9707	0.9483	0.9313	0.9178	0.9063	0.8958	0.8857
256 blocks	0.9920	0.9845	0.9775	0.9709	0.9647	0.9590	0.9536
1023 blocks	0.9980	0.9960	0.9940	0.9920	0.9901	0.9882	0.9864
binary tree	0.9889	0.9817	0.9758	0.9704			
random							
standard	0.5481	0.3421	0.2748				
blocks							
4 blocks	0.0625	0.0000	0.0625				
16 blocks	0.4451	0.2464	0.2291				
64 blocks	0.4690	0.2847	0.2579				
256 blocks	0.4812	0.2874	0.2675				
binary tree							
family	0.7040	0.5401	0.4225	0.3356	0.2615	0.2769	0.4462
	(0.6879)	(0.6073)	(0.6879)				
generation	0.9889	0.9809	0.9768	0.9709			
	(0.9985)	(0.9980)	(0.9985)				
subtree	0.8349	0.7474	0.6486	0.6408	0.7103		

Observations about Table 2:

- All algorithms converge faster in the overdetermined consistent case than in the underdetermined case. That is not surprising: since we have four times more equations than we actually need; one complete run through all equations is comparable to four complete run-throughs for the underdetermined case.
- For the same reason, the parallel block algorithm with four blocks (or fewer) for $\omega = 1$ converges in a single step.
- We observe that random algorithms are still slower in the overdetermined case, even though the argument from the underdetermined case does not apply here.

Author Contributions: F.K. designed the numerical experiments and produced the associated code. E.S.W. developed the algorithms and proved the qualitative results. Conceptualization, F.K. and E.S.W.; methodology, F.K.; software, F.K.; validation, F.K.; writing—original draft preparation, E.S.W.; writing—review and editing, F.K. All authors have read and agreed to the published version of the manuscript.

Funding: Fritz Keinert and Eric S. Weber were supported in part by the National Science Foundation and the National Geospatial Intelligence Agency under award #1830254. Eric S. Weber was supported in part by the National Science Foundation under award #1934884.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kaczmarz, S. Angenäherte Auflösung von Systemen linearer Gleichungen. *Bull. Int. Acad. Pol. Sci. Lett. Cl. Sci. Math. Nat. Ser. A Sci. Math.* **1937**, *35*, 355–357.
2. Tanabe, K. Projection Method for Solving a Singular System of Linear Equations and its Application. *Numer. Math.* **1971**, *17*, 203–214. [CrossRef]

3. Eggermont, P.P.B.; Herman, G.T.; Lent, A. Iterative Algorithms for Large Partitioned Linear Systems, with Applications to Image Reconstruction. *Linear Alg. Appl.* **1981**, *40*, 37–67. [CrossRef]
4. Natterer, F. *The Mathematics of Computerized Tomography*; Teubner: Stuttgart, Germany, 1986.
5. Hegde, C.; Keinert, F.; Weber, E.S. A Kaczmarz Algorithm for Solving Tree Based Distributed Systems of Equations. In *Excursions in Harmonic Analysis*; Balan, R., Benedetto, J.J., Czaja, W., Dellatorre, M., Okoudjou, K.A., Eds.; Applied and Numerical Harmonic Analysis; Birkhäuser/Springer: Cham, Switzerland, 2021; Volume 6, pp. 385–411. [CrossRef]
6. West, D.B. *Introduction to Graph Theory*; Prentice Hall, Inc.: Upper Saddle River, NJ, USA, 1996; p. xvi+512.
7. Hamaker, C.; Solmon, D.C. The angles between the null spaces of X rays. *J. Math. Anal. Appl.* **1978**, *62*, 1–23. [CrossRef]
8. Strohmer, T.; Vershynin, R. A randomized Kaczmarz algorithm with exponential convergence. *J. Fourier Anal. Appl.* **2009**, *15*, 262–278. [CrossRef]
9. Zouzias, A.; Freris, N.M. Randomized extended Kaczmarz for solving least squares. *SIAM J. Matrix Anal. Appl.* **2013**, *34*, 773–793. [CrossRef]
10. Needell, D.; Zhao, R.; Zouzias, A. Randomized block Kaczmarz method with projection for solving least squares. *Linear Algebra Appl.* **2015**, *484*, 322–343. [CrossRef]
11. Needell, D.; Srebro, N.; Ward, R. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Math. Progr.* **2016**, *155*, 549–573. [CrossRef]
12. Cimmino, G. Calcolo approssimato per soluzioni dei sistemi di equazioni lineari. In *La Ricerca Scientifica XVI, Series II, Anno IX 1*; Consiglio Nazionale delle Ricerche: Rome, Italy, 1938; pp. 326–333.
13. Censor, Y.; Gordon, D.; Gordon, R. Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel Comput.* **2001**, *27*, 777–808. [CrossRef]
14. Necoara, I. Faster randomized block Kaczmarz algorithms. *SIAM J. Matrix Anal. Appl.* **2019**, *40*, 1425–1452. [CrossRef]
15. Moorman, J.D.; Tu, T.K.; Molitor, D.; Needell, D. Randomized Kaczmarz with averaging. *BIT Numer. Math.* **2021**, *61*, 337–359. [CrossRef]
16. Tsitsiklis, J.; Bertsekas, D.; Athans, M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. Autom. Control* **1986**, *31*, 803–812. [CrossRef]
17. Xiao, L.; Boyd, S.; Kim, S.J. Distributed average consensus with least-mean-square deviation. *J. Parallel Distrib. Comput.* **2007**, *67*, 33–46. [CrossRef]
18. Shah, D. Gossip Algorithms. *Found. Trends Netw.* **2008**, *3*, 1–125. [CrossRef]
19. Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.* **2011**, *3*, 1–122. [CrossRef]
20. Nedic, A.; Ozdaglar, A. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Autom. Control* **2009**, *54*, 48. [CrossRef]
21. Johansson, B.; Rabi, M.; Johansson, M. A randomized incremental subgradient method for distributed optimization in networked systems. *SIAM J. Optim.* **2009**, *20*, 1157–1170. [CrossRef]
22. Yuan, K.; Ling, Q.; Yin, W. On the convergence of decentralized gradient descent. *SIAM J. Optim.* **2016**, *26*, 1835–1854. [CrossRef]
23. Sayed, A.H. Adaptation, learning, and optimization over networks. *Found. Trends Mach. Learn.* **2014**, *7*, 311–801. [CrossRef]
24. Zhang, X.; Liu, J.; Zhu, Z.; Bentley, E.S. Compressed Distributed Gradient Descent: Communication-Efficient Consensus over Networks. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 2431–2439. [CrossRef]
25. Scaman, K.; Bach, F.; Bubeck, S.; Massoulié, L.; Lee, Y.T. Optimal algorithms for non-smooth distributed optimization in networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 2740–2749.
26. Loizou, N.; Richtárik, P. Revisiting Randomized Gossip Algorithms: General Framework, Convergence Rates and Novel Block and Accelerated Protocols. *arXiv* **2019**, arXiv:1905.08645.
27. Necoara, I.; Nesterov, Y.; Glineur, F. Random block coordinate descent methods for linearly constrained optimization over networks. *J. Optim. Theory Appl.* **2017**, *173*, 227–254. [CrossRef]
28. Necoara, I.; Nesterov, Y.; Glineur, F. Linear convergence of first order methods for non-strongly convex optimization. *Math. Progr.* **2019**, *175*, 69–107. [CrossRef]
29. Bertsekas, D.P.; Tsitsiklis, J.N. *Parallel and Distributed Computation: Numerical Methods*; Athena Scientific: Nashua, NH, USA, 1997. Available online: http://hdl.handle.net/1721.1/3719 (accessed on 1 December 2021).
30. Kamath, G.; Ramanan, P.; Song, W.Z. Distributed Randomized Kaczmarz and Applications to Seismic Imaging in Sensor Network. In Proceedings of the 2015 International Conference on Distributed Computing in Sensor Systems, Fortaleza, Brazil, 10–12 June 2015; pp. 169–178. [CrossRef]
31. Herman, G.T.; Hurwitz, H.; Lent, A.; Lung, H.P. On the Bayesian approach to image reconstruction. *Inform. Control* **1979**, *42*, 60–71. [CrossRef]
32. Hansen, P.C. *Discrete Inverse Problems: Insight and Algorithms*; Fundamentals of Algorithms; Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, USA, 2010; Volume 7, p. xii+213. [CrossRef]
33. Liu, J.; Wright, S.J.; Sridhar, S. An asynchronous parallel randomized Kaczmarz algorithm. *arXiv* **2014**, arXiv:1401.4780.
34. Herman, G.T.; Lent, A.; Hurwitz, H. A storage-efficient algorithm for finding the regularized solution of a large, inconsistent system of equations. *J. Inst. Math. Appl.* **1980**, *25*, 361–366. [CrossRef]

35. Chi, Y.; Lu, Y.M. Kaczmarz method for solving quadratic equations. *IEEE Signal Process. Lett.* **2016**, *23*, 1183–1187. [CrossRef]
36. Crombez, G. Finding common fixed points of strict paracontractions by averaging strings of sequential iterations. *J. Nonlinear Convex Anal.* **2002**, *3*, 345–351.
37. Crombez, G. Parallel algorithms for finding common fixed points of paracontractions. *Numer. Funct. Anal. Optim.* **2002**, *23*, 47–59. [CrossRef]
38. Nikazad, T.; Abbasi, M.; Mirzapour, M. Convergence of string-averaging method for a class of operators. *Optim. Methods Softw.* **2016**, *31*, 1189–1208. [CrossRef]
39. Reich, S.; Zalas, R. A modular string averaging procedure for solving the common fixed point problem for quasi-nonexpansive mappings in Hilbert space. *Numer. Algorithms* **2016**, *72*, 297–323. [CrossRef]
40. Censor, Y.; Zaslavski, A.J. Convergence and perturbation resilience of dynamic string-averaging projection methods. *Comput. Optim. Appl.* **2013**, *54*, 65–76. [CrossRef]
41. Zaslavski, A.J. Dynamic string-averaging projection methods for convex feasibility problems in the presence of computational errors. *J. Nonlinear Convex Anal.* **2014**, *15*, 623–636.
42. Witt, M.; Schultze, B.; Schulte, R.; Schubert, K.; Gomez, E. A proton simulator for testing implementations of proton CT reconstruction algorithms on GPGPU clusters. In Proceedings of the 2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC), Anaheim, CA, USA, 27 October–3 November 2012; pp. 4329–4334. [CrossRef]
43. Censor, Y.; Nisenbaum, A. String-averaging methods for best approximation to common fixed point sets of operators: The finite and infinite cases. *Fixed Point Theory Algorithms Sci. Eng.* **2021**, *21*, 9. [CrossRef]
44. Censor, Y.; Tom, E. Convergence of string-averaging projection schemes for inconsistent convex feasibility problems. *Optim. Methods Softw.* **2003**, *18*, 543–554. [CrossRef]
45. Haddock, J.; Needell, D. Randomized projections for corrupted linear systems. In Proceedings of the AIP Conference Proceedings, Thessaloniki, Greece, 25–30 September 2017; Volume 1978, p. 470071.
46. Borgard, R.; Harding, S.N.; Duba, H.; Makdad, C.; Mayfield, J.; Tuggle, R.; Weber, E.S. Accelerating the distributed Kaczmarz algorithm by strong over-relaxation. *Linear Algebra Appl.* **2021**, *611*, 334–355. [CrossRef]
47. Needell, D. Randomized Kaczmarz solver for noisy linear systems. *BIT* **2010**, *50*, 395–403. [CrossRef]