AN ACCELERATED RANDOMIZED KACZMARZ ALGORITHM

JI LIU AND STEPHEN J. WRIGHT

ABSTRACT. The randomized Kaczmarz (RK) algorithm is a simple but powerful approach for solving consistent linear systems $Ax = b$. This paper proposes an accelerated randomized Kaczmarz (ARK) algorithm with better convergence than the standard RK algorithm on ill-conditioned problems. The per-iteration cost of RK and ARK are similar if A is dense, but RK is much more able to exploit sparsity in A than is ARK. To deal with the sparse case, an efficient implementation for ARK, called SARK, is proposed. A comparison of convergence rates and average per-iteration complexities among RK, ARK, and SARK is given, taking into account different levels of sparseness and conditioning. Comparisons with the leading deterministic algorithm — conjugate gradient applied to the normal equations — are also given. Finally, the analysis is validated via computational testing.

1. INTRODUCTION

We consider the problem of finding a solution to a consistent linear system

$$(1.1) \qquad Ax = b,$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. We denote the rows of A by a_i^T and the elements of b by b_i, $i = 1, 2, \ldots, m$. That is,

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

(Our convergence results do not apply directly to inconsistent systems. For inconsistent systems, we can solve instead the least-squares problem $\min_x \|Ax - b\|^2$, whose solution x can be found by solving the consistent linear system $Ax = y, A^T y = A^T b$.) Besides consistency of $Ax = b$, we assume throughout that A has no zero rows. (Such rows can be detected and eliminated in a trivial preprocessing step.) We assume for purposes of analysis — though not for purposes of deriving and specifying the algorithms — that the rows of A are normalized:

$$(1.2) \qquad \|a_i\|_2 = 1, \ \ i = 1, 2, \ldots, m.$$

This assumption does not add significantly to the cost of implementation: We could simply normalize each a_i the first time it is encountered by the algorithm. Normalization simplifies the analysis in the appendix considerably, and in most cases will improve the conditioning of the problem, leading to faster convergence. However, in the description of algorithms in the main body of the paper, we do not assume (1.2), and include factors $\|a_i\|^2$ as needed. Our randomized algorithms generate the same sequence of iterates whether or not normalization is carried out (provided, of course, that a corresponding scaling is applied to b).

The randomized Kaczmarz (RK) algorithm is an algorithm for solving (1.1) that requires only $O(n)$ storage and has a linear (geometric) rate of convergence. In some situations, it is even more efficient than the conjugate gradient (CG) method, which is the most popular iterative algorithm for solving large linear systems. At each iteration, the RK algorithm randomly selects a row $i \in \{1, 2, \ldots, m\}$ of the linear system and does an orthogonal projection of the current estimate vector onto the hyperplane:

$$(1.3) \qquad x_{k+1} = x_k - \frac{(a_i^T x_k - b_i)}{\|a_i\|^2} a_i.$$

The RK update (1.3) is equivalent to one step of coordinate descent applied to the dual problem

$$\min_y \quad \frac{1}{2} \|A^T y\|^2 - b^T y,$$

(specifically, a negative gradient step in the ith component of y with steplength $1/\|a_i\|_2^2$), where the primal variables x and duals y are related through $x = A^T y$; see [8]. We denote by $i(k)$ the index selected at iteration k, and note that x_k depends on all the indices selected up to iteration k, namely, $i(0), i(1), \ldots, i(k-1)$.

The RK method overcomes two drawbacks of the original Kaczmarz algorithm [7]. First, the original algorithm selects rows of A cyclically (not randomly) and may converge very slowly when the data order is poor, for example, when many neighboring rows are identical. Second, it is difficult to analyze the convergence rate for the original Kaczmarz algorithm, whereas the expected convergence rate of RK can be proved in a few lines.

By applying an acceleration scheme due to Nesterov to the standard RK algorithm, we obtain an accelerated randomized Kaczmarz algorithm (ARK) in Section 3 and show (Section 5) that its linear convergence is faster than the original method when the linear system has poor conditioning, as measured by the minimum nonzero eigenvalue of $A^T A$. The cost per iteration of both RK and ARK is $O(n)$ if the matrix A is dense. If A is sparse, however, the calculus changes. The cost of an iteration of RK is proportional to the number of nonzeros in a_i, whereas the cost of each ARK iteration is still $O(n)$ in general. We therefore propose in Section 4 a scheme called SARK in which the ARK updates are cached, to preserve sparsity in the intermediate vectors. (In the absence of numerical error, the iterates generated by ARK and SARK are identical.) The average cost per iteration of SARK is $O(\sqrt{\delta}n)$, where δ is the fraction of nonzero elements in A. In Section 5.2, we compare the theoretical performance of RK, ARK, and SARK for different values of the sparsity ratio δ and the minimal eigenvalue $\lambda_{\min}$, thus giving guidance about how to choose between these algorithms under various scenarios. We illustrate the computational performance of the algorithm on some random problems in Section 6.

1.1. **Notation.** We summarize notations used in the remainder of the paper.

- $\lambda_{\min}$ and $\lambda_{\max}$ are defined to be the minimum and maximum nonzero eigenvalues $A^T A$, respectively.
- $\|X\|$ is the spectral norm of the matrix, while $\|X\|_F$ denotes the Frobenius norm.
- X^+ is the Moore-Penrose pseudoinverse of X. Denoting the compact singular value decomposition of $X \in \mathbb{R}^{m \times n}$ as $X = U\Sigma V^T$ where U and V are orthonormal matrices (that is, $U^T U = I$ and $V^T V = I$) and Σ is nonsingular and diagonal, we have $X^+ = V\Sigma^{-1}U^T$. Note that $\lambda_{\min} = 1/\|(A^T A)^+\|$.
- Given a positive semidefinite matrix M, $\|X\|_M$ is defined as $\sqrt{\operatorname{trace}(X^T M X)}$.
- Define $\mathcal{P}_{c,d}(x)$ as the orthogonal projection of x onto the hyperplane given by $c^T x = d$, that is,

$$\mathcal{P}_{c,d}(x) = x - \frac{c}{\|c\|^2}(c^T x - d).$$

- $\mathcal{P}_{A,b}(x)$ denotes the (Euclidean-norm) projection of x onto the solution set of $Ax = b$.
- $e_j \in \mathbb{R}^n$, $j = 1, 2, \ldots, n$, denotes the jth Euclidean basis vector — a vector of n zeros except for 1 in position j.

2. RELATED WORK

The Kaczmarz algorithm was proposed by Kaczmarz [7], who used the cyclic projection procedure to solve consistent linear systems $Ax = b$. He proved the convergence to the unique solution if A is a square nonsingular matrix. The cyclic ordering of the iterates made it difficult to obtain iteration-based convergence results, but Galantai [3] proved a linear convergence rate in terms of cycles. Since the 1980s, the Kaczmarz algorithm has found an important application area in Algebraic Reconstruction Techniques (ART) for image reconstruction; see for example [4] and [5].

Censor et al. [1] proposed a component averaging method to solve (1.1): Parallel-project the current x onto all hyperplanes and apply an average scheme on all projections to obtain the next iterate x. This method is essentially a gradient descent method for solving $\frac{1}{2}\|Ax - b\|^2$, and can thus handle inconsistent systems.

Strohmer and Vershynin [14] studied the behavior of RK in the case of a consistent system $Ax = b$ in which A has full column rank (making the solution unique). They proved the linear convergence rate for RK in expectation. Needell [9] also assumed full column rank, but dropped the assumption of consistency, showing that the RK algorithm converges linearly to a ball of fixed radius centered at the solution. The radius is proportional to the distance of b from the image space of A. Eldar and Needell [2] presented a modified version of the randomized Kaczmarz method which at each iteration selects the optimal projection from a randomly chosen set. This technique improves the convergence rate but requires more computation cost in each iteration.

Leventhal and Lewis [8] extended the RK algorithm for consistent linear equalities $Ax = b$ to the more general setting of consistent linear inequalities and equalities: $A_I x \geq b_I$, $A_E x = b_E$. The basic idea is quite similar to the RK algorithm: iteratively update x_{k+1} by projecting x_k onto the randomly selected hyperplane or half space. The linear convergence rate was proven to be $1 - 1/(L^2\|A\|_F^2)$, where L is the Hoffman constant [6] for the system $A_I x \geq b_I$, $A_E x = b_E$.

Zouzias and Freris [16] considered the case of possibly inconsistent (1.1). They proposed a randomized extended Kaczmarz algorithm by first projecting b orthogonally onto the image space of A to obtain $b_\perp$, then orthogonally projecting the initial point x_0 onto the hyperplane $Ax = b_\perp$. Essentially, the RK algorithm is applied twice. The convergence rate is proven to be $1 - \lambda_{\min}/\|A\|_F^2$, which is the same as the RK algorithm for consistent linear systems. This method can be considered as a randomized variant of the extended Kaczmarz method proposed by Popa [13].

3. ALGORITHM

In this section, we review the randomized Kaczmarz algorithm (RK, Algorithm 1) and propose an accelerated variant called ARK (Algorithm 2). Finally, we describe an equivalent version of ARK that can be implemented with fewer operations (Algorithm 3).

Each iteration of RK randomly selects a hyperplane $a_i^T x = b_i$, for some $i \in \{1, 2, \ldots, m\}$, and obtains x_{k+1} by orthogonally projecting x_k onto this hyperplane. As shown at the start of Section 5, this algorithm guarantees linear convergence in the expectation sense.

Algorithm 1. Randomized Kaczmarz: $x_{K+1} = \text{RK}(A, b, x_0, K)$

1: Initialize $k \leftarrow 0$;
2: **while** $k \leq K$ **do**
3:    Choose $i = i(k)$ from $\{1, 2, 3, \ldots, m\}$ with equal probability;
4:    Set $x_{k+1} \leftarrow \mathcal{P}_{a_i,b_i}(x_k)$, that is, $x_{k+1} = x_k - a_i(a_i^T x_k - b_i)/\|a_i\|^2$;
5:    $k \leftarrow k + 1$;
6: **end while**

We note again that Step 4 does not change if we omit the normalization step, that is, if $\|a_i\| \neq 1$. However, when the rows are not normalized, our RK algorithm becomes inconsistent with the versions described in [8, 14], which select the index i in Step 3 with probability $\|a_i\|_2^2/\|A\|_F^2$. We could simulate the effects of non-normalized rows by defining a matrix $\bar{A}$ in which row a_i is replaced by $\|a_i\|^2$ copies of the normalized rows $a_i/\|a_i\|$. Our Algorithm 1 applied to this virtual matrix $\bar{A}$ would then be equivalent to Algorithm 1 of [14] applied to A, with the same convergence results as in that paper (see (5.1)), with $\|A\|_F^2$ replacing m in the denominator of the rate constant). Analysis of the accelerated algorithm to be discussed below could also be performed without the assumption of normalization, but the situation becomes considerably more complicated in this case. In particular, several subtle issues related to allowable scalings of A (not dealt with in existing analyses of accelerated methods) must be addressed. We believe that any additional generality to be gained by dropping our assumptions of normalization and uniform probabilities is minor, and would be obscured by the additional complication in the analysis.

The ARK algorithm applies Nesterov's accelerated procedure [10] — more familiar in the context of gradient descent for optimization — to the standard RK algorithm. When applied to $\min_x f(x)$, gradient descent sets $x_{k+1} \leftarrow x_k - \theta_k \nabla f(x_k)$,

where ∇f is the objective gradient and θ_k is the stepsize. Nesterov's accelerated procedure introduces the sequences $\{y_k\}$ and $\{v_k\}$ and defines the following iterative scheme:

$$y_k \leftarrow \alpha_k v_k + (1 - \alpha_k)x_k,$$
$$x_{k+1} \leftarrow y_k - \theta_k \nabla f(y_k),$$
$$v_{k+1} \leftarrow \beta_k v_k + (1 - \beta_k)y_k - \gamma_k \nabla f(y_k).$$

With appropriate choices of α_k, β_k, and γ_k, this procedure yields better convergence rates than standard gradient descent.

If we treat the projection operation of step 4 in Algorithm 1 analogously to the gradient descent step, we can obtain an accelerated version of the RK algorithm. This accelerated randomized Kaczmarz (ARK) procedure is detailed in Algorithm 2. The scalars α_k, β_k, and γ_k in Algorithm 2 are independent of the vector sequences $\{x_k\}$, $\{y_k\}$, and $\{v_k\}$, and can be calculated offline.

Algorithm 2. Accelerated Randomized Kaczmarz: $x_{K+1} = \mathrm{ARK}(A, b, \lambda, x_0, K)$

1: Check that $\lambda \in [0, \lambda_{\min}]$;
2: Initialize $v_0 \leftarrow x_0$, $\gamma_{-1} \leftarrow 0$, $k \leftarrow 0$;
3: **while** $k \leq K$ **do**
4:     Choose γ_k to be the larger root of

$$(3.1) \qquad \gamma_k^2 - \frac{\gamma_k}{m} = \left(1 - \frac{\gamma_k \lambda}{m}\right)\gamma_{k-1}^2;$$

5:     Set α_k and β_k as follows:

$$(3.2) \qquad \alpha_k \leftarrow \frac{m - \gamma_k \lambda}{\gamma_k(m^2 - \lambda)},$$

$$(3.3) \qquad \beta_k \leftarrow 1 - \frac{\gamma_k \lambda}{m};$$

6:     Set $y_k \leftarrow \alpha_k v_k + (1 - \alpha_k)x_k$;
7:     Choose $i = i(k)$ from $\{1, 2, 3, \ldots, m\}$ with equal probability;
8:     Set $x_{k+1} \leftarrow \mathcal{P}_{a_i,b_i}(y_k)$, that is, $x_{k+1} \leftarrow y_k - a_i(a_i^T y_k - b_i)/\|a_i\|^2$;
9:     Set $v_{k+1} \leftarrow \beta_k v_k + (1 - \beta_k)y_k - \gamma_k a_i(a_i^T y_k - b_i)/\|a_i\|^2$;
10:    $k \leftarrow k + 1$;
11: **end while**

We now describe the complexity of these methods for the case of dense A. We make the standing assumption that the quantities $\|a_i\|^2$, $i = 1, 2, \ldots, m$ are precomputed via an initial pass through the matrix. The main computation in Algorithm 1 is in step 4, which requires about $4n$ operations per iteration. The cost per iteration of Algorithm 2 is about $11n$, incurred in steps 6, 8, and 9.

Although Algorithm 2 is useful for purposes of convergence analysis of the accelerated Kaczmarz algorithm, we describe an equivalent implementation in Algorithm 3 that has a lower cost per iteration. Denoting

$$g_k := a_i(a_i^T y_k - b_i)/\|a_i\|^2,$$

we have $x_{k+1} = y_k - g_k$. Substituting for v_k from step 6 into step 9, we obtain

$$v_{k+1} = \beta_k v_k + (1 - \beta_k)y_k - \gamma_k g_k$$
$$= \left(\frac{\beta_k}{\alpha_k} + 1 - \beta_k\right)y_k - \beta_k \frac{1 - \alpha_k}{\alpha_k}x_k - \gamma_k g_k.$$

By substituting for v_{k+1} in step 6 of Algorithm 2, for iterate $k + 1$, we obtain

$$y_{k+1} = \alpha_{k+1}v_{k+1} + (1 - \alpha_{k+1})x_{k+1}$$
$$= \alpha_{k+1}\left(\frac{\beta_k}{\alpha_k} + 1 - \beta_k\right)y_k - \alpha_{k+1}\beta_k\frac{1 - \alpha_k}{\alpha_k}x_k$$
$$\quad - \alpha_{k+1}\gamma_k g_k + (1 - \alpha_{k+1})(y_k - g_k)$$
$$= \left[1 + \alpha_{k+1}\beta_k\left(\frac{1 - \alpha_k}{\alpha_k}\right)\right]y_k - \alpha_{k+1}\beta_k\frac{1 - \alpha_k}{\alpha_k}x_k$$

(3.4)
$$\quad - (1 - \alpha_{k+1} + \alpha_{k+1}\gamma_k)\,g_k.$$

From (3.2) and (3.3), we have

$$-\beta_k\left(\frac{1 - \alpha_k}{\alpha_k}\right) = \frac{m - \gamma_k\lambda}{m}\left(1 - \frac{\gamma_k(m^2 - \lambda)}{m - \gamma_k\lambda}\right)$$
$$= \frac{m - \gamma_k\lambda}{m} - \frac{\gamma_k(m^2 - \lambda)}{m}$$
$$= 1 - m\gamma_k.$$

Thus, by substituting into (3.4), we obtain

$$y_{k+1} = (1 - m\gamma_k)\alpha_{k+1}x_k + (1 - \alpha_{k+1} + m\alpha_{k+1}\gamma_k)y_k - (1 - \alpha_{k+1} + \alpha_{k+1}\gamma_k)g_k.$$

By making these substitutions into Algorithm 2, we obtain the equivalent implementation of Algorithm 3.

Algorithm 3. Efficient ARK: $x_{K+1} = \text{ARK}(A, b, \lambda, x_0, K)$

1: Check that $\lambda \in [0, \lambda_{\min}]$;
2: Initialize $y_0 = x_0$, $\gamma_{-1} = 0$, $k = 0$.
3: Generate the sequences $\{\gamma_k : k = 0, 1, \ldots, K+1\}$ and $\{\alpha_k : k = 0, 1, \ldots, K+1\}$ as in (3.1) and (3.2);
4: **while** $k \leq K$ **do**
5:    Choose $i = i(k)$ from $\{1, 2, 3, \ldots, m\}$ with equal probability;
6:    Set $s_k \leftarrow (a_i^T y_k - b_i)/\|a_i\|^2$;
7:    Set $g_k \leftarrow s_k a_i$;
8:    Set $g_k' \leftarrow (1 - \alpha_{k+1} + \alpha_{k+1}\gamma_k)\,s_k a_i$;
9:    Set $y_{k+1}' \leftarrow (1 - m\gamma_k)\alpha_{k+1}x_k + (1 - \alpha_{k+1} + m\alpha_{k+1}\gamma_k)y_k$;
10:   Set $y_{k+1} \leftarrow y_{k+1}' - g_k'$;
11:   Set $x_{k+1} \leftarrow y_k - g_k$;
12:   $k \leftarrow k + 1$;
13: **end while**

The main computations are in step 6 to step 11 which have operation counts of about $2n$, n, n, $3n$, n, and n, respectively, giving a total of $9n$. If parallel computation is possible, then steps 6, 7, and 8 can be performed simultaneously with step 9 (in time complexity about $3n$) while steps 10 and 11 can be performed

simultaneously (in time about n). In this setting, the total complexity can be reduced to about $4n$ — a count identical to the RK algorithm.

4. Efficient implementation for sparse data

This section considers the case in which the data matrix A is sparse, with a fraction of δ nonzeros (with $0 < \delta \ll 1$) and seeks an efficient implementation of Algorithm 3 for this case. We assume that the nonzeros are not concentrated in certain rows of A, that is, the sparsity of each row a_i^T is also approximately δ.

Note that the ARK approach starts at a significant disadvantage in the sparse setting. While sparsity can be exploited easily in RK — the average number of operations for each iteration of Algorithm 1 is approximately $4\delta n$ — the operation counts of the ARK algorithms remain at $O(n)$, since the vectors x_k, y_k, and v_k are dense in general. (Algorithm 3 has a count of approximately $3n + 6\delta n$ per iteration.) We now seek a modification of Algorithm 3 that "caches" the updates in order to maintain some sparsity in the update vectors, thus reducing the average complexity of each ARK iteration.

We start by writing the main updating steps in Algorithm 3 as follows:

$$(4.1a) \qquad s_k = (a_{i(k)}^T y_k - b_{i(k)})/\|a_{i(k)}\|^2,$$

$$(4.1b) \qquad x_{k+1} = y_k - s_k a_{i(k)},$$

$$(4.1c) \qquad y_{k+1} = P_k x_k + Q_k y_k - R_k s_k a_{i(k)},$$

where

$$(4.2a) \qquad P_k = \alpha_{k+1}(1 - m\gamma_k),$$

$$(4.2b) \qquad Q_k = 1 - \alpha_{k+1} + m\alpha_{k+1}\gamma_k,$$

$$(4.2c) \qquad R_k = 1 - \alpha_{k+1} + \alpha_{k+1}\gamma_k.$$

Since updating x_{k+1} and y_{k+1} is quite expensive, we only update them once on each cycle (that is, once per T iterations). We see by recursive application of (4.1) that each iterate x_{k+t}, y_{k+t} for $t \geq 1$ can be expressed as a linear combination of x_k and y_k, plus one other vector. The successive updates from rows $a_{i(k)}, a_{i(k+1)}, \ldots, a_{i(k+t)}$ can be "cached" in vectors z_t and w_t, so that x_{k+t} and y_{k+t} can be written as follows:

$$(4.3a) \qquad x_{k+t} = \rho_t x_k + \tau_t y_k + z_t,$$

$$(4.3b) \qquad y_{k+t} = \sigma_t x_k + \nu_t y_k + w_t,$$

where ρ_t, τ_t, σ_t, and ν_t are scalars. Rather than forming x_{k+t} and y_{k+t} explicitly, we could instead update the quantities ρ_t, τ_t, σ_t, ν_t, z_t, and w_t at each iteration. The advantage of doing so is that, provided t is not too large, the vectors z_t and w_t are not dense, so the cost of updating this implicit representation is usually lower than the explicit version. At some point, when t grows too large, the vectors z_t and w_t "fill in" enough that the advantages of implicit representation are lost. At this point — after T steps, say — we can store the latest vectors x_{k+T} and y_{k+T} explicitly, and start a new cycle of T iterations.

We now obtain the update formulae for the quantities ρ_t, τ_t, σ_t, ν_t, z_t, and w_t. At the starting point of a cycle, we set $t = 0$ and

$$\rho_0 = 1, \tau_0 = 0, \sigma_0 = 0, \nu_0 = 1, z_0 = w_0 = 0,$$

so that (4.3) holds for $t = 0$. In the step from iteration t to iteration $(t + 1)$ of a cycle, we have

$$x_{k+t+1} = y_{k+t} - s_{k+t}a_{i(k+t)} = \sigma_t x_k + \nu_t y_k + w_t - s_{k+t}a_{i(k+t)},$$

implying that

$$\rho_{t+1} = \sigma_t,$$
$$\tau_{t+1} = \nu_t,$$
$$z_{t+1} = w_t - s_{k+t}a_{i(k+t)}.$$

Similarly, from

$$\begin{aligned}
y_{k+t+1} &= P_{k+t}x_{k+t} + Q_{k+t}y_{k+t} - R_{k+t}s_{k+t}a_{i(k+t)} \\
&= P_{k+t}(\rho_t x_k + \tau_t y_k + z_t) + Q_{k+t}(\sigma_t x_k + \nu_t y_k + w_t) - R_{k+t}s_{k+t}a_{i(k+t)} \\
&= (P_{k+t}\rho_t + Q_{k+t}\sigma_t)x_k + (P_{k+t}\tau_t + Q_{k+t}\nu_t)y_k \\
&\quad + (P_{k+t}z_t + Q_{k+t}w_t - R_{k+t}s_{k+t}a_{i(k+t)}),
\end{aligned}$$

we have

$$\sigma_{t+1} = P_{k+t}\rho_t + Q_{k+t}\sigma_t,$$
$$\nu_{t+1} = P_{k+t}\tau_t + Q_{k+t}\nu_t,$$
$$w_{t+1} = P_{k+t}z_t + Q_{k+t}w_t - R_{k+t}s_{k+t}a_{i(k+t)}.$$

The scalar s_{k+t} can be computed from

$$\begin{aligned}
s_{k+t} &= (a_{i(k+t)}^T y_{k+t} - b_{i(k+t)})/\|a_{i(k+t)}\|^2 \\
&= [a_{i(k+t)}^T(\sigma_t x_k + \nu_t y_k + w_t) - b_{i(k+t)}]/\|a_{i(k+t)}\|^2 \\
(4.4) \qquad &= [\sigma_t a_{i(k+t)}^T x_k + \nu_t a_{i(k+t)}^T y_k + a_{i(k+t)}^T w_t - b_{i(k+t)}]/\|a_{i(k+t)}\|^2.
\end{aligned}$$

We show this approach in full detail, for cycles of fixed length T, in Algorithm 4.

Note that w_t and z_t have nonzeros in locations where any of the vectors $a_{i(k)}, a_{i(k+1)}, \ldots, a_{i(k+t-1)}$ contain nonzeros. Thus, assuming that these vectors do not overlap significantly, and that each of them has about δn nonzeros, we can estimate that w_t and z_t have about $t\delta n$ nonzeros, in the same locations as each other. The major costs at each iteration are as follows:

- s_{k+t} costs about $6\delta n$ operations when evaluated according to (4.4), since $a_{i(k+t)}$ has about δn nonzeros.
- z_{t+1} costs about $2\delta n$ operations, for the same reason.
- w_{t+1} costs about $3t\delta n + 2\delta n$ operations, since z_t and w_t both have about $\delta n t$ nonzeros, in the same locations, and $a_{i(k+t)}$ has about δn nonzeros.

The cost of updating x_k and y_k in Step 15 is about $3n + T\delta n$ each. Therefore, over a complete cycle of T iterations, we expect an approximate operation count of

$$\sum_{t=1}^{T-1}(3t\delta n + 10\delta n) + 6n + 2T\delta n \approx 1.5(T-1)T\delta n + 6n + 12T\delta n,$$

giving an approximate average cost per iteration of

$$1.5(T-1)\delta n + \frac{6n}{T} + 12\delta n.$$

This count is minimized by setting $T = T^* = 2/\sqrt{\delta}$; for this value we obtain an average count per iteration of $6\sqrt{\delta}n + 10.5\delta n$. This is still worse than the iteration

---

Algorithm 4. Efficient ARK for Sparse A: $x_{K+1} = \text{SARK}(A, b, \lambda, x_0, T, K)$

1: Check that $\lambda \in [0, \lambda_{\min}]$;
2: Initialize $y_0 = x_0$, $\gamma_{-1} = 0$, $k = 0$;
3: Generate the sequences

$$P_k = \alpha_{k+1}(1 - m\gamma_k), \qquad k = 0, 1, \ldots, K,$$
$$Q_k = 1 - \alpha_{k+1} + m\alpha_{k+1}\gamma_k, \quad k = 0, 1, \ldots, K,$$
$$R_k = 1 - \alpha_{k+1} + \alpha_{k+1}\gamma_k, \quad k = 0, 1, \ldots, K;$$

4: **while** $k \leq K$ **do**
5:    Set $t \leftarrow 0$, $\bar{x} \leftarrow x_k$, $\bar{y} \leftarrow y_k$, $\rho_0 \leftarrow 1$, $\tau_0 \leftarrow 0$, $\sigma_0 \leftarrow 0$, $\nu_0 \leftarrow 1$, $z_k \leftarrow 0$, $w_k \leftarrow 0$;
6:    **while** $t < T$ **do**
7:      **if** $k + t \geq K$ **then**
8:       break;
9:      **end if**
10:     Choose $i = i(k)$ from $\{1, 2, 3, \ldots, m\}$ with equal probability;
11:     Set

$$s_{k+t} = (a_i^T(\sigma_t\bar{x} + \nu_t\bar{y} + w_t) - b_i)/\|a_i\|^2,$$
$$\rho_{t+1} = \sigma_t,$$
$$\tau_{t+1} = \nu_t,$$
$$\sigma_{t+1} = P_{k+t}\rho_t + Q_{k+t}\sigma_t,$$
$$\nu_{t+1} = P_{k+t}\tau_t + Q_{k+t}\nu_t,$$
$$z_{t+1} = w_t - s_{k+t}a_i,$$
$$w_{t+1} = P_{k+t}z_t + Q_{k+t}w_t - R_{k+t}s_{k+t}a_i;$$

12:     Set $t \leftarrow t + 1$;
13:    **end while**
14:    Set $k \leftarrow k + T$;
15:    Set

$$x_k = \rho_t\bar{x} + \tau_t\bar{y} + z_t,$$
$$y_k = \sigma_t\bar{x} + \nu_t\bar{y} + w_t;$$

16: **end while**

---

cost for RK (which is $O(\delta n)$) but much better than that of ARK (which is $O(n)$). We show in the next section that the total number of iterations required by ARK to achieve a prescribed accuracy is lower than for RK, in general, which makes Algorithm 4 competitive in some regimes.

5. CONVERGENCE RATE

In this section, we study the convergence behavior of Algorithms 1, 3, and 4, estimating, in particular, the total number of operations required to achieve a specified level of accuracy. We also compare the approach with the conjugate gradient (CG)

algorithm, applied to the "normal equations" system $A^T A x = A^T b$. To simplify the comparisons, we assume throughout that (1.2) holds, so that $\|A\|_F^2 = m$.

The convergence of RK (Algorithm 1) is studied in [8, 14][1] It is shown that

$$(5.1) \qquad \mathbb{E}(\|x_{k+1} - \mathcal{P}_{A,b}(x_{k+1})\|^2) \leq \left(1 - \frac{\lambda_{\min}}{m}\right)^{k+1} \|x_0 - \mathcal{P}_{A,b}(x_0)\|^2,$$

where the expectation is taken over the indices $i(0), i(1), i(2), \dots$ selected at each iteration.

For ARK, we have the following result. The proof can be found in the appendix. It is quite technical, and follows to some extent the framework developed by Nesterov [11] for the accelerated coordinate descent method.

Theorem 5.1. *Apply* ARK *to the problem* (1.1) *with* $\lambda \in [0, \lambda_{\min}]$, *and define* $\sigma_1 = 1 + \frac{\sqrt{\lambda}}{2m}$ *and* $\sigma_2 = 1 - \frac{\sqrt{\lambda}}{2m}$. *Then we have for any* $k \geq 0$ *that*

$$(5.2) \qquad \mathbb{E}(\|v_{k+1} - x^*\|_{(A^T A)^+}^2) \leq \frac{4\|x_0 - x^*\|_{(A^T A)^+}^2}{(\sigma_1^{k+1} + \sigma_2^{k+1})^2}$$

and

$$(5.3) \qquad \mathbb{E}(\|x_{k+1} - x^*\|^2) \leq \frac{4\lambda\|x_0 - x^*\|_{(A^T A)^+}^2}{(\sigma_1^{k+1} - \sigma_2^{k+1})^2},$$

where $x^* := \mathcal{P}_{A,b}(x_0) = x_0 + A^+(b - Ax_0)$.

Essentially, Theorem 5.1 ensures that the ARK algorithm converges in expectation to the projection of the initial point x_0 onto the affine space defined by $Ax = b$.

Theorem 5.1 shows that when $\lambda > 0$, the ARK algorithm converges at a linear rate. If the value of $\lambda = 0$, we can obtain a sublinear rate. By taking limits as $\lambda \to 0^+$ in (5.3), we have

$$\lim_{\lambda \to 0^+} \frac{4\lambda\|x_0 - x^*\|_{(A^T A)^+}^2}{(\sigma_1^{k+1} - \sigma_2^{k+1})^2}$$

$$= \lim_{\lambda \to 0^+} \frac{4\lambda\|x_0 - x^*\|_{(A^T A)^+}^2}{\left(\left(1 + \frac{(k+1)\sqrt{\lambda}}{2m} + o(\sqrt{\lambda})\right) - \left(1 - \frac{(k+1)\sqrt{\lambda}}{2m} + o(\sqrt{\lambda})\right)\right)^2}$$

$$= \lim_{\lambda \to 0^+} \frac{4\lambda\|x_0 - x^*\|_{(A^T A)^+}^2}{\left(\frac{(k+1)\sqrt{\lambda}}{m} + o(\sqrt{\lambda})\right)^2}$$

$$(5.4) \qquad = \frac{4m^2\|x_0 - x^*\|_{(A^T A)^+}^2}{(k+1)^2}.$$

Next, we compare convergence rates of RK, ARK, and CG. We assume further that λ is set to its optimal value $\lambda_{\min}$ in ARK. Since all algorithms converge rapidly when $\lambda_{\min}(A^T A)$ is large, we are particularly interested in the case in which $\lambda_{\min}$ is small, that is, the linear system is ill-conditioned.

---

[1]In [14], it is required that A has full column rank, but this requirement is removed in [8, Theorem 4.3], where the Hoffman constant L is equivalent to $1/\sqrt{\lambda_{\min}}$.

TABLE 1. Operation and iteration counts to achieve expected accuracy ϵ for randomized Kaczmarz variants.

	Approx Operations per Iteration	Approx Iterations
RK (Algorithm 1)	$4\delta n$	$|\log \epsilon|(m/\lambda_{\min})$
ARK (Algorithm 3)	$3n + 6\delta n$	$|\log \epsilon|(m/\sqrt{\lambda})$
SARK (Algorithm 4)	$6\sqrt{\delta}n + 10.5\delta n$	$|\log \epsilon|(m/\sqrt{\lambda})$

5.1. **Comparison between** RK **and** ARK **for dense** A. The right-hand side of the bound (5.1) decreases by a factor of $1 - \lambda/m$ at each iteration. For ARK, we have that $\sigma_2^k \to 0$, so the decrease of the right-hand side is governed mainly by the behavior of the σ_1 term in the denominator. Asymptotically, we have a decrease factor per iteration of approximately

$$(5.5) \qquad \sigma_1^{-2} = \left(1 + \frac{\sqrt{\lambda}}{2m}\right)^{-2} \approx 1 - \frac{\sqrt{\lambda}}{m}.$$

We conclude that for small values of λ, the ARK approach will have significantly faster linear convergence. Even if we measure convergence rate *per operation*, ARK is still faster in general, since in the implementation of Algorithm 3, it requires only twice as many operations per iteration as RK.

5.2. **Comparison among** RK, ARK, **and** SARK **for sparse** A. When the coefficient matrix A is sparse, the comparisons change, because each iteration of RK costs less than each iteration of either ARK or SARK. On the other hand, fewer iterations of ARK are required to reduce the expected error below a specified tolerance. From (5.1), we deduce that the number N of iterations needed to reduce $\mathbb{E}(\|x_N - \mathcal{P}_{A,b}(x_N)\|^2)$ below a target threshold ϵ is $O((m/\lambda_{\min})|\log \epsilon|)$. We have from (5.3) and (5.5) (and ignoring a $\log \lambda$ term) that the number of iterations N of ARK and SARK needed to reduce $\mathbb{E}(\|x_k - x^*\|^2)$ below ϵ is $O((m/\sqrt{\lambda})|\log \epsilon|)$. Assuming approximately δn nonzeros in each row of each row of A, we summarize the operation and iteration counts for RK, ARK, and SARK in Table 1.

From the data in Table 1, and assuming that λ is set to its optimal value $\lambda_{\min}$ in the ARK and SARK algorithms, we conclude the following for the relative performance of these three approaches for various values of δ and $\lambda_{\min}$.

- RK will be approximately the best option if

$$\lambda \geq \max\left\{\left(\frac{4\delta}{3 + 6\delta}\right)^2, \left(\frac{4\sqrt{\delta}}{6 + 10.5\sqrt{\delta}}\right)^2\right\};$$

- SARK will be approximately best if

$$\lambda \leq \left(\frac{4\sqrt{\delta}}{6 + 10.5\sqrt{\delta}}\right)^2 \text{ and } \delta \leq 0.1;$$

- ARK will be approximately best, otherwise.

We illustrate these claims in Figure 1. Note that our comparison is based on approximate and worst-case analyses, which is why we claim only "approximate" superiority for each set of values in question. We can confidently say, however, RK will be superior for larger values of $\lambda_{\min}$, while ARK favors small $\lambda_{\min}$ and large δ, and SARK is superior to ARK for small values of δ. For small fixed values of $\lambda_{\min}$, RK will be superior for small values of δ, then SARK will be superior for intermediate δ values, and ARK superior for larger δ values.
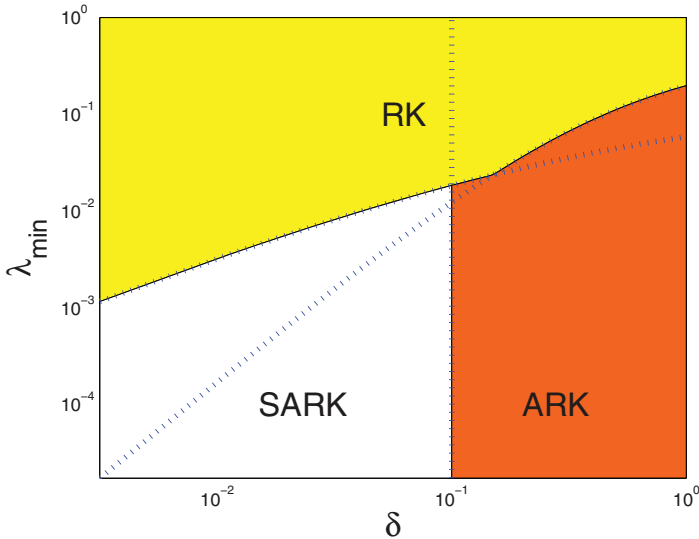

FIGURE 1. Illustration of the regions of the $(\delta, \lambda_{\min})$ space for which RK, ARK, and SARK are approximately superior (that is, $\lambda_{\min}$ in the graph). The white (yellow, red) area indicates that SARK (RK, ARK) is approximately best for the given combination of values.

5.3. **Comparison among** RK, SARK, **and** CG. Next, we compare RK and ARK with conjugate gradient (CG) applied to the normal-equations system $A^T A x = A^T b$. CG is a deterministic algorithm that requires matrix-vector multiplications with the entire data matrix A and its transpose at every iteration, while RK and ARK are randomized algorithms for which each iteration requires access to just one row of A, but which require many more iterations than CG in general. CG does not require estimates of parameters such as $\lambda_{\min}$ (though we show in the next section that estimation of this parameter can be incorporated into RK algorithms efficiently). Because the CG and RK approaches have very different convergence properties, and because their data access requirements are quite different, there are situations in which one or another of them will have an advantage. Here we do a simple comparison between CG and the RK methods based only on convergence rate as a function of operation count, and put aside the issues of suitability of one class or the other to various contexts and various computational platforms.

The asymptotic convergence rate for CG is

$$(5.6) \qquad \|Ax_{k+1} - b\|^2 \le \left(\frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}} \right)^{2(k+1)} \|Ax_0 - b\|^2.$$

(See, for example, formula (5.36) in [12].) The decrease factor per iteration is thus approximately

$$(5.7) \qquad 1 - 4\frac{\sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}}}.$$

If A has sparsity δ, the cost of the main operation of CG — multiplication by $A^T A$ — is about $4\delta mn$ operations. This is the approximate cost of m iterations of RK and about $(2/3)\sqrt{\delta}m$ iterations of SARK. Thus, for a roughly equivalent number of operations, assuming again that $\lambda = \lambda_{\min}$, we obtain the following approximate decrease factors for RK and SARK:

$$(5.8a) \qquad \text{RK}: \left(1 - \frac{\lambda_{\min}}{m} \right)^m \qquad \approx 1 - \lambda_{\min},$$

$$(5.8b) \qquad \text{ARK}: \left(1 - \frac{\sqrt{\lambda_{\min}}}{m} \right)^{(2/3)\sqrt{\delta}m} \qquad \approx 1 - \frac{2}{3}\sqrt{\delta}\sqrt{\lambda_{\min}}.$$

By comparing (5.7) and (5.8a), we see that RK may be competitive with CG if $\sqrt{\lambda_{\min}\lambda_{\max}}$ (the geometrically averaged eigenvalue of $A^T A$) is significantly larger than 1. From (5.7) and (5.8b), we see that SARK may be competitive with CG if $\delta\lambda_{\max}$ is significantly great than 1.

We note, however, that the asymptotic rate (5.6) for CG is somewhat pessimistic. In practice, performance of CG depends on the distribution of the eigenvalues of $A^T A$. Rapid convergence is often seen on early iterations, as the largest eigenvalues are "resolved," but the method often settles into a steady linear rate on later iterations.

6. Computational results

In this section, we study the computational behavior of RK, ARK, SARK, and CG on a variety of test problems. We start by comparing RK and ARK for dense A, then compare RK, ARK, and SARK for sparse A. Finally, we compare the randomized algorithms (RK and ARK) to the deterministic algorithm CG.

Since we need to supply the parameter λ to ARK, we introduce three ways of setting this parameter:

- ARK($\lambda_{\min}$): set $\lambda = \lambda_{\min}$. This choice gives the theoretically best convergence rate, and should be used if $\lambda_{\min}$ is known.
- ARK(0): set $\lambda = 0$. This choice requires no additional knowledge of A and guarantees convergence, though at a sublinear rate (see (5.4)).
- ARK(auto): λ determined automatically. Run RK for K_2 iterations and record x_{K_1+1} and x_{K_2+1}, where $K_2 = \lceil \frac{K}{10} \rceil$ and $K_1 = \max(1, K_2 - 10m)$. From (5.1), we can say roughly that $\mathbb{E}(\|Ax_k - b\|^2) \sim (1 - \lambda_{\min}/m)^k$, so

by setting $k = K_1$ and $k = K_2$, we deduce that $\lambda_{\min}$ could be estimated by the formula

$$m \left[1 - \left(\frac{\|Ax_{K_2} - b\|}{\|Ax_{K_1} - b\|} \right)^{\frac{2}{K_2 - K_1}} \right].$$

We find that a more conservative estimate of $\lambda_{\min}$ works better in practice, in which we replace the exponent $2/(K_2 - K_2)$ by $0.5/(K_2 - K_1)$ in our experiments. If the entire matrix A cannot be obtained at one time, one could estimate $\|Ax - b\|^2$ by using a sample of the rows of $Ax - b$.

We measure performance by plotting residual error $\|Ax - b\|$ against the number of iterations and the number of operations. The initial point $x_0 = 0$ is used in all algorithms.

6.1. Comparison between RK **and** ARK **for dense data.** Synthetic data for these tests is generated as follows: All elements of the data matrix $A \in \mathbb{R}^{m \times n}$ and the optimal solution $x^* \in \mathbb{R}^n$ are chosen to be i.i.d. $\mathcal{N}(0, 1)$. The length of all rows in A is normalized to 1. The right-hand side b is set to $b = Ax^*$. We run all algorithms 20 times (with 20 different sample sequences) and report the averaged performance.

Figures 2 and 3 show residual errors for RK and ARK with different values of λ. Figure 2 focuses on small problems while Figure 3 shows larger cases. In the graphs in the left column, the horizontal axis is iteration number, while in the right column, the horizontal axis is operation count, which is our proxy for computation cost. Operation count is obtained by scaling the number of iterations by our estimate of the average number of floating-point operations per iteration (see Table 1). From these figures, we observe the following.

- ARK($\lambda_{\min}$) and ARK(auto) converge much faster than RK (in both iterations and operations), except for very well conditioned problems.
- After the initial phase in which $\lambda_{\min}$ is estimated, ARK(auto) converges at about the same rate as ARK($\lambda_{\min}$).
- ARK(0) is not competitive with the other variants of ARK, but is competitive with RK on ill conditioned problems.

6.2. Comparison among RK**,** ARK**, and** SARK **for sparse data.** We compare RK, ARK(auto), and SARK on sparse data. Each element of A is set to 0 with the probability $1 - \delta$, so that the proportion of nonzero entries in A is approximately δ. The nonzero entries are chosen to be i.i.d. Gaussian $\mathcal{N}(0, 1)$, then the zeros rows are removed from A and the nonzero rows are normalized. The optimal solution x^* and right-hand side b are generated as in the dense case.

Figure 4 fixes $m = 1000$ and $n = 950$, and chooses $\delta = 0.8$, 0.08, and 0.01 for different levels of sparsity. For this small value of $\lambda_{\min}$ (about .0006 in all three cases), ARK/SARK outperforms RK with respect to number of iterations, as we see in the graphs in the left column of Figure 4. For the highest density ($\delta = 0.8$; top right graph), both ARK and SARK take fewer operations than RK, and ARK is more efficient than SARK. For moderate sparsity $\delta = 0.08$ (middle right), ARK is dominated by RK in operation count, while SARK is the best option of the three. For the most sparse case ($\delta = 0.01$; bottom right), RK dominates both ARK and SARK in the number of operations. These observations are consistent with our analysis of Section 5.2.
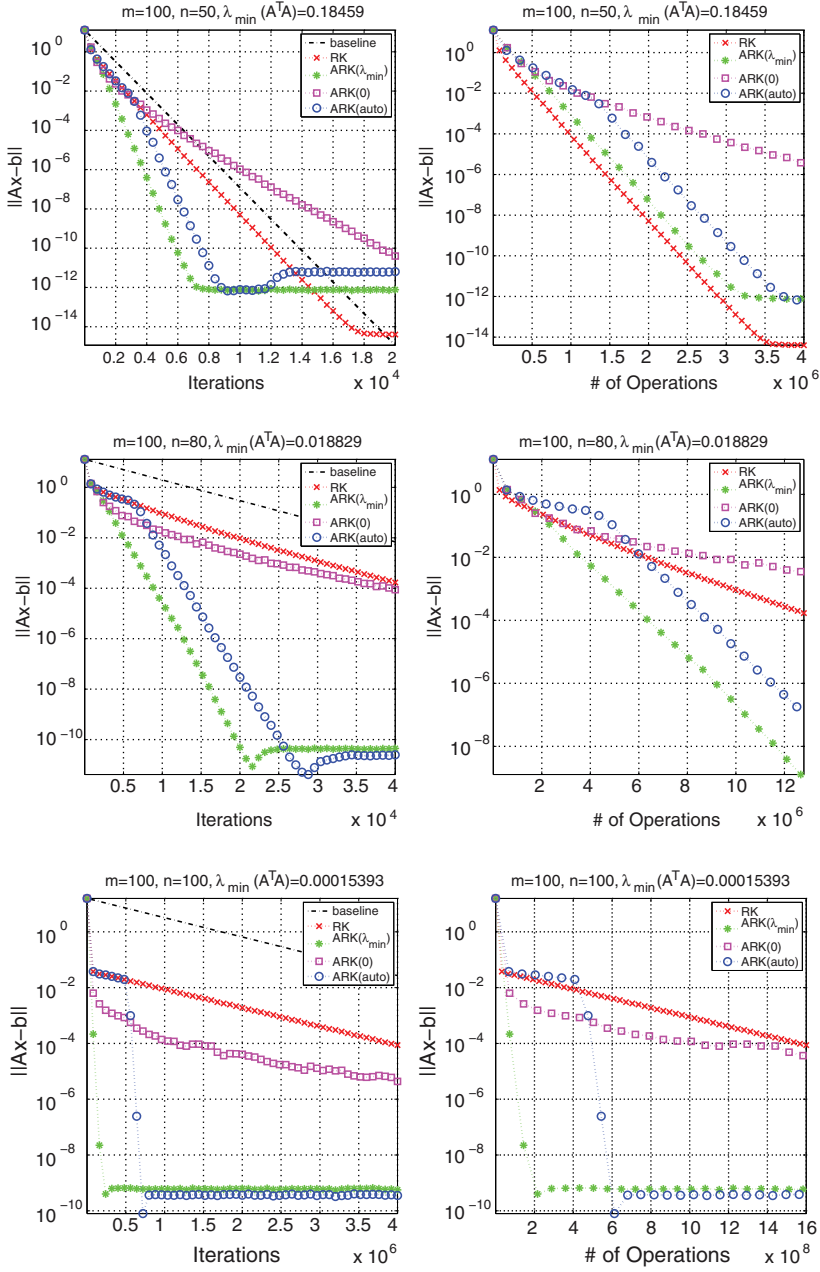
FIGURE 2. Comparison among RK, ARK($\lambda_{\min}$), ARK(0), and ARK(auto) on the dense data for $m = 100$ and $n = 50, 80, 100$. The graphs on the left (right) column plot iterations (operations) against residual error, averaged over 20 trials. The left graphs show a reference baseline sequence $\{(1 - \lambda_{\min}/m)^k : k = 0, 1, \dots\}$.
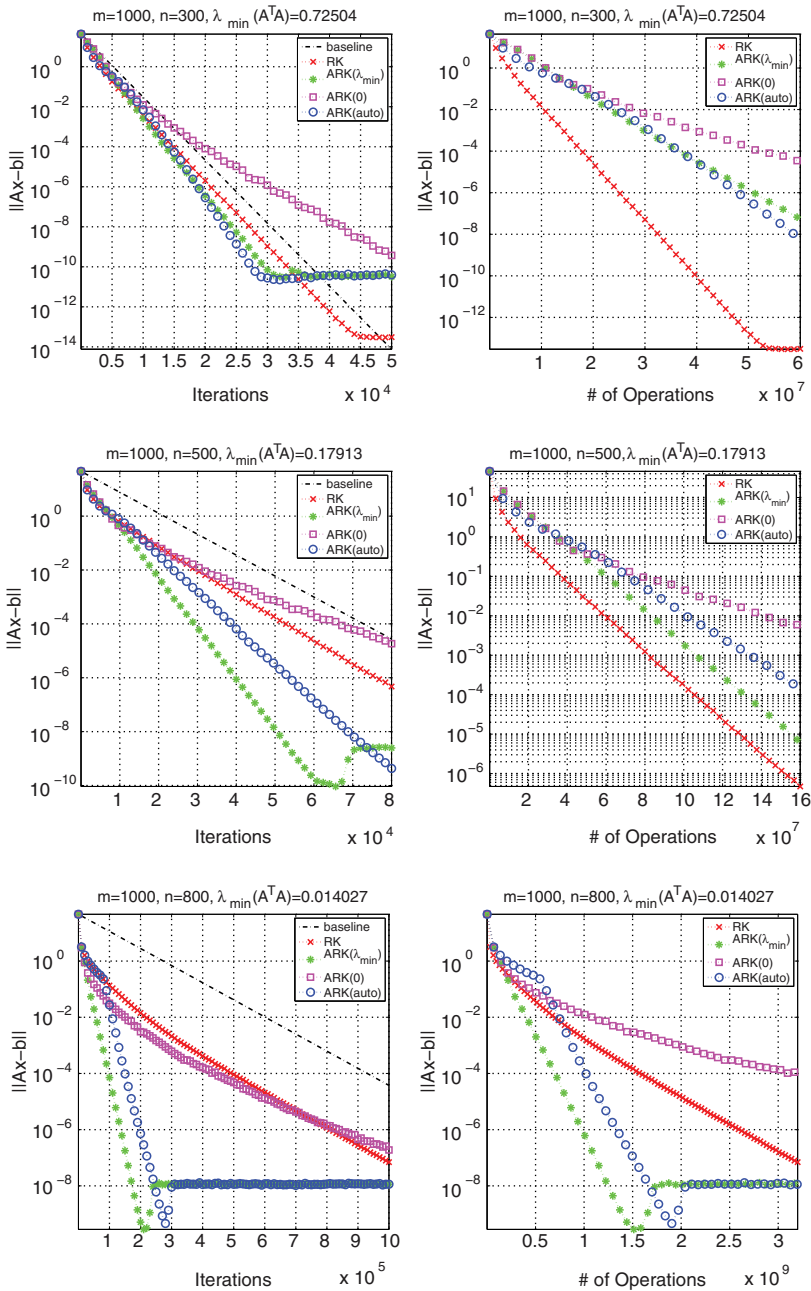
FIGURE 3. Comparison among RK, ARK($\lambda_{\min}$), ARK(0), and ARK(auto) on dense data for $m = 1000$ and $n = 300, 500, 800$. The graphs on the left (right) plot iterations (operations) against residual error, averaged over 20 trials. A reference baseline showing $\{(1 - \lambda_{\min}/m)^k : k = 0, 1, \dots\}$ is shown in the left plots.
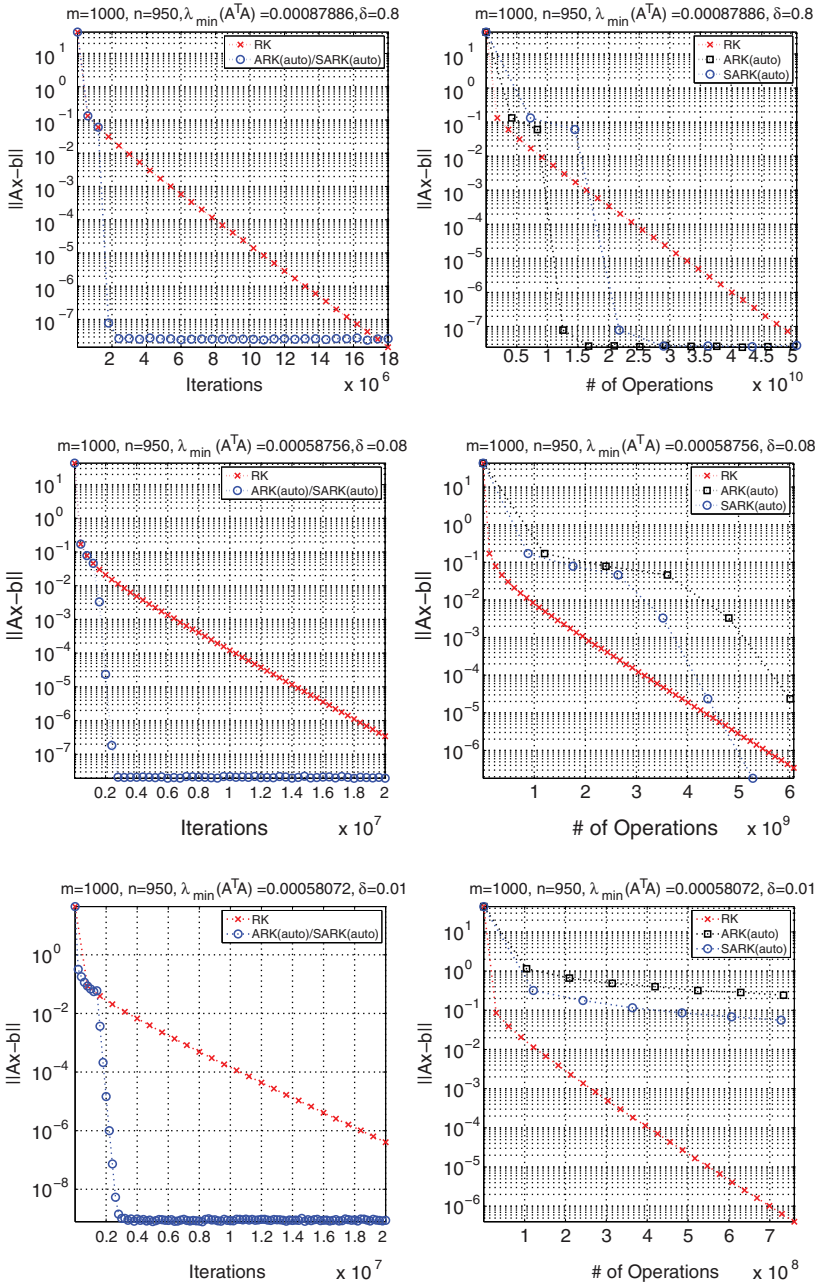
FIGURE 4. Comparison among RK, ARK(0), and SARK(auto) on sparse data with $m = 1000$, $n = 950$, and $\delta = 0.01$, 0.08, and 0.8. The graphs on the left (right) column plot iterations (operations) against residual errors, averaged over 20 trials.

6.3. **Comparison among** RK, ARK, **and** CG. A comparison between CG and RK was made in [14], where A is chosen to be Gaussian (elements are i.i.d. from $N(0, 1/n)$) with $m \gg n$. Problems of this type are particularly advantageous for RK. From random matrix theory [15], we have for these matrices that $\lambda_{\min} \approx (\sqrt{m/n}-1)^2$ and $\lambda_{\max} \approx (\sqrt{m/n}+1)^2$, so that when $m \gg n$, we have $\sqrt{\lambda_{\min}\lambda_{\max}} \gg 1$. The convergence rates observed in [14] are thus consistent with our analysis of Section 5.3. We do not consider the case $m \gg n$ further here, because $\lambda_{\min}$ is large in this setting, so all algorithms converge rapidly. We focus instead on cases in which $m = n$ and A is ill-conditioned.

For a given choice of $\lambda_{\min}$, we see from Section 5.3 that CG favors a smaller *maximum* eigenvalue, while RK and ARK favor a smaller *geometric average* eigenvalue. We control the distribution of eigenvalues of $A^T A$ by generating our test matrices as follows. First, find the SVD $U\Lambda V^T$ of a random $n \times n$ Gaussian matrix. Next, define an $n \times n$ diagonal matrix $\tilde{\Lambda}$ by $\tilde{\Lambda}_{ii} = i^{-\alpha}$, $i = 1, 2, \ldots, n$, for some parameter $\alpha > 0$, and compute $U\tilde{\Lambda}V^T$. Finally, normalize the rows of this matrix to obtain A. We generate x^* and b in the same way as in Section 6.1. The rows of A are normalized, so trace $(A^T A) = n$ and the average eigenvalue of $A^T A$ is 1. The parameter α controls the distribution of eigenvalues of $A^T A$; as α increases, $\lambda_{\max}$ tends to grow while $\lambda_{\min}$ shrinks.

We choose three values of α — 0.5, 0.75, and 0.9 — and fix $n = 500$ in Figure 5. Each row of plots in Figure 5 corresponds to a particular value of α, increasing from top to bottom. The left column plots the number of iterations of each method, but since the complexity of CG per iteration is $O(n^2)$ while that of other algorithms is $O(n)$, we do a rough calibration by making each iteration of CG occupy n units on the horizontal axes of the graphs in this column. We note that CG converges rapidly in its early iterations but then slows. This behavior is consistent with the analysis of CG, which shows that the asymptotic rate (5.7) is somewhat pessimistic, and that early iterations tend to behave in a manner dictated by the distribution of eigenvalues of $A^T A$ rather than the ratio of the extreme eigenvalues. Rapid initial convergence is enabled by the fact that each iteration of CG does a sweep over the entire matrix, giving it a global view of the data which is lacking in the randomized approaches. By contrast with CG the convergence of randomized algorithms is consistent and stable, and well predicted by the analysis.

As the value of α increases (that is, as we move from the top row of plots to the bottom row in Figure 5), we observe the following changes.

- $\lambda_{\min}$ becomes smaller, $\lambda_{\max}$ becomes larger, and $\sqrt{\lambda_{\min}\lambda_{\max}}$ becomes smaller, as α increases.
- The asymptotic convergence rate of CG, after resolution of the leading eigenspaces, becomes slower as α increases.
- The performance of RK becomes worse compared to CG as α grows. This observation is consistent with our analysis in Section 5.3, which predicts poorer performance as $\sqrt{\lambda_{\min}\lambda_{\max}}$ decreases.
- The performance of ARK (including ARK(λ) and ARK(auto)) is comparable to CG. CG decreases faster in the beginning but ARK is better at achieving high precision. An effective hybrid strategy might be to run CG in early iterations and turn to RK or ARK in later iterations.
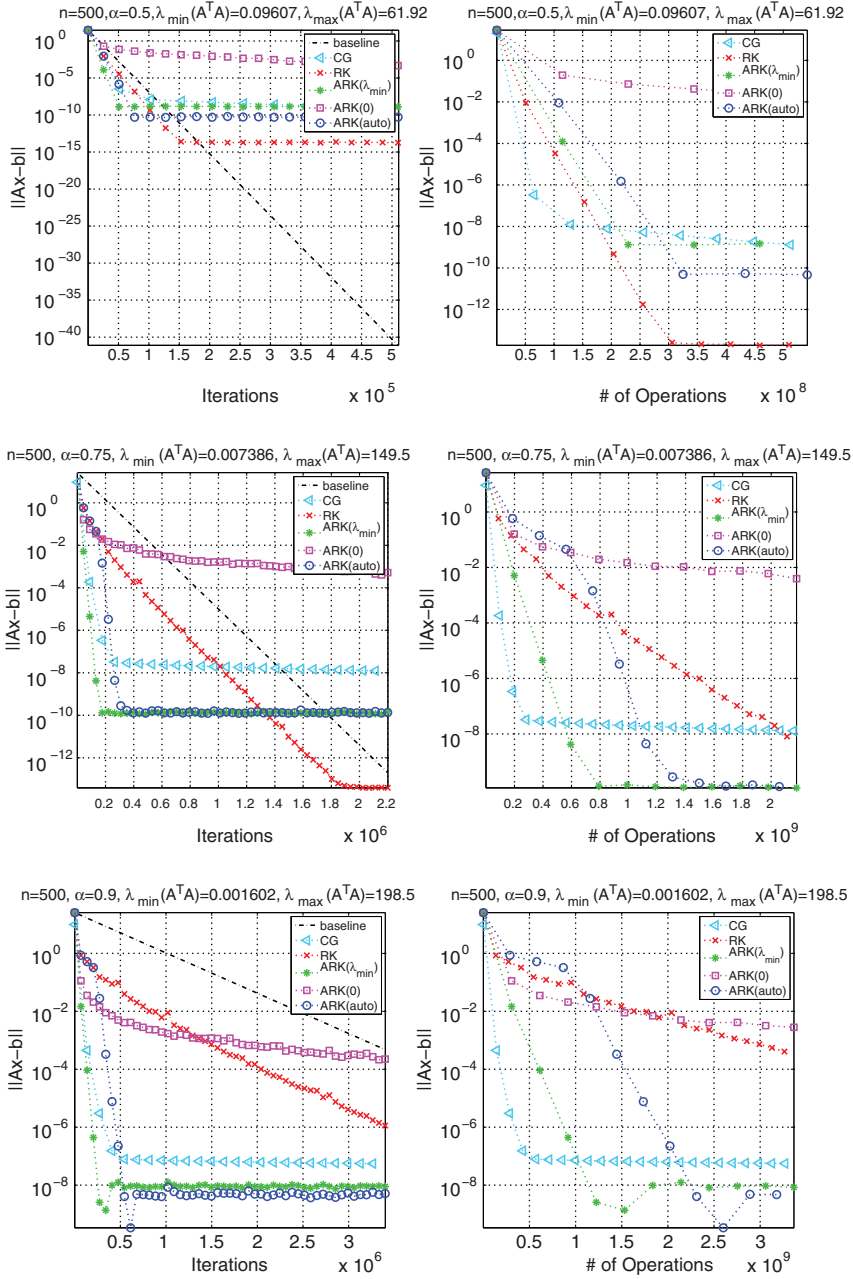
FIGURE 5. Comparison amomg CG, RK, ARK($\lambda_{\min}$), ARK(0), and ARK(auto) on dense data. The figures on the left (right) plot residual against iterations (operations). A reference baseline sequence of $\{(1 - \lambda_{\min}/m)^k : k = 0, 1, 2, \dots\}$ is shown in the left plots.

Appendix A. Proof of Theorem 5.1

In proving Theorem 5.1, we refer to the particular implementation in Algorithm 2 of ARK. We assume throughout that $\|a_i\|_2 = 1$ for $i = 1, 2, \ldots, m$.

We start with two useful technical lemmas.

Lemma A.1. *For any $y \in \mathbb{R}^n$, we have*

$$(A.1) \qquad \mathbb{E}_i\left(\left\|a_i(a_i^T y - b_i)\right\|_{(A^T A)^+}^2\right) \leq \frac{1}{m}\|Ay - b\|^2,$$

where the random variable i follows the uniform distribution over the set $\{1, 2, \ldots, m\}$.

Proof. Define the compact singular value decomposition of A as $A = U\Sigma V^T$, where $U^T U = I$, $V^T V = I$, and Σ is positive diagonal, so that $(A^T A)^+ = V\Sigma^{-2}V^T$. Denoting $U^T = [u_1\, u_2\, \ldots\, u_m]$, it is easy to show that $\|u_i\|_2 \leq 1$ for all $i = 1, 2, \ldots, m$. Using $\mathbb{E}_i$ to denote expectation with respect to the index i, we have

$$
\begin{aligned}
&\mathbb{E}_i\left(\left\|a_i(a_i^T y - b_i)\right\|_{(A^T A)^+}^2\right) \\
&= \frac{1}{m}\sum_{i=1}^m \langle (A^T A)^+ a_i(a_i^T y - b_i),\ a_i(a_i^T y - b_i)\rangle \\
&= \frac{1}{m}\operatorname{trace}\left[(A^T A)^+ \sum_{i=1}^m a_i(a_i^T y - b_i)^2 a_i^T\right] \\
&= \frac{1}{m}\operatorname{trace}\left[(A^T A)^+ A^T \operatorname{diag}(Ay - b)^2 A\right] \\
&= \frac{1}{m}\operatorname{trace}\left[V\Sigma^{-1}U^T \operatorname{diag}(Ay - b)^2 U\Sigma V^T\right] \\
&= \frac{1}{m}\operatorname{trace}\left[U^T \operatorname{diag}(Ay - b)^2 U\right] \\
&= \frac{1}{m}\|\operatorname{diag}(Ay - b)U\|_F^2 [4pt] \\
&= \frac{1}{m}\sum_{i=1}^m (a_i^T y - b)^2 \|u_i\|^2 \\
&\leq \frac{1}{m}\|Ay - b\|^2. \qquad \qquad \square
\end{aligned}
$$

Lemma A.2. *For any solution x^* to (1.1) and any $y \in \mathbb{R}^n$, we have*

$$(A.2) \qquad \mathbb{E}_i(\|\mathcal{P}_{a_i, b_i}(y) - x^*\|^2) = \|y - x^*\|^2 - \frac{1}{m}\|Ay - b\|^2,$$

where the random variable i follows the uniform distribution over the set $\{1, 2, \ldots, m\}$.

Proof. We have

$$\mathbb{E}_i(\|\mathcal{P}_{a_i,b_i}(y) - x^*\|^2)$$
$$= \mathbb{E}_i\left(\left\|y - a_i(a_i^T y - b_i) - x^*\right\|^2\right)$$
$$= \|y - x^*\|^2 + \mathbb{E}_i\left(\|a_i^T y - b_i\|^2\right) - 2\left\langle y - x^*, \ \mathbb{E}_i\left(a_i(a_i^T y - b_i)\right)\right\rangle$$
$$= \|y - x^*\|^2 + \frac{1}{m}\|Ay - b\|^2 - \frac{2}{m}\left\langle A(y - x^*), Ay - b\right\rangle$$
$$= \|y - x^*\|^2 - \frac{1}{m}\|Ay - b\|^2,$$

where the last equality uses $Ax^* = b$. □

The proof of Theorem 5.1 below essentially follows the proof for accelerated coordinate descent algorithm in [11] to construct the key inequality (A.12).

Proof. From Algorithm 2 one can verify that if the sequence $\{x_k, y_k, v_k\}$ is generated from $\mathrm{ARK}(A, b, \lambda, x_0, K)$, then the sequence generated from $\mathrm{ARK}(A, b - Ax_0, \lambda, 0, K)$ must be $\{x_k - x_0, y_k - x_0, v_k - x_0\}$. Thus, solving $Ax = b$ is equivalent to solving $Ax = b - Ax_0$ from initial point 0. It therefore suffices to study convergence from the zero initial point.

Recall from (3.1) that γ_k is the larger root of the following convex quadratic function:

$$t(\gamma) := \gamma^2 - \frac{\gamma}{m}(1 - \lambda\gamma_{k-1}^2) - \gamma_{k-1}^2.$$

Since $\lambda \leq \lambda_{\min} \leq m$, and using $\gamma_{-1} = 0$, we can note the following, from a simple recursive argument:

$$t(0) = -\gamma_{k-1}^2 \leq 0, \quad t(1/m) = \gamma_{k-1}^2(\lambda/m^2 - 1) \leq 0,$$

and thus $\gamma_k \geq 1/m$ for all $k \geq 0$. We can also verify that if $\gamma_{k-1} \leq 1/\sqrt{\lambda}$, we have

$$t(\gamma_{k-1}) = -(\gamma_{k-1}/m)(1 - \lambda\gamma_{k-1}^2) \leq 0$$
$$t\left(\frac{1}{\sqrt{\lambda}}\right) = \frac{1}{\lambda} - \frac{1}{m\sqrt{\lambda}}(1 - \lambda\gamma_{k-1}^2) - \gamma_{k-1}^2$$
$$= \frac{1}{\lambda} - \frac{1}{m\sqrt{\lambda}} + \gamma_{k-1}^2\left(\frac{\sqrt{\lambda}}{m} - 1\right)$$
$$\geq \frac{1}{\lambda} - \frac{1}{m\sqrt{\lambda}} + \frac{1}{\lambda}\left(\frac{\sqrt{\lambda}}{m} - 1\right) = 0,$$

which together imply that

$$\gamma_k \in \left[\gamma_{k-1}, \frac{1}{\sqrt{\lambda}}\right].$$

It follows from these bounds (together with the initialization $\gamma_{-1} = 0$) that $\{\gamma_k\}_{k=0}^\infty$ is an increasing sequence, bounded below by $1/m$ and above by $1/\sqrt{\lambda}$. It follows from these bounds and from $\lambda \leq m$ that α_k and β_k both lie in the interval $[0, 1]$ for all k.

Recalling that $x_0 = 0$, we have $x^* = A^+ b$. It can be verified that x_k, y_k, v_k, and x^* are all in $\mathcal{R}(A^T)$. We observe some useful relationships among the scalars in the

algorithm. We have from (3.1) and (3.2) that

$$(A.3) \qquad \frac{1 - \alpha_k}{\alpha_k} = \frac{m^2 \gamma_k - m}{m - \gamma_k \lambda} = \frac{m}{\gamma_k} \frac{m \gamma_k^2 - \gamma_k}{m - \gamma_k \lambda} = \frac{m \gamma_{k-1}^2}{\gamma_k}.$$

From (3.1) and (3.3), we have

$$(A.4) \qquad \gamma_k^2 - \frac{\gamma_k}{m} - \beta_k \gamma_{k-1}^2 = 0.$$

Defining

$$(A.5) \qquad r_k := \|v_k - x^*\|_{(A^T A)^+},$$

we consider the following expansion of r_{k+1}^2.

$$\begin{aligned}
r_{k+1}^2 &= \|v_{k+1} - x^*\|_{(A^T A)^+}^2 \\
&= \left\| \beta_k v_k + (1 - \beta_k) y_k - \gamma_k a_i (a_i^T y_k - b_i) - x^* \right\|_{(A^T A)^+}^2 \\
&= \|\beta_k v_k + (1 - \beta_k) y_k - x^*\|_{(A^T A)^+}^2 + \gamma_k^2 \|a_i(a_i^T y_k - b_i)\|_{(A^T A)^+}^2 \\
&\quad - 2\gamma_k \left\langle \beta_k v_k + (1 - \beta_k) y_k - x^*, \ (A^T A)^+ a_i(a_i^T y_k - b_i) \right\rangle \\
&= \|\beta_k v_k + (1 - \beta_k) y_k - x^*\|_{(A^T A)^+}^2 + \gamma_k^2 \|a_i(a_i^T y_k - b_i)\|_{(A^T A)^+}^2 \\
&\quad - 2\gamma_k \left\langle \beta_k \left(\frac{1}{\alpha_k} y_k - \frac{1 - \alpha_k}{\alpha_k} x_k \right) + (1 - \beta_k) y_k - x^*, \right. \\
&\qquad\qquad\qquad\qquad\qquad\qquad \left. (A^T A)^+ a_i(a_i^T y_k - b_i) \right\rangle \\
&= \|\beta_k v_k + (1 - \beta_k) y_k - x^*\|_{(A^T A)^+}^2 + \gamma_k^2 \|a_i(a_i^T y_k - b_i)\|_{(A^T A)^+}^2 \\
&\quad + 2\gamma_k \left\langle x^* - y_k + \frac{1 - \alpha_k}{\alpha_k} \beta_k (x_k - y_k), \ (A^T A)^+ a_i(a_i^T y_k - b_i) \right\rangle.
\end{aligned}$$

(A.6)

Denote by $i(k)$ the index randomly generated at iteration k, and let $I(k)$ denote all random indices seen at or before iteration k, that is,

$$I(k) := \{i(k), i(k-1), \ldots, i(0)\}.$$

Note that x_{k+1}, y_{k+1}, and v_{k+1} are determined by $I(k)$. In the remainder of the proof, we use $\mathbb{E}_{i(k)|I(k-1)}(\cdot)$ to denote the expectation of a random variable with respect to the index $i(k)$, conditioned on $I(k-1)$. Note that $\mathbb{E}_{I(k)}(\cdot) = E_{I(k-1)}(E_{i(k)|I(k-1)}(\cdot))$. When the context is clear, we use i in place of $i(k)$.

We consider the three terms in (A.6) in turn. From the convexity of $\|.\|_{(A^T A)^+}^2$ and the definition of β_k, the first item can be bounded as follows:

$$\begin{aligned}
&\|\beta_k v_k + (1 - \beta_k) y_k - x^*\|_{(A^T A)^+}^2 \\
&\leq \beta_k \|v_k - x^*\|_{(A^T A)^+}^2 + (1 - \beta_k) \|y_k - x^*\|_{(A^T A)^+}^2 \\
&= \beta_k \|v_k - x^*\|_{(A^T A)^+}^2 + \frac{\gamma_k \lambda}{m} \|y_k - x^*\|_{(A^T A)^+}^2 \\
&\leq \beta_k \|v_k - x^*\|_{(A^T A)^+}^2 + \frac{\gamma_k}{m} \|y_k - x^*\|^2,
\end{aligned}$$

(A.7)

where the last inequality is a consequence of $\lambda \leq \lambda_{\min}$ and the fact that y_k and x^* are in $\mathcal{R}(A^T)$. Using Lemmas A.1 and A.2, the second item in (A.6) can be

bounded in the expectation sense as follows:

$$\mathbb{E}_{i(k)|I(k-1)}\left(\left\|a_i(a_i^T y_k - b_i)\right\|^2_{(A^T A)^+}\right)$$

$$(A.8) \qquad \leq \frac{1}{m}\|Ay_k - b\|^2 = \|y_k - x^*\|^2 - \mathbb{E}_{i(k)|I(k-1)}(\|x_{k+1} - x^*\|^2).$$

For the third term in (A.6), we have by taking an expectation that

$$\mathbb{E}_{i(k)|I(k-1)}\left\langle x^* - y_k + \frac{1-\alpha_k}{\alpha_k}\beta_k(x_k - y_k),\ (A^T A)^+\left(a_i(a_i^T y_k - b_i)\right)\right\rangle$$

$$= \left\langle x^* - y_k + \frac{1-\alpha_k}{\alpha_k}\beta_k(x_k - y_k),\ (A^T A)^+ \mathbb{E}_{i(k)|I(k-1)}\left(a_i(a_i^T y_k - b_i)\right)\right\rangle$$

$$= \frac{1}{m}\left\langle x^* - y_k + \frac{1-\alpha_k}{\alpha_k}\beta_k(x_k - y_k),\ (A^T A)^+ \sum_i a_i(a_i^T y_k - b_i)\right\rangle$$

$$= \frac{1}{m}\left\langle x^* - y_k + \frac{1-\alpha_k}{\alpha_k}\beta_k(x_k - y_k),\ (A^T A)^+ A^T A(y_k - x^*)\right\rangle$$

$$\leq \frac{1}{m}\left\langle x^* - y_k + \frac{1-\alpha_k}{\alpha_k}\beta_k(x_k - y_k),\ y_k - x^*\right\rangle$$

$$= \frac{1}{m}\left(-\|y_k - x^*\|^2 + \frac{1-\alpha_k}{\alpha_k}\beta_k\langle x_k - y_k,\ y_k - x^*\rangle\right)$$

$$= \frac{1}{m}\left(-\|y_k - x^*\|^2 + \frac{1-\alpha_k}{2\alpha_k}\beta_k\left(\|x_k - x^*\|^2 - \|y_k - x^*\|^2 - \|x_k - y_k\|^2\right)\right)$$

$$= \frac{1}{m}\left(-\left(1 + \frac{1-\alpha_k}{2\alpha_k}\beta_k\right)\|y_k - x^*\|^2 + \frac{1-\alpha_k}{2\alpha_k}\beta_k\left(\|x_k - x^*\|^2 - \|x_k - y_k\|^2\right)\right)$$

$$= -\left(\frac{1}{m} + \frac{\beta_k\gamma_{k-1}^2}{2\gamma_k}\right)\|y_k - x^*\|^2 + \frac{\beta_k\gamma_{k-1}^2}{2\gamma_k}\left(\|x_k - x^*\|^2 - \|x_k - y_k\|^2\right)$$

$$\text{(from (A.3))}$$

$$(A.9) \quad \leq -\left(\frac{1}{m} + \frac{\beta_k\gamma_{k-1}^2}{2\gamma_k}\right)\|y_k - x^*\|^2 + \frac{\beta_k\gamma_{k-1}^2}{2\gamma_k}\|x_k - x^*\|^2.$$

By substituting (A.7), (A.8), and (A.9) into (A.6), we obtain

$$\mathbb{E}_{i(k)|I(k-1)}(r_{k+1}^2)$$

$$\leq \beta_k\|v_k - x^*\|^2_{(A^T A)^+} + \frac{\gamma_k}{m}\|y_k - x^*\|^2$$

$$\quad + \gamma_k^2(\|y_k - x^*\|^2 - \mathbb{E}_{i(k)|I(k-1)}(\|x_{k+1} - x^*\|^2))$$

$$\quad - \left(\frac{2\gamma_k}{m} + \beta_k\gamma_{k-1}^2\right)\|y_k - x^*\|^2 + \beta_k\gamma_{k-1}^2\|x_k - x^*\|^2$$

$$\leq \beta_k\|v_k - x^*\|^2_{(A^T A)^+} - \gamma_k^2\mathbb{E}_{i(k)|I(k-1)}(\|x_{k+1} - x^*\|^2) + \beta_k\gamma_{k-1}^2\|x_k - x^*\|^2$$

$$\quad + \left(\gamma_k^2 - \frac{\gamma_k}{m} - \beta_k\gamma_{k-1}^2\right)\|y_k - x^*\|^2$$

$$(A.10) \quad = \beta_k\|v_k - x^*\|^2_{(A^T A)^+} - \gamma_k^2\mathbb{E}_{i(k)|I(k-1)}(\|x_{k+1} - x^*\|^2) + \beta_k\gamma_{k-1}^2\|x_k - x^*\|^2,$$

where the final equality is a consequence of (A.4).

We now define two scalar sequences $\{A_k\}$ and $\{B_k\}$ as follows:

$$(A.11) \qquad A_k \geq 0,\ B_k \geq 0,\ B_0 \neq 0,\ B_{k+1}^2 = \frac{B_k^2}{\beta_k},\ A_{k+1}^2 = \gamma_k^2 B_{k+1}^2.$$

We set $A_0 = 0$ (to be consistent with the definition (A.11) and the fact that $\gamma_{-1} = 0$ in Algorithm 2) and note that $B_{k+1} \geq B_k$, since $\beta_k \in (0,1]$. Since from

(A.11) together with (3.1) and (3.3), we have

$$A_{k+1}^2 = \frac{B_k^2 \gamma_k^2}{\beta_k} = \frac{\gamma_k^2 A_k^2}{\beta_k \gamma_{k-1}^2} = \frac{A_k^2 \gamma_k^2}{\gamma_k^2 - \gamma_k/m},$$

we obtain that $\{A_k\}$ is also an increasing sequence.

Multiplying the last inequality (A.10) by B_{k+1}^2, and using the definition of r_k (A.5) along with (A.11) (in particular, the identities $B_{k+1}^2 \gamma_k^2 = A_{k+1}^2$, $B_{k+1}^2 \beta_k = B_k^2$, and $B_{k+1}^2 \beta_k \gamma_{k-1}^2 = A_k^2$), we obtain

$$B_{k+1}^2 \mathbb{E}_{i(k)|I(k-1)}(r_{k+1}^2) + A_{k+1}^2 \mathbb{E}_{i(k)|I(k-1)}(\|x_{k+1} - x^*\|^2)$$

(A.12)
$$\leq B_k^2 r_k^2 + A_k^2 \|x_k - x^*\|^2.$$

It follows that

$$\mathbb{E}_{I(k)}(B_{k+1}^2 r_{k+1}^2 + A_{k+1}^2(\|x_{k+1} - x^*\|^2))$$
$$= \mathbb{E}_{I(k-1)}(B_{k+1}^2 \mathbb{E}_{i(k)|I(k-1)}(r_{k+1}^2) + A_{k+1}^2 \mathbb{E}_{i(k)|I(k-1)}(\|x_{k+1} - x^*\|^2))$$
$$\leq \mathbb{E}_{I(k-1)}(B_k^2 r_k^2 + A_k^2 \|x_k - x^*\|^2).$$

By applying this inequality recursively, we obtain

$$\mathbb{E}_{I(k)}(B_{k+1}^2 r_{k+1}^2 + A_{k+1}^2(\|x_{k+1} - x^*\|^2)) \leq \mathbb{E}_{I(0)}(B_1^2 r_1^2 + A_1^2 \|x_1 - x^*\|^2)$$
$$\leq B_0^2 r_0^2 + A_0^2 \|x_0 - x^*\|^2 = B_0^2 r_0^2,$$

where we dropped the last term because $A_0 = 0$. It follows from this bound that

(A.13) $$\mathbb{E}(r_{k+1}^2) \leq \frac{B_0^2}{B_{k+1}^2} r_0^2 \quad \text{and} \quad \mathbb{E}(\|x_{k+1} - x^*\|^2) \leq \frac{B_0^2}{A_{k+1}^2} r_0^2.$$

We now need to estimate the growth of two sequences $\{A_k\}$ and $\{B_k\}$. Here we follow the proof for the accelerated coordinate descent algorithm of [11], but spelling out some details skipped in that paper. We have

$$B_k^2 = B_{k+1}^2 \beta_k = \left(1 - \frac{\lambda}{m} \gamma_k\right) B_{k+1}^2 = \left(1 - \frac{\lambda A_{k+1}}{m B_{k+1}}\right) B_{k+1}^2,$$

which implies that

$$\frac{\lambda}{m} A_{k+1} B_{k+1} = B_{k+1}^2 - B_k^2 = (B_k + B_{k+1})(B_{k+1} - B_k),$$

so by recalling that $B_{k+1} \geq B_k$, we obtain

(A.14) $$B_{k+1} \geq B_k + \frac{\lambda}{2m} A_k.$$

We have

$$\frac{A_{k+1}^2}{B_{k+1}^2} - \frac{A_{k+1}}{B_{k+1} m} = \gamma_k^2 - \frac{\gamma_k}{m} \qquad \text{(from (A.11))}$$

$$= (1 - \frac{\gamma_k \lambda}{m}) \gamma_{k-1}^2 \quad \text{(from (3.1))}$$

$$= \frac{\beta_k A_k^2}{B_k^2} = \frac{A_k^2}{B_{k+1}^2} \quad \text{(from (3.3) and (A.11)),}$$

so we obtain by multiplying both sides of this expression by B_{k+1}^2 and using $A_{k+1} \geq A_k$ that

$$\frac{1}{m}A_{k+1}B_{k+1} = A_{k+1}^2 - A_k^2 = (A_{k+1} + A_k)(A_{k+1} - A_k) \leq 2A_{k+1}(A_{k+1} - A_k)$$

and therefore

(A.15)
$$A_{k+1} \geq A_k + \frac{B_{k+1}}{2m} \geq A_k + \frac{B_k}{2m}.$$

By combining the inequalities (A.14) and (A.15) and applying a recursive argument, we can estimate A_{k+1} and B_{k+1} as follows:

$$\begin{bmatrix} A_{k+1} \\ B_{k+1} \end{bmatrix} \geq \begin{bmatrix} 1 & \frac{1}{2m} \\ \frac{\lambda}{2m} & 1 \end{bmatrix}^{k+1} \begin{bmatrix} A_0 \\ B_0 \end{bmatrix}.$$

The Jordan decomposition of the matrix in this expression is

$$\begin{bmatrix} 1 & \frac{1}{2m} \\ \frac{\lambda}{2m} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \sqrt{\lambda} & -\sqrt{\lambda} \end{bmatrix}^{-1} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ \sqrt{\lambda} & -\sqrt{\lambda} \end{bmatrix},$$

with

$$\sigma_1 = 1 + \frac{\sqrt{\lambda}}{2m}, \quad \sigma_2 = 1 - \frac{\sqrt{\lambda}}{2m}.$$

Thus we have

$$\begin{bmatrix} 1 & \frac{1}{2m} \\ \frac{\lambda}{2m} & 1 \end{bmatrix}^{k+1} = \frac{1}{2} \begin{bmatrix} 1 & \frac{1}{\sqrt{\lambda}} \\ 1 & -\frac{1}{\sqrt{\lambda}} \end{bmatrix} \begin{bmatrix} \sigma_1^{k+1} & 0 \\ 0 & \sigma_2^{k+1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ \sqrt{\lambda} & -\sqrt{\lambda} \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} \sigma_1^{k+1} + \sigma_2^{k+1} & (\sigma_1^{k+1} - \sigma_2^{k+1})/\sqrt{\lambda} \\ (\sigma_1^{k+1} - \sigma_2^{k+1})\sqrt{\lambda} & \sigma_1^{k+1} + \sigma_2^{k+1} \end{bmatrix}$$

which implies $A_{k+1} \geq B_0(\sigma_1^{k+1} - \sigma_2^{k+1})/(2\sqrt{\lambda})$ and $B_{k+1} \geq (\sigma_1^{k+1} + \sigma_2^{k+1})B_0/2$. By combining these bounds with (A.13), we obtain

$$\mathbb{E}(r_{k+1}^2) = \mathbb{E}(\|v_{k+1} - x^*\|_{(A^TA)^+}^2) \leq \frac{B_0^2}{B_{k+1}^2}r_0^2 \leq \frac{4\|x_0 - x^*\|_{(A^TA)^+}^2}{(\sigma_1^{k+1} + \sigma_2^{k+1})^2},$$

$$\mathbb{E}(\|x_{k+1} - x^*\|^2) \leq \frac{B_0^2}{A_{k+1}^2}r_0^2 \leq \frac{4\lambda\|x_0 - x^*\|_{(A^TA)^+}^2}{(\sigma_1^{k+1} - \sigma_2^{k+1})^2},$$

completing the proof. $\qquad\qquad\square$

References

[1] Y. Censor, D. Gordon, and R. Gordon, *Component averaging: an efficient iterative parallel algorithm for large and sparse unstructured problems*, Parallel Comput. **27** (2001), no. 6, 777–808, DOI 10.1016/S0167-8191(00)00100-9. MR1823354 (2002a:65211)

[2] Y. C. Eldar and D. Needell, *Acceleration of randomized Kaczmarz method via the Johnson-Lindenstrauss lemma*, Numer. Algorithms **58** (2011), no. 2, 163–177, DOI 10.1007/s11075-011-9451-z. MR2835851 (2012k:65043)

[3] A. Galántai, *On the rate of convergence of the alternating projection method in finite dimensional spaces*, J. Math. Anal. Appl. **310** (2005), no. 1, 30–44, DOI 10.1016/j.jmaa.2004.12.050. MR2160671 (2006m:65101)

[4] G. T. Herman, *Image Reconstruction from Projections*, The fundamentals of computerized tomography; Computer Science and Applied Mathematics, Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], New York-London, 1980. MR630896 (83e:92008)

[5] G. T. Herman, *Fundamentals of Computerized Tomography*, Springer, 2009.

[6] A. J. Hoffman, *On approximate solutions of systems of linear inequalities*, J. Research Nat. Bur. Standards **49** (1952), 263–265. MR0051275 (14,455b)

[7] S. Kaczmarz, *Angenaherte auflsung von systemen linearer gleichungen*, Bulletin International de l'Acadmie Polonaise des Sciences et des Letters **35** (1937), 355–357.

[8] D. Leventhal and A. S. Lewis, *Randomized methods for linear constraints: convergence rates and conditioning*, Math. Oper. Res. **35** (2010), no. 3, 641–654, DOI 10.1287/moor.1100.0456. MR2724068 (2012a:65083)

[9] D. Needell, *Randomized Kaczmarz solver for noisy linear systems*, BIT **50** (2010), no. 2, 395–403, DOI 10.1007/s10543-010-0265-5. MR2640019 (2011d:65086)

[10] Y. Nesterov, *Introductory lectures on convex optimization, A basic course*, Applied Optimization, vol. 87, Kluwer Academic Publishers, Boston, MA, 2004. MR2142598 (2005k:90001)

[11] Yu. Nesterov, *Efficiency of coordinate descent methods on huge-scale optimization problems*, SIAM J. Optim. **22** (2012), no. 2, 341–362, DOI 10.1137/100802001. MR2968857

[12] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer Series in Operations Research and Financial Engineering, Springer, New York, 2006. MR2244940 (2007a:90001)

[13] C. Popa, *Characterization of the solutions set of inconsistent least-squares problems by an extended Kaczmarz algorithm*, Korean J. Comput. Appl. Math. **6** (1999), no. 1, 51–64. MR1669567 (2000c:65035)

[14] T. Strohmer and R. Vershynin, *A randomized Kaczmarz algorithm with exponential convergence*, J. Fourier Anal. Appl. **15** (2009), no. 2, 262–278, DOI 10.1007/s00041-008-9030-4. MR2500924 (2010f:60126)

[15] R. Vershynin, *Introduction to the non-asymptotic analysis of random matrices*, Compressed sensing, Cambridge Univ. Press, Cambridge, 2012, pp. 210–268. MR2963170

[16] A. Zouzias and N. M. Freris, *Randomized extended Kaczmarz for solving least-squares*, Preprint arXiv:1205.5770v2, 2012.

Department of Computer Sciences, University of Wisconsin-Madison, Madison, Wisconsin 53706-1685

Current address: Department of Computer Sciences, 1210 W. Dayton St., Madison, Wisconsin 53706-1685

E-mail address: `ji.liu.uwisc@gmail.com`

Department of Computer Sciences, University of Wisconsin-Madison, Madison, Wisconsin 53706-1685

E-mail address: `swright@cs.wisc.edu`