A Fast Restarted Surrounding Algorithm for Solving Large Consistent Linear Algebraic Equations

Ke Zhang, Xiang-Xiang Chen and Xiang-Long Jiang

***Abstract*—We present a deterministic restarted surrounding algorithm to solve large-scale consistent linear systems. It utilizes a sketch of the coefficient matrix for implementation. Our theoretical analysis shows that the proposed algorithm converges to the unique least Euclidean-norm solution of the linear system, given a full-rank coefficient matrix. Numerical examples demonstrate that the new algorithm converges faster than existing algorithms in terms of CPU time and iteration counts.**

***Index Terms*—consistent linear system, surrounding algorithm, Kaczmarz method, sketching technique.**

I. Introduction

In this work, we are interested in solving the consistent linear system of equations in the form

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$ and $m \neq n$. Assume henceforth that the matrix A is of full rank. Then we only consider the least Euclidean-norm solution of the linear system (1).

Solving systems of linear equations is one of the most critical challenges in scientific computing and engineering. In the linear system (1), if $m = n$, then Krylov subspace methods, which access the matrix only in matrix-vector product form, are often preferred [14]. If $m > n$ or $m < n$, then the row-action methods are handy, particularly if the size of the underlying problem is huge. The row-action algorithms exploit the rows of the coefficient matrix A (one row at a time) without making any changes to A [4]. Among them, the randomized Kaczmarz (RK) algorithm [17], which overcomes the difficulty in analyzing the convergence rate of the classical Kaczmarz algorithm [10], becomes an efficient candidate for solving consistent linear systems. Many promising variants have been developed ever since. In addition, two closely related algorithms to RK are the randomized coordinate descent (RCD) algorithm [12] and the randomized extended Kaczmarz (REK) algorithm [22] which are proposed for solving inconsistent linear systems or least-squares problems. For more recent developments of these algorithms, we refer to [1], [2], [3], [4], [8], [9], [13], [19], [20], [21].

In [16], Steinerberger comes up with a novel randomized surrounding (RS) algorithm that is based on a sequence of Householder reflections. Geometrically, the RS algorithm can be interpreted as approximating the center of a sphere from given points on it. At each iteration, the RS algorithm reflects the current iterate through a randomly-chosen hyperplane which results in the next iterate. The iterations proceed until M reflections are applied, where M is a positive integer. Then the average of the previous M iterates is taken as an approximation to the least Euclidean-norm solution. It is shown that RS enjoys a linear convergence rate that is comparable with RK. More theoretical and algorithmic details about the RS algorithm can be found in Section II-A; see also [16]. As stated above, a new approximation to the solution in the RS algorithm is obtained only after M iterations, which can be time-consuming especially when M is large. To overcome this drawback, Yin et al. [18] propose a restarted randomized surrounding (RRS) algorithm for the consistent linear systems (1). Instead of exploiting M intermediate iterates computed previously, RRS takes the average of merely s iterates as the approximate solution per iteration, where the restarting frequency s is a user-prescribed positive integer that can be far less than M. This restarting technique turns out to be very efficient in that the RRS algorithm is superior to the original RS algorithm in terms of CPU time and the number of iteration steps. Recently, some algorithms related with the surrounding algorithm have been proposed in [7], [15].

In this work, we propose a fast restarted surrounding (FRS) algorithm for solving large consistent linear systems. Unlike the RK and RRS algorithms, the FRS algorithm utilizes the sketching of A by deeming the entries of residual vectors as weights, and imposes the Householder reflections deterministically. We prove that FRS converges to the unique least Euclidean-norm. Numerical examples show that FRS outperforms RK and RRS in terms of computational time and iterations steps for most test problems. In the rest of this work, we shall adopt the following traditional nomenclature. We denote by $\|\cdot\|_2$ the Euclidean norm of a vector and $\|\cdot\|_F$ the matrix Frobenius norm. The ith row of a matrix A is represented by $A^{(i)}$ while A^T stands for the transpose of A. The notation B^{-1} signifies the inverse of the matrix B. The unique least Euclidean-norm solution of (1) is given by $x_\star$. The identity matrix is denoted by I with its size deducible from the context. The notation $\sigma_{\min}(A)$ symbolizes the smallest (nonzero) singular value of a matrix A.

Manuscript received June 28, 2024; revised October 5, 2024.

This work was supported in part by the National Natural Science Foundation of China under Grants 12271342 and 12401498.

Ke Zhang is a lecturer of the Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China (corresponding author to provide phone:+86-21-3828-2215; e-mail: kezhang@shmtu.edu.cn).

Xiang-Xiang Chen is a postgraduate student of the Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China (e-mail: xxchen01@163.com).

Xiang-Long Jiang is a lecturer of the Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China (e-mail: jiangxl@shmtu.edu.cn).

The outline of this paper is as follows. In Section II, we recapitulate the RS and RRS algorithms. In Section III, we present the new algorithm and analyze its convergence. In Section IV, we evaluate the effectiveness of FRS by comparing it with RK and RRS. In Section V, we summarize this work and point out some possible future work.

II. The randomized surrounding algorithm and its restarted variant

In this section, we recap the randomized surrounding (RS) algorithm and its restarted variant (RRS). These two surrounding algorithms lay the foundation for the exposition of the new algorithm in Section III.

A. The RS algorithm

The RS algorithm is originally proposed for solving nonsingular square linear systems, i.e., $m = n$ in (1). Denote by $x_\star$ the unique least Euclidean-norm solution of the linear system (1). At the kth iteration, the iterate y_{k+1} is obtained by reflecting y_k through a randomly-chosen hyperplane $A^{(i_k)}x_\star = b^{(i_k)}$, that is,

$$y_{k+1} = y_k + 2 \cdot \frac{b^{(i_k)} - A^{(i_k)}y_k}{\|A^{(i_k)}\|_2^2}(A^{(i_k)})^T, \tag{2}$$

where the index i_k of the working row $A^{(i_k)}$ is picked according to the probability

$$\Pr(\text{row} = i_k) = \frac{\|A^{(i_k)}\|_2^2}{\|A\|_F^2}.$$

From (2), it is trivial to show that

$$y_{k+1} - x_\star = \left(I - 2 \cdot \frac{(A^{(i_k)})^T A^{(i_k)}}{\|A^{(i_k)}\|_2^2}\right)(y_k - x_\star)$$

for $k = 1, 2, \ldots, M-1$. To put it another way, one can verify that the distances between y_i and $x_\star$ remain unchanged during the reflections in that

$$\|y_M - x_\star\|_2 = \|y_{M-1} - x_\star\|_2 = \cdots = \|y_1 - x_\star\|_2.$$

Note that the iteration sequence $\{y_k\}_{k=1}^M$ itself does not converge to the solution $x_\star$. However, one can take the average of the previously computed iterates $y_1, \ldots, y_M$, i.e., $\sum_{k=1}^M y_k/M$, as the approximation to the least-norm solution of the linear system (1). The above process is geometrically inspiring and can be recast as a problem of locating the center of a sphere with given points on it.

In [16], Steinerberger analyzes the convergence of the aforementioned process and gets the following result

$$\mathbb{E}\|\frac{1}{M}\sum_{k=1}^M y_k - x_\star\|_2 \le \frac{1 + \|A\|_F\|A^{-1}\|_2}{\sqrt{M}}\|x_1 - x_\star\|_2,$$

where $x_1 \in \mathbb{R}^n$ is the initial guess.

Finally, we mention some recent work related with the RS algorithm. In [15], Shao presents a deterministic Kaczmarz method for solving (1), which provides an answer to the open problem given in [16] on how to find deterministically a more efficient way to approximate the solution of (1). By exploiting the randomized Douglas-Rachford approach, Han et al. propose the randomized r-sets-Douglas-Rachford algorithm as well as its variant with momentum acceleration [7]. More discussions on the differences among these algorithms can be found in [7], [15], [16].

Algorithm 1 The RRS algorithm

Input: A, b, x_0, s and l
Output: x_l

```
for k = 0, 1, ..., l − 1 do
    y_{k,0} = x_k.
    for i = 1, ..., s − 1 do
        Select i_k ∈ {1, ..., m} as per
            Pr(row = i_k) = ||A^{(i_k)}||_2^2 / ||A||_F^2.
        y_{k,i} = y_{k,i−1} + 2 · (b^{(i_k)} − A^{(i_k)} y_{k,i−1}) / ||A^{(i_k)}||_2^2 · (A^{(i_k)})^T.
    end for
    x_{k+1} = (1/s) Σ_{i=0}^{s−1} y_{k,i}.
end for
```

B. The RRS algorithm

In this subsection, we give a brief introduction to the restarted randomized surrounding (RRS) algorithm.

As explained in Section II-A, an approximate solution to $x_\star$ is attained by taking average of the previously computed M points. However, the computational overhead of the RS algorithm may increase as M goes up. It is noted in [16, p. 420] that we can run the RS algorithm for a while, update the current approximate solution with the average of previously computed iterates, and then restart RS with this average as a new initial guess. This gives rise to the RRS algorithm [18]; see Algorithm 1.

The process of RRS is in essence an inner-outer iteration; the inner iteration imposes $s-1$ Householder reflections and collects the intermediate points $y_{k,0}, y_{k,1}, \ldots, y_{k,s-1}$, where s is the restarting frequency. The outer iteration computes an updated approximate solution x_{k+1} from the average of $y_{k,0}, y_{k,1}, \ldots, y_{k,s-1}$. Then x_{k+1} is used as a new initial point and proceeds with the $s-1$ Householder reflections in the inner iteration. Numerical results in [18] show that such restarting technique turns out to be very efficient when equipped with suitable choices of s. We refer to [18] for more algorithmic details.

The convergence of the RRS algorithm is investigated in [18] and restated below.

Theorem 1. *For any initial guess $x_0 \in \mathbb{R}^n$, the iteration sequence $\{x_k\}_{k=0}^\infty$ generated from Algorithm 1 converges to the unique least Euclidean-norm solution $x_\star$ in expectation. Moreover, it holds that*

$$\mathbb{E}\|x_k - x_\star\|_2^2 \le \alpha^k\|x_0 - x_\star\|_2^2,$$

where s is the restarting frequency, and the convergence factor $\alpha < s^{-1} + 2s^{-2}\sum_{i=1}^{s-1}(s-i)\left(1 - 2\sigma_{\min}^2(A)/\|A\|_F^2\right)^i$.

III. A fast restarted surrounding algorithm

In this section, we present a fast restarted surrounding (FRS) algorithm for solving the consistent linear system (1).

Unlike the RS and RRS algorithms, the FRS algorithm does not choose the hyperplane for reflections randomly. Instead it deterministically uses the sketching of A with entries of the residual vectors as weights. Then the Householder

Algorithm 2 The fast restarted surrounding (FRS) algorithm

Input: A, b, x_0, s and l
Output: x_l
1: **for** $k = 0, 1, \ldots, l-1$ **do**
2: $y_{k,0} = x_k$.
3: **for** $i = 1, \ldots, s$ **do**
4: $c_{k,i-1} = b - Ay_{k,i-1}$.
5: $y_{k,i} = y_{k,i-1} + 2 \cdot \frac{\|c_{k,i-1}\|_2^2}{\|A^T c_{k,i-1}\|_2^2} A^T c_{k,i-1}$.
6: **end for**
7: $x_{k+1} = \frac{1}{s}\sum_{i=1}^{s} y_{k,i}$.
8: **end for**

reflections in the inner iteration are applied via the iterative scheme

$$y_{k,i} = y_{k,i-1} + 2 \cdot \frac{\|c_{k,i-1}\|_2^2}{\|A^T c_{k,i-1}\|_2^2} A^T c_{k,i-1},$$

where $c_{k,i-1} = b - Ay_{k,i-1}$. After s steps of inner iterations, the new approximate solution x_{k+1} is obtained by taking the average of $y_{k,1}, \ldots, y_{k,s}$, that is, $x_{k+1} = (\sum_{i=1}^{s} y_{k,i})/s$.

The result below implies that the iteration sequence $\{x_k\}_{k=0}^{\infty}$ generated in Algorithm 2 is convergent.

Theorem 2. *Assume that A is of full rank in* (1)*. The iteration sequence $\{x_k\}_{k=0}^{\infty}$ generated from Algorithm 2 converges to the unique least Euclidean-norm solution $x_\star$ of the linear system* (1)*. In addition, the error satisfies*

$$\|x_{k+1} - x_\star\|_2 = \delta_k \|x_k - x_\star\|_2,$$

where $\delta_k = \|\sum_{i=1}^{s} g_{k,i}\|_2 / \sum_{i=1}^{s} \|g_{k,i}\|_2 < 1$ and $g_{k,i} = y_{k,i} - x_\star$ for $k = 0, 1, \ldots$.

Proof: Let $g_{k,i} = y_{k,i} - x_\star$ for $i = 1, \ldots, s$. Then $c_{k,i-1} = b - Ay_{k,i-1} = -Ag_{k,i-1}$. It follows from step 5 of Algorithm 2 that

$$\begin{aligned} g_{k,i} &= g_{k,i-1} + 2 \cdot \frac{\|c_{k,i-1}\|_2^2}{\|A^T c_{k,i-1}\|_2^2} A^T c_{k,i-1} \\ &= g_{k,i-1} - 2 \cdot \frac{(g_{k,i-1})^T A^T A g_{k,i-1}}{\|A^T A g_{k,i-1}\|_2^2} A^T A g_{k,i-1} \\ &= (I - 2P_{k,i-1})\, g_{k,i-1}, \end{aligned} \tag{3}$$

where $P_{k,i-1} = (A^T A g_{k,i-1} (g_{k,i-1})^T A^T A)/\|A^T A g_{k,i-1}\|_2$. One can check that $I - 2P_{k,i-1}$ is an orthogonal matrix since

$$(I - 2P_{k,i-1})^T (I - 2P_{k,i-1}) = I.$$

Therefore, by taking norm on both sides of (3), we get

$$\|g_{k,i}\|_2 = \|(I_n - 2P_{k,i-1})\, g_{k,i-1}\|_2 = \|g_{k,i-1}\|_2, \tag{4}$$

for $i = 1, \ldots, s$.

Now we denote the error by $e_k = x_k - x_\star$ for $k = 0, 1, \ldots$. It follows from step 7 of Algorithm 2 that

$$\begin{aligned} e_{k+1} &= \frac{1}{s}\sum_{i=1}^{s} y_{k,i} - x_\star \\ &= \frac{1}{s}\sum_{i=1}^{s} (y_{k,i} - x_\star) \\ &= \frac{1}{s}\sum_{i=1}^{s} g_{k,i}. \end{aligned} \tag{5}$$

Taking the Euclidean-norm on both sides of (5), coupled with the equation (4), yields the inequality

$$\|e_{k+1}\|_2 = \|\frac{1}{s}\sum_{i=1}^{s} g_{k,i}\|_2 = \delta_k \cdot \frac{1}{s}\sum_{i=1}^{s} \|g_{k,i}\|_2 = \delta_k \|g_{k,1}\|_2, \tag{6}$$

where $\delta_k = \|\sum_{i=1}^{s} g_{k,i}\|_2 / \sum_{i=1}^{s} \|g_{k,i}\|_2$. From the definition of δ_k, we know that $\delta_k \leq 1$ and the equality holds when the vectors $g_{k,1}, \ldots, g_{k,s}$ are collinear. However, this cannot happen in practice because $g_{k,1}, \ldots, g_{k,s}$ can never be collinear simultaneously; for instance, $g_{k,2}$ is the vector reflected from $g_{k,1}$ by applying a certain Householder reflection and hence $g_{k,2}$ cannot parallel $g_{k,1}$. Therefore, it always holds that $\delta_k < 1$. Furthermore, it can be deduced from (6) and $y_{k,0} = x_k$ that

$$\begin{aligned} \|g_{k,1}\|_2 &= \|y_{k,0} - x_\star + 2 \cdot \frac{\|c_{k,0}\|_2^2}{\|A^T c_{k,0}\|_2^2} A^T c_{k,0}\|_2^2 \\ &= \|x_k - x_\star + 2 \cdot \frac{\|b - Ax_k\|_2^2}{\|A^T(b - Ax_k)\|_2^2} A^T (b - Ax_k)\|_2^2 \\ &= \|e_k - 2 \cdot \frac{\|Ae_k\|_2^2}{\|A^T A e_k\|_2^2} A^T A e_k\|_2^2 \\ &= \|e_k - 2 \cdot \frac{e_k^T A^T A e_k}{\|A^T A e_k\|_2^2} A^T A e_k\|_2^2 \\ &= \|(I_n - 2 \cdot \frac{A^T A e_k e_k^T A^T A}{\|A^T A e_k\|_2^2}) e_k\|_2^2 \\ &= \|e_k\|_2^2, \end{aligned} \tag{7}$$

where in the first equation the relation $c_{k,0} = b - Ax_k$ is used. Combining (6) and (7) completes the proof. ■

IV. Numerical experiments

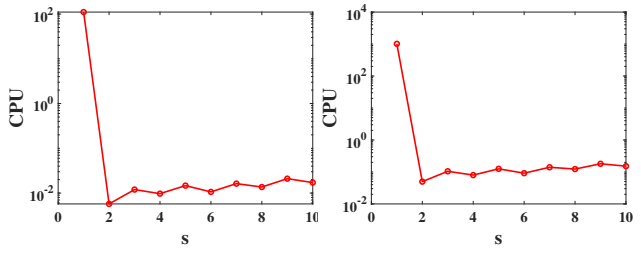
We are now in a position to appraise the numerical performance of of the FRS algorithm by comparing it with the randomized Kaczmarz (RK) algorithm [17] and the restarted randomized surrounding (RRS) [18] algorithm with varying choices of restarting frequency s. We run all experiments on a laptop with AMD Ryzen 7 5825U with Radeon Graphics @2.00 GHZ 16.0 GB RAM using MATLAB (R2022a) under Windows 11 operating system. The effectiveness of these three algorithms are assessed regarding the averages of CPU time in seconds (CPU) and the number of iteration steps (IT). By "average", we mean that running the RK and RRS algorithms 20 times and taking the averages of total CPU or IT. Since the FRS algorithm is deterministic, then it is run only once. To manifest the acceleration of FRS over RK and RRS, we adopt the CPU speed-up defined as follows

$$\begin{aligned} \text{speed} - \text{up}_{\text{RK}} &= \frac{\text{CPU of RK}}{\text{CPU of FRS}}, \\ \text{speed} - \text{up}_{\text{RRS}} &= \frac{\text{CPU of RRS}}{\text{CPU of FRS}}. \end{aligned}$$

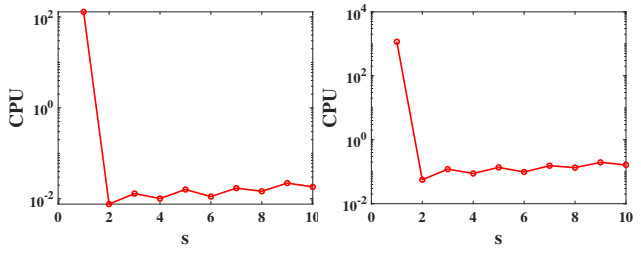
Here the quantity $\text{speed} - \text{up}_{\text{RRS}}$ is obtained by computing the ratio of the shortest CPU time of RRS(s) to that of FRS for $s = 5, 10$ and 20. The choices for s with these values are recommended for RRS in [18].

The initial guess for all algorithms is set to be $x_0 = \mathbf{0}$. We terminate the underlying algorithm once IT exceeds 200,000 steps or when the relative solution error (RSE) at the kth iteration satisfies $\text{RSE} < 10^{-6}$, where

$$\text{RSE} = \frac{\|x_k - x_\star\|_2^2}{\|x_\star\|_2^2}.$$

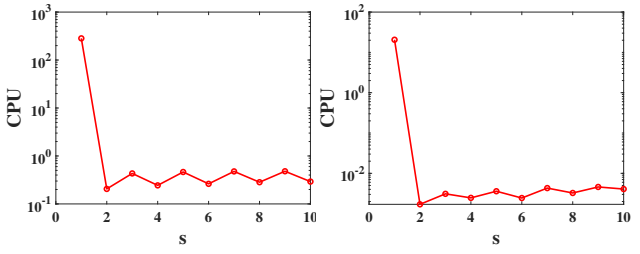


(a) `randn`$(5000, 300)$ (b) `randn`$(10000, 1000)$

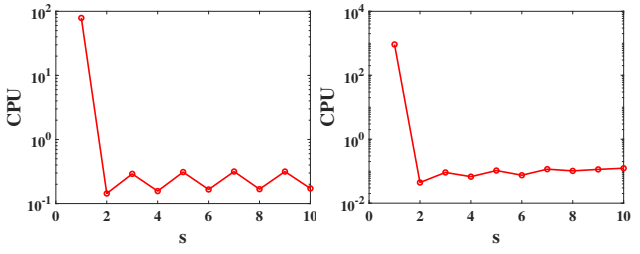


(c) `randn`$(300, 5000)$ (d) `randn`$(1000, 10000)$

Fig. 1: CPU against varying s for `randn` matrices.



(a) `abtaha1` (b) `ash958`



(c) `crew1` (d) `df2177`

Fig. 2: CPU against varying s for real-world matrices.

The coefficient matrix A in the consistent linear system (1) is generated by the MATLAB build-in function `randn`, from some real-world applications [11] or from tomography test problems [5], [6]. Properties of the real-world test matrices, including the matrix name, matrix size, density, condition number, rank and corresponding application fields, are tabulated in Table I. The right-hand vector b is given by $Ax_\star$, where $x_\star \equiv \texttt{pinv}(A) \cdot b$ is derived from `randn`.

This section is split into two parts. In Section IV-A, we look into the issue that how the restarting frequency s affects the performance of FRS. In Section IV-B, we compare FRS with RK and RRS to illustrate its effectiveness.

A. Selection of the restarting frequency s

In the first part, we investigate the impact of s on the performance of the FRS algorithm, from which we give some sound choice of s. To this end, we consider applying FRS to solving linear systems (1) with the coefficient matrix A generated by `randn` or from the SuiteSparse matrix collection [11]. The results are illustrated in Figures 1-2.

In Figure 1, the history of CPU time against varying s are recorded for `randn` matrices. It is easy to check that CPU drops quickly form $s = 1$ to $s = 2$ and fluctuates slightly afterwards. Thus it is reasonable to choose $s = 2$ in implementing FRS. A similar observation is witnessed in Figure 2, where the history of CPU time against varying s are recorded for four real-world matrices. Still the choice $s = 2$ yields the shortest CPU time.

In light of this, we choose $s = 2$ for FRS in the subsequent experiments. As for the restarting frequency in RRS, we pick $s = 5$, 10 and 20, the experimentally appealing choices as suggested in the original work [18].

B. Numerical comparisons among RK, RRS and FRS

In the second part, we compare the numerical performance of FRS with RK and RRS. In Example 1, we consider solving overdetermined or underdetermined linear systems with these three algorithms. In Example 2, we solve the linear systems (1) with coefficient matrices from the real-world applications. In Example 3, we test the efficiency of the proposed algorithm for solving test problems from medical and seismic tomography [5].

Example 1. *In this example, we verify the effectiveness of FRS when solving linear systems with coefficient matrix being Gaussian matrix. The results are depicted in Tables II-III. Some remarks are in order. As for the performance of RRS, we follow the practice done in [18] by adopting three different choices of s, i.e., $s = 5, 10$ and 20. As indicated in Table II, when solving overdetermined linear systems, RRS(20) is superior to RRS(5) and RRS(10) regarding CPU and IT. Furthermore, FRS(2) outperforms RK and RRS(20) for all tests in terms of CPU and IT; the CPU speed-up of FRS over RK ranges from 36.2 to 51.8 while that for RRS varies from 14.8 to 20.2. Similar conclusions can be drawn from Table III where underdetermined linear systems are solved; for instance, the CPU speed-up of FRS over RK can be as large as 44.0. Therefore, the effectiveness of the FRS algorithm in solving linear systems is justified.*

Example 2. *In this example, we assess the numerical performance of FRS for solving linear systems from various real-world applications [11]. The results are tabulated in Tables IV-V. In Table IV, we list CPU and IT for thin ($m > n$) test matrices. In Table V, we display CPU and IT for fat ($m < n$) test matrices. Take Table IV for example. The superiority of the FRS algorithms over the two counterparts is pronounced; the CPU speed-up of FRS over RK can be up to 82.18 while that for FRS reaches 22.89. In Table V, FRS outperforms RK and RRS for most test problems except for the case* `gen2` *where RRS(20) exceeds FRS(2) marginally.*

Example 3. *In this example, we test the FRS algorithm for solving a number of random phantom problems generated from AIR Tools [5], [6]. All algorithms are given a budget of 10 seconds CPU time to reconstruct the images. The*

TABLE I: The properties of the real-world test matrices.

Matrix name	Size	Density	cond(A)	Rank	Application field
abtaha1	14596×209	1.68%	12.23	209	Combinatorial Problem
abtaha2	37932×331	1.09%	12.22	331	Combinatorial Problem
n4c6 − b15	60×920	1.74%	1.10	60	Combinatorial Problem
ash608	608×188	1.06%	3.37	188	Least Squares Problem
ash958	958×292	0.68%	3.20	292	Least Squares Problem
WorldCities	315×100	23.87%	66.00	100	Weighted Bipartite Graph
gen2	1121×3264	2.24%	33.47	1121	Linear Programming Problem
crew1	135×6469	5.38%	18.20	135	Linear Programming Problem
df2177	630×10358	0.34%	2.01	630	Linear Programming Problem
lp_grow22	440×946	1.98%	5.75	440	Linear Programming Problem

TABLE II: Numerical results for randn(5000, n) matrices with different n.

$m \times n$		5000×200	5000×400	5000×600	5000×800	5000×1000
RK	IT	2835.9	5970.8	9532.5	13751.1	18810.3
	CPU	0.2029	0.4172	0.7055	1.0512	1.5191
RRS(5)	IT	879.0	1815.3	2852.2	3978.9	5328.1
	CPU	0.2904	0.6210	1.0029	1.3961	1.9175
RRS(10)	IT	429.6	877.4	1367.5	1894.3	2490.5
	CPU	0.1522	0.3376	0.5196	0.7296	0.9994
RRS(20)	IT	213.6	437.6	676.1	934.8	1225.2
	CPU	0.0830	0.1806	0.2911	0.4227	0.5904
FRS(2)	IT	4.0	5.0	5.0	5.0	6.0
	CPU	0.0056	0.0097	0.0143	0.0213	0.0293
$\text{speed} - \text{up}_{\text{RK}}$		36.2321	43.0103	49.3357	49.3521	51.8464
$\text{speed} - \text{up}_{\text{RRS}}$		14.8214	18.6186	20.3566	19.8451	20.1502

TABLE III: Numerical results for randn(m, 5000) matrices with different m.

$m \times n$		200×5000	400×5000	600×5000	800×5000	1000×5000
RK	IT	2295.2	5361.4	8805.4	13588.6	18522.8
	CPU	0.1151	0.3145	0.6061	1.0372	1.5416
RRS(5)	IT	807.5	1745.0	2738.7	3866.9	5266.8
	CPU	0.1080	0.3338	0.5669	0.9657	1.3984
RRS(10)	IT	391.8	848.6	1307.1	1866.7	2455.8
	CPU	0.0884	0.2312	0.4356	0.7351	1.0063
RRS(20)	IT	198.0	418.1	656.8	913.3	1212.7
	CPU	0.0794	0.2126	0.3751	0.6199	0.8449
FRS(2)	IT	4.0	5.0	5.0	5.0	6.0
	CPU	0.0043	0.0128	0.0183	0.0236	0.0350
$\text{speed} - \text{up}_{\text{RK}}$		26.7674	24.5703	33.1202	43.9492	44.0457
$\text{speed} - \text{up}_{\text{RRS}}$		18.4651	16.6094	20.4973	26.2669	24.1400

results are depicted in Figures 3-6. In Figure 3, we employ the function paralleltomo *in AIR Tools to generate a 2D parallel-beam tomography problem with the coefficient matrix* $A \in \mathbb{R}^{14400\times10000}$. *As illustrated, after running each algorithm for 10 seconds, the FRS algorithm recovers the original image efficiently. In Figure 4, we exploit the function* seismictomo *in AIR Tools to construct a 2D seismic tomography problem with coefficient matrix* $A \in \mathbb{R}^{20000\times10000}$. *As depicted, the FRS Algorithm outperforms the other two peers in restoring the seismic tomography problem. In Figures 5 and 6, we use the function* phantomgallery *in AIR Tools to create two 2D phantoms for use in tomography test problems. One is the* binary *image with binary pixel values arranged in domains (Figure 5) while the other is the* ppower *random image with patterns of nonzero pixels (Figure 6). This leads to a linear system in Figure 5 with* $A \in \mathbb{R}^{4320\times10000}$ *while that in Figure 6 with* $A \in \mathbb{R}^{5400\times10000}$. *As indicated, FRS(2) performs better than the RK and RRS methods in reconstructing the original images.*

V. Conclusions

We have developed a fast deterministic algorithm with restarting to solve large linear equations. We proved the convergence of the proposed algorithm both theoretically and numerically. We regard refining the upper bound presented in Theorem 2 as an important future research.

TABLE IV: Numerical results for thin matrices from real-world applications.

Matrix		abtbha1	abtaha2	ash608	ash958	WorldCities
RK	IT	67480.3	101578.4	4423.4	5895.9	36001.0
	CPU	10.6039	60.2757	0.0651	0.0997	0.4074
RRS(5)	IT	16756.5	24988.5	1212.7	1709.8	8938.1
	CPU	14.7990	61.3154	0.0596	0.1139	0.2727
RRS(10)	IT	7757.7	11432.9	565.4	787.8	3914.2
	CPU	7.1626	29.0235	0.0329	0.0576	0.1420
RRS(20)	IT	3693.3	5387.5	271.0	403.0	1875.4
	CPU	3.7276	14.5719	0.0206	0.0343	0.0896
FRS(2)	IT	68.0	56.0	9.0	7.0	800.0
	CPU	0.1904	0.7335	0.0009	0.0016	0.0212
$\text{speed} - \text{up}_{\text{RK}}$		55.6928	82.1755	72.3333	62.3125	19.2170
$\text{speed} - \text{up}_{\text{RRS}}$		19.5777	19.8663	22.8889	21.4375	4.2264

TABLE V: Numerical results for fat matrices from real-world applications.

$m \times n$		crew1	df2177	gen2	n4c6 − b15	lp_grow22
RK	IT	14378.9	7870.7	90024.0	456.1	31249.8
	CPU	0.8942	0.8561	5.1913	0.0050	0.4713
RRS(5)	IT	3520.6	2755.7	23111.7	213.7	7778.1
	CPU	0.6044	1.1153	4.4339	0.0049	0.3633
RRS(10)	IT	1557.6	1304.9	10341.0	104.0	3484.8
	CPU	0.4317	0.9752	3.0726	0.0036	0.2097
RRS(20)	IT	796.4	639.2	4977.7	53.1	1645.6
	CPU	0.4228	0.9174	2.4186	0.0029	0.1417
FRS(2)	IT	181.0	4.0	878.0	3.0	22.0
	CPU	0.1422	0.0356	3.3291	0.0002	0.0072
$\text{speed} - \text{up}_{\text{RK}}$		6.2883	24.0478	1.5594	25.0000	65.4583
$\text{speed} - \text{up}_{\text{RRS}}$		2.9733	25.7697	0.7265	14.5000	19.6806

REFERENCES

[1] Z.-Z. Bai and L. Wang, "On convergence rates of Kaczmarz-type methods with different selection rules of working rows," *Appl. Numer. Math.*, vol. 186, pp. 289-319, 2023.

[2] Z.-Z. Bai and W.-T. Wu, "On greedy randomized Kaczmarz method for solving large sparse linear systems," *SIAM J. Sci. Comput.*, vol. 40, no. 1, pp. A592-A606, 2018.

[3] Z.-Z. Bai and W.-T. Wu, "Randomized Kaczmarz iteration methods: Algorithmic extensions and convergence theory," *Jpn. J. Ind. Appl. Math.*, vol. 40, pp. 1421-1443, 2023.

[4] Y. Censor, "Row-action methods for huge and sparse systems and their applications," *SIAM Rev.*, vol. 23, no. 4, pp. 444-466, 1981.

[5] P. C. Hansen and M. Saxild-Hansen, "AIR tools-a MATLAB package of algebraic iterative reconstruction methods," *J. Comput. Appl. Math.*, vol. 236, no. 8, pp. 2167-2178, 2012.

[6] P. C. Hansen and J. S. Jørgensen, "AIR Tools II: algebraic iterative reconstruction methods, improved implementation," *Numer. Algor.*, vol. 79, no. 1, pp. 107-137, 2018.

[7] D. Han, Y. Su, and J. Xie, "Randomized Douglas-Rachford methods for linear systems: improved accuracy and efficiency," *SIAM J. Optim.*, vol. 34, no. 1, pp. 1045-1070, 2024.

[8] X.-L. Jiang and K. Zhang, "A randomized block extended Kaczmarz method with hybrid partitions for solving large inconsistent linear systems," *Appl. Math. Lett.*, vol. 152, no. 109027, pp. 1-7, 2024.

[9] X.-L. Jiang, K. Zhang, and J.-F. Yin, "Randomized block Kaczmarz methods with k-means clustering for solving large linear systems," *J. Comput. Appl. Math.*, vol. 403, no. 113828, pp. 1-14, 2022.

[10] S. Kaczmarz, "Angenäherte Auflösung von systemen linearer gleichungen," *Bull. Int. Acad. Pol. Sic. Let. A*, pp. 355-357, 1937.

[11] S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson, Y. Hu, and R. Sandstrom, "The SuiteSparse matrix collection website interface," *J. Open Source Softw.*, vol. 4, no. 1244, 2019.

[12] D. Leventhal and A. S. Lewis, "Randomized methods for linear constraints: convergence rates and conditioning," *Math. Oper. Res.*, vol. 35, no. 3, pp. 641-654, 2010.

[13] Y. Liu and C. Gu, "Two greedy subspace Kaczmarz algorithm for image reconstruction," *IAENG International Journal of Applied Mathematics*, vol. 50, no. 4, pp. 853-859, 2020.

[14] Y. Saad, "Iterative Methods for Sparse Linear Systems," second ed., *SIAM*, Philadelphia, 2003.

[15] C. Shao, "A deterministic Kaczmarz algorithm for solving linear systems," *SIAM J. Matrix Anal. Appl.*, vol. 44, no. 1, pp. 212-239, 2023.

[16] S. Steinerberger, "Surrounding the solution of a linear system of equations from all sides," *Q. Appl. Math.*, vol. 79, no. 3, pp. 419-429, 2021.

[17] T. Strohmer and R. Vershynin, "A randomized Kaczmarz algorithm with exponential convergence," *J. Fourier Anal. Appl.*, vol. 15, pp. 262-278, 2009.

[18] J.-F. Yin, N. Li, and N. Zheng, "Restarted randomized surrounding methods for solving large linear equations," *Appl. Math. Lett.*, vol. 133, no. 108290, pp. 1-9, 2022.

[19] K. Zhang, X.-X. Chen, and X.-L. Jiang, "A residual-based surrogate hyperplane extended Kaczmarz algorithm for large least squares problems," *Calcolo*, vol. 61, no. 51, pp. 1-28, 2024.

[20] K. Zhang, F.-T. Li, and X.-L. Jiang, "Multi-step greedy Kaczmarz algorithms with simple random sampling for solving large linear systems," *Comput. Appl. Math.*, vol. 41, no. 332, pp. 1-25, 2022.

[21] K. Zhang, C.-T. Liu, and X.-L. Jiang, "A greedy randomized block coordinate descent algorithm with k-means clustering for solving large linear least-squares problems," *IAENG International Journal of Computer Science*, vol. 51, no. 5, pp. 511-518, 2024.

[22] A. Zouzias and N. M. Freris, "Randomized extended Kaczmarz for solving least squares," *SIAM J. Matrix Anal. Appl.*, vol. 34, pp. 773-793, 2013.

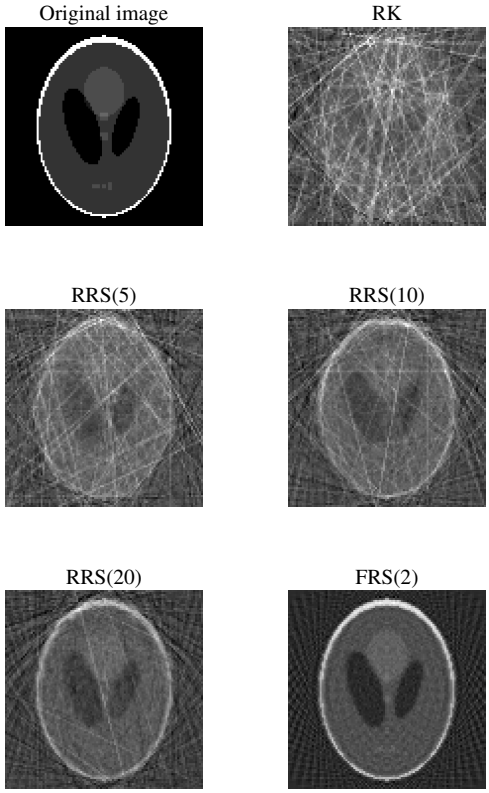


Fig. 3: Numerical results of RK, RRS and FRS for solving the 2D parallel beam tomography test problem.

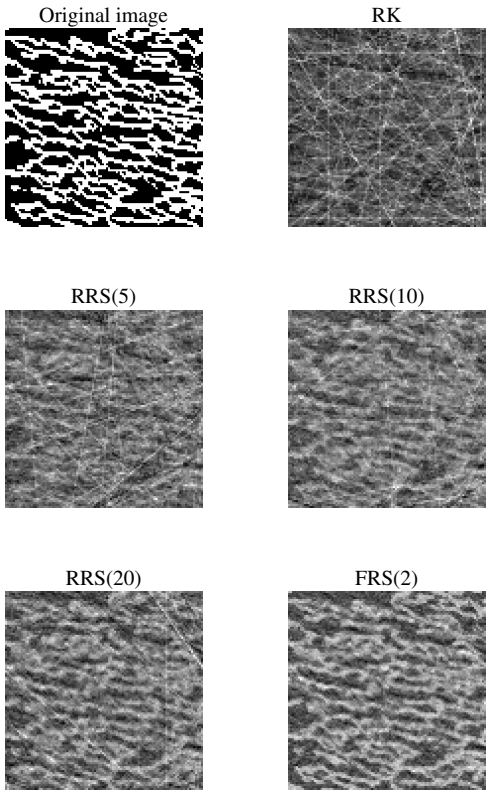


Fig. 5: Numerical results of RK, RRS and FRS for solving the 2D binary phantom problem.

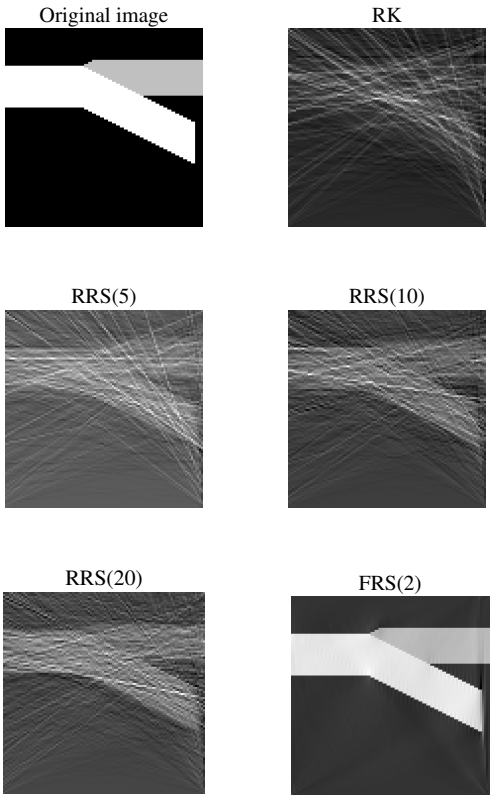


Fig. 4: Numerical results of RK, RRS and FRS for solving the 2D seismic tomography test problem.

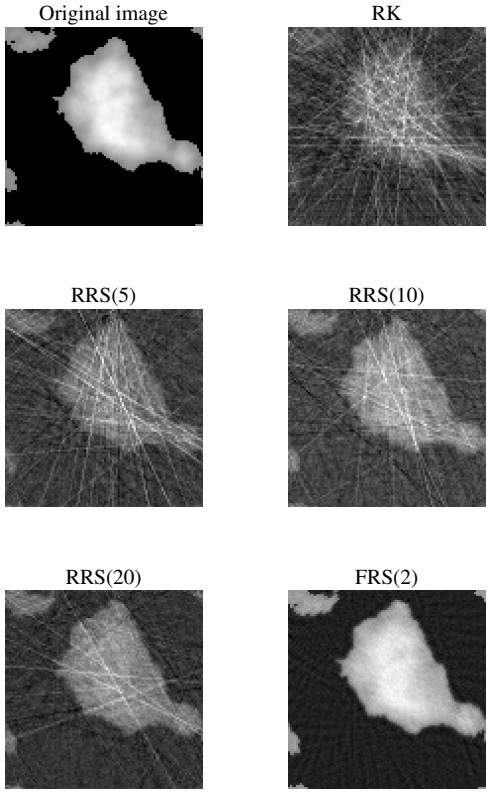


Fig. 6: Numerical results of RK, RRS and FRS for solving the 2D ppower phantom problem.