Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 7464212, 11 pages
https://doi.org/10.1155/2020/7464212

Research Article

On Randomized Sampling Kaczmarz Method with Application in Compressed Sensing

Mei-Lan Sun,[1,2] **Chuan-Qing Gu**,[1] **and Peng-Fei Tang**[1]

[1]*Department of Mathematics, Shanghai University, Shanghai 200444, China*
[2]*Institute of Artificial Intelligence and Big Data, Hefei University, Hefei 230601, China*

Correspondence should be addressed to Chuan-Qing Gu; cqgu@shu.edu.cn

Received 21 December 2019; Accepted 19 February 2020; Published 30 March 2020

Academic Editor: Richard I. Avery

We propose a randomized sampling Kaczmarz algorithm for the solution of very large systems of linear equations by introducing a maximal sampling probability control criterion, which is aimed at grasping the largest entry of the absolute sampling residual vector at each iteration. This new method differs from the greedy randomized Kaczmarz algorithm, which needs not to compute the residual vector of the whole linear system to determine the working rows. Numerical experiments show that the proposed algorithm has the most significant effect when the selected row number, i.e, the size of samples, is equal to the logarithm of all rows. Finally, we extend the randomized sampling Kaczmarz to signal reconstruction problems in compressed sensing. Signal experiments show that the new extended algorithm is more effective than the randomized sparse Kaczmarz method for online compressed sensing.

1. Introduction

In this paper, we mainly consider the iterative solution of large-scale consistent linear systems of the form

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

where $A \in \mathbb{C}^{m\times n}$, $b \in \mathbb{C}^m$, and $x \in \mathbb{C}^n$ denotes the n-dimensional unknown vector. The Kaczmarz algorithm [1] is a typical row-action method [2–5] for solving such a linear system (1) and is known as algebraic reconstruction technique in the field of computerized tomography [6, 7] and image reconstruction [5, 8–10]; see also [11–13] for additional references. More specifically, the $(t+1)$th iterative vector $xt+1$ is generated by the following Kaczmarz updates:

$$x_{t+1} = x_t + \frac{b^{(i_t)} - A^{(i_t)}x_t}{\left\|A^{(i_t)}\right\|_2^2}\left(A^{(i_t)}\right)^*, \quad t = 0, 1, 2, \ldots, \tag{2}$$

where $A^{(i_t)}$ denotes the i_tth row of the matrix A, $(\cdot)*$ denotes the conjugate transpose of the corresponding vector or matrix, $b^{(i_t)}$ denotes the i_tth entry of the vector b, and $i_t = (t \bmod m) + 1$. Thus, the Kaczmarz algorithm is very suitable for the solution of very large linear equations since the main computation in the procedure of this algorithm is an inner product; however, this algorithm sometimes converges very slowly, see [14, 15] and the references therein. In order to improve the convergence of this algorithm, in 2009, Strohmer and Vershynin [16] proposed the randomized Kaczmarz (RK) algorithm with an expected exponential rate of convergence by using the rows of the coefficient matrix A in a random manner rather than in a given order, i.e., the RK chooses row $i_t \in \{1, 2, \ldots, m\}$ with probability

$$\Pr\left(\text{row} = i_t\right) = \frac{\left\|A^{(i_t)}\right\|_2^2}{\|A\|_F^2}. \tag{3}$$

Recently, Bai and Wu proposed a different but more effective probability criterion and, based on it, they constructed the greedy randomized Kaczmarz (GRK) algorithm for solving the linear system (1) in [17]. This probability criterion makes the corresponding GRK algorithm to converge significantly faster than the RK method in both theory and experiments. We refer to [15, 18, 19] for more

details on convergence theory and algorithmic generalizations for the GRK algorithm.

We observe that the most expensive component at the tth iterate of the GRK algorithm is computing the square norm of the residual vector (i.e., $\|b - Ax_t\|_2^2$) of the linear system, and it is the main factor that affects the computing time. Consequently, in this paper, we first randomly select k indices of rows, say, $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots,\}$, of the coefficient matrix of the linear system (1) and then compute the largest absolute relative residual, i.e, $\max(|b^{(i_l)} - A^{(i_l)}x_t|/\|A\|_F)$, $i_l \in \{i_1, i_2, \ldots, i_k\}$ and construct a randomized sampling Kaczmarz (RSK) algorithm. Therefore, our method needs not to calculate the residuals of all rows as the GRK method does and can be expected to be more effective than GRK in terms of the computing time. In theory, we provide a proof that shows linear convergence in expectation to the unique least-norm solution of the consistent linear system (1) for our RSK algorithm. And in computations, we show that the RSK algorithm significantly outperforms the RK algorithm in terms of both iteration counts and computing times and costs much less CPU time than the GRK algorithm but slightly requires more iteration steps than GRK algorithm. In addition, we extend the RSK algorithm to signal reconstruction problems in compressed sensing [20, 21] and establish the corresponding randomized sparse sampling Kaczmarz (RaSSK) algorithm and show that RaSSK performs much better than the randomized sparse Kaczmarz (RaSK) algorithm, which is a special case of the Bregman iterative algorithm [22] for online compressed sensing.

The remainder of the paper is organized as follows. In Section 2, we review the RK and GRK algorithms. In Section 3, we give the RSK algorithm and establish its convergence theory. Numerical experiments and comparisons are shown in Section 4. In Section 5, we extend the RSK to signal reconstruction in compressed sensing and give some signal experiments in this section. Finally, we end this paper with a few remarks and conclusions in Section 6.

2. The RK and GRK Algorithms

In this section, we briefly introduce the RK and GRK algorithms for solving systems of linear equations. First, let us give some necessary explanations. In this paper, we use $\mathbb{E}_t$ to denote the expected value conditional on the first t iterations, that is,

$$\mathbb{E}_t[.] \overset{\text{def}}{=} \mathbb{E}_t[.\,|\, i_1, i_2, \ldots, i_t], \tag{4}$$

where i_l ($l = 1, 2, \ldots, t$) is the i_lth row chosen at the lth iterate. Then, it is easy to get that $\mathbb{E}[\mathbb{E}_t[.]] = \mathbb{E}[.]$ (see, e.g., the work of Bai and Wu [17]).

When the linear system (1) is consistent and its coefficient matrix $A \in \mathbb{C}^{m\times n}$ ($m \geq n$) is of the full-column rank, Strohmer and Vershynin [16] proposed the randomized Kaczmarz algorithm, as given in Algorithm 1.

The RK method is convergent to the unique least-norm solution of the consistent linear system (1) when the coefficient matrix A is of the full-column rank with $m \geq n$ [16], and later in [23], Ma, Needell, and Ramdas gave the same convergence rate of the RK algorithm, but for the case that $m < n$. Compared with the Kaczmarz method, we can find, according to probability criterion (3), that the order in which the RK method selects the action row is based on the probability corresponding to the size of each row norm. It can improve the convergence of the Kaczmarz method discussed in [1].

In [17], Bai and Wu proposed a new probability criterion (i.e., Steps 2–5 of Algorithm 2) for the RK algorithm, and based on it, they constructed a greedy randomized Kaczmarz algorithm, as given in Algorithm 2, which converges faster than the RK algorithm.

According to Algorithm 2, we can find that, at the tth iterate, the corresponding residual vector is $r_t = b - Ax_t$ and, if $|r_t^{(i)}| > |r_t^{(j)}|$, $i, j \in \{1, 2, \ldots, m\}$, then the GRK algorithm is to make the ith row be selected with a larger probability than the jth row. Actually, only some entries of r_t whose relative residual is greater than a threshold are nonzero. Convergence property of the GRK method has been studied by Bai and Wu in [17]. In addition, from [17], we can find that if the GRK method is convergent, then it must converge to the least-norm solution $x_* = A^\dagger b$ of the linear system (1), where $A^\dagger$ indicates the Moore–Penrose pseudoinverse of the matrix A.

3. A Randomized Sampling Kaczmarz Method

3.1. The Randomized Sampling Kaczmarz Method. In this part, a new algorithm will be given. Through the contents of the previous sections, we note that at each iteration of GRK, it will cost a lot of money to calculate the residual vector r_t. If the dimension of the matrix is too large relative to the memory of the computer, the performance of the GRK method will be greatly reduced. In response to this situation, we propose a new method to solve the linear system (1). The purpose of our method is to make full use of the residual information as the GRK method does, while avoiding calculating the residual r_t at each iteration. Let $A = (a_{ij}) \in \mathbb{C}^{m\times n}$ in the linear system (1) with $1 \leq i \leq m$. Our method is carried out as follows: first, according to uniform distribution, randomly select k rows $i_1, i_2, \ldots, i_k$ from the m rows of the coefficient matrix A and put them in a set $V_t = \{i_1, i_2, \ldots, i_k\}$; second, calculate the relative residuals of all corresponding rows in the set V_t; then, select the row with the largest absolute relative residual value (i.e., $\max((|b^{(i_l)} - A^{(i_l)}x_t|)/\|A\|_F)$, $i_l \in \{i_1, i_2, \ldots, i_k\}$) to implement the Kaczmarz update in (2). The RSK algorithm proposed in this paper is expressed in Algorithm 3.

According to Algorithm 3, we can see that the selection of the appropriate number of rows k plays an important role in the RSK algorithm. More specifically, if the number of rows k is too small, it is difficult to ensure the efficiency of selecting iterative action rows in each iteration, and, if the number of rows k is too large, then the RSK method will have higher requirement for the memory of the computer, especially when the matrix dimension is large enough. In the following numerical examples, we will discuss the selection of parameter k in the RSK method.

In fact, according to Algorithms 1–3, the RSK method will cost $(2k+2)\ n+1+k$ flopping operations (flops) at each iteration, while the GRK method and the RK method will cost $7m+2\ (n+1)$ and $4n+1$ flops, respectively [17]. We list the flops

Require: A, b, l, and x_0.
Ensure: x_l.
(1) **for** $t = 0, 1, \ldots, l$ **do**
(2) Select $i_t \in \{0, 1, \ldots, m\}$ with probability $\Pr(\text{row} = i_t) = \|A^{(i_t)}\|_2^2 / \|A\|_F^2$
(3) Set $x_{t+1} = x_t + ((b^{(i_t)} - A^{(i_t)} x_t) / \|A^{(i_t)}\|_2^2)(A^{(i_t)})^*$
(4) **end for**

Algorithm 1: The randomized Kaczmarz (RK) algorithm.

Require: A, b, l, and x_0.
Ensure: x_l.
(1) **for** $t = 0, 1, \ldots, l$ **do**
(2) Compute
$\varepsilon_t = (1/2)((1/(\|b - Ax_t\|_2^2))\max_{1 \le i_t \le m}\left\{(|b^{(i_t)} - A^{(i_t)} x_t|^2)/\|A^{(i_t)}\|_2^2\right\} + 1/\|A\|_F^2)$
(3) Determine the index set of positive integers
$U_t = \left\{i_t \mid |b^{(i_t)} - A^{(i_t)} x_t|^2 \ge \varepsilon_t \|b - Ax_t\|_2^2 \|A^{(i_t)}\|_2^2\right\}$
(4) Compute the ith entry $\widetilde{r}_t^{(i)}$ of the vector $\widetilde{r}_t$ according to
$\widetilde{r}_t^{(i)} = \begin{cases} b^{(i)} - A^{(i)} x_t, & \text{if } i \in U_t, \\ 0, & \text{otherwise} \end{cases}$
(5) Select $i_t \in U_t$ with probability $\Pr(\text{row} = i_t) = |\widetilde{r}_t^{(i)}|^2 / \|\widetilde{r}_t\|_2^2$
(6) Set $x_{t+1} = x_t + ((b^{(i_t)} - A^{(i_t)} x_t) / \|A^{(i_t)}\|_2^2)(A^{(i_t)})^*$
(7) **end for**

Algorithm 2: The greedy randomized Kaczmarz (GRK) algorithm.

Require: A, b, l, k and x_0.
Ensure: x_l.
(1) **for** $t = 0, 1, \ldots, l$ **do**
(2) Randomly generate by uniform distribution k different elements from 1 to m, indicated by $V_t = \{i_1, i_2, \ldots, i_k\}$
(3) Compute the lth entry $r_t^{(l)}$ of the vector r_t according to
$r_t^{(l)} = (|b^{(i_l)} - A^{(i_l)} x_t| / \|A\|_F)$, if $i_l \in V_t$, $l = 1, 2, \ldots, k$
(4) For $l = 1, 2, \ldots, k$, select the row according to $\max r_t^{(l)}$ for $i_l \in V_t$ and assume that the index $i_l = j$
(5) Set $x_{t+1} = x_t + ((b^{(i)} - A^{(j)} x_t) / \|A^{(j)}\|_2^2)(A^{(j)})^*$
(6) **end for**

Algorithm 3: The randomized sampling Kaczmarz (RSK) algorithm.

of these three algorithms at each iteration in Table 1. Obviously, when the row number m is extremely large, the RSK method costs less flops than the GRK method. When $k = 1$, the RSK method degenerates into the uniform sampling RK method.

3.2. Convergence Analysis of the RSK Algorithm. In this section, we pay attention to the convergence property of our algorithm. The following theorem guarantees the convergence of the RSK method.

Theorem 1. *Let $x_* = A^\dagger b$ be the solution of the consistent linear system (1) and $x_0 \in R(A^*)$, where $R(A^*)$ denotes the column space of the matrix A^*. Then, the iteration sequence $\{x_t\}_{t=0}^{\infty}$, generated by the RSK algorithm, converges to the solution x_* in expectation. Moreover, the solution error in expectation for the iteration sequence $\{x_t\}_{t=0}^{\infty}$ obeys*

Table 1: The computational complexity of RK, GRK, and RSK algorithms at each iteration.

Algorithm	RK	GRK	RSK
Flopping operations (flops)	$4n + 1$	$7m + 2(n + 1)$	$(2k + 2)n + 1 + k$

$$\mathbb{E}\left\|x_{t+1} - x_*\right\|_2^2 \le \left(1 - \frac{\lambda_{\min}(A^*A)}{m^2 \times \max_j \|A^{(j)}\|_2^2}\right)\mathbb{E}\left\|x_t - x_*\right\|_2^2,$$
$$\mathbb{E}\left\|x_t - x_*\right\|_2^2 \le \left(1 - \frac{\lambda_{\min}(A^*A)}{m^2 \times \max_j \|A^{(j)}\|_2^2}\right)^t \left\|x_0 - x_*\right\|_2^2, \quad t = 0, 1, 2, \ldots, \tag{5}$$

where $\lambda_{\min}(A^*A)$ is the smallest positive eigenvalue of A^*A.

Proof. Update rule of the RSK algorithm yields

$$x_{t+1} - x_t = \frac{b^{(i_t)} - A^{(i_t)} x_t}{\left\| A^{(i_t)} \right\|_2^2} \left(A^{(i_t)} \right)^*, \tag{6}$$

which indicates that $x_{t+1} - x_t$ is parallel to $(A^{(i_t)})^*$. Using $A^{(i_t)} x_* = b^{(i_t)}$, we compute

$$\begin{aligned} A^{(i_t)} (x_{t+1} - x_*) &= A^{(i_t)} \left(x_t - x_* + \frac{b^{(i_t)} - A^{(i_t)} x_t}{\left\| A^{(i_t)} \right\|_2^2} \left(A^{(i_t)} \right)^* \right) \\ &= A^{(i_t)} (x_t - x_*) + \left(b^{(i_t)} - A^{(i_t)} x_t \right) \\ &= b^{(i_t)} - A^{(i_t)} x_* = 0. \end{aligned} \tag{7}$$

In other words, $x_{t+1} - x_*$ is orthogonal to $A^{(i_t)}$. Consequently, by the Pythagorean theorem, we obtain

$$\left\| x_{t+1} - x_* \right\|_2^2 = \left\| x_t - x_* \right\|_2^2 - \left\| x_{t+1} - x_t \right\|_2^2. \tag{8}$$

Based on this equality, we have

$$\begin{aligned} \mathbb{E}_t \left\| x_{t+1} - x_* \right\|_2^2 &= \left\| x_t - x_* \right\|_2^2 - \mathbb{E}_t \left\| x_{t+1} - x_t \right\|_2^2 \\ &= \left\| x_t - x_* \right\|_2^2 - \mathbb{E}_t \left\| \frac{b^{(j)} - A^{(j)} x_t}{\left\| A^{(j)} \right\|_2^2} \left(A^{(j)} \right)^* \right\|_2^2 \\ &= \left\| x_t - x_* \right\|_2^2 - \mathbb{E}_t \frac{\left| b^{(j)} - A^{(j)} x_t \right|^2}{\left\| A^{(j)} \right\|_2^2}. \end{aligned} \tag{9}$$

Note that the population mean can be expressed as the expectation of the sample mean, thus,

$$\begin{aligned} \mathbb{E}_t \frac{\left| b^{(j)} - A^{(j)} x_t \right|^2}{\left\| A^{(j)} \right\|_2} &= \frac{1}{k} \mathbb{E}_t \frac{\left| k \left(b^{(j)} - A^{(j)} x_t \right) \right|^2}{\left\| A^{(j)} \right\|_2} \\ &\geq \frac{1}{k} \mathbb{E}_t \frac{\left| \sum_{i=i_1}^{i_t} \left(b^{(i)} - A^{(i)} x_t \right) \right|^2}{\left\| A^{(i)} \right\|_2} \\ &= \frac{1}{m} \frac{\left| \sum_{i=1}^{m} \left(b^{(i)} - A^{(i)} x_t \right) \right|^2}{\left\| A^{(i)} \right\|_2}. \end{aligned} \tag{10}$$

So, we upper bound

$$\begin{aligned} \mathbb{E}_t \left\| x_{t+1} - x_* \right\|_2^2 &\leq \left\| x_t - x_* \right\|_2^2 - \frac{1}{m^2} \sum_{i=1}^{m} \frac{\left| b^{(i)} - A^{(i)} x_t \right|^2}{\max_j \left\| A^{(j)} \right\|_2^2} \\ &= \left\| x_t - x_* \right\|_2^2 - \frac{1}{m^2} \frac{\left\| A (x_t - x_*) \right\|_2^2}{\max_j \left\| A^{(j)} \right\|_2^2} \\ &= \left\| x_t - x_* \right\|_2^2 \left(1 - \frac{1}{m^2} \frac{\left\| A (x_t - x_*) \right\|_2^2}{\max_j \left\| A^{(j)} \right\|_2^2 \left\| x_t - x_* \right\|_2^2} \right). \end{aligned} \tag{11}$$

By the assumption that $x_0 \in R(A^*)$, we have from the RSK algorithm that x_t also does for each t. Since $x_* = A^\dagger b \in R(A^*)$, we obtain $x_t - x_* \in R(A_*)$, which implies that

$$\left\| A (x_t - x_*) \right\|_2^2 \geq \lambda_{\min}(A^* A) \left\| x_t - x_* \right\|_2^2. \tag{12}$$

Hence, it holds that

$$\mathbb{E}_t \left\| x_{t+1} - x_* \right\|_2^2 \leq \left(1 - \frac{\lambda_{\min} (A^* A)}{m^2 \times \max_j \left\| A^{(j)} \right\|_2^2} \right) \left\| x_t - x_* \right\|_2^2. \tag{13}$$

In addition, by taking full expectation on both sides of (13) and using the law of iterated expectation $\mathbb{E}[\mathbb{E}_k[\cdot]] = \mathbb{E}[\cdot]$, we immediately get

$$\mathbb{E} \left\| x_{t+1} - x_* \right\|_2^2 \leq \left(1 - \frac{\lambda_{\min} (A^* A)}{m^2 \times \max_j \left\| A^{(j)} \right\|_2^2} \right) \mathbb{E} \left\| x_t - x_* \right\|_2^2. \tag{14}$$

It follows from induction on the iteration index t, and we finally obtain that

$$\mathbb{E} \left\| x_t - x_* \right\|_2^2 < \left(1 - \frac{\lambda_{\min} (A^* A)}{m^2 \times \max_j \left\| A^{(j)} \right\|_2^2} \right)^t \left\| x_0 - x_* \right\|_2^2. \tag{15}$$

□

Remark 1. Actually, compared with the GRK algorithm in [17], the RSK algorithm cannot improve the convergence for the number of iterations for the linear system (1). However, when the row number m of the coefficient matrix A is much larger than k, in each iteration of the GRK algorithm, we must calculate the residuals of all m rows, while in the RSK algorithm, we just calculate the residual of k rows. This explains how and why the RSK algorithm uses less calculation time to achieve convergence accuracy when compared with the RK and GRK algorithms.

3.2.1. Sensitivity Analysis. The essence of the RaSK algorithm is to solve the least square problem. Consider least square problem (16), given matrix $A \in R^{m \times n}$, observation vector $b \in R^m$, and $x \in R^n$ is the vector to be solved and satisfies

$$\min \| Ax - b \|^2. \tag{16}$$

Let A be accurate and vector b be the measured data, and then we consider the sensitivity of the solution x of problem (16) with respect to each component of data b_j. Let the sensitivity of x with respect to b_j be

$$S_{ij} = \frac{\partial x_i}{\partial b_j}. \tag{17}$$

According to articles [24, 25], we have the following conclusions.

Theorem 2. *In the least square problem (16), the sensitivity of x with respect to b_j is*

$$S_{ij} = \frac{\partial x_i}{\partial b_j} = \sum_{k=1}^{n} \frac{u_{ik}}{\lambda_k} \left(\sum_{\substack{l=1 \\ i \neq k}}^{n} v_{lk} b_l + v_{jk} \right), \tag{18}$$

where u_{ik} and v_{lk} are the elements of orthogonal matrix U and V, respectively, and it satisfies singular value decomposition formula $A = V\Lambda U^T$ with $\Lambda = diag\,(\lambda_1, \lambda_2, \ldots)$ being the singular value of matrix A.

Proof. Without losing generality, let rank $(A) = n(m \geq n)$, and according to the singular value decomposition formula of the matrix, we have

$$A = V\Lambda U^T, \tag{19}$$

where the orthogonal matrix $V \in R^{m \times m}$, $U \in R^{n \times n}$ and $\Lambda \in R^{m \times n}$.

Let $\overline{\Lambda} = \mathrm{diag}\,(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, then we have

$$\Lambda = \begin{pmatrix} \overline{\Lambda} \\ 0 \end{pmatrix}. \tag{20}$$

For any x and b, let

$$\begin{aligned} \alpha &= U^T x, \\ \beta &= V^T b, \end{aligned} \tag{21}$$

then we can get

$$Ax - b = V\lambda U^T U\alpha - V\beta = V\left[\begin{pmatrix} \overline{\Lambda} \\ 0 \end{pmatrix} \alpha - \beta \right]. \tag{22}$$

Let $\beta = (\overline{\beta}_1 / \overline{\beta}_2)$, where $\overline{\beta}_1 \in R^n$ and $\overline{\beta}_2 \in R^{m-n}$. We have

$$\begin{aligned} Ax - b &= V \begin{pmatrix} \overline{\Lambda}\alpha - \overline{\beta}_1 \\ -\overline{\beta}_2 \end{pmatrix}, \\ \|(Ax - b)\|_2^2 &= \left\| \begin{pmatrix} \overline{\Lambda}\alpha - \overline{\beta}_1 \\ -\overline{\beta}_2 \end{pmatrix} \right\|_2^2 = \left\| \overline{\Lambda}\alpha - \overline{\beta}_1 \right\|_2^2 + \left\| \overline{\beta}_2 \right\|_2^2. \end{aligned} \tag{23}$$

So x should satisfy

$$\begin{aligned} \overline{\Lambda}\alpha &= \overline{\beta}_1, \\ \overline{\Lambda} U^T x &= \overline{\beta}_1. \end{aligned} \tag{24}$$

Thus, we can get

$$\begin{aligned} x &= \left(\overline{\Lambda} U^T \right)^{-1} \overline{\beta}_1 = U^{-T} \overline{\Lambda}^{-1} \overline{\beta}_1 = U \overline{\Lambda}^{-1} \overline{\beta}_1, \\ x_i &= \sum_{k=1}^{n} \frac{1}{\lambda_k} u_{ik} \beta_k. \end{aligned} \tag{25}$$

According to $\beta = V^T b$, we have

$$\beta_k = \sum_{l=1}^{n} v_{lk} b_l. \tag{26}$$

Thus, we can get

$$x_i = \sum_{k=1}^{n} \frac{1}{\lambda_k} u_{ik} \sum_{l=1}^{n} v_{lk} b_l = \sum_{k=1}^{n} \sum_{l=1}^{n} \frac{u_{ik} v_k b_l}{\lambda_k} = \sum_{l=1}^{n} \sum_{k=1}^{n} \frac{u_{ik} v_{lk}}{\lambda_k} b_l, \tag{27}$$

So we have

$$\frac{\partial x_i}{\partial b_j} = \sum_{k=1}^{n} \frac{u_{ik}}{\lambda_k} \left(\sum_{\substack{l=1 \\ l \neq k}}^{n} v_{lk} b_l + v_{jk} \right). \tag{28}$$

In fact, S_{ij} reflects the rate of change of x with respect to data b_j, which depends on the singular value of matrix A and its decomposition orthogonal matrix U, V. In particular, when $m = n$, the least square problem is reduced to solve the system of equations. □

4. Numerical Experiments

4.1. Algorithm Comparison. In this section, we use RK, GRK, and RSK algorithms to solve equations (1) with different matrices A. The numerical behaviors of these algorithms are tested and evaluated in terms of the computing time in seconds (denoted as "CPU") and the number of iteration steps (denoted as "IT"), and the CPU and IT mean the medians of the CPU time and iteration steps with respect to 50 times repeated runs of the corresponding method. In addition, we also report the speed-up of RSK (k) (k represents the number of rows selected from the coefficient matrix A in the RSK algorithm) against GRK, which is defined as

$$\text{speed} - \text{up} = \frac{\text{CPU}_{\text{GRK}}}{\text{CPU}_{\text{RSK}(k)}}. \tag{29}$$

All algorithms started from the initial vector $x_0 = \mathbf{0}$ and terminated once the relative solution error (RSE), defined as

$$\text{RSE} = \frac{\left\| x_t - x_* \right\|_2^2}{\left\| x_* \right\|_2^2}, \tag{30}$$

at the current iterate x_t, satisfies RSE $< 10^{-6}$, or the number of iteration steps exceeds 200,000. The latter is given a label "–" in the numerical tables. For part of the tested matrices, we give their Euclidean condition number, denoted by cond(A), and their density is defined as

$$\text{density} = \frac{\text{number of nonzeros of an } m \times n \text{ matrix}}{m \times n}. \tag{31}$$

All experiments are carried out using MATLAB (R2015b) on a personal laptop with 2.5 GHz (Intel(R) Core (TM) CPU i5-7300HQ), 8.00 GB memory, and Windows operating system (Windows 10).

Example 1. In this example, test matrices A are randomly generated by the MATLAB function *randn*, which produces independent entries subject to the standard normal distribution N (0,1). In our implementations, the random vectors $\tilde{x} \in \mathbb{R}^n$ is randomly generated by using the MATLAB

Table 2: IT and CPU of RK, GRK, and RSK (k) for $m \times n$ matrices A with $m = 100$ and 200 and different n.

$m \times n$		100×1000	100×3000	100×5000	200×1000	200×3000	200×5000
RK	IT	1127.3	906.0	825.6	3109.3	2137	2049.2
	CPU	0.1335	0.1572	0.1907	0.3740	0.4142	0.5292
GRK	IT	319.4	232.1	212.5	932.6	552.6	481.4
	CPU	0.0496	0.0626	0.0896	0.1574	0.2127	0.4336
RSK (2)	IT	739.9	595.4	548.0	2005.4	1374.4	1257.8
	CPU	0.0396	0.0615	0.1043	0.1092	0.1907	0.3539
Speed-up		1.25	1.02	0.86	1.44	1.12	1.23
RSK (7)	IT	369.7	275.6	254.9	1060.8	649.2	583.8
	CPU	0.0259	0.0456	0.0733	0.0757	0.1364	0.2564
Speed-up		1.92	1.37	**1.22**	**2.08**	1.56	1.69

Table 3: IT and CPU of RK, GRK, and RSK (k) for $m \times n$ matrices A with $n = 100$ and 200 and different m.

$m \times n$		1000×100	3000×100	5000×100	1000×200	3000×200	5000×200
RK	IT	1439.2	1329.5	1284.3	3550.4	2794.0	2655.7
	CPU	0.1760	0.2407	0.3204	0.4416	0.5147	0.6745
GRK	IT	223.8	175.1	161.9	608.5	375.0	335.9
	CPU	0.0377	0.0537	0.0809	0.1101	0.1553	0.2692
RSK (2)	IT	928.2	860.6	844.0	2246.0	1799.9	1733.7
	CPU	0.0320	0.0365	0.0437	0.0803	0.0863	0.1057
Speed-up		1.18	1.47	1.85	1.37	1.80	2.55
RSK (10)	IT	399.6	372.3	368.5	978.1	773.8	753.5
	CPU	0.0157	0.0231	0.0308	0.0460	0.0586	0.0771
Speed-up		2.40	**2.32**	2.63	2.39	2.65	**3.49**

Table 4: IT and CPU of RK, GRK, and RSK (k) for $m \times n$ matrices A with different m and n.

Name		refine	cage5	*bibd*−16−8	WorldCities	flower−5−1	relat6	football
$m \times n$		29×62	37×37	120×12870	315×100	211×201	2340×157	35×35
Density		8.51%	17.02%	23.33%	23.87%	1.42%	2.21%	9.63%
Cond(A)		66.67	15.42	9.54	66.00	Inf	Inf	Inf
RK	IT	25963.3	10046.1	3058.5	24641	43611	9027.0	—
	CPU	2.6477	1.0146	2.3381	2.5843	4.5635	1.4171	—
GRK	IT	689.1	556.1	1060.9	8925.8	9034.8	1672.9	102331
	CPU	0.0858	0.0676	1.8835	1.3617	1.2300	0.4332	11.8878
RSK (2)	IT	1046.3	1173.5	1932.1	19249.1	17504.2	7578.5	189040.5
	CPU	0.0311	0.0405	1.1868	0.5917	0.5764	0.2641	5.1743
Speed-up		**2.76**	1.67	**1.59**	2.30	2.13	1.64	2.30
RSK (5)	IT	632.4	614.3	1182.9	11435.0	11157.3	3808.3	127467.4
	CPU	0.0193	0.0259	1.0222	0.3645	0.3762	0.1535	3.4984
Speed-up		**4.45**	2.61	**1.84**	3.74	3.27	2.82	3.40

function *randn*, and $b \in \mathbb{R}^m$ is taken to be $A\tilde{x}$; in addition, the solution vectors $x_* \in \mathbb{R}^n$ is taken to be $pinv(A)b$.

In Tables 2 and 3, we report iteration counts and CPU times for RK, GRK, RSK (2), and RSK (7) when the linear system (1) is consistent. As the results in Table 2 show that the RSK (7) vastly outperform both the RK and GRK in terms of CPU times with significant speed-ups when the corresponding linear system is fat (i.e., $m < n$) and the minimum speed-up is 1.22 and the maximum reaches 2.08. From Table 3, we see that the CPU times of RSK (10) are considerably less than those of RK and GRK when the corresponding linear system is thin (i.e., $m > n$), with the speed-up being at least 2.32 and at most attaining 3.49. In addition, we can find that the "fatter" the matrix is, the RSK algorithm shows less advantages, and the "thinner" the matrix is, the RSK algorithm shows more advantages. It is in line with our intuition because if the row number m is extremely large, the RSK algorithm can reduce more computational complexity, for the RSK algorithm is independent of m while the GRK algorithm is not.

Example 2. In this example, we select full-rank sparse matrices from [26], which originate in different applications such as linear programming, combinatorial optimization, DNA electrophoresis model, and Pajek or world city network. They possess certain structures and properties, such as square ($m = n$) (e.g., cage5), thin ($m > n$) (e.g., WorldCities), or fat ($m < n$) (e.g., *bibd*_16_8, and refine). There are also rank-deficient sparse matrices from [26], which come from applications like combinatorial problem and Pajek or world

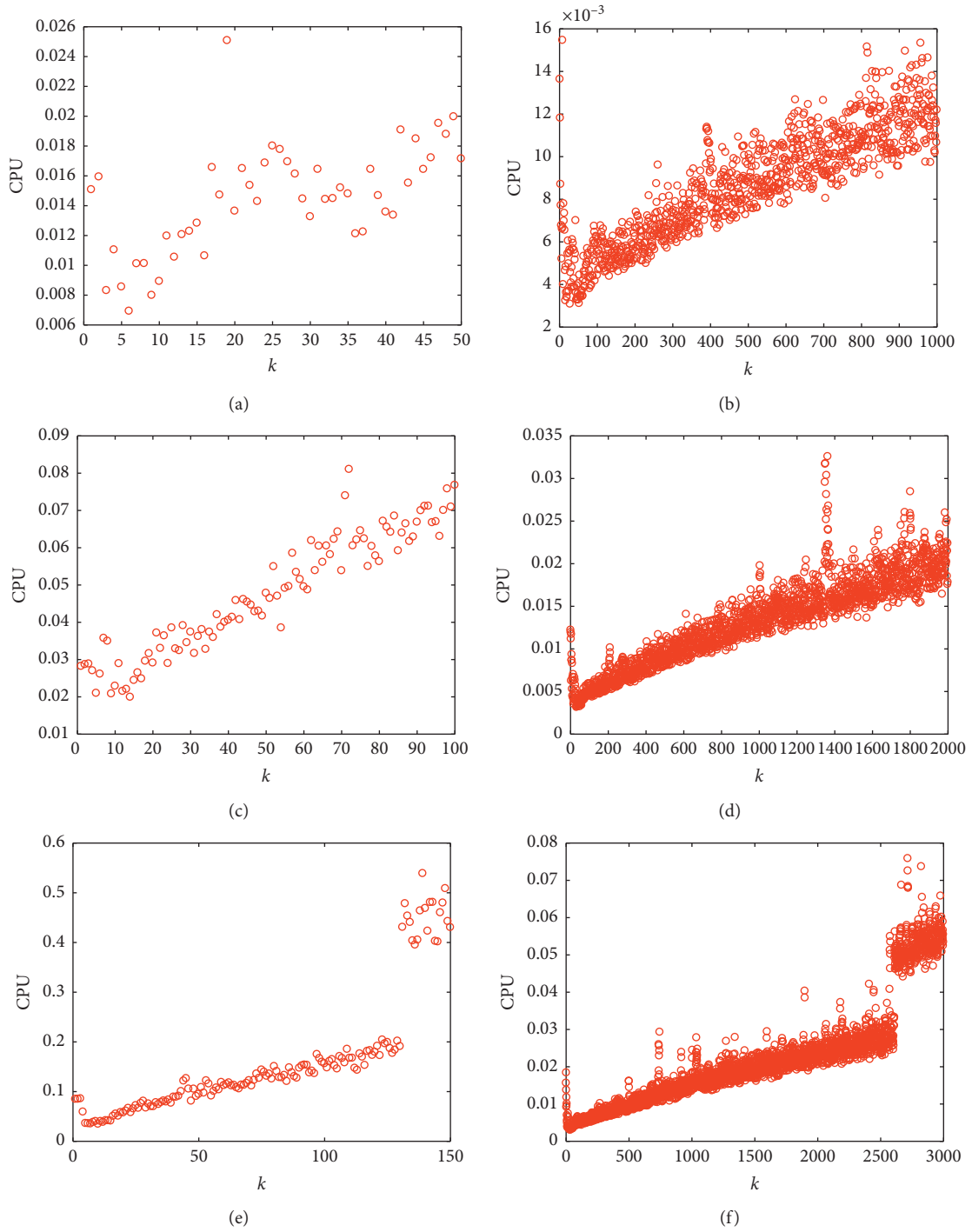


FIGURE 1: Pictures of CPU for RSK versus k for a random matrix with different m and n. (a) Simulation results for $m = 50$, $n = 1000$, and $\log_2$ (m)≈5.6. (b) Simulation results for $m = 1000$, $n = 50$, and $\log_2$ (m)≈10. (c) Simulation results for $m = 100$, $n = 1000$, and $\log_2$ (m)≈6.6. (d) Simulation results for $m = 2000$, $n = 0$, and $\log_2$ (m)≈11. (e) Simulation results for $m = 150$, $n = 1000$, and $\log_2$ (m)≈7.2. (f) Simulation results for $m = 3000$, $n = 50$, and $\log_2$ (m)≈12.

soccer network, such as square (e.g., football) and thin (e.g., relat6 and flower_5_1).

In Table 4, we list the numbers of iteration steps and the computing times for RK, GRK, RSK (2), and RSK (5) algorithms. According to the test results in Table 4, we can see that the RSK algorithm performs better than the GRK algorithm in terms of CPU, even if the matrix is square or not, full-rank or not, and sparse or not and the condition number is infinite or not. More specifically, in terms of the RSK (2) algorithm, the speed-up is at least 1.59 and the biggest is 2.76; and in terms of the RSK (5) algorithm, the speed-up is at least 1.84 and the biggest even attains 4.45.

Require: x_0, A, b, and l.
Ensure: x_l.
(1) **for** $t = 0, 1, \ldots, l$ **do**
(2) Generate i_t randomly by $\Pr(\text{row} = i_t) = \|A^{(i_t)}\|_2^2 / \|A\|_F^2$
(3) Set $x_{t+1/2} = x_t + ((b^{(i_t)} - A^{(i_t)} x_t) / \|A^{(i_t)}\|_2^2)(A^{(i_t)})^*$
(4) Set $x_{t+1} = S\lambda(x_{t+1/2})$
(5) **end for**

Algorithm 4: The randomized sparse Kaczmarz (RaSK) algorithm [22].

Require: x_0, A, b, l, k.
Ensure: x_l.
(1) **for** $t = 0, 1, \ldots, l$ **do**
(2) Randomly generate by uniform distribution k different elements from 1 to m, indicated by $V_t = \{i_1, i_2, \ldots, i_k\}$
(3) Compute the lth entry $r_t^{(l)}$ of the vector rt according to
$r_t^{(l)} = ((|b^{(i_l)} - A^{(i_l)} x_t|) / \|A\|_F)$, if $i_l \in V_t$, $l = 1, 2, \ldots, k$
(4) For $l = 1, 2, \ldots, k$, select the row according to $\max r_t^{(l)}$ for $il \in Vt$ and assume that the index $il = j$
(5) Set $x_{t+1/2} = x_t + ((b^{(j)} - A^{(j)} x_t) / \|A^{(j)}\|_2^2)(A^{(j)})^*$
(6) Set $x_{t+1} = S\lambda(x_{t+1/2})$
(7) **end for**

Algorithm 5: The randomized sampling sparse Kaczmarz (RaSSK) algorithm.

4.2. The Choice of Parameter k. There is no doubt that the value of parameter k makes a huge difference in the performance of the RSK algorithm. In this section, we will have a tentative discussion on k and the matrix row number m. We simulate the relationship between the CPU of the RSK algorithm (with RSE $< 10^{-6}$) and the size of k under different A. The simulation results are shown in Figure 1.

Through a large number of our numerical experiments, we find that $k = [\log_2(m)]$ may be a good choice.

5. Application of RSK Algorithm in Compressed Sensing

5.1. The RaSSK Algorithm. Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{m \times n}$, $m < n$, x is a sparse n dimension vector, and b is an m-dimension vector. By solving the following regularized basis pursuit problem

$$\min_{x \in \mathbb{R}^n} \quad f(x) = \lambda \|x\|_1 + \frac{1}{2}\|x\|_2^2, \qquad \text{s.t.} \quad Ax = b, \tag{32}$$

we can find that the least Euclidean norm solution satisfies the sparsity requirement. In 2014, Lorenz et al. [22] proposed the RaSK algorithm for solving the regularized basis pursuit problem as given in Algorithm 4).

Shrink function $S_\lambda(x) = \max\{|x| - \lambda, 0\} \cdot \operatorname{sign}(x)$, where $\lambda(>0)$ is set according to different applications and is to control the sparsity of the solution. Lorenz et al. proved that, for a consistent linear system $Ax = b$, the RaSK algorithm converges to the unique solution of the regularized basis pursuit problem in [22].

Similar to the construction method of the RaSK algorithm, we can give the randomized sampling sparse Kaczmarz (RaSSK) algorithm which is listed in Algorithm 5.

The proof of the convergence of RaSSK is similar to RaSK. It can be seen as a special case of the Bregman projections for split feasible problems (BPSFP) algorithm in [22]; if we change its feasibility question to "Find $x \in \cap_{k=1}^m A^k$" (i.e., the hyperplane formed by the rows of the matrix A) and define $f(x) = \lambda\|x\|_1 + (1/2)|x\|_2^2$, then, the convergence can be easily obtained by Theorem 2.8 in [22].

5.2. Signal Experiments. In this section, we will show the efficiency of the RaSSK method by several numerical examples and compare it with the RaSK algorithm. We implement the numerical experiments, by MATLAB (R2015b) on a personal laptop with 2.5 GHz (Intel(R) Core(TM) CPU i5-7300HQ), 8.00 GB memory, and Windows operating system (Windows 10).

Example 3. In this example, the test signal is randomly generated with length 256 and limit its sparsity to 10, that is, only 10 nonzero coefficients. The signal is reconstructed by RaSK and RaSSK, respectively. The recovery signal map, relative error map, and relative residual map are given in Figure 2. As shown in this figure, the parameter λ in the RaSSK method and the RaSK method is 50. We can see that both algorithms can reconstruct the original signal. It is worth mentioning that the RaSSK method is more efficient compared to the RaSK method.

Example 4. In this example, we use the famous image lena with 64×64 and 128×128 pixels to test our algorithm. The

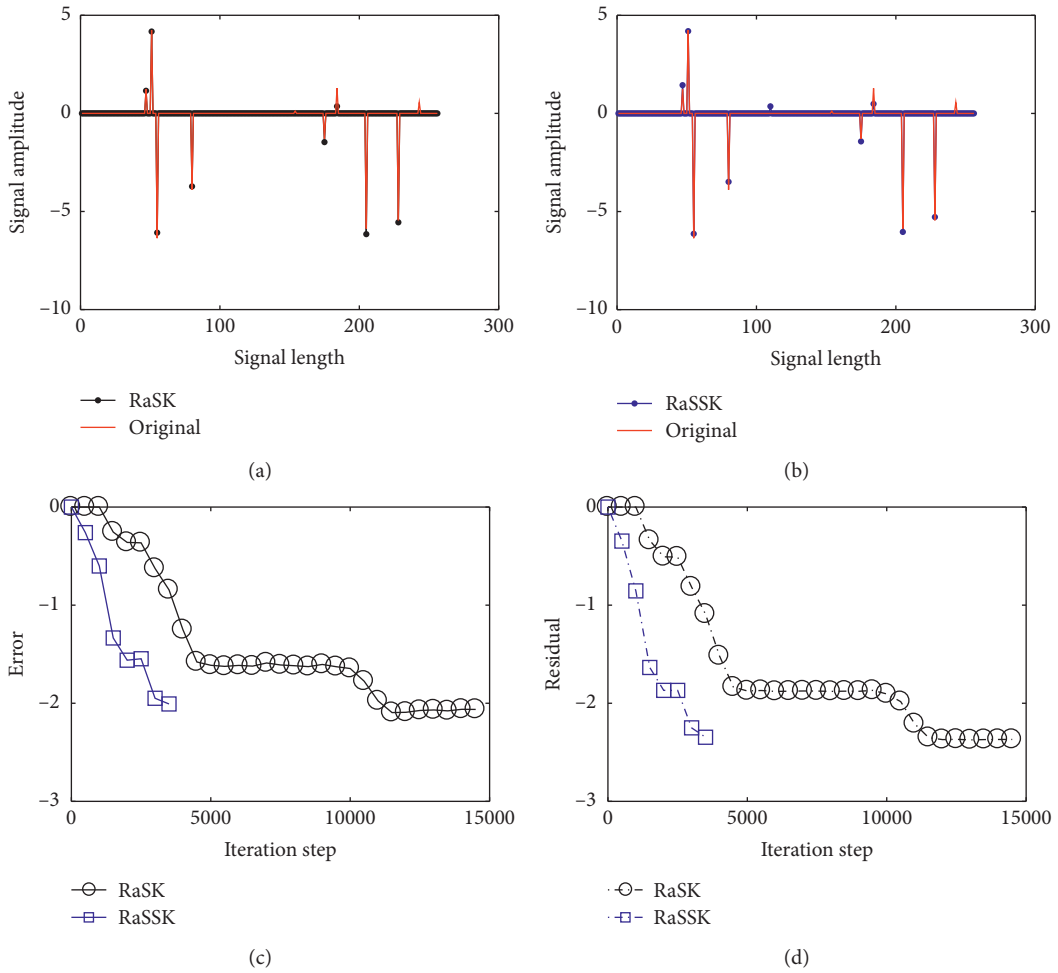


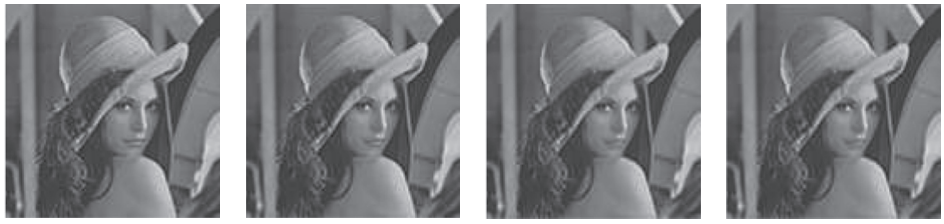
Figure 2: Recovery map, relative error map, and relative residual map in signal reconstruction by RaSK and RaSSK algorithms. (a) Signal recovery map by RaSK. (b) Signal recovery map by RaSSK. (c) Relative error $\|x - x\,\text{real}\|^2/\|x\,\text{real}\|^2$. (d) Relative error $\|Ax - b\|^2/\|b\|^2$.

Table 5: Numerical results of RaSK and RaSSK for the 64×64 and 128×128 image to be reconstructed.

64×64	RaSK	RaSSK	128×128	RaSK	RaSSK
Observation matrix size $m \times n$	1000×2051	1000×2051	Observation matrix size$m \times n$	3000×8195	3000×8195
Original image sparsity	25.03%	25.03%	Original image sparsity	50.02%	50.02%
Sparse image sparsity	6.63%	6.63%	Sparse image sparsity	2.66%	2.66%
λ	50	50	λ	50	50
Iteration step	21483	47	Iteration step	50000	723
CPU	32.1677	0.0743	CPU	704.1457	10.6620

(a) (b) (c) (d)

Figure 3: The RaSK and RaSSK algorithms for a 64×64 reconstructed image. (a) is the original image, (b) is the image after sparse representation, (c) is the image obtained by the RaSK algorithm, and (d) is the image obtained by the RaSSK algorithm.



(a) (b) (c) (d)

Figure 4: The RaSK and RaSSK algorithms for a 128×128 reconstructed image. (a) is the original image, (b) is the image after sparse representation, (c) is the image obtained by the RaSK algorithm, and (d) is the image obtained by the RaSSK algorithm.

MATLAB function *randn* generates an $m \times n$ random matrix which is independently distributed in the standard normal distribution $N(0,1)$ as the observation matrix. For the sparse representation of the image, we use MATLAB function *dwt*, which is a wavelet transform process. The error is defined by

$$\frac{\|x - \text{real}_x\|_2^2}{\|\text{real}_x\|_2^2}, \tag{33}$$

and the stopping criterion of the algorithm is error < 0.1 or reaches the maximum number of iteration steps 50,000. The value of the parameter λ and the results are shown in Table 5. In this table, "CPU" denotes the computing time and "IT" denotes the number of iteration steps.

From Table 5, it is easy to obtain that the RaSSK algorithm significantly performs better than the RaSK algorithm. It greatly reduces the number of iteration steps and the computing time. Meanwhile, from Figures 3 and 4, we can find that for human beings, there is almost no perception loss whenever some information is discarded.

6. Conclusion

Variants of the RK algorithm are effective iteration methods for large-scale linear systems. In this paper, based on the randomized Kaczmarz algorithm and the greedy randomized Kaczmarz algorithm, we propose a new algorithm which makes use of the residual information, while it need not calculate all the residuals. This algorithm converges faster than RK [16] and GRK [17] in experiments. Furthermore, after a large amount of numerical experiments, we recommend the parameter k to take $[\log 2(m)]$. As an application, we apply it to the signal reconstruction in compressed sensing. The experiments show that our algorithm has a good performance.

Data Availability

The table and figure data used to support the findings of this study are included within the article. The software code data used to support the findings of this study are available from the corresponding author upon request. We respect and implement the relevant provisions of Mathematical Problems in Engineering, and our article implements

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was funded by the National Natural Science Foundation of China, under Grant no. 11371243, Key Disciplines of Shanghai Municipality, under Grant no. S30104, Fostering Master's Degree Empowerment Point Project of Hefei University, under Grant no. 2018xs03, Key Natural Science Research Project of University of Anhui Province, Education Department of Anhui Province, under Grant nos. KJ2019A0846 and KJ2017A547, and Innovation Program of Shanghai Municipal Education Commission (13ZZ068).

References

[1] S. Kaczmarz, "Angenäherte auflösung von systemen linearer gleichungen," *Bulletin International de l'Académie Polonaise des Sciences et des Lettres A*, vol. 35, pp. 355–357, 1937.

[2] C. Brezinski, *Projection Methods for Systems of Equations*, North-Holland Publishing, Amsterdam, Netherlands, 1997.

[3] C. L. Byrne, *Applied Iterative Methods*, A K Peters, Wellesley, MA, USA, 2008.

[4] R. Ansorge, "Connections between the Cimmino-method and the Kaczmarz-method for the solution of singular and regular systems of equations," *Computing*, vol. 33, no. 3-4, pp. 367–375, 1984.

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

[6] Y. Censor, "Parallel application of block-iterative methods in medical imaging and radiation therapy," *Mathematical Programming*, vol. 42, no. 1–3, pp. 307–325, 1988.

[7] F. Natterer, *The Mathematics of Computerized Tomography*, SIAM, Philadelphia, PA, USA, 2001.

[8] P. C. Hansen and M. Saxild-Hansen, "AIR Tools—a MATLAB package of algebraic iterative reconstruction methods," *Journal of Computational and Applied Mathematics*, vol. 236, no. 8, pp. 2167–2178, 2012.

[9] P. P. B. Eggermont, G. T. Herman, and A. Lent, "Iterative algorithms for large partitioned linear systems, with applications to image reconstruction," *Linear Algebra and Its Applications*, vol. 40, pp. 37–67, 1981.

[10] G. T. Herman and L. B. Meyer, "Algebraic reconstruction techniques can be made computationally efficient (positron

emission tomography application)," *IEEE Transactions on Medical Imaging*, vol. 12, no. 3, pp. 600–609, 1993.

[11] C. Byrne, "A unified treatment of some iterative algorithms in signal processing and image reconstruction," *Inverse Problems*, vol. 20, no. 1, pp. 103–120, 2004.

[12] J. M. Elble, N. V. Sahinidis, and P. Vouzis, "GPU computing with Kaczmarz's and other iterative algorithms for linear systems," *Parallel Computing*, vol. 36, no. 5-6, pp. 215–231, 2010.

[13] F. Pasqualetti, R. Carli, and F. Bullo, "Distributed estimation via iterative projections with application to power network monitoring," *Automatica*, vol. 48, no. 5, pp. 747–758, 2012.

[14] Z.-Z. Bai and X.-G. Liu, "On the meany inequality with applications to convergence analysis of several row-action iteration methods," *Numerische Mathematik*, vol. 124, no. 2, pp. 215–236, 2013.

[15] Z.-Z. Bai and W.-T. Wu, "On convergence rate of the randomized Kaczmarz method," *Linear Algebra and Its Applications*, vol. 553, pp. 252–269, 2018.

[16] T. Strohmer and R. Vershynin, "A randomized Kaczmarz algorithm with exponential convergence," *Journal of Fourier Analysis and Applications*, vol. 15, no. 2, pp. 262–278, 2009.

[17] Z.-Z. Bai and W.-T. Wu, "On greedy randomized Kaczmarz method for solving large sparse linear systems," *SIAM Journal on Scientific Computing*, vol. 40, pp. 592–606, 2018.

[18] Z.-Z. Bai and W.-T. Wu, "On relaxed greedy randomized Kaczmarz methods for solving large sparse linear systems," *Applied Mathematics Letters*, vol. 83, pp. 21–26, 2018.

[19] Y. Liu and C.-Q. Gu, "Variant of greedy randomized Kaczmarz for ridge regression," *Applied Numerical Mathematics*, vol. 143, pp. 223–246, 2019.

[20] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[21] E. J. Candes, J. Romberg, and T. Tao, *Robust Uncertainty Principles: Exact Signal Reconstruction from Highly Incomplete Frequency Information*, IEEE Press, Piscataway, NJ, USA, 2006.

[22] D. A. Lorenz, S. Wenger, and F. Schöpfer, "A sparse Kaczmarz solver and a linearized Bregman method for online compressed sensing," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Paris, France, October 2014.

[23] A. Ma, D. Needell, and A. Ramdas, "Convergence properties of the randomized extended gauss--seidel and Kaczmarz methods," *SIAM Journal on Matrix Analysis and Applications*, vol. 36, no. 4, pp. 1590–1604, 2015.

[24] C. F. Lang, "Numerical sensitivity analysis in numerical computation," Graduate thesis, Jilin University, Changchun, China, 2017.

[25] U. Goktas and W. Hereman, "Perturbation analysis and randomized algorithms for large-Scale total least squares problems," *Physica D*, vol. 123, pp. 425–436, 1998.

[26] T. A. Davis and Y. Hu, "Algorithm 915, SuiteSparseQR," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, pp. 1–22, 2011.