Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 9(1): 2473–2499.
DOI: 10.3934/math.2024122
Received: 06 November 2023
Revised: 11 December 2023
Accepted: 19 December 2023
Published: 25 December 2023

Research article

An efficient variant of the greedy block Kaczmarz algorithm for solving large linear systems

Ke Zhang[*]**, Hong-Yan Yin and Xiang-Long Jiang**

Department of Mathematics, Shanghai Maritime University, Shanghai 201306, China

* **Correspondence:** Email: kezhang@shmtu.edu.cn.

Abstract: By exploiting the concept of row partitioning, we propose an efficient variant of the greedy block Kaczmarz algorithm for solving consistent large linear systems. The number of blocks is determined a priori through numerical experiments. The new algorithm works with a reduced linear system, which dramatically diminishes the computational overhead per iteration. The theoretical result validates that this method converges to the unique least-norm solution of the linear system. The effectiveness of the proposed algorithm is also justified by comparing it with some block Kaczmarz algorithms in extensive numerical experiments.

Keywords: linear system; Kaczmarz method; greedy Kaczmarz method; block Kaczmarz method
Mathematics Subject Classification: 65F10, 65F20, 15A06

1. Introduction

A mainstay of scientific computing is solving large linear systems of the form

$$Ax = b, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$.

In what follows, we assume that the linear system is consistent and consider only the least-norm solution x_*. For linear systems of extremely large size, iterative methods are often the only option. These methods date back to the early 19th century that were initiated by the work of Gauss. An explosion of activities has been sparked ever since due to demands from the engineering and scientific fields. Among those candidates, the Kaczmarz algorithm proposed in 1937 by Stefan Kaczmarz is one that is still very appealing nowadays [22]. Based on the `BLAS` level-1 subroutines, it has successfully found its way into a variety of applications in image reconstruction [16], computed tomography [20] and signal processing [9], to mention a few. In its simplest form, the classical Kaczmarz algorithm is given as

$$x_{k+1} = x_k + \frac{b^{(i_k)} - A^{(i_k)} x_k}{\|A^{(i_k)}\|_2^2} (A^{(i_k)})^T, \tag{1.2}$$

where $A^{(i)}$ denotes the ith row of the matrix A, $b^{(i)}$ represents the ith entry of the vector b and $(A^{(i)})^T$ stands for the transpose of $A^{(i)}$. At each iteration, the current iterate x_k is projected onto the hyperplane $A^{(i_k)}x = b^{(i_k)}$, where the working row in (1.2) is selected cyclically by applying $i_k = (k \bmod m) + 1$.

The philosophy behind the classical Kaczmarz algorithm is to apply sequences of orthogonal projections onto hyperplanes. Its convergence is warranted. However, it is not easy to analyze the convergence rate. For more than a decade, the classical Kaczmarz algorithm was in oblivion. Fortunately, it resurged sporadically in the works of [8,15,30]. In 1970, it was rediscovered in the field of electron microscopy and X-ray photography, termed the algebraic reconstruction technique [16]. A breakthrough in the research of Kaczmarz-type algorithms is the randomized Kaczmarz algorithm, which adopts a non-uniform selection rule by picking the working row with probability that is proportional to the norm of the rows [29]. Agaskar et al. [1] propose a complete characterization of the randomized Kaczmarz algorithm, which includes an exact formula for the mean squared error and the annealed error exponent portraying its decay rate. Several approaches for determining working rows have been proposed ever since; for instance, Bai and Wu propose a greedy randomized Kaczmarz (GRK) method which selects the working row by applying the residual rule [5,6]. Gower et al. analyze adaptive sampling rules in the sketch-and-project setting, with an emphasis on the distance rule [17]. Griebel and Oswald investigate the maximal residual rule that greedily chooses the subspace with the largest residual norm for the next update [19]. Eldar and Needell consider a Johnson-Lindenstrauss-type projection when implementing the maximal distance rule for solving linear systems [13]. More about the selection rules for working rows and convergence analysis can be found in [3,4,7,31,34] and the references therein.

While the selection rules for working rows are of great importance in the implementation of the Kaczmarz-type algorithms, the idea of exploiting blocks also matters. To put it simply, the block Kaczmarz method is an iterative scheme for solving linear systems. At each iteration step, the block Kaczmarz algorithm projects the current iterate onto the solution space of a subset of the constraints [27]. In [14], the coefficient matrix is partitioned by blocks of rows whose generalized inverse is then applied in Jacobi or successive overrelaxation iterative schemes. Needell and Tropp [27] put forward a block Kaczmarz algorithm that uses a randomized control scheme to choose the subset per iteration. It lays the path for the research of "row paving" and is the first block Kaczmarz method with an expected linear rate of convergence that is expressed in the geometric properties of the matrix and its submatrices. By using the block Meany inequality, Bai and Liu [2] improve some existing results and develop new convergence theorem for row-action iterative schemes like the block Kaczmarz and the Householder-Bauer methods for solving large linear systems and least-squares problems. Necoara presents a group of randomized block Kaczmarz algorithms that involve a subset of the constraints and extrapolated step sizes [26]. Niu and Zheng investigate a block Kaczmarz algorithm which incorporates the greedy row-selection rule [28]. To capture large block entries of the residual vector, Liu and Gu propose a greedy randomized block Kaczmarz algorithm for solving consistent linear systems [24]. Inspired by the work of [5], Miao and Wu present a greedy randomized average block Kaczmarz method for solving linear systems [25]. Motivated by the Gaussian Kaczmarz method [18] and the greedy rule for selecting working rows in [5, 6], Chen and Huang consider a fast deterministic block Kaczmarz method [10]. Based on the Kaczmarz-Motzkin method, Zhang and Li develop several efficient block Kaczmarz-Motzkin variants by exploiting different sampling strategies [36,37]. Recently, the block Kaczmarz-type algorithms have been generalized to solve least squares problems and linear systems with multiple right-hand sides; for instance, Du et al. [12] present a randomized extended average block Kaczmarz algorithm that is suitable for solving least squares problems. Wu and Chang propose semi-randomized block Kaczmarz methods with simple random sampling for linear systems with multiple right-hand sides [32].

In this work, we propose an efficient variant of the greedy block Kaczmarz (VGBK) algorithm for solving consistent linear systems in the form of (1.1). It proceeds with a partition of rows which can dramatically reduce the computational complexity per iteration. Numerical experiments show that the new algorithm is competitive with some existing block Kaczmarz-type methods. Throughout this work, we denote by $\|A\|_F$ the matrix Frobenius norm and $\|\cdot\|_2$ the 2-norm of a matrix or vector if there is no ambiguity. The notion $A^\dagger$ defines the Moore-Penrose pseudoinverse. The symbol $(\cdot)^T$ stands for the transpose of a matrix or vector. The smallest nonzero singular value and its largest peer of a matrix are represented by $\sigma_{\min}(\cdot)$ and $\sigma_{\max}(\cdot)$, respectively. Unless otherwise stated, matrices are denoted by capital letters, while vectors are usually represented by lowercase ones; capital letters with superscripts are occasionally used to indicate a vector, e.g., $A^{(i_k)}$ for the i_kth row of a matrix A. Scalars are usually signified by Greek letters, except those, such as m, n, i, j, k, s and p, which are used for indexing. The set $\{1,\ldots,m\}$ is abbreviated as $[m]$. An empty set is denoted by $\emptyset$. An index set is usually indicated by a calligraphic capital letter like $\mathcal{I}$, whose cardinality reads as $|\mathcal{I}|$.

This work is organized as follows. In Section 2, we review three related GBK-type algorithms. In Section 3, we first make clear the motivation for this paper, and then we develop a new GBK algorithm with analysis that includes the convergence rate and complexity. In Section 4, we present some numerical experiments to justify the effectiveness of the proposed algorithm. In Section 5, some conclusions and the future work are given.

2. The greedy block Kaczmarz algorithm and its efficient variants

In this section, we recapitulate the GBK algorithm [28], the fast deterministic block Kaczmarz (FDBK) algorithm [10] and the fast greedy block Kaczmarz (FGBK) algorithm [33]. These three algorithms facilitate the delineation of our new algorithm in Section 3.

2.1. The greedy block Kaczmarz algorithm

In the classical Kaczmarz algorithm, the current iterate x_k is projected onto a single hyperplane. In its block counterpart, however, the current iterate x_k is projected onto a combination of several hyperplanes, i.e.,

$$x_{k+1} = x_k + A^\dagger_{\mathcal{I}^G_k}(b_{\mathcal{I}^G_k} - A_{\mathcal{I}^G_k} x_k), \tag{2.1}$$

where $\mathcal{I}^G_k$ is a set that consists of the indexed rows [14, 27]. In general, the block Kaczmarz algorithms take advantage of the `BLAS` level-2 operations and benefit more from exploiting multiple rows simultaneously.

In [28], the block technique is integrated with the idea of maximal distance, which yields the GBK algorithm. In that setting, an index set $\mathcal{I}^G_k$ is determined once the squared distance of the current iterate to the hyperplane $A^{(i_k)}x_k = b^{(i_k)}$ exceeds ε_k which is a parameter varying from zero to the maximal squared distance. In what follows, we do not elaborate on the algorithmic implementation, but refer to Algorithm 1 for details.

Algorithm 1 The GBK algorithm

```
Input: $A, b, x_0, l$ and $\alpha \in (0,1]$
Output: $x_{l+1}$
1: for $k = 0, 1, \ldots, l$ do
2:     Compute
           $$\varepsilon_k = \alpha \cdot \max_{1\le i\le m}\left\{\frac{|b^{(i)} - A^{(i)}x_k|^2}{\|A^{(i)}\|_2^2}\right\}.$$
3:     Determine the index set of positive integers
           $$\mathcal{I}_k^G = \left\{i_k \,\middle|\, |b^{(i_k)} - A^{(i_k)}x_k|^2 \ge \varepsilon_k \|A^{(i_k)}\|_2^2\right\}.$$
4:     Set $x_{k+1} = x_k + A^{\dagger}_{\mathcal{I}_k^G}(b_{\mathcal{I}_k^G} - A_{\mathcal{I}_k^G}x_k)$.
5: end for
```

The convergence result of the GBK algorithm is detailed in the following theorem.

Theorem 1. *For the consistent linear system (1.1), the iterative sequence of $\{x_k\}_{k=0}^{\infty}$, as generated by Algorithm 1, converges to the unique least-norm solution $x_* = A^{\dagger}b$. For $k \ge 0$, the norm of error satisfies*

$$\|x_{k+1} - x_*\|_2^2 \le \left(1 - \beta_k \cdot \frac{\sigma_{\min}^2(A)}{\|A\|_F^2}\right)\|x_k - x_*\|_2^2,$$

where $\alpha \in (0,1]$, $\beta_k = \dfrac{\alpha\|A\|_F^2\|A_{\mathcal{I}_k^G}\|_F^2}{\sigma_{\max}^2(A_{\mathcal{I}_k^G})(\|A\|_F^2 - \|A_{\mathcal{I}_{k-1}^G}\|_F^2)}$ with $\beta_0 = \dfrac{\alpha\|A_{\mathcal{I}_0^G}\|_F^2}{\sigma_{\max}^2(A_{\mathcal{I}_0^G})}$ and $\sigma_{\min}(\cdot)$ and $\sigma_{\max}(\cdot)$ denote the smallest nonzero and largest singular values of a matrix, respectively.

2.2. *A fast deterministic block Kaczmarz algorithm*

Another way to implement the block Kaczmarz algorithm is to make use of the Gaussian Kaczmarz method [18, p. 1677] defined by

$$x_{k+1} = x_k + \frac{g^T(b - Ax_k)}{\|A^Tg\|_2^2}A^Tg,$$

where $g \in \mathbb{R}^m$ is a random Gaussian vector with a zero mean and the covariance matrix being the identity matrix.

Motivated by the Gaussian Kaczmarz method and the greedy rule for selecting working rows in [5, 6], Chen and Huang propose the FDBK algorithm [10]. When updating the iteration, recall from Algorithm 1 that one needs to compute the Moore-Penrose pseudoinverse of sub-matrices of A. In FDBK, however, one only needs to compute a linear combination of submatrices of A^T. The algorithmic implementation of FDBK is presented in Algorithm 2; see [10, Algorithm 1] for more detail.

Algorithm 2 The FDBK algorithm

```
Input: $A$, $b$, $x_0$ and $l$
Output: $x_{l+1}$
1: for $k = 0, 1, \dots, l$ do
2:     Compute
           $$\varepsilon_k = \frac{1}{2}\left(\frac{1}{\|b - Ax_k\|_2^2} \cdot \max_{1\le i\le m}\left\{\frac{\left|b^{(i)} - A^{(i)}x_k\right|^2}{\|A^{(i)}\|_2^2}\right\} + \frac{1}{\|A\|_F^2}\right),$$
       where $A^{(i)}$ draws the $i$th row of $A$ and $b^{(i)}$ the $i$th entry of $b$.
3:     Determine the index set of positive integers
           $$\mathcal{I}_k^{FD} = \left\{i_k \middle| |b^{(i_k)} - A^{(i_k)}x_k|^2 \ge \varepsilon_k \|b - Ax_k\|_2^2 \|A^{(i_k)}\|_2^2\right\}.$$
4:     Compute $c_k = \sum_{i_k \in \mathcal{I}_k^{FD}} \left(b^{(i_k)} - A^{(i_k)}x_k\right) e_{i_k}$, where $e_{i_k}$ is the $i_k$th canonical basis vector in $\mathbb{R}^m$.
5:     Set
           $$x_{k+1} = x_k + \frac{c_k^T(b - Ax_k)}{\|A^T c_k\|_2^2} A^T c_k.$$
6: end for
```

The convergence result of the FDBK algorithm is wrapped up in Theorem 2.

Theorem 2. *Suppose that the linear system (1.1) is consistent and there is no row with entries all zero in A. For any initial guess x_0 in the column space of A^T, the iterative sequence of $\{x_k\}_{k=0}^{\infty}$, as generated by Algorithm 2, converges to the unique least-norm solution $x_* = A^\dagger b$, with the error satisfying*

$$\|x_{k+1} - x_*\|_2^2 \le \left(1 - \gamma_k \cdot \frac{\|A_{\mathcal{I}_k^{FD}}\|_F^2}{\sigma_{\max}^2(A_{\mathcal{I}_k^{FD}})} \cdot \frac{\sigma_{\min}^2(A)}{\|A\|_F^2}\right)\|x_k - x_*\|_2^2,\ k \ge 0,$$

where

$$\gamma_k = \frac{1}{2}\left(\frac{\|A\|_F^2}{\theta_k\|A_{\xi_k}\|_F^2 + (1-\theta_k)\|A_{\mathcal{I}_k^{FD}}\|_F^2} + 1\right) \ge 1,\ k \ge 0,$$

with

$$\xi_k = \left\{i \middle| b^{(i)} - A^{(i)}x_k \ne 0,\ i \in [m]\right\},\ k \ge 0,$$

and

$$\theta_k = \begin{cases} \dfrac{\max\limits_{i\in\xi_k\setminus\mathcal{I}_k^{FD}}\left\{\dfrac{|b^{(i)}-A^{(i)}x_k|^2}{\|A^{(i)}\|_2^2}\right\}}{\max\limits_{1\le i\le m}\left\{\dfrac{|b^{(i)}-A^{(i)}x_k|^2}{\|A^{(i)}\|_2^2}\right\}} < 1, & \text{if } \mathcal{I}_k^{FD}\neq[m] \text{ and } \xi_k\neq\mathcal{I}_k^{FD},\ k\ge 0,\\ 1, & \text{otherwise.}\end{cases}$$

2.3. *The fast greedy block Kaczmarz algorithm*

As explained in the previous two subsections, the GBK algorithm captures hyperplanes onto which the current iterate is projected with relatively large distance, while the FDBK algorithm follows a Gaussian Kaczmarz practice that makes use of all rows of A and b simultaneously.

In order to speed up the convergence of the GBK and FDBK algorithms, Xiao et al. [33] propose an FGBK algorithm that exploits the advantages of the GBK and FDBK algorithms. In particular, a parameter p is introduced to allow for more flexibility in implementing the algorithm. More algorithmic details can be found in Algorithm 3.

Algorithm 3 The FGBK algorithm

```
Input: $A, b, x_0, \alpha \in (0,1], p \in [1,+\infty)$
1: for all $k = 0, 1, 2, \dots$ do
2:     Compute
           $$\varepsilon_k = \alpha\cdot\max_{1\le i\le m}\left\{\frac{|b^{(i)}-A^{(i)}x_k|^p}{\|A^{(i)}\|_p^p}\right\}.$$
3:     Determine the index set of positive integers
           $$\mathcal{I}_k^{FG} = \left\{i \,\middle|\, |b^{(i)}-A^{(i)}x_k|^p \ge \varepsilon_k\|A^{(i)}\|_p^p\right\}.$$
4:     Compute $c_k = \sum_{i\in\mathcal{I}_k^{FG}}\left(b^{(i)}-A^{(i)}x_k\right)e_i$.
5:     Set
           $$x_{k+1} = x_k + \frac{c_k^T(b-Ax_k)}{\|A^Tc_k\|_2^2}A^Tc_k.$$
6: end for
```

The convergence result of the FGBK algorithm is characterized by the following theorem [33].

Theorem 3. *Suppose that the linear system (1.1) is consistent. For any initial guess x_0 in the column space of A^T, the iterative sequence $\{x_k\}_{k=0}^{\infty}$, as generated by Algorithm 3, converges to the unique minimum norm solution $x_* = A^\dagger b$, with the error satisfying*

$$\|x_{k+1}-x_*\|_2^2 \le (1-\beta_k(\alpha,p)\sigma_{\min}^2(A))\|x_k-x_*\|_2^2,\ k\ge 0,$$

where $\beta_k(\alpha, p) = \dfrac{\alpha^{\frac{2}{p}}}{\sum\limits_{i\in[m]\setminus\mathcal{I}_{k-1}^{FG}} \|A^{(i)}\|_p^2} \cdot \dfrac{\sum\limits_{i\in\mathcal{I}_k^{FG}} \|A^{(i)}\|_p^2}{\sigma_{\max}^2(A_{\mathcal{I}_k^{FG}})}$, $\alpha \in (0, 1]$ *and* $p \in [1, +\infty)$.

Remark 1. *We are in a position to give some remarks on the origins of the above-mentioned block Kaczmarz-type algorithms. In [5], Bai and Wu propose a novel GRK algorithm that outperforms the prototypical randomized Kaczmarz algorithm in many scenarios. The GRK algorithm exploits the greedy selection strategy prioritizing large entries of the residual vector with higher probabilities. As shown in Algorithms 1–3, the GBK, FDBK and FGBK algorithms fall into the category of block Kaczmarz-type algorithms. In fact, they can be regarded as extensions of the GRK algorithm with some effective modifications. For instance, the GBK algorithm combines the maximum-distance rule and a modified index set originating from the GRK algorithm. Following the line of the GRK algorithm, the FDBK algorithm employs the same index set as in GRK and incorporates the idea of Gaussian-Kaczmarz iteration.*

3. An efficient variant of the greedy block Kaczmarz algorithm

In this section, we first clarify the initial rationale for developing the new algorithm, and then we give an in-depth analysis of the algorithmic implementation as well as the theoretical results. In Section 2, recall that two existing algorithms, i.e., GBK and FDBK, offer two different ways to incorporate the block techniques; the GBK algorithm takes advantage of rows selected from the index set, while the FDBK algorithm avoids the computation of the Moore-Penrose pseudoinverse and updates the current iterate in a manner analogous to that of the Gaussian Kaczmarz method. In the actual computations, however, both algorithms require one to find the maximum squared distance to m hyperplanes; see Step 2 in Algorithms 1 and 2. It may pose a challenge to the numerical performance when the size of A is huge. Now we switch tacks. Intuitively, it would be desirable to seek the maximum distance to merely a small fraction of the hyperplanes since the computational cost of working out the maximum distance shall be reduced. Therefore, it is interesting to investigate the case in which these desirable attributes are merged. Motivated by this intuition, we construct an efficient variant of the greedy block Kaczmarz (VGBK) algorithm for solving large linear system (1.1). A complete coverage of its algorithmic implementation is displayed in Algorithm 4.

Now, let us present the details of the VGBK algorithm. To reduce the computational complexity in computing the maximum distance, a user-prescribed row partition of the set $[m]$ is imposed at the outset, namely,

$$\cup_{j=1}^{s}\tau_j = [m], \text{ and } \tau_i \cap \tau_j = \emptyset \text{ for } i \neq j,$$

where $\tau_j \subset [m]$ is a subset comprising the row indices of the jth partition for $j = 1, \ldots, s$. As for the choices of partition before the iteration (Step 1 in Algorithm 4), several candidates are available; see, for instance, [21, 27, 35]. In the numerical experiments, we adopt a simple approach by partitioning $[m]$ with $\tau_j^T = [\mathtt{j} : \mathtt{s} : \mathtt{m}]$, where the MATLAB colon operator ":" is used to create a vector with evenly spaced increments for $j = 1, \ldots, s$. It should be stressed that this practice is not necessarily the optimal one, but it turns out to be quite successful as compared with other block Kaczmarz algorithms; see the numerical examples in Section 4. The following toy example gives the reader a sense of the appearance of the partitioning. Assume that $m = 11$ and $s = 3$. It follows from the above discussion that

$$\tau_1 = \{1, 4, 7, 10\},\ \tau_2 = \{2, 5, 8, 11\},\ \tau_3 = \{3, 6, 9\}.$$

The corresponding partitions of A and b can thus be given; for example, if $A \in \mathbb{R}^{11\times 5}$, then $A_{\tau_3} \in \mathbb{R}^{3\times 5}$, assembled by $A(3, :)$, $A(6, :)$ and $A(9, :)$, is the third submatrix (block) from the partition.

After the partition, we have at hand s submatrices $A_{\tau_1}, \dots, A_{\tau_s}$ that mimic the coefficient matrix A. Assume that there are no rows with entries all zero in A and, thus, $\|A_{\tau_j}\|_2 \neq 0$ for $j = 1, \dots, s$. We select cyclically the index for the working submatrix $A_{\tau_{j_k}}$ and vector $b_{\tau_{j_k}}$ with $j_k = (k \bmod s) + 1$. It turns out that the maximum squared distance

$$\max_{i\in\tau_{j_k}} \left\{ \frac{|b^{(i)}_{\tau_{j_k}} - A^{(i)}_{\tau_{j_k}} x_k|^2}{\|A^{(i)}_{\tau_{j_k}}\|_2^2} \right\}$$

is now unraveled from a subset τ_{j_k} instead of $[m]$. Then, a set $\mathcal{I}_k$ is used to collect the indices i_k's of rows when

$$\frac{|b^{(i_k)}_{\tau_{j_k}} - A^{(i_k)}_{\tau_{j_k}} x_k|^2}{\|A^{(i_k)}_{\tau_{j_k}}\|_2^2} \geq \alpha \cdot \max_{i\in\tau_{j_k}} \left\{ \frac{|b^{(i)}_{\tau_{j_k}} - A^{(i)}_{\tau_{j_k}} x_k|^2}{\|A^{(i)}_{\tau_{j_k}}\|_2^2} \right\}$$

for $\alpha \in (0, 1]$. It is easy to verify that $\mathcal{I}_k$ is a subset of τ_{j_k} and is non-empty. Inspired by the Gaussian Kaczmarz method, we update the iterate x_{k+1} by

$$x_{k+1} = x_k + \frac{c_k^T (b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)}{\|A^T_{\tau_{j_k}} c_k\|_2^2} A^T_{\tau_{j_k}} c_k,$$

where $c_k = \sum_{i_k\in\mathcal{I}_k} \left(b^{(i_k)}_{\tau_{j_k}} - A^{(i_k)}_{\tau_{j_k}} x_k \right) e_{i_k}$, with e_{i_k} being the ith canonical basis vector in $\mathbb{R}^{|\tau_{j_k}|}$.

Remark 2. *We are ready to give some remarks. As introduced in Section 2, the novelty of the related FGBK algorithm lies in the choice of exponent p, a parameter encountered in ε_k and the index set. At each iteration, however, the FGBK algorithm requires one to compute the pth power of the maximum distance to m hyperplanes, which may be increasingly time-consuming as the size of A increases; see Algorithm 3 for more details. Such difficulty is circumvented in our VGBK algorithm since one only needs to figure out the maximum distance from a much smaller subset τ_{j_k} per iteration. Numerical experiments in Section 4 also corroborate that the VGBK algorithm improves upon the FGBK method in terms of CPU time.*

Algorithm 4 A VGBK algorithm

```
Input: $A$, $b$, $x_0$, $\alpha \in (0,1]$, $l$ and the number of blocks $s$
Output: $x_{l+1}$
1: Partition before iteration:
   Assume that $\tau_1, \ldots, \tau_s$ is a user-prescribed partition of $[m]$, i.e., $\cup_{j=1}^{s}\tau_j = [m]$ and $\tau_i \cap \tau_j = \emptyset$ for $i \neq j$, where $\tau_j \subset [m]$ is a subset comprising the row indices of the $j$th block for $j = 1, \ldots, s$.
2: for $k = 0, 1, \ldots, l$ do
3:     Compute the index for the working block with $j_k = (k \mod s) + 1$.
4:     Compute
           $$\varepsilon_k = \alpha \cdot \max_{i \in \tau_{j_k}} \left\{ \frac{|b_{\tau_{j_k}}^{(i)} - A_{\tau_{j_k}}^{(i)} x_k|^2}{\|A_{\tau_{j_k}}^{(i)}\|_2^2} \right\},$$
       where $A_{\tau_{j_k}} \in \mathbb{R}^{|\tau_{j_k}| \times n}$ and $b_{\tau_{j_k}} \in \mathbb{R}^{|\tau_{j_k}|}$ denote the submatrix and subvector with rows indexed in the set $\tau_{j_k}$, respectively.
5:     Determine the index set of positive integers
           $$\mathcal{I}_k = \left\{ i_k \middle| \, |b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)} x_k|^2 \geq \varepsilon_k \|A_{\tau_{j_k}}^{(i_k)}\|_2^2 \right\}.$$
6:     Compute $c_k = \sum_{i_k \in \mathcal{I}_k} \left( b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)} x_k \right) e_{i_k} \in \mathbb{R}^{|\tau_{j_k}|}$, where $e_{i_k}$ is the $i_k$th canonical basis vector in $\mathbb{R}^{|\tau_{j_k}|}$.
7:     Set
           $$x_{k+1} = x_k + \frac{c_k^T (b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)}{\|A_{\tau_{j_k}}^T c_k\|_2^2} A_{\tau_{j_k}}^T c_k.$$
8: end for
```

The theorem below establishes the convergence result of the VGBK algorithm.

Theorem 4. *For the consistent linear system (1.1), assume that there are no zero rows in A. Then, the iterative sequence $\{x_k\}_{k=0}^{\infty}$, as generated by Algorithm 4, converges to the unique least-norm solution $x_* = A^\dagger b$ in expectation for any initial guess x_0 that is in the column space of A^T. Moreover, the norm of error satisfies*

$$\|x_{k+1} - x_*\|_2^2 \leq (1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A_{\tau_{j_k}})}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \cdot \frac{\|A_{\mathcal{I}_k}\|_F^2}{\|A_{\tau_{j_k}}\|_F^2})\|x_k - x_*\|_2^2, \tag{3.1}$$

where τ_{j_k} is the partition block chosen at the kth iteration, $\mathcal{I}_k$ is the index set defined in Algorithm 4, $\alpha \in (0,1]$ and $\omega_k \in (0,1]$ for $k = 0, 1, \ldots$.

Proof. The proof is partly inspired by [10, Theorem 3.1]. It follows from Steps 6 and 7 of Algorithm 4 that

$$x_{k+1} = x_k + \frac{c_k^T (b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)}{\|A_{\tau_{j_k}}^T c_k\|_2^2} A_{\tau_{j_k}}^T c_k, \tag{3.2}$$

where $c_k = \sum_{i_k \in \mathcal{I}_k} \left(b^{(i_k)} - A^{(i_k)} x_k \right) e_i \in \mathbb{R}^{|\tau_{j_k}|}$. Subtracting x_* from both sides of (3.2) yields

$$x_{k+1} - x_* = x_k - x_* + \frac{c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)}{\|A_{\tau_{j_k}}^T c_k\|_2^2} A_{\tau_{j_k}}^T c_k = (I - P_k)(x_k - x_*), \tag{3.3}$$

where $P_k = \frac{A_{\tau_{j_k}}^T c_k c_k^T A_{\tau_{j_k}}}{\|A_{\tau_{j_k}}^T c_k\|_2^2} \in \mathbb{R}^{n\times n}$ is an orthogonal projection matrix, namely, $P_k^T = P_k$ and $P_k^2 = P_k$. From (3.2) and (3.3), we have

$$\begin{aligned}(x_{k+1} - x_*)^T(x_{k+1} - x_k) &= (x_k - x_*)^T \cdot (I - P_k)^T \cdot \frac{c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)}{\|A_{\tau_{j_k}}^T c_k\|_2^2} A_{\tau_{j_k}}^T c_k \\ &= -(x_k - x_*)^T \cdot (I - P_k)^T \cdot P_k(x_k - x_*) \\ &= -(x_k - x_*)^T \cdot (P_k - P_k^2) \cdot (x_k - x_*) \\ &= 0,\end{aligned}$$

which implies that $x_{k+1} - x_*$ is orthogonal to $x_{k+1} - x_k$.

Computing the squared 2-norm of $x_{k+1} - x_*$ in (3.3), together with the use of the Pythagorean theorem, renders

$$\begin{aligned}\|x_{k+1} - x_*\|_2^2 &= \|(I - P_k)(x_k - x_*)\|_2^2 \\ &= \|x_k - x_*\|_2^2 - \|P_k(x_k - x_*)\|_2^2 \\ &= \|x_k - x_*\|_2^2 - \left\|\frac{A_{\tau_{j_k}}^T c_k c_k^T A_{\tau_{j_k}}}{\|A_{\tau_{j_k}}^T c_k\|_2^2}(x_k - x_*)\right\|_2^2 \\ &= \|x_k - x_*\|_2^2 - \frac{|c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)|^2}{\|A_{\tau_{j_k}}^T c_k\|_2^2}.\end{aligned} \tag{3.4}$$

Let $|\mathcal{I}_k|$ be the number of elements in the index set $\mathcal{I}_k$ and $E_k \in \mathbb{R}^{|\tau_{j_k}|\times|\mathcal{I}_k|}$ be a matrix consisting of the canonical basis vectors $e_{i_k} \in \mathbb{R}^{|\tau_{j_k}|}$ for $i_k \in \mathcal{I}_k$. Therefore, we get

$$\hat{c}_k = E_k^T c_k \in \mathbb{R}^{|\mathcal{I}_k|} \tag{3.5}$$

and

$$A_{\mathcal{I}_k} = E_k^T A_{\tau_{j_k}} \in \mathbb{R}^{|\mathcal{I}_k|\times n}. \tag{3.6}$$

In a nutshell, $\hat{c}_k$ and $A_{\mathcal{I}_k}$ are condensed counterparts of c_k and $A_{\tau_{j_k}}$ by encapsulating rows indexed in the set $\mathcal{I}_k$. A straightforward computation using (3.5) and (3.6) yields

$$\|\hat{c}_k\|_2^2 = c_k^T E_k E_k^T c_k = \|c_k\|_2^2 = \sum_{i_k\in\mathcal{I}_k} |b^{(i_k)} - A^{(i_k)} x_k|^2, \tag{3.7}$$

$$\|A_{\tau_{j_k}}^T c_k\|_2^2 = \hat{c}_k^T E_k^T A_{\tau_{j_k}} A_{\tau_{j_k}}^T E_k \hat{c}_k = \hat{c}_k^T A_{\mathcal{I}_k} A_{\mathcal{I}_k}^T \hat{c}_k = \|A_{\mathcal{I}_k}^T \hat{c}_k\|_2^2 \le \sigma_{\max}^2(A_{\mathcal{I}_k})\|\hat{c}_k\|_2^2, \tag{3.8}$$

where $\sigma_{\max}^2(A_{\mathcal{I}_k})$ is the largest singular value of $A_{\mathcal{I}_k}$. Besides, it follows from the definition of c_k and (3.7) that

$$c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k) = \sum_{i_k\in\mathcal{I}_k} |b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)} x_k|^2 = \|\hat{c}_k\|_2^2. \tag{3.9}$$

Combining (3.7)–(3.9), we have

$$\begin{aligned}
\frac{|c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k)|^2}{\|A_{\tau_{j_k}}^T c_k\|_2^2} &= \frac{\sum\limits_{i_k\in\mathcal{I}_k} |b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)}x_k|^2\|\hat{c}_k\|_2^2}{\|A_{\mathcal{I}_k}^T\hat{c}_k\|_2^2} \\
&\geq \frac{\sum\limits_{i_k\in\mathcal{I}_k} |b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)}x_k|^2\|\hat{c}_k\|_2^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})\|\hat{c}_k\|_2^2} \\
&= \frac{\sum\limits_{i_k\in\mathcal{I}_k} |b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)}x_k|^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \\
&= \sum_{i_k\in\mathcal{I}_k}\left(\frac{|b_{\tau_{j_k}}^{(i_k)} - A_{\tau_{j_k}}^{(i_k)}x_k|^2}{\|A_{\tau_{j_k}}^{(i_k)}\|_2^2}\cdot\frac{\|A_{\tau_{j_k}}^{(i_k)}\|_2^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})}\right) \\
&\geq \alpha\cdot\max_{i\in\tau_{j_k}}\left\{\frac{|b_{\tau_{j_k}}^{(i)} - A_{\tau_{j_k}}^{(i)}x_k|^2}{\|A_{\tau_{j_k}}^{(i)}\|_2^2}\right\}\cdot\frac{\sum\limits_{i_k\in\mathcal{I}_k}\|A_{\tau_{j_k}}^{(i_k)}\|_2^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})}. \qquad (3.10)
\end{aligned}$$

Moreover, $\|b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k\|_2^2$ can be recast as

$$\|b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k\|_2^2 = \sum_{i\in\tau_{j_k}}\frac{|b_{\tau_{j_k}}^{(i)} - A_{\tau_{j_k}}^{(i)}x_k|^2}{\|A_{\tau_{j_k}}^{(i)}\|_2^2}\|A_{\tau_{j_k}}^{(i)}\|_2^2 \leq \max_{i\in\tau_{j_k}}\left\{\frac{|b_{\tau_{j_k}}^{(i)} - A_{\tau_{j_k}}^{(i)}x_k|^2}{\|A_{\tau_{j_k}}^{(i)}\|_2^2}\right\}\sum_{i\in\tau_{j_k}}\|A_{\tau_{j_k}}^{(i)}\|_2^2,$$

or, equivalently,

$$\frac{\|b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k\|_2^2}{\sum\limits_{i\in\tau_{j_k}}\|A_{\tau_{j_k}}^{(i)}\|_2^2} \leq \max_{i\in\tau_{j_k}}\left\{\frac{|b_{\tau_{j_k}}^{(i)} - A_{\tau_{j_k}}^{(i)}x_k|^2}{\|A_{\tau_{j_k}}^{(i)}\|_2^2}\right\}. \qquad (3.11)$$

Denote $u_k = x_k - x_*$ and $u_k = u_{k1} + u_{k2}$, where u_{k1} is in the column space of $A_{\tau_{j_k}}^T$ and u_{k2} is in the subspace orthogonal to the column space of $A_{\tau_{j_k}}^T$, i.e., $A_{\tau_{j_k}}u_{k2} = 0$. Let $\|u_{k1}\|_2^2 = \omega_k\|u_k\|_2^2$ with $\omega_k \in (0, 1]$. Therefore,

$$\|b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k\|_2^2 = \|A_{\tau_{j_k}}u_k\|_2^2 = \|A_{\tau_{j_k}}u_{k1}\|_2^2 \geq \sigma_{\min}^2(A_{\tau_{j_k}})\|u_{k1}\|_2^2 = \omega_k\sigma_{\min}^2(A_{\tau_{j_k}})\|u_k\|_2^2.$$

Multiplying both sides of (3.11) with α, we attain that

$$\alpha\cdot\max_{i\in\tau_{j_k}}\left\{\frac{|b_{\tau_{j_k}}^{(i)} - A_{\tau_{j_k}}^{(i)}x_k|^2}{\|A_{\tau_{j_k}}^{(i)}\|_2^2}\right\} \geq \frac{\alpha\cdot\|b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k\|_2^2}{\sum\limits_{i\in\tau_{j_k}}\|A_{\tau_{j_k}}^{(i)}\|_2^2} \geq \frac{\alpha\omega_k\sigma_{\min}^2(A_{\tau_{j_k}})\cdot\|u_k\|_2^2}{\sum\limits_{i\in\tau_{j_k}}\|A_{\tau_{j_k}}^{(i)}\|_2^2}, \qquad (3.12)$$

where $\sigma_{\min}^2(A_{\tau_{j_k}})$ is the smallest nonzero singular value of $A_{\tau_{j_k}}$.

Substituting the inequality (3.12) into (3.10) gives

$$\frac{|c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}}x_k)|^2}{\|A_{\tau_{j_k}}^T c_k\|_2^2} \geq \frac{\alpha\omega_k\sigma_{\min}^2(A_{\tau_{j_k}})}{\sum\limits_{i\in\tau_{j_k}}\|A_{\tau_{j_k}}^{(i)}\|_2^2}\cdot\frac{\sum\limits_{i_k\in\mathcal{I}_k}\|A_{\tau_{j_k}}^{(i_k)}\|_2^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})}\|u_k\|_2^2. \qquad (3.13)$$

Finally, it follows from (3.4) and (3.13) that

$$\begin{aligned}
\|u_{k+1}\|_2^2 &= \|u_k\|_2^2 - \frac{|c_k^T(b_{\tau_{j_k}} - A_{\tau_{j_k}} x_k)|^2}{\|A_{\tau_{j_k}}^T c_k\|_2^2} \\
&\le \|u_k\|_2^2 - \frac{\alpha\omega_k \sigma_{\min}^2(A_{\tau_{j_k}})}{\sum\limits_{i\in\tau_{j_k}} \|A_{\tau_{j_k}}^{(i)}\|_2^2} \cdot \frac{\sum\limits_{i_k\in\mathcal{I}_k} \|A_{\tau_{j_k}}^{(i_k)}\|_2^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \|u_k\|_2^2 \\
&= (1 - \frac{\alpha\omega_k \sigma_{\min}^2(A_{\tau_{j_k}})}{\sum\limits_{i\in\tau_{j_k}} \|A_{\tau_{j_k}}^{(i)}\|_2^2} \cdot \frac{\sum\limits_{i_k\in\mathcal{I}_k} \|A_{\tau_{j_k}}^{(i_k)}\|_2^2}{\sigma_{\max}^2(A_{\mathcal{I}_k})}) \|u_k\|_2^2 \\
&= (1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A_{\tau_{j_k}})}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \cdot \frac{\|A_{\mathcal{I}_k}\|_F^2}{\|A_{\tau_{j_k}}\|_F^2}) \|u_k\|_2^2,
\end{aligned}$$

which completes the proof. □

Remark 3. *Some remarks are in order. Since* $\|A_{\mathcal{I}_k}\|_F^2 \ge \sigma_{\max}^2(A_{\mathcal{I}_k})$*, it follows that*

$$1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A_{\tau_{j_k}})}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \cdot \frac{\|A_{\mathcal{I}_k}\|_F^2}{\|A_{\tau_{j_k}}\|_F^2} \le 1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A_{\tau_{j_k}})}{\|A_{\tau_{j_k}}\|_F^2} < 1, \tag{3.14}$$

where $\alpha \in (0, 1]$ *and* $\omega_k \in (0, 1]$*. Therefore, the VGBK algorithm converges to the least-norm solution under the assumption of Theorem 4. A special case of the VGBK algorithm exists when the number of blocks* $s = 1$*, i.e.,* $A_{\tau_{j_k}} = A$*. If so, then* $\omega_k = 1$*; thus* (3.14) *is simplified to*

$$1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A_{\tau_{j_k}})}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \cdot \frac{\|A_{\mathcal{I}_k}\|_F^2}{\|A_{\tau_{j_k}}\|_F^2} \le 1 - \alpha \cdot \frac{\sigma_{\min}^2(A)}{\|A\|_F^2}.$$

More generally, if $\sigma_{\min}^2(A_{\tau_{j_k}})/\|A_{\tau_{j_k}}\|_F^2 \ge \sigma_{\min}^2(A)/\|A\|_F^2$ *holds in* (3.14)*, then the inequality* (3.14) *reduces to*

$$1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A_{\tau_{j_k}})}{\sigma_{\max}^2(A_{\mathcal{I}_k})} \cdot \frac{\|A_{\mathcal{I}_k}\|_F^2}{\|A_{\tau_{j_k}}\|_F^2} \le 1 - \alpha\omega_k \cdot \frac{\sigma_{\min}^2(A)}{\|A\|_F^2},$$

which resembles those error bounds given in Theorems 1–3. It should be noted that the condition $\sigma_{\min}^2(A_{\tau_{j_k}})/\|A_{\tau_{j_k}}\|_F^2 \ge \sigma_{\min}^2(A)/\|A\|_F^2$ *is not too stringent; for instance, if A is a matrix of normally distributed random numbers, then* $\|A_{\tau_{j_k}}\|_F^2/\|A\|_F^2 \approx s^{-1}$ *for sufficiently large m, where s is the number of blocks and m is the number of rows in A. Consequently, the condition* $\sigma_{\min}^2(A_{\tau_{j_k}})/\|A_{\tau_{j_k}}\|_F^2 \ge \sigma_{\min}^2(A)/\|A\|_F^2$ *holds as long as* $\sigma_{\min}^2(A_{\tau_{j_k}})/\sigma_{\min}^2(A) \gtrsim s^{-1}$*. In the numerical experiments, s is always chosen as* $s = \lfloor 0.8\% m \rfloor$ *or* $s = \lfloor 4\% m \rfloor$*, which implies that* s^{-1} *approaches 0 if m is large enough. In the implementation of the VGBK algorithm, we only work on a small fraction of A and b, instead of exploiting the full information of A and b as done in the GBK, FDBK and FGBK methods. For this reason, the VGBK algorithm generally requires more iteration steps than these three algorithms to reach the required accuracy. Nevertheless, this does not necessarily imply that the VGBK algorithm* *is numerically inferior to the three counterparts. On the contrary, as we will explain soon, the VGBK algorithm consumes far less computational cost than the other three algorithms per iteration step, which can readily reduce the total computing time. Numerical examples in the next section also validate that the VGBK algorithm enjoys noticeable gains in CPU time as compared to the other three algorithms.*

Before ending this section, we investigate the computational overhead of VGBK at each iteration step. The complexities of the block Kaczmarz algorithms outlined in Section 2, including the GBK, FDBK and FGBK algorithms, are also analyzed for comparison. For each method, the total computational overhead per iteration consists of the cost for determining the pseudoinverse of a matrix, finding the maximum of a vector and updating the iteration and other costs. To keep the exposition simple, we assume that the coefficient matrix A in (1.1) is dense and $p = 2$ in the FGBK algorithm. The results are tabulated in Table 1.

Table 1. Comparison of computational complexities for GBK, FDBK, FGBK and VGBK with m-D standing for m-dimensional.

Method	Pseudoinverse-free	No. of finding maximums	Iteration update (FLOPS)	Others (FLOPS)
GBK	No	1 (m-D vector)	$(2\lvert\mathcal{I}_k^G\rvert + 1)n$	$(2n+3)m+1$
FDBK	Yes	1 (m-D vector)	$(2\lvert\mathcal{I}_k^{FD}\rvert + 4)n + 2\lvert\mathcal{I}_k^{FD}\rvert + 1$	$(2n+5)m+4$
FGBK	Yes	1 (m-D vector)	$(2\lvert\mathcal{I}_k^{FG}\rvert + 4)n + 2\lvert\mathcal{I}_k^{FG}\rvert + 1$	$(2n+3)m+1$
VGBK	Yes	1 ($\lvert\tau_{j_k}\rvert$-D vector)	$(2\lvert\mathcal{I}_k\rvert + 4)n + 2\lvert\mathcal{I}_k\rvert + 1$	$(2n+3)\lvert\tau_{j_k}\rvert + 1$

The second column of Table 1 indicates whether the implementation of the corresponding method involves computing the pseudoinverse. We see that the FDBK, FGBK and VGBK algorithms are pseudoinverse-free, while the GBK method involves computing the pseudoinverse of a submatrix of A. The computation involving the pseudoinverse of a matrix is expensive and often constitutes the bulk of the total computational cost. A standard work-around refers to iterative methods such as CGLS [27]. In Table 1, we do not specify the method for addressing the pseudoinverse in GBK as we just qualitatively point out that the GBK algorithm is not pseudoinverse-free; a detailed analysis can be made available if the method is prescribed, such as CGLS [28, p. 5]. The third column shows the number of maximum to be found. Take the FDBK algorithm for example. We need to figure out $\max_{1\le i\le m}\{|b^{(i)} - A^{(i)}x_k|^2/\|A^{(i)}\|_2^2\}$ at Step 2 of Algorithm 2, which is equivalent to finding the maximum of an m-dimensional vector. The fourth column displays the FLOPS needed when updating the iterate x_k. The last column collects FLOPS that are not counted in the previous three columns; for instance, in the case of FDBK, it includes the cost of computing the parameter ε_k, determining the index set $\mathcal{I}_k^{FD}$ and accessing the vector c_k.

In the VGBK algorithm, recall that $\mathcal{I}_k \subset \tau_{j_k}$ and $\tau_{j_k} \subset [m]$. It follows that $|\mathcal{I}_k| < |\tau_{j_k}| \ll m$. Furthermore, it generally holds that $|\mathcal{I}_k| \ll \min\{|\mathcal{I}_k^G|, |\mathcal{I}_k^{FD}|, |\mathcal{I}_k^{FG}|\}$ since $\mathcal{I}_k$ is a subset of τ_{j_k}, which itself is a subset of $[m]$ while $\mathcal{I}_k^G$, $\mathcal{I}_k^{FD}$, $\mathcal{I}_k^{FG}$ are subsets of $[m]$. Therefore, the total cost per iteration for the VGBK algorithm can be much less than that for its three counterparts. To see this intuitively, we consider the number of blocks $s = \lfloor 0.8\% m \rfloor$, which is a practical choice for the overdetermined linear system given in Section 4. Thus each block τ_{j_k} has about $m/s \approx 125$ rows on average. In light of this, the computational cost per iteration for the VGBK algorithm roughly accounts for $1/s$ of those for the

GBK, FDBK and FGBK methods, which greatly reduces the computing time and offsets the excessive iterations due to the initial partitioning. Such an advantage becomes more pronounced when m is large since $1/s = 1/\lfloor 0.8\% m \rfloor$ goes to 0 as m increases; see Section 4 for more details.

4. Numerical experiments

In this section, some numerical experiments are presented to verify the effectiveness of the VGBK algorithm in comparison with three block Kaczmarz counterparts, i.e., GBK [28], FDBK [10] and FGBK [33]. All experiments were run on a laptop with an AMD R5-5600H CPU @3.30GHZ 16 GB RAM by operating MATLAB 2022a on a Windows 11 system. Numerical performance of the four algorithms is assessed by means of the CPU time in seconds (CPU) and the number of iteration steps (IT). To demonstrate the effectiveness, we determine the speed-up values of VGBK against GBK, FDBK and FGBK, which are respectively defined by

$$\text{speed} - \text{up1} = \frac{\text{CPU time of GBK}}{\text{CPU time of VGBK}},$$
$$\text{speed} - \text{up2} = \frac{\text{CPU time of FDBK}}{\text{CPU time of VGBK}},$$
$$\text{speed} - \text{up3} = \frac{\text{CPU time of FGBK}}{\text{CPU time of VGBK}}.$$

As stated in Section 1, we are only concerned with the solution of consistent linear system (1.1) in this work. To this end, the right-hand side vector b is set to be Ax_* in all examples, where x_* is generated by the MATLAB built-in function `randn`. The coefficient matrices A are either given by `randn` or taken from real-world applications [11, 23]; see Table 2 for detailed properties of these matrices. As such, we are guaranteed to work on the consistent linear system (1.1). The initial guess is given by $x_0 = 0$. All computations are terminated once the relative solution error (RSE) at the current iteration satisfies that RSE $< 10^{-6}$ or when the number of iteration steps exceeds the maximum of 200000, where

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

In Algorithm 4, recall that the implementation of VGBK involves two parameters, i.e., the number of blocks s and the tuning parameter α. Numerical experiments show that the VGBK algorithm often offers sound performance with appropriate choices of s and α. The experimental parameters in the GBK, FDBK, FGBK and VGBK algorithms are set as follows:

I. GBK [28]: The parameter α in Algorithm 1 is chosen to be

$$\alpha = \frac{1}{2} + \frac{\|b - Ax_k\|_2^2}{2\|A\|_F^2}\left(\max_{1\le i\le m}\left\{\frac{|b^{(i)} - A^{(i)}x_k|^2}{\|A^{(i)}\|_2^2}\right\}\right)^{-1},$$

which is recommended in the original work on the GBK [28].

II. FDBK [10]: Algorithm 2 is parameter-free.

III. FGBK [33]: In [33], FGBK(p) with $p = 2$ requires the least CPU time for most tests. Therefore, we set $p = 2$. Besides, the parameter α is set to be 0.05 or 0.1, which are preferred in [33].

IV. VGBK: The tuning parameter α and the number of blocks s are required in Algorithm 4. We set $s = \lfloor 0.8\% m \rfloor$ (overdetermined linear system), $s = \lfloor 4\% m \rfloor$ (underdetermined linear system) and $\alpha = 0.1$, the reason for which is given in Examples 1–2. Here, $\lfloor \cdot \rfloor$ denotes the floor function.

The purpose of this section is two-fold. In Section 4.1, we look into choices of the two parameters s and α for various types of test matrices. In Section 4.2, the numerical performance of the VGBK algorithm is appraised by comparing it with that of the GBK, FDBK and FGBK algorithms for abundant experiments.

Table 2. Properties of the test matrices.

Matrix	Size	Density	cond(A)	Application field
`ash958`	958×292	0.6849%	3.2014	Least Squares Problem
`abtaha1`	14596×209	1.6819%	12.2284	Combinatorial Problem
`abtaha2`	37932×331	1.0930%	12.2199	Combinatorial Problem
`ch7-9-b2`	17640×1512	0.1984%	$1.6077 \cdot 10^{15}$	Combinatorial Problem
`ch8-8-b2`	18816×1568	0.1913%	$1.6326 \cdot 10^{15}$	Combinatorial Problem
`Franz10`	19588×4164	0.1195%	$1.2694 \cdot 10^{16}$	Combinatorial Problem
`mk11-b3`	17325×6930	0.0577%	$5.8553 \cdot 10^{15}$	Combinatorial Problem
`bibd_49_3`	1176×18424	0.2551%	1.7701	Combinatorial Problem
`cat_ears_3_4`	5226×13271	0.0571%	17.9503	Combinatorial Problem
`GL7d25`	2798×21074	0.1385%	$1.4241 \cdot 10^{19}$	Combinatorial Problem
`df2177`	630×10358	0.3423%	2.0066	Linear Programming Problem
`lp22`	2958×16392	0.1413%	25.7823	Linear Programming Problem
`lp_maros_r7`	3136×9408	0.4910%	2.3231	Linear Programming Problem

4.1. Choices of s and α

Theoretically identifying the optimal values of s and α in the VGBK method, if they exist, is not available for now, and we shall not pursue it here. Instead, we attempt to experimentally determine practical choices of s and α in this subsection. In Example 1, we empirically analyze the influence of s on VGBK. In Example 2, we tackle the problem of choosing α.

Example 1. *This example serves to show how the number of blocks s impacts the numerical performance of the VGBK algorithm. For the moment, α is selected to be* 0.1*; numerical experiments in Example 2 indicate that setting $\alpha = 0.1$ turns out to be practical. We consider linear system (1.1) with thin ($m \geq n$) or fat ($m \leq n$) coefficient matrices A which are either generated by the MATLAB function* `randn` *or adapted from the SuiteSparse Matrix Collection [23]. Numerical results with varying values of s are illustrated in Figures 1–3.*

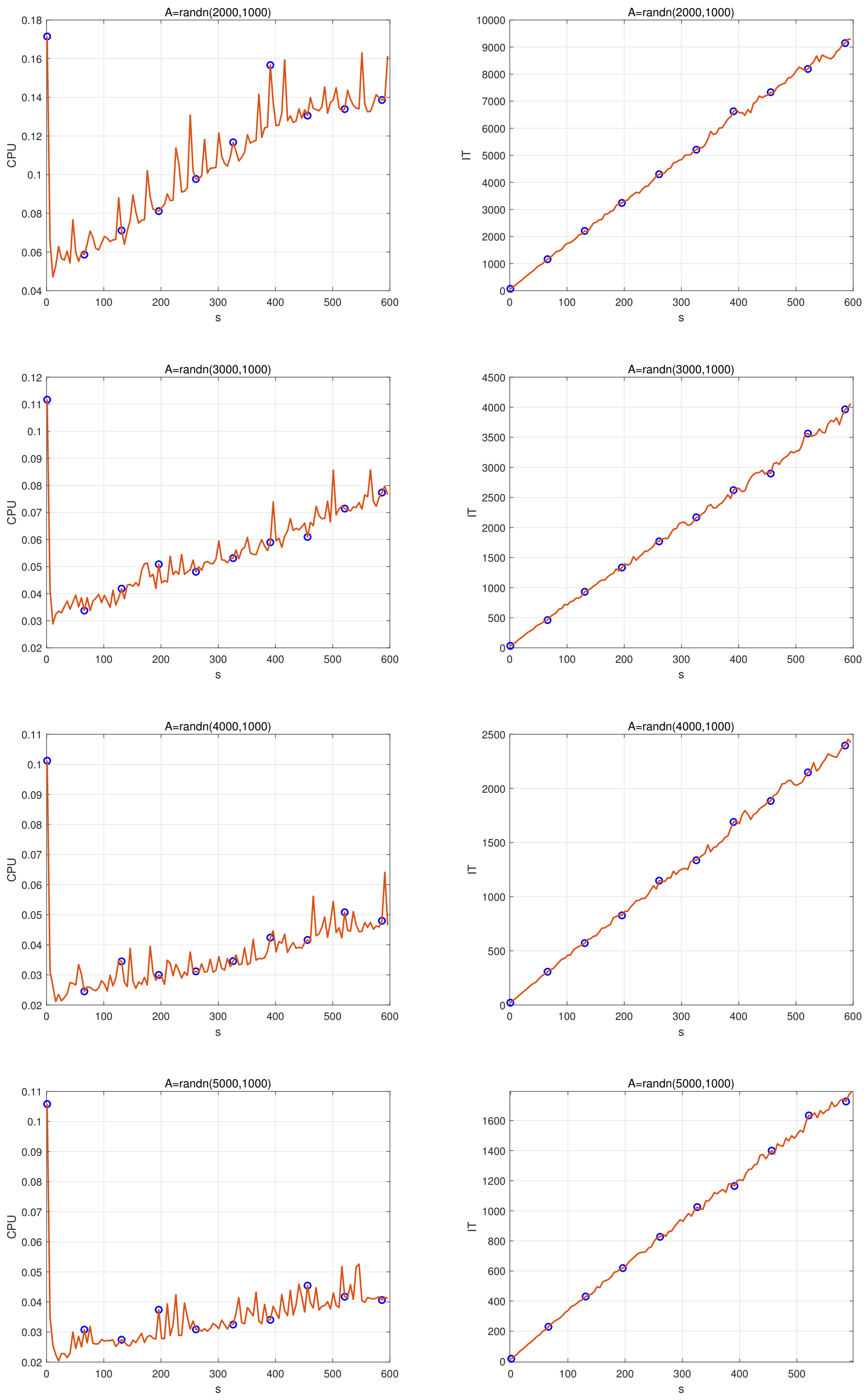

Figure 1. CPU and IT results for thin coefficient matrices $\texttt{randn}(m, 1000)$ with varying s for the VGBK algorithm, where $m = 2000$, 3000, 4000 and 5000, respectively.

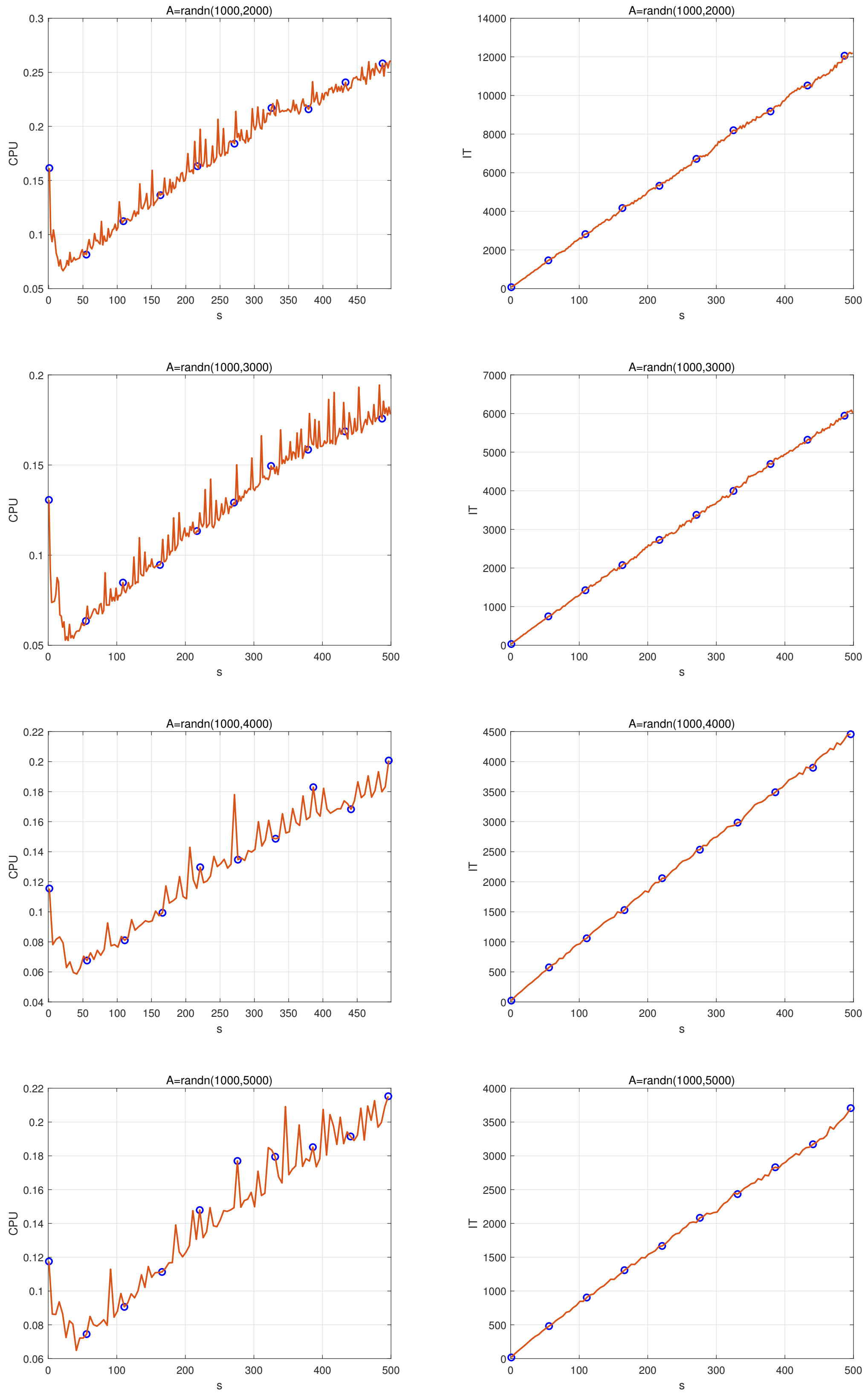

Figure 2. CPU and IT results for fat coefficient matrices $\texttt{randn}(1000, n)$ with varying s for the VGBK algorithm, where $n = 2000$, 3000, 4000 and 5000, respectively.

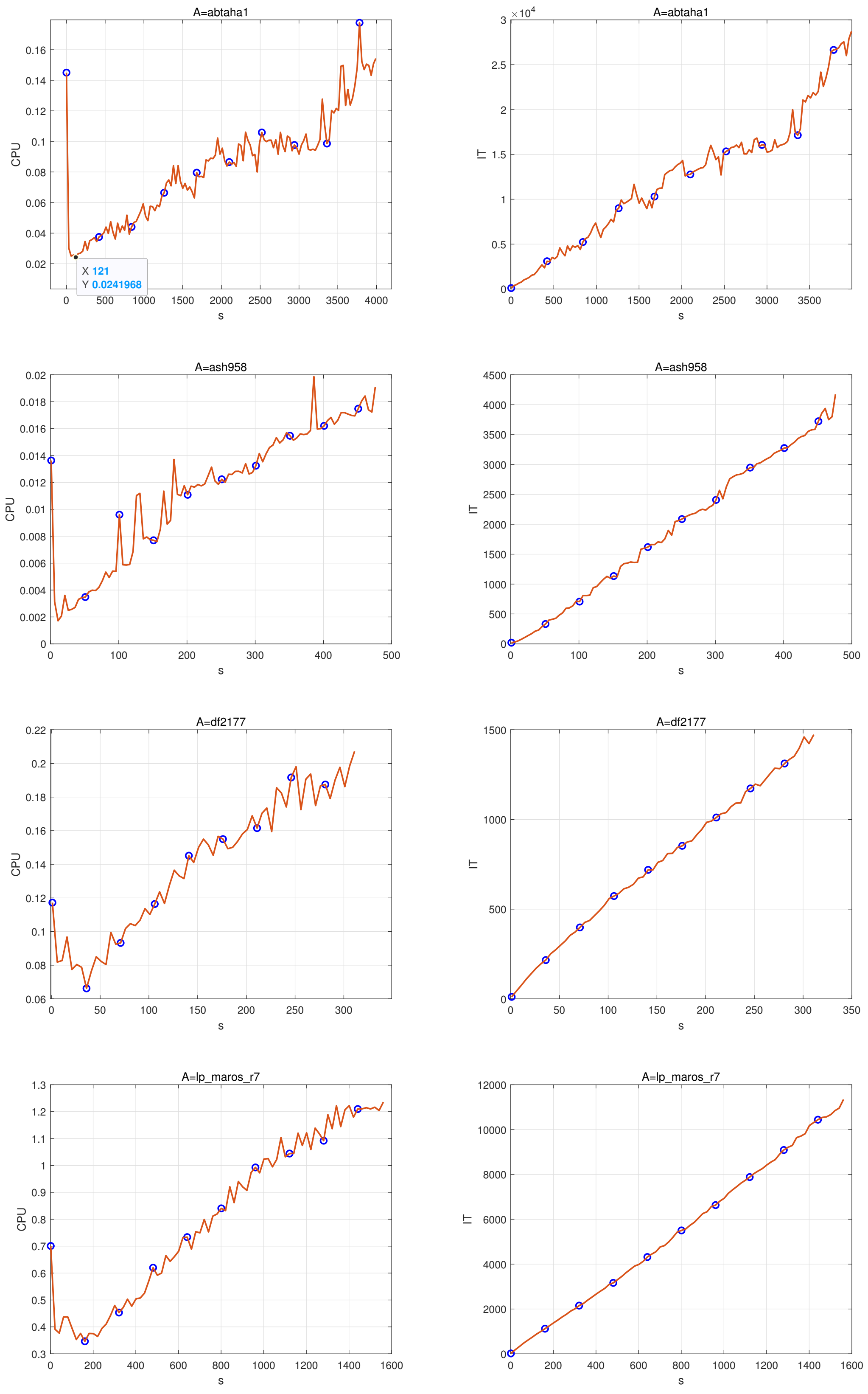

Figure 3. CPU and IT results for thin (`abtaha1, ash958`) and fat coefficient matrices (`df2177, lp_maros_r7`) with varying s for the VGBK algorithm.

We are now in a position to give some remarks. For either thin or fat coefficient matrices, as demonstrated in Figures 1–3, the CPU time (though with wiggles) exhibits a roughly V-shaped curve while the number of IT increases in general as s increases. It can be explained as follows. As for the CPU time, if s is small, or put it another way, if each submatrix/vector associated with τ_j from the partition in the VGBK algorithm is of large size, then the computation overhead tends to increase, which in turn may consume more CPU time. On the contrary, if s is large, i.e., each submatrix/vector indexed in τ_j is of small size, then only a linear system of small size is handled; more iterations are required in exchange for such inexpensive computation, which probably boosts the total CPU time. This explains why the CPU time first drops and then bounces up as s grows. Nevertheless, the scenario for the number of IT differs. In fact, small values of s often lead to large blocks which maintain most information of the original linear system; computation with such large blocks tends to less demanding in the number of iterations, which is quite in line with the observations reported in Figures 1–3.

In practice, it is often the CPU time that counts. Therefore, we restrict ourselves to diminishing the CPU time here. As portrayed in Figures 1–3, one needs to strike a balance when choosing the number of blocks s. To this end, we pinpoint the "experimental minimum" of the CPU time by tracking the MATLAB data tips. Take the thin matrix `abtaha1` *as an example; the first subplot in Figure 3 illustrates that the"experimental minimum" of the CPU time is achieved at $s = 121$, with approximately 0.82% of the row size of* `abtaha1` *($m = 14596$). For all experiments in this example, the percentage s/m ranges from 0.37% to 1.14% for thin matrices while that for fat matrices ranges from 2.10% to 5.71%. Heuristically, any value in the intervals $[\lfloor 0.37\% m\rfloor, \lfloor 1.14\% m\rfloor]$ and $[\lfloor 2.10\% m\rfloor, \lfloor 5.71\% m\rfloor]$, separately, can be a reasonable choice for thin and fat matrices, where $\lfloor\cdot\rfloor$ is the floor function and m is the row size of A. In light of this, we pick two intermediate values by setting $s = \lfloor 0.8\% m\rfloor$ for overdetermined linear systems and $s = \lfloor 4\% m\rfloor$ for underdetermined ones in Examples 2–5.*

Example 2. *In this example, we examine how the tuning parameter α influences the numerical performance of the VGBK algorithm. In Algorithm 4, if the value of α is large, then only a small fraction of rows are extracted from A and b to update the iterate x_{k+1}. To ameliorate the possible loss of information, more iterations (and thus CPU time) are required to reach the prescribed accuracy; see Steps 5–7 in Algorithm 4. Conversely, if α is chosen to be small, more rows will be retained, which amounts to updating the iterate x_{k+1} for a relatively large linear system. As a result, the computational overhead per iteration increases, which results in more CPU time. Such an observation tallies with Figures 4–6 for various test matrices. Therefore, one should exercise caution in picking the value of α. As shown in Figures 4–6, values of α that entail the least CPU time for either thin or fat matrices lie in some neighborhood of 0.1. For this reason, the choice $\alpha = 0.1$ can be practical for the VGBK algorithm. In what follows, we assume that $\alpha = 0.1$ in all experiments. It should also be stressed that the value 0.1 is also a candidate of choice for α in the FGBK algorithm [33, Section 3].*

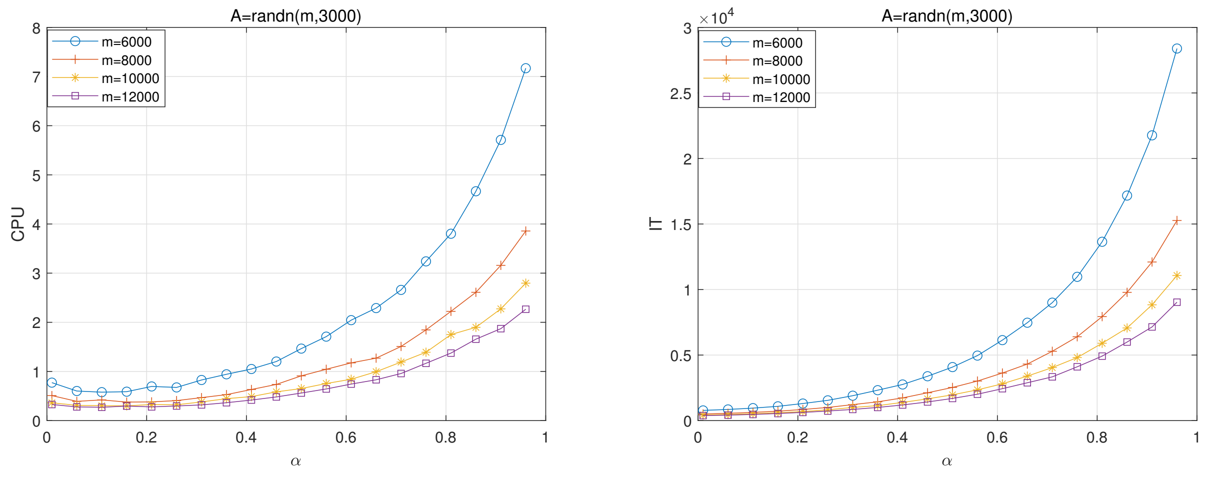

Figure 4. CPU and IT results for thin coefficient matrices $\mathtt{randn}(m, 3000)$, where $m = 6000$, 8000, 10000 and 12000, with varying α for the VGBK algorithm.

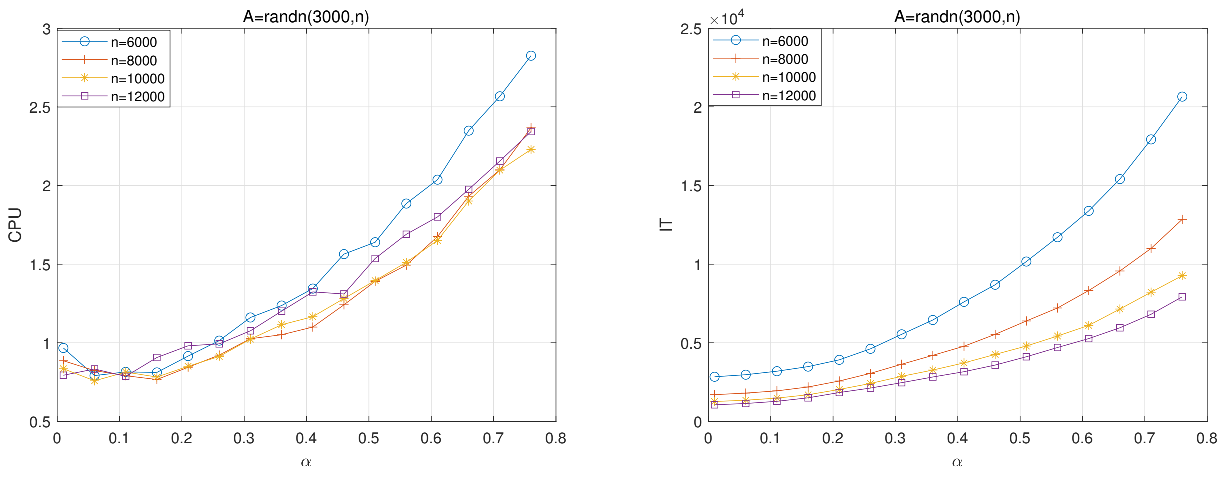

Figure 5. CPU and IT results for fat coefficient matrices $\mathtt{randn}(3000, n)$, where $n = 6000$, 8000, 10000 and 12000, with varying α for the VGBK algorithm.

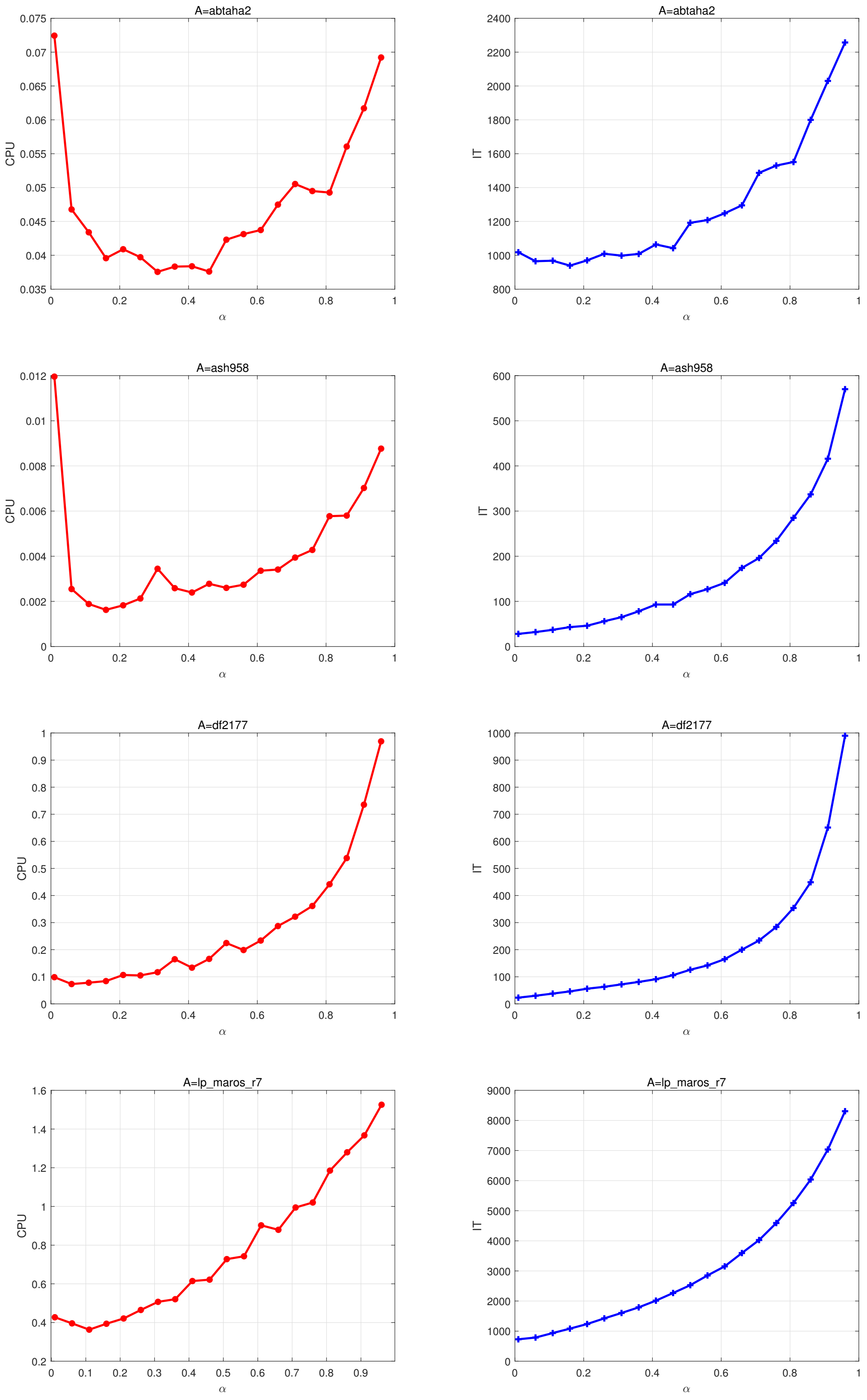

Figure 6. CPU and IT results for thin (`abtaha2, ash958`) and fat coefficient matrices (`df2177, lp_maros_r7`), with varying α for the VGBK algorithm.

4.2. Comparison of block Kaczmarz algorithms

In this subsection, the effectiveness of the VGBK algorithm is justified by comparing it with the GBK, FDBK and FGBK algorithms. Reminiscent of Section 4.1, we consider consistent linear system (1.1) with coefficient matrices adapted from the MATLAB built-in function `randn` or the SuiteSparse Matrix Collection [23]. The structure of this subsection is as follows. In Example 3, we consider solving overdetermined linear system (1.1). In Example 4, we look on solving underdetermined linear system (1.1). In Example 5, we touch upon solving linear system (1.1) with large coefficient matrices from real-world problems.

Example 3. *In this example, we are concerned with solving linear systems with thin coefficient matrices generated by the MATLAB built-in function* `randn`*. As recommended in Examples 1 and 2, we set* $s = \lfloor 0.8\%m \rfloor$ *and* $\alpha = 0.1$ *for the VGBK algorithm. The number of IT and CPU time for the GBK, FDBK, FGBK and VGBK algorithms are tabulated in Table 3. A rough conclusion is that FDBK, FGBK and VGBK outperform their prototype GBK in terms of CPU time; see, e.g., the case of* 10000×5000 *where the speed-up of VGBK against GBK is up to 13.8089. Moreover, VGBK surpasses all block Kaczmarz peers regarding CPU time, albeit with more iteration steps. This can be interpreted as follows. In the VGBK, we only need to work on a small linear system with just* $\lfloor 0.8\%m \rfloor$ *rows once the partition is applied, which dramatically reduces the computational cost (and thus CPU) per iteration. In general, the gain in CPU time reduction becomes more pronounced as m grows. It tends to give us a hint that the VGBK algorithm is suitable for solving large overdetermined linear systems. Finally, we explain how the comparison with the FGBK method is made. In fact, the speed-up3 is obtained by FGBK with less CPU time for either* $\alpha = 0.05$ *or* $\alpha = 0.1$ *as compared to that of VGBK; for example, speed-up3=* 2.2912 *is computed from a ratio of* 3.4606 *to* 1.5104 *in the case of* 10000×5000.

Table 3. Numerical results for overdetermined linear systems in Example 3.

$m \times n$	m	10000	12000	14000	16000	18000	20000
	n	5000	5000	5000	5000	5000	5000
GBK	IT	466	289	231	185	158	131
	CPU	20.8569	14.0318	11.7329	10.0255	8.9900	8.2728
FDBK	IT	489	307	229	177	152	136
	CPU	9.0201	6.6980	5.5091	4.7936	4.5030	4.5538
FGBK	IT	71	46	36	30	25	22
(α=0.05)	CPU	4.3284	3.2734	3.0066	2.7094	2.6355	2.6696
FGBK	IT	74	47	37	31	26	23
(α=0.1)	CPU	3.4606	2.6517	2.3205	2.2693	2.0589	1.9840
	IT	1522	1107	968	867	813	744
VGBK	CPU	1.5104	1.0828	0.8782	0.8355	0.7402	0.6693
	s	80	96	112	128	144	160
speed-up1		13.8089	12.9588	13.3602	11.9994	12.1454	12.3604
speed-up2		5.9720	6.1858	6.2732	5.7374	6.0835	6.8038
speed-up3		2.2912	2.4489	2.6423	2.7161	2.7815	2.9643

Example 4. *In this example, we consider solving linear systems with fat coefficient matrices generated by* `randn`*. The number of IT and CPU time for the GBK, FDBK, FGBK and VGBK algorithms are listed in Table 4. We choose* $s = \lfloor 4\%m \rfloor$ *and* $\alpha = 0.1$ *for VGBK, as suggested in Examples 1 and 2. For all test matrices, VGBK outperforms the other three algorithms regarding CPU time, with the speed-up ranging from* 1.0887 *to* 18.0378*. Besides, the speed-up becomes noticeable as m increases; for instance, the speed-up of VGBK against GBK reaches as high as* 18.0378 *for the case* 12000×15000*. It should be noted that FGBK renders a performance that is comparable to that of the VGBK when m is relatively small. As m increases, however, the VGBK algorithm has a remarkable advantage over FGBK; see the results for speed-up3 in Table 4.*

Table 4. Numerical results for underdetermined linear systems in Example 4.

$m \times n$	m	2000	4000	6000	8000	10000	12000
	n	15000	15000	15000	15000	15000	15000
GBK	IT	70	158	329	735	1628	5285
	CPU	4.6013	17.3493	48.2193	136.6090	323.2486	1156.3538
FDBK	IT	71	157	338	746	1664	5292
	CPU	1.2994	4.8180	13.8537	38.0042	104.2648	373.6110
FGBK	IT	13	24	45	90	189	573
(α=0.05)	CPU	0.5065	1.8356	5.0438	13.7018	34.3899	124.1716
FGBK	IT	15	26	47	92	195	589
(α=0.1)	CPU	0.4837	1.6001	4.3786	11.3237	29.5134	106.1159
VGBK	IT	628	1756	4218	10190	24687	82799
	CPU	0.4443	1.3651	3.2769	8.9984	19.1812	64.1071
	s	80	160	240	320	400	480
speed-up1		10.3563	12.7092	14.7149	15.1815	16.8524	18.0378
speed-up2		2.9246	3.5294	4.2277	4.2234	5.4358	5.8279
speed-up3		1.0887	1.1721	1.3362	1.2584	1.5387	1.6553

Example 5. *In this example, we shall compare the performance of the VGBK algorithm with those of GBK, FDBK and FGBK for solving linear systems with coefficient matrices from the SuiteSparse Matrix Collection whose properties are listed in Table 2. The corresponding numerical results are displayed in Tables 5 and 6. The effectiveness of VGBK is again verified regarding CPU time; for instance, the speed-up of VGBK against GBK, FDBK and FGBK varies from* 2.0647 *to* 15.0433, 2.2771 *to* 7.2730 *and* 1.2906 *to* 9.3184*, respectively. In this sense, the VGBK algorithm is very appealing for solving linear systems from real-world problems.*

Table 5. Numerical results for overdetermined linear systems in Example 5.

Test Matrix		abtaha1	abtaha2	ch7-9-b2	ch8-8-b2	Franz10	mk11-b3
GBK	IT	22	18	23	23	104	90
	CPU	0.0351	0.1121	0.6100	0.6007	5.3835	9.3379
FDBK	IT	101	83	39	43	198	95
	CPU	0.0954	0.2982	0.3535	0.4401	5.2831	4.0183
FGBK	IT	82	72	5	6	47	14
(α=0.05)	CPU	0.1400	0.4799	0.1658	0.1873	4.6478	1.6653
FGBK	IT	89	92	9	8	49	20
(α=0.1)	CPU	0.1245	0.4771	0.1701	0.1900	3.7400	1.8330
VGBK	IT	467	965	502	540	990	736
	CPU	0.0170	0.0512	0.0935	0.1017	0.7264	1.0474
	s	116	303	141	150	156	138
speed-up1		2.0647	2.1895	6.5241	5.9066	7.4112	8.9153
speed-up2		5.6118	5.8242	3.7807	4.3274	7.2730	3.8365
speed-up3		7.3235	9.3184	1.7733	1.8417	5.1487	1.5899

Table 6. Numerical results for underdetermined linear systems in Example 5.

Test Matrix		bibd_49_3	cat_ears_3_4	df2177	GL7d25	lp22	lp_maros_r7
GBK	IT	36	4045	41	57	1457	91
	CPU	2.4013	317.1875	0.6528	8.0641	61.2920	4.7440
FDBK	IT	37	4044	42	276	2447	92
	CPU	0.6679	112.6446	0.2235	7.7782	45.7603	1.4195
FGBK	IT	10	465	11	277	589	18
(α=0.05)	CPU	0.3229	29.8907	0.1087	7.4476	25.3378	0.7734
FGBK	IT	11	580	12	275	657	18
(α=0.1)	CPU	0.3400	30.8552	0.0978	7.0040	23.5825	0.6171
VGBK	IT	276	25568	159	3790	15019	912
	CPU	0.2502	21.0849	0.0686	3.4158	11.4409	0.3425
	s	47	209	25	111	118	125
speed-up1		9.5975	15.0433	9.5160	2.3608	5.3573	13.8511
speed-up2		2.6695	5.3424	3.2580	2.2771	3.9997	4.1445
speed-up3		1.2906	1.4176	1.4257	2.0505	2.0612	1.8018

5. Conclusions

By imposing a row partitioning, we put forth a VGBK algorithm for solving consistent linear systems. We establish theoretical results that characterize the convergence of the proposed algorithm. Numerical experiments demonstrate that the new algorithm offers good performance for a broad class of problems and outperforms other block Kaczmarz-type counterparts in terms of CPU time for large linear systems. The choice of partitioning approach is crucial for the performance of the proposed algorithm, and it can be regarded as important future work.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The authors are grateful to Miss Xiang-Xiang Chen of Shanghai Maritime University for helpful discussions. The authors are grateful to the referees for their insightful comments and valuable suggestions, which greatly improved the original version of this paper.

This work is supported by the National Natural Science Foundation of China (No. 12271342).

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. A. Agaskar, C. Wang, Y. M. Lu, *Randomized Kaczmarz algorithms: Exact MSE analysis and optimal sampling probabilities*, 2014 IEEE Global Conference on Signal and Information Processing (Global-SIP), 389–393. https://doi.org/10.1109/GlobalSIP.2014.7032145
2. Z. Z. Bai, X. G. Liu, On the Meany inequality with applications to convergence analysis of several row-action iteration methods, *Numer. Math.,* **124** (2013), 215–236. https://doi.org/10.1007/s00211-012-0512-6
3. Z. Z. Bai, L. Wang, On convergence rates of Kaczmarz-type methods with different selection rules of working rows, *Appl. Numer. Math.,* **186** (2023), 289–319. https://doi.org/10.1016/j.apnum.2023.01.013
4. Z. Z. Bai, W. T. Wu, On convergence rate of the randomized Kaczmarz method, *Linear Algebra Appl.,* **553** (2018), 252–269. https://doi.org/10.1016/j.laa.2018.05.009
5. Z. Z. Bai, W. T. Wu, On greedy randomized Kaczmarz method for solving large sparse linear systems, *SIAM J. Sci. Comput.,* **40** (2018), A592–A606. https://doi.org/10.1137/17M1137747
6. Z. Z. Bai, W. T. Wu, On relaxed greedy randomized Kaczmarz methods for solving large sparse linear systems, *Appl. Math. Lett.,* **83** (2018), 21–26. https://doi.org/10.1016/j.aml.2018.03.008
7. Z. Z. Bai, W. T. Wu, Randomized Kaczmarz iteration methods: Algorithmic extensions and convergence theory, *Jpn. J. Ind. Appl. Math.,* **40** (2023), 1421–1443. https://doi.org/10.1007/s13160-023-00586-7
8. E. Bodewig, Bericht über die verschiedenen Methoden zur Lösung eines System linearer Gleichungen mit reellen Koeffizienten, *Cons. Naz. Ric.,* **324** (1948), 441–452.
9. C. Byrne, A unified treatment of some iterative algorithms in signal processing and image reconstruction, *Inverse Probl.,* **20** (2004), 103–120. https://doi.org/10.1088/0266-5611/20/1/006
10. J. Q. Chen, Z. D. Huang, On a fast deterministic block Kaczmarz method for solving large-scale linear systems, *Numer. Algorithms,* **89** (2022), 1007–1029. https://doi.org/10.1007/s11075-021-01143-4

11. T. A. Davis, Y. Hu, The university of Florida sparse matrix collection, *ACM T. Math. Software,* **38** (2011), 1–25. https://doi.org/10.1145/2049662.2049663

12. K. Du, W. T. Si, X. H. Sun, Randomized extended average block Kaczmarz for solving least squares, *SIAM J. Sci. Comput.,* **42** (2020), A3541–A3559. https://doi.org/10.1137/20M1312629

13. Y. C. Eldar, D. Needell, Acceleration of randomized Kaczmarz methods via the Johnson-Lindenstrauss lemma, *Numer. Algorithms,* **58** (2011), 163–177. https://doi.org/10.1007/s11075-011-9451-z

14. T. Elfving, Block-iterative methods for consistent and inconsistent linear equations, *Numer. Math.,* **35** (1980), 1–12. https://doi.org/10.1007/BF01396365

15. G. E. Forsythe, Solving linear algebraic equations can be interesting, *Bull. Amer. Math. Soc.,* **59** (1953), 299–329.

16. R. Gordon, R. Bender, G. T. Herman, Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography, *J. Theoret. Biol.,* **29** (1970), 471–481. https://doi.org/10.1016/0022-5193(70)90109-8

17. R. M. Gower, D. Molitor, J. Moorman, D. Needell, On adaptive sketch-and-project for solving linear systems, *SIAM J. Matrix Anal. A.,* **42** (2021), 954–989. https://doi.org/10.1137/19M1285846

18. R. M. Gower, P. Richtárik, Randomized iterative methods for linear systems, *SIAM J. Matrix Anal. A.,* **36** (2015), 1660–1690. https://doi.org/10.1137/15M1025487

19. M. Griebel, P. Oswald, Greedy and randomized versions of the multiplicative Schwarz method, *Linear Algebra Appl.,* **437** (2012), 1596–1610. https://doi.org/10.1016/j.laa.2012.04.052

20. G. T. Herman, *Fundamentals of computerized tomography: Image reconstruction from projections*, 2 Eds., Springer, Dordrecht, 2009. https://doi.org/10.1007/978-1-84628-723-7

21. X. L. Jiang, K. Zhang, J. F. Yin, Randomized block Kaczmarz methods with k-means clustering for solving large linear systems, *J. Comput. Appl. Math.,* **403** (2022), 113828. https://doi.org/10.1016/j.cam.2021.113828

22. S. Kaczmarz, Angenäherte Auflösung von systemen linearer gleichungen, *Bull. Int. Acad. Pol. Sci. Lett. A,* **35** (1937), 355–357.

23. S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson et al., The SuiteSparse matrix collection website interface, *J. Open Source Softw.,* **4** (2019), 1244–1248. https://doi.org/10.21105/joss.01244

24. Y. Liu, C. Q. Gu, On greedy randomized block Kaczmarz method for consistent linear systems, *Linear Algebra Appl.,* **616** (2021), 178–200. https://doi.org/10.1016/j.laa.2021.01.024

25. C. Q. Miao, W. T. Wu, On greedy randomized average block Kaczmarz method for solving large linear systems, *J. Comput. Appl. Math.,* **413** (2022), 114372. https://doi.org/10.1016/j.cam.2022.114372

26. I. Necoara, Faster randomized block Kaczmarz algorithms, *SIAM J. Matrix Anal. A.,* **40** (2019), 1425–1452. https://doi.org/10.1137/19M1251643

27. D. Needell, J. A. Tropp, Paved with good intentions: Analysis of a randomized block Kaczmarz method, *Linear Algebra Appl.,* **441** (2014), 199–221. https://doi.org/10.1016/j.laa.2012.12.022

28. Y. Q. Niu, B. Zheng, A greedy block Kaczmarz algorithm for solving large-scale linear systems, *Appl. Math. Lett.,* **104** (2020), 106294. https://doi.org/10.1016/j.aml.2020.106294

29. T. Strohmer, R. Vershynin, A randomized Kaczmarz algorithm with exponential convergence, *J. Fourier Anal. Appl.,* **15** (2009), 262–278. https://doi.org/10.1007/s00041-008-9030-4

30. C. Tompkins, Projection methods in calculation of some linear problems, *Bull. Amer. Math. Soc.,* **55** (1949), 520.

31. L. Wen, F. Yin, Y. M. Liao, G. X. Huang, A greedy average block Kaczmarz method for the large scaled consistent system of linear equations, *AIMS Math.,* **7** (2022), 6792–6806. https://doi.org/10.3934/math.2022378

32. G. Wu, Q. Chang, Semi-randomized block Kaczmarz methods with simple random sampling for large-scale linear systems, *arXiv:2212.08797*, 2023. https://doi.org/10.48550/arXiv.2212.08797

33. A. Q. Xiao, J. F. Yin, N. Zheng, On fast greedy block Kaczmarz methods for solving large consistent linear systems, *Comput. Appl. Math.,* **42** (2023), 119. https://doi.org/10.1007/s40314-023-02232-x

34. F. Yin, B. Y. Zhang, G. X. Huang, A partially block randomized extended Kaczmarz method for solving large overdetermined inconsistent linear systems, *AIMS Math.,* **8** (2023), 18512–18527. https://doi.org/10.3934/math.2023941

35. K. Zhang, F. T. Li, X. L. Jiang, Multi-step greedy Kaczmarz algorithms with simple random sampling for solving large linear systems, *Comput. Appl. Math.,* **41** (2022), 332. https://doi.org/10.1007/s40314-022-02044-5

36. Y. Zhang, H. Li, Block sampling Kaczmarz-Motzkin methods for consistent linear systems, *Calcolo,* **58** (2021), 39. https://doi.org/10.1007/s10092-021-00429-2

37. Y. Zhang, H. Li, Randomized block subsampling Kaczmarz-Motzkin method, *Linear Algebra Appl.,* **667** (2023), 133–150. https://doi.org/10.1016/j.laa.2023.03.003