Math. Commun. **31** (2026), 179–195
DOI: 10.64785/mc.31.2.3

Mathematical Communications

The randomized block steepest descent iterative method for solving linear systems and its extension to the tensor framework

Fatemeh Panjeh Ali Beik[1,*] and Ke Zhang[2]

[1] Department of Mathematics, Vali-e-Asr University of Rafsanjan, P.O. Box 518, Rafsanjan, Iran
[2] Department of Mathematics, Shanghai Maritime University, Shanghai 201 306, China

Abstract. We present a new randomized block variant of the residual norm steepest descent iterative method for solving a consistent linear system of equations. The proposed algorithm is also applied both in matrix form for solving a class of matrix equations and in the t-product framework for solving a class of tensor equations. We prove that the residual norm of the computed approximate solutions converges linearly in expectation. Numerical experiments are reported to illustrate the validity of established results and superiority of the proposed approach over three recently examined methods for solving a linear system of equations with randomly generated coefficient matrices, a matrix equation and a tensor equation arising from color image restoration problems in the literature.

AMS subject classifications: 65F10
Keywords: linear system; randomized iterative method; steepest descent; convergence; t-product
Received February 20, 2026; accepted September 25, 2026

1. Introduction and preliminaries

Consider the consistent linear system of equations

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

where A is an $m \times n$ real-valued matrix ($m \geq n$) with full column rank ($\text{rank}(A) = n$), and the right-hand side vector $b \in \mathbb{R}^m$ is given. Our goal is to determine an approximation for the unique solution of normal equation $A^T Ax = A^T b$, i.e.,

$$x^* = (A^T A)^{-1} A^T b = A^\dagger b.$$

Here $A^\dagger$ denotes the Moore–Penrose pseudoinverse of A, and the superscript T denotes the transpose operation.

Randomized variants of Kaczmarz type and residual coordinate descent iterative methods are among popular approaches for solving (1), see [3, 4, 12, 22, 23, 28, 29]. Alternative variants of randomized Kaczmarz have been further investigated, e.g., the relaxed greedy randomized Kaczmarz method [2], the accelerated randomized Kaczmarz method [14], and the randomized block Kaczmarz methods [8, 7, 13, 15, 20, 25, 30]. In particular, a randomized block residual steepest descent method with k-means clustering (RBRSD) was developed in [13] for solving a consistent linear system of equations of the form (1). A key advantage of the RBRSD method, compared with the previously examined approach in [8], as well as similar iterative methods in references therein, is relaxing the requirement to compute the Moore-Penrose pseudoinverse of sub-matrices. More recently, Li et al. [12] developed a Randomized Achlioptas Block Residual Steepest Descent method with k-means clustering (RABRSD-k) for solving a consistent linear system of equations (1), see Algorithm 1 for more details[1]. Indeed, using the Achlioptas random projection [1], the authors employed a dimensionality reduction technique in Line 4 of Algorithm 1 to further improve

Email addresses: f.beik@vru.ac.ir (F. P. A. Beik), kezhang@shmtu.edu.cn (K. Zhang)
*Corresponding author.

[1] Throughout this paper, the symbol $\|\cdot\|$ denotes the 2-norm for vectors/matrices, while the notation $\|\cdot\|_F$ refers to the Frobenius norm of matrices/tensors. Given a set of natural numbers θ and a matrix Z, the notations Z_θ, $Z(\theta, :)$, and $Z^{(\theta)}$ all denote the submatrix of Z formed by extracting the rows whose indices belong to θ.

© 2026 School of Applied Mathematics and Informatics, University of Osijek

the computational costs of the RBRSD method. Additionally, several randomized Kaczmarz type methods have been developed for approximating the solution of the consistent linear matrix equation

$$AXB = C,$$

where $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, and $C \in \mathbb{R}^{m\times q}$ are given, and $X \in \mathbb{R}^{n\times p}$ is the unknown matrix, see [19, 24] and the references therein. The above matrix equation appears in a variety of applications, including image processing and control theory, see [24, 26, 27] and the references therein for further details. In numerical experiments, we examine a test problem corresponding to the forward model for noise-free color image restoration, which takes the form

$$AXA_c^T = C, \tag{2}$$

as discussed in [24, 27]. For this specific application, A is typically a large square matrix, while A_c is a small 3×3 matrix.

Inspired by the randomized method based on the row/column action for solving (1), several iterative methods have been recently proposed for solving the following tensor equation:

$$\mathcal{A} * \mathcal{X} = \mathcal{B}, \tag{3}$$

where $\mathcal{A} \in \mathbb{R}^{m\times n\times p}$ and $\mathcal{B} \in \mathbb{R}^{m\times 1\times p}$ are given, and $\mathcal{X} \in \mathbb{R}^{n\times 1\times p}$ is unknown, see [6, 16, 17] and the references therein. Here, the t-product is denoted by "$*$" as a tensor-tensor multiplication which was first introduced in [10], and it was found to be a useful tool when working with high-dimensional data in comparison with matricization, see [10, 9].

Algorithm 1 The RABRSD Method [12]

1: **Input:** A, b, k, ϵ, initial guess x_0 and $\text{iter}_{\max}$
2: **Output:** Approximate solution of (1)
3: Set $d = \left\lceil \frac{\log_2(n)}{\epsilon^2} \right\rceil$
4: Generate an Achlioptas random projection matrix $T_A \in \mathbb{R}^{d\times m}$ to obtain $\hat{A} = T_A A$, $\hat{b} = T_A b$.
5: Apply the clustering method to $\hat{A}$ and $\hat{b}$ to get a partition $\{\tau_1, \ldots, \tau_k\}$ of the row indices $\{1, \ldots, d\}$, yielding k groups of $\hat{A}_{\tau_i}$ and $\hat{b}_{\tau_i}$ for $i = 1, \ldots, k$.
6: **for** $j = 0, 1, \ldots, \text{iter}_{\max}$ **do**
7: Select $\tau_j \in \{\tau_1, \tau_2, \ldots, \tau_k\}$ with probability $\Pr = \frac{\|\hat{A}_{\tau_j}\|_F^2}{\|\hat{A}\|_F^2}$
8: Compute $\hat{r}_{\tau_j} = \hat{b}_{\tau_j} - \hat{A}_{\tau_j} x_j$
9: Update

$$x_{j+1} = x_j + \frac{\|\hat{r}_{\tau_j}\|^2}{\|\hat{A}_{\tau_j}^T \hat{r}_{\tau_j}\|^2} \hat{A}_{\tau_j}^T \hat{r}_{\tau_j}$$

10: **end for**

In this paper, we study the performance of a new variant of randomized block residual steepest descent method. The proposed approach exploits a similar probability distribution as in [13], but utilizes a different orthogonality condition (see [21, Chapter 5]) which corresponds to the minimizing norm of a sub-block of the residual vector. Let π denote a permutation on $\{1, 2, \ldots, m\}$ obtained by uniformly sampling the indices from $\{1, 2, \ldots, m\}$. In the proposed method, the working blocks are selected from the random partition

$$T = \{\tau_j\}_{j=1}^{\ell}, \quad \tau_j = \left\{\pi(k) \ : \ k = \left\lfloor \frac{(j-1)m}{\ell} \right\rfloor + 1, \left\lfloor \frac{(j-1)m}{\ell} \right\rfloor + 2, \ldots, \left\lfloor \frac{jm}{\ell} \right\rfloor \right\}, \tag{4}$$

instead of clustering. It is known that the above partition divides the rows of A into ℓ blocks whose singular values have a moderate upper bound with high probability; see [18] for more details. It is worth mentioning that, Guo et al. [5] have recently exploited partition (4) for column selection in a greedy quasi block coordinate descent method.

The remainder of this paper is organized as follows. Section 2 introduces a new randomized block approach for solving (1), called the RBRNSD method, which utilizes a one-dimensional minimum residual

norm technique at each iterate. Moreover, the matrix form of the RBRNSD method is implemented for solving (2). In Section 3, the RBRNSD method is developed for solving (3) as well. The disclosed numerical results in Section 4 for test problems, taken from [12, Example 3.1], [24, Experiment 6.2] and [16, Subsection 5.2], illustrate the effectiveness and feasibility of our method in comparison with three recently examined approaches in the literature employed for solving (1), (2), and (3), respectively. Finally, some concluding remarks are given in Section 5.

2. RBRNSD: The Randomized Block Residual Norm Steepest Descent Method

In this section, we discuss the implementation of the RBRNSD approach for solving the consistent linear system (1). Given a prescribed integer $\ell > 0$, throughout this section, we consider the partition T (see (4)) on the rows of A. At the j-th step, the new approximate solution is obtained as follows:

$$x_{j+1} = x_j + \bar{\alpha}_j A_{\tau_j}^T r_j^{(\tau_j)}, \tag{5}$$

where $r_j^{(\tau_j)} = b_{\tau_j} - A_{\tau_j} x_j$ and

$$\bar{\alpha}_j = \frac{\left\langle r_j^{(\tau_j)},\ A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)} \right\rangle}{\|A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)}\|^2}, \tag{6}$$

provided that $\|A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)}\|$ is not zero, where $\tau_j \in T$ is randomly chosen based on a specific probability distribution. The proposed approach is summarized in Algorithm 2.

Evidently, for $k = 1, 2, \ldots, \ell$, one can observe that

$$\left\langle r_j^{(\tau_k)},\ A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)} \right\rangle = \left\langle A_{\tau_k}^T r_j^{(\tau_k)},\ A_{\tau_k}^T r_j^{(\tau_k)} \right\rangle = \|A_{\tau_k}^T r_j^{(\tau_k)}\|^2.$$

The above relation illustrates that if $\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\| = 0$, then $\|A_{\tau_k}^T r_j^{(\tau_k)}\| = 0$. Thus, the condition $\|A_{\tau_k}^T r_j^{(\tau_k)}\| \neq 0$ ensures $\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\| \neq 0$.

It is worth mentioning that if $A_{\tau_k}^T r_j^{(\tau_k)}$ is also zero for $k = 1, 2, \ldots, \ell$, then $A^T r_j$ is zero. In fact, we have $A^T r_j = \sum_{k=1}^{\ell} A_{\tau_k}^T r_j^{(\tau_k)}$. Notice that $A^T r_j = 0$ is equivalent to $A^T A x_j = A^T b$, i.e., x_j is the exact solution.

Algorithm 2 The Randomized Block Residual Norm Steepest Descent (RBRNSD) Method

Input: A, b, partition T on the rows of A, initial guess x_0 and $\text{iter}_{\max}$
Output: Approximate solution of (1)
Set $r_0 = b - Ax_0$
for $j = 0, 1, \ldots, \text{iter}_{\max}$ **do**
  Select $\tau_j \in T$ with probability $\Pr = \frac{\|A_{\tau_j}\|_F^2}{\|A\|_F^2}$ where T is defined in (4)
  Compute $r_j^{(\tau_j)} = b_{\tau_j} - A_{\tau_j} x_j$
  if $\|A_{\tau_j}^T r_j^{(\tau_j)}\| \neq 0$ **then**
    Update

$$x_{j+1} = x_j + \frac{\|A_{\tau_j}^T r_j^{(\tau_j)}\|^2}{\|A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)}\|^2} A_{\tau_j}^T r_j^{(\tau_j)}$$

  end if
end for

2.1. Convergence analysis

In this part, we study the convergence behavior of the RBRNSD method for solving the consistent linear system of equations (1). We show that the residual norm of approximate solutions computed by the proposed algorithm converges linearly in expectation. To this end, first, we present the following proposition.

Proposition 1. *Let the linear system (1) be consistent and x_j be an approximate solution for Eq. (1) with the corresponding residual $r_j = b - Ax_j$. Given a prescribed integer $\ell > 0$, consider partition (4) on the rows of A. Assume that $\mathcal{N}_j$ is a subset of $\{1, 2, \ldots, \ell\}$ defined as follows:*

$$\mathcal{N}_j = \{k \mid \|A_{\tau_k}^T r_j^{(\tau_k)}\| \neq 0\}. \tag{7}$$

Then, the following inequality holds:

$$\|A_{\tau_k}^T r_j^{(\tau_k)}\|^2 \geq \bar{\sigma}_{\min}^2(A_{\tau_k})\|r_j^{(\tau_k)}\|^2 \quad \text{for} \quad k \in \mathcal{N}_j,$$

where $\bar{\sigma}_{\min}(A_{\tau_k})$ denotes the smallest positive singular value of A_{τ_k}.

Proof. Let $\mu_k = \operatorname{rank}(A_{\tau_k})$ and $\sigma_1^{(\tau_k)} \geq \sigma_2^{(\tau_k)} \geq \ldots \geq \sigma_{\mu_k}^{(\tau_k)} > 0$ be the nonzero singular values of A_{τ_k} ($\bar{\sigma}_{\min}(A_{\tau_k}) = \sigma_{\mu_k}^{(\tau_k)}$). Let $r_j^{(\tau_k)}$ denote a sub-block of r_j indexed by a set of indices τ_k, or equivalently that $r_j^{(\tau_k)} = b_{\tau_k} - A_{\tau_k} x_j$. Consider a singular value decomposition of A_{τ_k}, i.e., $A_{\tau_k} = U_{\tau_k}\Sigma_{\tau_k}V_{\tau_k}^T$. Therefore, we conclude that

$$A_{\tau_k}^T r_j^{(\tau_k)} = \sum_{\nu=1}^{\mu_k} (u_{\tau_k,\nu}^T r_j^{(\tau_k)})\sigma_\nu^{(\tau_k)} v_{\tau_k,\nu},$$

where $u_{\tau_k,\nu}$ and $v_{\tau_k,\nu}$ are the ν-th columns of U_{τ_k} and V_{τ_k}, respectively. It is known that the set of first μ_k columns of U_{τ_k} forms a basis for the column space of A_{τ_k}. Since b belongs to the column space of A, we deduce that $r_j^{(\tau_k)}$ belongs to the column space of A_{τ_k}. As a result, one can deduce that

$$\|A_{\tau_k}^T r_j^{(\tau_k)}\|^2 \geq \bar{\sigma}_{\min}^2(A_{\tau_k}) \sum_{\nu=1}^{\mu_k} (u_{\tau_k,\nu}^T r_j^{(\tau_k)})^2 = \bar{\sigma}_{\min}^2(A_{\tau_k})\|r_j^{(\tau_k)}\|^2,$$

which completes the proof. □

In view of (5), it is observed that

$$r_{j+1}^{(\tau_j)} = b_{\tau_j} - A_{\tau_j} x_{j+1} = r_j^{(\tau_j)} - \bar{\alpha}_j A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)}.$$

Invoking Eq. (6), we deduce that

$$\left\langle r_{j+1}^{(\tau_j)},\ A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)} \right\rangle = 0.$$

Thus, we derive

$$\begin{aligned} \|r_{j+1}^{(\tau_j)}\|^2 &= \left\langle r_{j+1}^{(\tau_j)},\ r_{j+1}^{(\tau_j)} \right\rangle = \left\langle r_{j+1}^{(\tau_j)},\ r_j^{(\tau_j)} - \bar{\alpha}_j A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)} \right\rangle \\ &= \left\langle r_j^{(\tau_j)} - \bar{\alpha}_j A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)},\ r_j^{(\tau_j)} \right\rangle \\ &= \left\langle r_j^{(\tau_j)},\ r_j^{(\tau_j)} \right\rangle - \bar{\alpha}_j \left\langle r_j^{(\tau_j)},\ A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)} \right\rangle. \end{aligned}$$

Substituting $\bar{\alpha}_j$ from (6) into the preceding relation, we obtain

$$\|r_{j+1}^{(\tau_j)}\|^2 = \left\langle r_j^{(\tau_j)},\ r_j^{(\tau_j)} \right\rangle - \frac{\left\langle r_j^{(\tau_j)},\ A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)} \right\rangle^2}{\|A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)}\|^2} = \|r_j^{(\tau_j)}\|^2 - \frac{\|A_{\tau_j}^T r_j^{(\tau_j)}\|^4}{\|A_{\tau_j} A_{\tau_j}^T r_j^{(\tau_j)}\|^2}.$$

Theorem 1. *Let the linear system (1) be consistent. If x_j is the j-th computed approximate solution by Algorithm 2 with the corresponding residual $r_j = b - Ax_j$. Then, there exists $0 < \tilde{l}_j < 1$ such that*

$$\mathbb{E}\|r_{j+1}\|^2 \leq (1 - \tilde{l}_j)\|r_j\|^2.$$

Proof. For ease in notation, we set

$$\tilde{p}_k = \frac{\|A_{\tau_k}\|_F^2}{\|A\|_F^2}, \quad k = 1, 2, \ldots, \ell. \tag{8}$$

By definition, for $t = 1, 2, \ldots$, it is known that $\mathbb{E}\|r_t\|^2 = \sum_{k=1}^{\ell} \|r_t^{(\tau_k)}\|^2 \tilde{p}_k$. Let $\mathcal{N}_j$ be defined by (7), in view of (2.1) and noting that $\tilde{p}_k \leq 1$, we conclude

$$\begin{aligned}
\mathbb{E}\|r_{j+1}\|^2 &\leq \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \sum_{k\in\mathcal{N}_j} \frac{\|A_{\tau_k}^T r_j^{(\tau_k)}\|^4}{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2} \tilde{p}_k \\
&= \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \frac{1}{\|A\|_F^2} \cdot \sum_{k\in\mathcal{N}_j} \frac{\|A_{\tau_k}^T r_j^{(\tau_k)}\|^4}{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2} \|A_{\tau_k}\|_F^2.
\end{aligned}$$

It is known that

$$\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\| \leq \|A_{\tau_k}\|_F \|A_{\tau_k}^T r_j^{(\tau_k)}\|.$$

Therefore, from the last relation, we can deduce that

$$\begin{aligned}
\mathbb{E}\|r_{j+1}\|^2 &\leq \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \frac{1}{\|A\|_F^2} \cdot \sum_{k\in\mathcal{N}_j} \frac{\|A_{\tau_k}^T r_j^{(\tau_k)}\|^2}{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2} \|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|_2^2 \\
&= \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \frac{1}{\|A\|_F^2} \cdot \sum_{k\in\mathcal{N}_j} \|A_{\tau_k}^T r_j^{(\tau_k)}\|^2.
\end{aligned}$$

Consequently, Proposition 1 implies that

$$\begin{aligned}
\mathbb{E}\|r_{j+1}\|^2 &\leq \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \frac{1}{\|A\|_F^2} \cdot \sum_{k\in\mathcal{N}_j} \bar{\sigma}_{\min}^2(A_{\tau_k}) \|r_j^{(\tau_k)}\|^2 \\
&\leq \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \frac{\min\limits_{k\in\mathcal{N}_j} \{\bar{\sigma}_{\min}^2(A_{\tau_k})\}}{\|A\|_F^2} \cdot \frac{\sum\limits_{k\in\mathcal{N}_j} \|r_j^{(\tau_k)}\|^2}{\|r_j\|^2} \|r_j\|^2 \\
&= \left(1 - \frac{\min\limits_{k\in\mathcal{N}_j} \{\bar{\sigma}_{\min}^2(A_{\tau_k})\}}{\|A\|_F^2} \cdot \frac{\sum\limits_{k\in\mathcal{N}_j} \|r_j^{(\tau_k)}\|^2}{\|r_j\|^2}\right) \|r_j\|^2,
\end{aligned}$$

so we can conclude the assumption with

$$\tilde{l}_j := \frac{\min\limits_{k\in\mathcal{N}_j} \{\bar{\sigma}_{\min}^2(A_{\tau_k})\}}{\|A\|_F^2} \cdot \frac{\sum\limits_{k\in\mathcal{N}_j} \|r_j^{(\tau_k)}\|^2}{\|r_j\|^2},$$

which completes the proof. □

Remark 1. By Theorem 1, it can be observed that

$$\mathbb{E}\|r_t\|^2 = \sum_{k=1}^{\ell} \|r_t^{(\tau_k)}\|^2 \tilde{p}_k = \sum_{k=1}^{\ell} \|r_t^{(\tau_k)}\|^2 \cdot \frac{\|A_{\tau_k}^T\|_F^2}{\|A\|_F^2} \geq \frac{1}{\|A\|_F^2} \cdot \sum_{k=1}^{\ell} \|A_{\tau_k}^T r_t^{(\tau_k)}\|^2 \geq 0.$$

Therefore, if $\mathbb{E}\|r_t\|^2 \to 0$ as $t \to \infty$, then $\lim\limits_{t\to\infty} A_{\tau_k}^T r_t^{(\tau_k)} = 0$ for $k = 1, 2, \ldots, \ell$. Consequently, one can see that $\lim\limits_{k\to\infty} A^T r_t = 0$, bearing in mind that $A^T r_t = \sum_{k=1}^{\ell} A_{\tau_k}^T r_t^{(\tau_k)}$.

Remark 2. Note that the working block is chosen at each step of Algorithm 2 provided that the condition in Line 7 of the algorithm is satisfied. It seems that a similar condition should have been set in Line 9 of Algorithm 1 as well.

As an alternative, when the condition in Line 7 of Algorithm 2 is not met, the working block can be selected according to the following probability distribution:

$$\hat{\tilde{p}}_k = \frac{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2}{\sum_{k=1}^{\ell} \|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2}, \quad k = 1, 2, \ldots, \ell. \tag{9}$$

Let $\mathcal{S}_j = \{k \mid \|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2 \neq 0\}$. Evidently, we have $\hat{\tilde{p}}_k = 0$ for $k \notin \mathcal{S}_j$. Thus, by Eq. (2.1), we deduce that

$$\mathbb{E}\|r_{j+1}\|^2 = \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 \hat{\tilde{p}}_k - \sum_{k\in\mathcal{S}_j} \frac{\|A_{\tau_k}^T r_j^{(\tau_k)}\|^4}{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2} \hat{\tilde{p}}_k. \tag{10}$$

By definition of $\hat{\tilde{p}}_k$ in (9), one can verify that

$$\frac{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2}{\|r_j\|^2 \max\limits_{1\le k\le \ell} \{\sigma_{\max}^4(A_{\tau_k})\}} \le \hat{\tilde{p}}_k \le 1.$$

Consequently, invoking Eq. (10), we derive

$$\begin{aligned}
\mathbb{E}\|r_{j+1}\|^2 &\le \sum_{k=1}^{\ell} \|r_j^{(\tau_k)}\|^2 - \frac{1}{\|r_j\|^2 \max\limits_{1\le k\le \ell} \{\sigma_{\max}^4(A_{\tau_k})\}} \cdot \sum_{k\in\mathcal{S}_j} \frac{\|A_{\tau_k}^T r_j^{(\tau_k)}\|^4}{\|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2} \|A_{\tau_k} A_{\tau_k}^T r_j^{(\tau_k)}\|^2 \\
&= \|r_j\|^2 - \frac{1}{\|r_j\|^2 \max\limits_{1\le k\le \ell} \{\sigma_{\max}^4(A_{\tau_k})\}} \cdot \sum_{k\in\mathcal{S}_j} \|A_{\tau_k}^T r_j^{(\tau_k)}\|^4.
\end{aligned}$$

In view of Proposition 1, from the preceding relation, we deduce that

$$\begin{aligned}
\mathbb{E}\|r_{j+1}\|^2 &\le \|r_j\|^2 - \frac{\min\limits_{k\in\mathcal{S}_j} \{\bar{\sigma}_{\min}^4(A_{\tau_k})\}}{\|r_j\|^2 \max\limits_{1\le k\le \ell} \{\sigma_{\max}^4(A_{\tau_k})\}} \cdot \sum_{k\in\mathcal{S}_j} \|r_j^{(\tau_k)}\|^4 \\
&= \|r_j\|^2 - \frac{\min\limits_{k\in\mathcal{S}_j} \{\bar{\sigma}_{\min}^4(A_{\tau_k})\}}{\max\limits_{1\le k\le \ell} \{\sigma_{\max}^4(A_{\tau_k})\}} \cdot \frac{\sum\limits_{k\in\mathcal{S}_j} \|r_j^{(\tau_k)}\|^4}{\|r_j\|^4} \|r_j\|^2 = (1 - \hat{\tilde{l}}_j)\|r_j\|^2
\end{aligned}$$

with

$$\hat{\tilde{l}}_j := \frac{\min\limits_{k\in\mathcal{S}_j} \{\bar{\sigma}_{\min}^4(A_{\tau_k})\}}{\max\limits_{1\le k\le \ell} \{\sigma_{\max}^4(A_{\tau_k})\}} \cdot \frac{\sum\limits_{k\in\mathcal{S}_j} \|r_j^{(\tau_k)}\|^4}{\|r_j\|^4}.$$

In our numerical experiments for solving Example 1, it was observed that the condition in Line 7 of Algorithm 2 was satisfied at every iteration. Consequently, it was not necessary to employ the probability distribution (9). In general, using the probability distribution (8) is computationally more efficient, as the probabilities can be calculated once outside the iteration loop and do not need to be updated at each iterate.

2.2. RBRSD method for $AXA_c^T = C$: An application to color image restoration

Evidently, as a direct consequence of the Kronecker product properties, the matrix equation (2) is equivalent to the following linear system:[2]

$$(A_c \otimes A)\mathrm{vec}(X) = \mathrm{vec}(C).$$

[2]The vectorization of an $n \times p$ matrix $X = [x_{ij}]_{n\times p}$, denoted by vec(X), is the $np \times 1$ vector obtained by stacking the columns of the matrix X on top of one another, i.e., vec(X)=$[x_{11}, \ldots, x_{n1}, x_{12}, \ldots, x_{n2}, \ldots, x_{1p}, \ldots, x_{np}]^T$.

Therefore, one can implement Algorithm 3 as a possible matrix version of Algorithm 2.

We aim to numerically compare the performance of Algorithm 3 with a recently proposed method in [24]. Based on the reported experimental results in [24], it can be seen that the proposed maximal weighted residual block Kaczmarz (MWRBK) method (see Algorithm 4) outperforms the other examined approaches. It is noted that, in the MWRBK method, the term "block" refers to the presence of A_c, rather than the meaning of working with A. In Algorithm 3, we select row blocks of A while keeping the entire matrix A_c similar to MWRBK. The numerical experiments in [24, Table 12] show that the examined approaches based on single-row selection of A require many iterations and a high CPU time to converge. Here, it can be numerically seen that Algorithm 3, which uses row block selection, substantially outperforms the MWRBK method.

Algorithm 3 The RBRNSD Method for solving $AXA_c^T = C$

Input: A, A_c, C, partition T on the rows of A, initial guess X_0 and $\text{iter}_{\max}$
Output: Approximate solution of (2)
Set $R_0 = C - AX_0A_c^T$
for $j = 0, 1, \ldots, \text{iter}_{\max}$ **do**
  Select $\tau_j \in T$ with probability $\Pr = \frac{\|A_{\tau_j}\|_F^2}{\|A\|_F^2}$ where T is defined in (4)
  Compute $U_j = A_{\tau_j}^T R_j^{(\tau_j)} A_c$.
  if $\|U_j\|_F \neq 0$ **then**
    Update

$$X_{j+1} = X_j + \frac{\|U_j\|_F^2}{\|A_{\tau_j} U_j A_c^T\|^2} U_j$$

$$R_{j+1} = R_j - \frac{\|U_j\|_F^2}{\|A_{\tau_j} U_j A_c^T\|^2} A U_j A_c^T$$

  end if
end for

Algorithm 4 The MWRBK method [24]

Input: A, A_c, C, initial guess X_0, scalar α and $\text{iter}_{\max}$
Output: Approximate solution of (2)
Set $R_0 = C - AX_0A_c^T$
for $k = 0, 1, 2, \ldots, \text{iter}_{\max}$ **do**
  Let i_k be the index maximizing $\frac{\|R_k(i_k, :)\|^2}{\|A_{i_k}\|^2}$; if there are multiple, choose the smallest
  Update $X_{k+1} = X_k + \frac{\alpha}{\|A_{i_k}\|^2} A_{i_k}^T R_k(i_k, :) A_c$
  Compute $R_{k+1} = R_k - \frac{\alpha}{\|A_{i_k}\|^2} A(A_{i_k}^T R_k(i_k, :) A_c) A_c^T$
end for

In Section 4, we compare the performance of the preceding two algorithms to solve (2) for color image restoration. Therefore, we end this part by briefly describing the considered experiments in [24], i.e., the forward model for color image restoration [27]. The original (clean) color image $\mathcal{X}$ and the observed (blurred) color image $\mathcal{C}$ are each represented as three-dimensional tensors. The red (R), green (G), and blue (B) channels correspond to the matrices

$$X_r := \mathcal{X}(:, :, 1), \qquad X_g := \mathcal{X}(:, :, 2), \qquad X_b := \mathcal{X}(:, :, 3),$$

and analogously,

$$C_r := \mathcal{C}(:, :, 1), \qquad C_g := \mathcal{C}(:, :, 2), \qquad C_b := \mathcal{C}(:, :, 3).$$

Suppose that

$$X = \begin{bmatrix} \text{vec}(X_r) & \text{vec}(X_g) & \text{vec}(X_b) \end{bmatrix},$$

and

$$C = \begin{bmatrix} \text{vec}(C_r) & \text{vec}(C_g) & \text{vec}(C_b) \end{bmatrix}.$$

The forward model is then given as follows:

$$C = AXA_c^T + E,$$

where A models the within-channel blurring, $A_c \in \mathbb{R}^{3\times 3}$ models the cross-channel blurring, and E stands for the additive noise. In the noise-free setting, where E is zero, the model reduces to the matrix equation (2). Similar to Wang et al. [24], here we consider the noise-free case.

3. Developing the RBRSD method in the t-product framework

Throughout this paper, the third-order tensors are denoted by calligraphic letters, e.g., $\mathcal{X}$, which are members of $\mathbb{R}^{m\times n\times p}$. The $|\tau_k|\times n\times p$ sub-tensor associated with $\mathcal{X}$ is denoted by $\mathcal{X}_{\tau_k::}$ or $\mathcal{X}(\tau_k,:,:)$, which is a block slice along the first mode including all elements whose first index belongs to τ_k.

For a tensor $\mathcal{X} \in \mathbb{R}^{m\times n\times p}$, we denote by $X_1, X_2, \ldots, X_p$ the p frontal slices of $\mathcal{X}$ and associate to $\mathcal{X}$ the block-circulant matrix:

$$\text{bcirc}(\mathcal{X}) = \begin{pmatrix} X_1 & X_p & X_{p-1} & \ldots & X_2 \\ X_2 & X_1 & X_p & \ldots & X_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ X_p & X_{p-1} & \ddots & X_2 & X_1 \end{pmatrix} \in \mathbb{R}^{mp\times np}.$$

Adopting the notations in [9], we also denote the unfold operation by *Ufold* and its inverse by *Fold*:

$$Ufold(\mathcal{X}) = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{pmatrix} \in \mathbb{R}^{mp\times n}, \qquad Fold(Ufold(\mathcal{X})) = \mathcal{X}.$$

Definition 1. [9] The **t-product**, denoted by $*$, between two tensors $\mathcal{X} \in \mathbb{R}^{m\times n\times p}$ and $\mathcal{Y} \in \mathbb{R}^{n\times \ell\times p}$ is the following $m\times \ell\times p$ tensor:

$$\mathcal{X} * \mathcal{Y} = Fold(\text{bcirc}(\mathcal{X})Ufold(\mathcal{Y})).$$

Evidently, Eq. (3) is equivalent to the following multiple linear system of equations:

$$\text{bcirc}(\mathcal{A})Ufold(\mathcal{X}) = Ufold(\mathcal{B}). \tag{11}$$

In our numerical experiments, we do not implement Algorithm 2 on (11) directly. Instead, we extend the method in the t-product framework and apply partition (4) on the first mode of tensor. To do so, first, we recall the definition of the transpose of a third-order tensor, see [10] for more details.

Definition 2. Let $\mathcal{A} \in \mathbb{R}^{m\times n\times p}$. The notation $\mathcal{A}^T$ stands for the $n\times m\times p$ tensor obtained by first transposing each frontal slice (from front to back) and then reversing the order of the transposed frontal slices from the second to the p-th.

The RBRNSD method based on tensor form is summarized in Algorithm 5. Considering the selection of a working block at each iterate, it should be mentioned that Algorithm 5 is not necessarily equivalent to Algorithm 3 for solving (11). Nevertheless, the following proposition can be proved using similar strategies used in Theorem 1.

We recall that the Frobenius inner product of two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{m\times n\times p}$ is given by (see [11])

$$\langle \mathcal{X}, \mathcal{Y} \rangle_F = \sum_{i_1=1}^{m} \sum_{i_2=1}^{n} \sum_{i_3=1}^{p} x_{i_1 i_2 i_3} y_{i_1 i_2 i_3}.$$

The proof of the following proposition follows similar techniques used for proving Theorem 1, so its proof is omitted.

Proposition 2. *Let the tensor equation (3) be consistent and $\mathcal{X}_j$ be the j-th computed approximate solution by Algorithm 5 with the corresponding residual $\mathcal{R}_j = \mathcal{B} - \mathcal{A} * \mathcal{X}_j$. There exists $0 < \hat{l}_j < 1$ such that*

$$\mathbb{E}\|\mathcal{R}_{j+1}\|_F^2 \leq (1 - \hat{l}_j)\|\mathcal{R}_j\|_F^2.$$

Algorithm 5 The RBRNSD Method in the t-product framework

Input: $\mathcal{A}, \mathcal{B}$ and partition T on horizontal slices of $\mathcal{A}$, initial guess $\mathcal{X}_0$ and $\text{iter}_{\max}$
Output: Approximate solution of (3)
Set $\mathcal{R}_0 = \mathcal{B} - \mathcal{A} * \mathcal{X}_0$
for $j = 0, 1, \ldots, \text{iter}_{\max}$ **do**
  Select $\tau_j \in T$ with probability $\Pr = \frac{\|\mathcal{A}_{\tau_j ::}\|_F^2}{\|\mathcal{A}\|_F^2}$
  Compute $\mathcal{R}_j^{(\tau_j)} = \mathcal{B}_{\tau_j ::} - \mathcal{A}_{\tau_j ::} * \mathcal{X}_j$
  Set $\mathcal{P}_j = \mathcal{A}_{\tau_j ::}^T * \mathcal{R}_j^{(\tau_j)}$
  if $\|\mathcal{P}_j\|_F \neq 0$ **then**
    Set $\mathcal{AP}_j = \mathcal{A}_{\tau_j ::} * \mathcal{P}_j$
    $\alpha_j = \|\mathcal{P}_j\|_F^2 / \|\mathcal{AP}_j\|_F^2$
    Update
$$\mathcal{X}_{j+1} = \mathcal{X}_j + \alpha_j \mathcal{A}_{\tau_j ::}^T * \mathcal{R}_j^{(\tau_j)}$$
  end if
end for

Assume that F_p denotes the $p \times p$ (normalized) discrete Fourier transform matrix. Given $\mathcal{A} \in \mathbb{R}^{m\times n\times p}$, it is known that there exist p matrices $\hat{A}_i \in \mathbb{C}^{m\times n}$ (for $i = 1, 2, \ldots, p$), possibly with complex entries, see [10], such that

$$(F_p \otimes I_m)\text{bcirc}(\mathcal{A})(F_p^H \otimes I_n) = \begin{pmatrix} \hat{A}_1 & 0 & \cdots & 0 \\ 0 & \hat{A}_2 & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & \hat{A}_p \end{pmatrix}.$$

There is no need to explicitly form $\text{bcirc}(\mathcal{A})$ to determine $\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_p$. Suppose that the function $\text{fft}(\mathcal{A}, [], 3)$ computes the fast Fourier transform (FFT) on the third dimension of $\mathcal{A}$, and its inverse is denoted as $\text{ifft}(\cdot, [], 3)$, i.e., $\text{ifft}\,(\text{fft}(\mathcal{A}, [], 3), [], 3) = \mathcal{A}$. The block matrices $\hat{A}_i$, for each i, are equal to $\hat{\mathcal{A}}(:, :, i)$, where $\hat{\mathcal{A}} = \text{fft}(\mathcal{A}, [], 3)$. Thus, for given tensors $\mathcal{A}$ and $\mathcal{R}$, the t-product $\mathcal{A} * \mathcal{R}$ is computed by transforming both $\mathcal{A}$ and $\mathcal{R}$ into the Fourier domain via FFT, multiplying the corresponding frontal slices as matrices, and then transforming back with inverse FFT, see [10, 9] for more details.

4. Numerical experiments

In this section, we present some numerical experiments to demonstrate the performance of Algorithms 2, 3 and 5 for solving consistent linear system (1), matrix equation (2) and tensor equation (3), respectively. The initial guesses in all algorithms are taken to be a zero vector/matrix/tensor.

All computations were carried out on a computer with an Intel Core i7-1355U CPU @ 1.70 GHz processor and 16.0GB RAM using MATLAB.R2023b. The required number of iterations and elapsed CPU times (in seconds), with respect to the specified stopping criteria, are reported under "Iter" and "CPU-time" in the tables, respectively. All algorithms are implemented with $\ell = 5$ in (4), unless stated otherwise.

In the first example, we disclose comparative results between RABRSD-GMM and Algorithm 2 for solving a consistent linear system of equations to demonstrate the superiority of the proposed approach over the best implementation of RABRSD reported in [12]. To solve the test problem, the examined iterative methods are terminated once

$$\text{ERR} := \frac{\|x_t - x^*\|}{\|x^*\|} \leq 10^{-4},$$

where x_t stands for the t-th approximate solution.

Example 1. The coefficient matrix A in (1) is produced by MATLAB command $A = \texttt{sprandn}(m, n, 0.01) + 10 * \texttt{sprandn}(m, n, 0.01)$.[3] Similar to [12, Example 3.1], we set $n = 1000$ and construct the right-hand side vector b such that $b = Ax^*$ with x^* being an all-one vector.

For implementing Algorithm 1, in the above example, we constructed the Achlioptas projection matrix T_A of size $d \times m$ as follows:

$$T_A(i,j) = \begin{cases} 1 & \text{if } R(i,j) < \frac{1}{6} \\ -1 & \text{if } R(i,j) > \frac{5}{6} \\ 0 & \text{otherwise} \end{cases},$$

where R is a $d \times m$ random matrix R generated with entries uniformly distributed in $[0, 1]$ and d is obtained by

$$d = \left\lceil \frac{\log_2(n)}{\epsilon^2} \right\rceil,$$

in which the value of ϵ is chosen according to the values used in [12, Table 3.1].

Li et al. [12] observed that the Gaussian Mixture Model (GMM) can cluster complex data more effectively than k-means clustering. The numerical results reported in [12, Section 3] indicate that RABRSD-GMM outperforms the RABRSD-k method. Thus, here we only compare the proposed approach with the RABRSD-GMM method. As shown in Table 1, the numerical results demonstrate that Algorithm 2 is significantly more effective than the RABRSD-GMM method. It is worth mentioning that the clustering parameter k in Algorithm 1 is set to be $\max(4, m/10000)$. For further details, we also disclose the relative norm of the residual vector of the computed approximation, i.e.,

$$\text{RES} := \frac{\|b - Ax_t\|}{\|b\|}. \tag{12}$$

It is noted that each reported value represents the average over ten independent test runs, with iteration counts rounded to the nearest integer.

Figure 1 illustrates the number of iterations against the base 10 logarithm of RES (cf. (12)) for RABRSD-GMM and Algorithm 2. Evidently, the figure shows the superiority of the proposed method over RABRSD-GMM. For more details, in Figure 2, we plot the performance of Algorithm 2 with respect to different values of ℓ in (4). It is observed that the convergence speed decreases with increasing ℓ for the smallest problem size, whereas for the largest problem size, the convergence rate remains largely independent of the choice of ℓ.

[3]The MATLAB function $\texttt{sprandn}(m, n, \nu)$ which creates an $m \times n$ sparse matrix with normally distributed random entries with $\nu\%$ density. It is noted that we fixed the MATLAB random seed to rng(0) for the sake of reproducibility.

Method		m = 20 000	40 000	80 000	150 000
RABRSD-GMM	CPU-time	7.495	9.934	16.415	24.231
	Iter	57	49	74	163
	RES	8.618e-05	8.771e-05	9.091e-05	9.559e-05
	Err	9.2234e-05	9.0705e-05	9.2224e-05	9.6548e-05
Algorithm 2	CPU-time	0.033	0.059	0.117	0.248
	Iter	18	11	8	6
	RES	6.429e-05	4.742e-05	5.553e-05	8.291e-05
	Err	7.6046e-05	5.0255e-05	5.6702e-05	8.4562e-05

Table 1: Numerical results for RABRSD-GMM [12] (Algorithm 1) and Algorithm 2 in Example 1

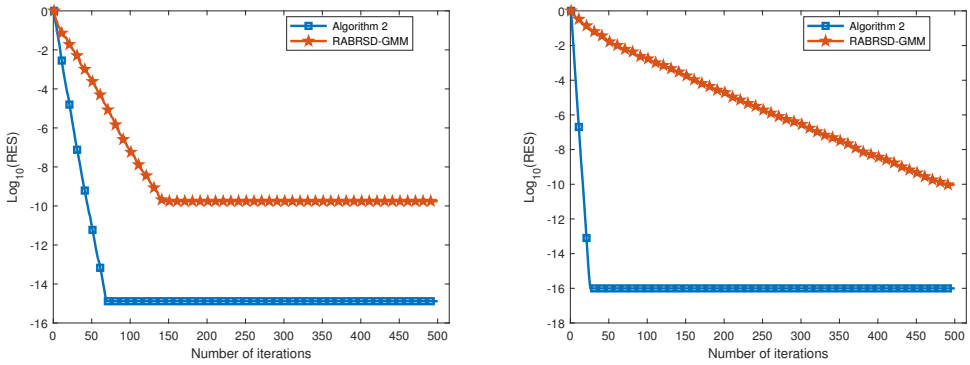


Figure 1: Number of iterations vs. $\text{Log}_{10}(\text{RES})$ associated with $m = 20\,000$ (left) and $m = 150\,000$ (right) in Example 1

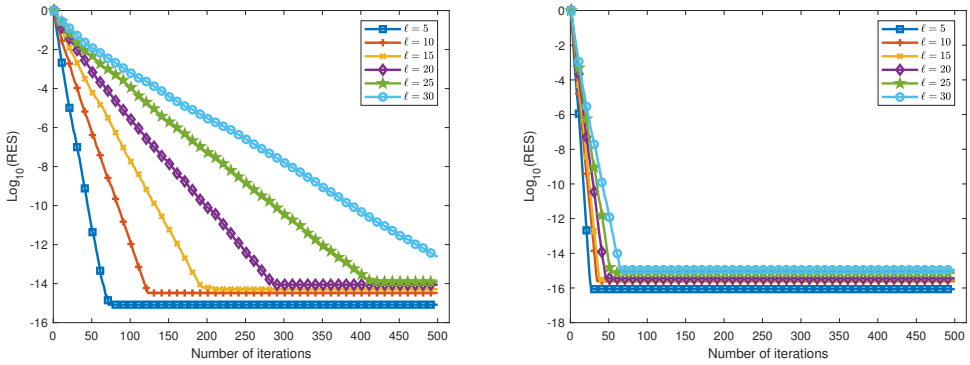


Figure 2: Number of iterations vs. $\text{Log}_{10}(\text{RES})$ for Algorithm 2 associated with $m = 20\,000$ (left) and $m = 150\,000$ (right) in Example 1

In what follows, we consider an application to color image restoration, see [24, Section 6]. To be more specific, for comparing the performance of Algorithms 3 and 4, we examine four test images "Airplane", "House", "Lake", and "Tree" which are illustrated in the top row of Figure 3, respectively. The exact images can be downloaded from the USC-SIPI image database[4]. Here, we take all images of size $128 \times 128 \times 3$ for which A in (2) is a 16384×16384 matrix.

Example 2. We consider the forward model for color image restoration, see Subsection 2.2 for more details. To obtain blurred images, the MATLAB function fspecial is applied to generate a rotationally symmetric Gaussian lowpass filter of size 5 with standard deviation 6 to simulate the blurred images caused by the

[4] https://sipi.usc.edu/database/

Gaussian filter. The cross-channel blurring matrix A_c is given as follows ([24, cf. Eq. (6.1)]):

$$A_c = \begin{bmatrix} 0.90 & 0.05 & 0.05 \\ 0.00 & 0.90 & 0.10 \\ 0.05 & 0.10 & 0.85 \end{bmatrix}.$$

The obtained blurred images are presented in the bottom row of Figure 3.

The iterative methods for solving Example 2 are terminated once $\|C - AX_tA_c^T\|_F \leq \eta\|C\|_F$, where X_t is the t-th approximate solution and $\eta = 0.0015$. In [24, Subsection 4.3], it is noted that the convergence factor for the MWRBK methods is minimized when $\alpha = \|A_c\|^{-2}$; therefore, this parameter value is used in the implementation of Algorithm 4. The obtained experimental results are reported in Table 2. As observed, Algorithm 3 significantly surpasses the MWRBK method [24] in both iteration count and CPU-time required for the convergence, although the MWRBK method provides slightly lower relative absolute errors in the approximate solutions. For more details, the restored images are shown in Figure 4 and their corresponding PSNR[5] values are included in Table 2 as well.

Test image	Method	Iter	Err	CPU-time	PSNR
Airplane	Algorithm 3	526	0.0673	0.9096	26.1586
	MWRBK	93806	0.0651	147.38	26.4586
House	Algorithm 3	503	0.0666	0.8454	28.1609
	MWRBK	97153	0.0652	155.46	28.4617
Lake	Algorithm 3	707	0.0799	1.1541	27.1655
	MWRBK	139621	0.0782	183.85	27.4577
Tree	Algorithm 3	675	0.0755	1.0485	27.1325
	MWRBK	128535	0.0737	171.06	27.4151

Table 2: Comparison of Algorithm 3 and the MWRBK method [24] (Algorithm 4)

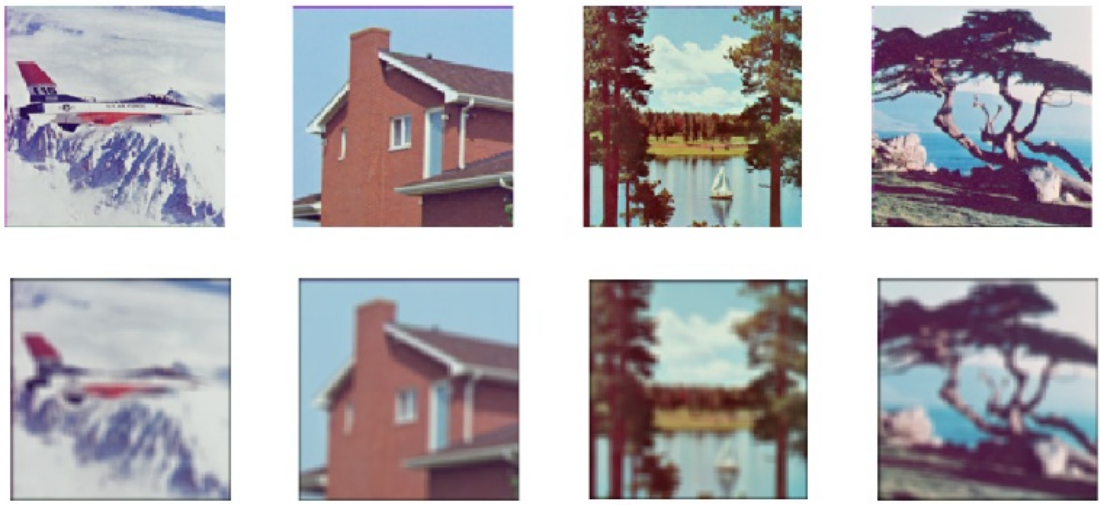


Figure 3: Test images (top row) and blurred images (bottom row) in Example 2.

[5] See Table 3.

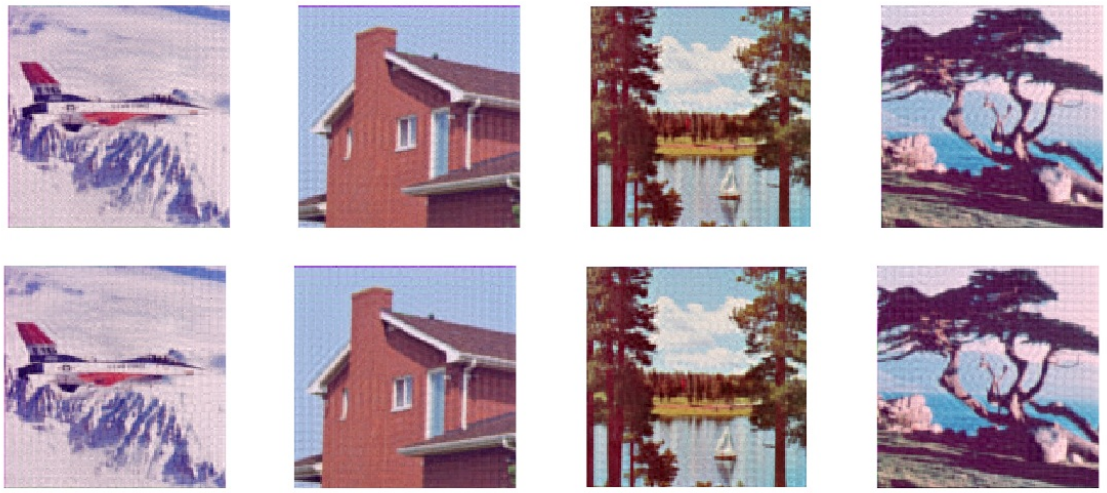


Figure 4: Restored images obtained by Algorithm 3 (top row) and Algorithm 4 (bottom row) in Example 2.

Recently, Luo and Ma [16] proposed the Block Gradient Descent with Cyclic Frontal Slices (BCFGD) method (see Algorithm 6) for solving (3). The experimental results indicate that the BCFGD method outperforms other examined approaches in [16] for solving the following test problem. In Algorithm 6, the tensor $\tilde{\mathcal{A}}_i^s$ denotes a frontal sub-block of $\mathcal{A}$, see [16, cf. Eq. (3.6)].

Algorithm 6 Block Gradient Descent with Cyclic Frontal Slices (BCFGD) [16, Algorithm 3.2]

Input: $\mathcal{A}$, $\mathcal{B}$, learning rate α_l, block size s and $\text{iter}_{\max}$
Initialize $\mathcal{R}(0) = \mathcal{B}$, $\mathcal{X}(0) = 0$, $t = 0$
for $t = 0, 1, 2, \ldots, \text{iter}_{\max}$ **do**
  $i = \text{mod}(t, n/s) + 1$
  $\mathcal{R}(t+1) = \mathcal{R}(t) - \tilde{\mathcal{A}}_i^s * \mathcal{X}(t) + \tilde{\mathcal{A}}_i^s * \mathcal{X}(t-n+1)$
    with the convention that $\mathcal{X}(t) = 0$ for $t \leq 0$
  $\mathcal{X}(t+1) = \mathcal{X}(t) + \alpha_l (\tilde{\mathcal{A}}_i^s)^T * \mathcal{R}(t+1)$
  $t = t + 1$
end for

For the sake of comparison, in Example 3, we compare the performance of Algorithm 5 with the BCFGD method with varying block sizes s. The MATLAB code for the numerical implementation of Algorithm 6, for Example 3, was downloaded from `https://github.com/anna-math/frontalslicemethods`.

Example 3. We consider the image deblurring test problem in [16, Subsection 5.2]. More precisely, the algorithms are used to solve $\mathcal{A} * \mathcal{X} = \mathcal{B}$, in which the right-hand side tensor $\mathcal{B}$ is the observed blurred image, i.e., $\mathcal{B} = \mathcal{A} * \mathcal{X}^*$, where $\mathcal{X}^*$ is the exact solution[6]. The blurring operator $\mathcal{A} \in \mathbb{R}^{60\times 60\times 60}$ is generated by

$$\mathcal{A}(:,:,i) = A_1(i,1)A_2, \quad i = 1, 2, \ldots, 60,$$

where matrices A_1 and A_2 are constructed by using the MATLAB commands

$$A_1 = \frac{1}{\sigma\sqrt{2\pi}}\text{toeplitz}(z_1, z_2) \quad \text{and} \quad A_2 = \frac{1}{\sigma\sqrt{2\pi}}\text{toeplitz}(z_1),$$

with

$$z_1 = \left[\exp(-([0:\text{band}-1]^2)/(2\sigma^2)),\ \text{zeros}(1, 60-\text{band})\right]$$

and

$$z_2 = \left[z_1(1)\ \ \text{fliplr}(z_1(\text{end}-\text{length}(z_1)+2:\text{end}))\right],$$

[6]The exact solution $\mathcal{X}^*$ is the Hubble Space Telescope image which is restored as a $60 \times 1 \times 60$ image.

where $\sigma = 3$ and band = 9. Finally, each horizontal slice $\mathcal{A}(i,:,:)$ is normalized as follows:

$$\mathcal{A}(i,:,:) = \frac{\mathcal{A}(i,:,:)}{\|\mathcal{A}(i,:,:)\|_F}, \quad i = 1, 2, \ldots, 60.$$

Metric	Description
Mean Squared Error (MSE)	Measures the average squared difference between corresponding pixels of the original and restored images.
Peak Signal-to-Noise Ratio (PSNR)	Logarithmic measure of the ratio between the maximum possible pixel value and the power of the noise or reconstruction error.
Structural Similarity Index (SSIM)	Evaluates perceived structural similarity based on luminance, contrast, and texture; values range from -1 to 1, with higher values indicating better structural preservation.

Table 3: Image quality evaluation metrics

For solving Example 3, we applied the BCFGD method with the same parameter α_l utilized in [16], i.e., we set $\alpha_l = 0.01$ in Algorithm 6. The experimental results are obtained by running Algorithms 5 and 6 in the following two settings:

- **Case I.** We set $\text{iter}_{\max} = 200$ and apply the algorithms for exactly 200 iterations.
- **Case II.** We terminated the algorithms once the following condition holds:

$$\|\mathcal{X}(t) - \mathcal{X}(t-1)\|_F \leq \gamma \|\mathcal{X}(t-1)\|_F, \tag{13}$$

  where $\mathcal{X}(t)$ denotes the t-th approximate solution and $\gamma = 0.5 \times 10^{-3}$.

Similar to [16, Subsection 5.2], the Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) error metrics are used to measure the quality of restored images, see Table 3 for more details. It is worth mentioning that a lower MSE and higher PSNR and SSIM indicate superior results. In fact, a lower MSE naturally gives a higher PSNR, since they are inversely related. In addition, the higher SSIM shows that the improvement is not just in pixel-wise error but also in structural quality.

The experimental results in Tables 4 and 5 together with Figures 5 and 6 show that Algorithm 5 provides a more accurate approximate solution than the BCFGD method. In Case I, the results in Table 4 indicate that the proposed approach is considerably faster than the BCFGD method [16]. Although the BCFGD method (with $s = 5$) requires less CPU time than other approaches under stopping criterion (13) in Case II, its produced approximate solution is of the lowest quality among all obtained approximations, as seen in Table 5 and Figures 5 and 6.

Method	block size (s)	MSE	PSNR	SSIM	CPU-time
BCFGD	20	1.221e-05	4.913e+01	9.883e-01	2.04e+00
BCFGD	10	1.221e-05	4.913e+01	9.883e-01	1.80e+00
BCFGD	5	1.414e-05	4.850e+01	9.859e-01	1.75e+00
Algorithm 5	-	1.084e-05	4.965e+01	9.899e-01	3.41e-01

Table 4: Performance of Algorithm 5 and the BCFGD [16] method (Algorithm 6) in Example 3 (Case I)

Method	block size (s)	MSE	PSNR	SSIM	CPU-time
BCFGD	20	1.328e-05	4.877e+01	9.870e-01	1.32e+00
BCFGD	10	1.328e-05	4.877e+01	9.870e-01	1.28e+00
BCFGD	5	1.975e-05	4.704e+01	9.793e-01	4.99e-01
Algorithm 5	-	1.021e-05	4.991e+01	9.906e-01	7.31e-01

Table 5: Performance of Algorithm 5 and the BCFGD [16] method (Algorithm 6) in Example 3 (Case II)

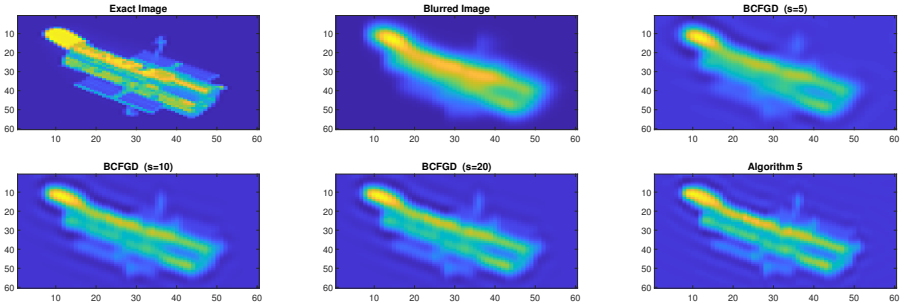


Figure 5: Exact, blurred and restored images by algorithms in Example 3 (Case II)

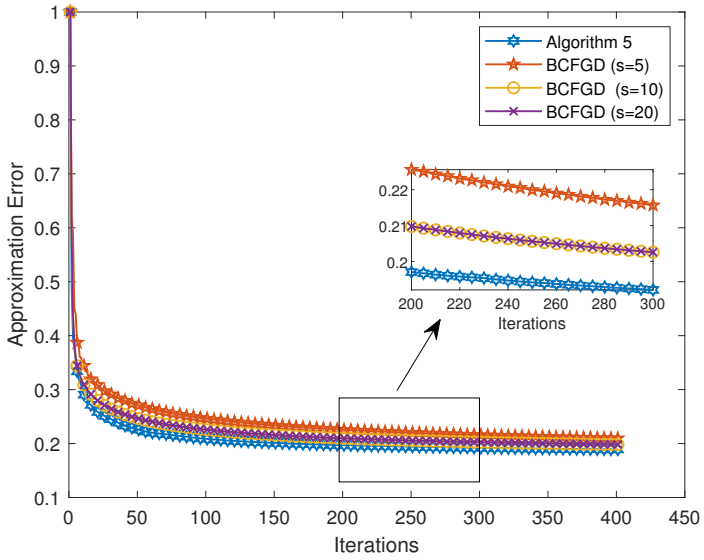


Figure 6: Number of iterations (t) vs. approximation error $\|\mathcal{X}(t) - \mathcal{X}^*\|_F$

Finally, to further validate the applicability of the proposed algorithm, we consider a randomly generated t-product tensor equation in Example 4. The random number generator is initialized with rng(0) to ensure reproducibility.

Example 4. Let $\mathcal{A} \in \mathbb{R}^{m\times 20\times m}$ and $\mathcal{X}^* \in \mathbb{R}^{20\times 1\times m}$ be constructed using the MATLAB function "randn" such that each horizontal slice $\mathcal{A}(i,:,:)$ is normalized as follows:

$$\mathcal{A}(i,:,:) = \frac{\mathcal{A}(i,:,:)}{\|\mathcal{A}(i,:,:)\|_F}, \quad i = 1, 2, \ldots, m. \tag{14}$$

Consider the tensor equation $\mathcal{A} * \mathcal{X} = \mathcal{B}$, where the right-hand tensor is generated setting $\mathcal{B} = \mathcal{A} * \mathcal{X}^*$.

Algorithms 5 and 6 are applied for solving Example 4 across ten independent test runs. In Table 6, we represent the average results including CPU time and iteration counts (rounded to the nearest integer). The algorithms are terminated when condition (13) holds. As no explicit formula is provided in [16] for choosing the optimal parameter α_l in Algorithm 6, we empirically determined its value for each problem size to minimize the required iteration numbers and CPU time for convergence with respect to stopping criterion (13). The notation "Err_T" in Table 6 stands for the relative solution error, i.e.,

$$\text{Err}_T := \frac{\|\mathcal{X}(t) - \mathcal{X}^*\|_F}{\|\mathcal{X}^*\|_F},$$

where $\mathcal{X}(t)$ stands for the t-th approximate solution. As seen in Table 6, Algorithm 5 requires less CPU time than the BCFGD method, while giving comparable accuracy. Using the larger block size s generally improves

performance of the BCFGD method in Example 4. Our experimental results show that Algorithm 6 fails to produce accurate approximation without normalization step (14), while Algorithm 5 converges reliably without normalization.

Method		m			
		50 ($\alpha = 0.6$)	100 ($\alpha = 0.4$)	150 ($\alpha = 0.3$)	200 ($\alpha = 0.3$)
BCFGD ($s = 5$)	CPU-time	2.232e-01	2.925e+00	6.662e+00	1.189e+01
	Iter	196	185	200	219
	Err_T	2.743e-03	2.189e-03	2.462e-03	X1.847e-03
BCFGD ($s = 10$)	CPU-time	1.261e-01	1.590e+00	3.684e+00	6.573e+00
	Iter	107	97	107	118
	Err_T	1.954e-03	1.548e-03	1.590e-03	1.168e-03
BCFGD ($s = 20$)	CPU-time	7.189e-02	8.344e-01	2.063e+00	3.471e+00
	Iter	62	51	60	64
	Err_T	1.478e-03	9.117e-04	1.022e-03	7.488e-04
Algorithm 5	CPU-time	5.602e-02	2.016e-01	3.180e-01	3.681e-01
	Iter	72	34	25	20
	Err_T	4.017e-03	1.671e-03	1.065e-03	8.098e-04

Table 6: Numerical results for Algorithm 5 and BCFGD [16] (Algorithm 6) in Example 4

5. Concluding remarks

A new randomized block variant of the residual norm steepest descent iterative method was developed for solving consistent linear systems $Ax = b$, matrix equations of the form $AXA_c^T = C$ and tensor equation $\mathcal{A} * \mathcal{X} = \mathcal{B}$. Convergence properties of the proposed approach were established and numerical experiments were reported to illustrate the applicability of the approach for a linear system of equations, matrix equations and tensor equations. Specifically, the superiority of the proposed approach was shown over the methods given in [12, 16, 24].

Acknowledgments

The authors sincerely thank the anonymous referees for their careful reading and valuable suggestions, which have been helpful in improving the quality of this work.

References

[1] D. Achlioptas, Database-friendly random projections: Johnson-Lindenstrauss with binary coins, *J. Comput. Syst. Sci.* **66** (2003), no. 4, 671–687.

[2] Z. Z. Bai and W. T. Wu, On relaxed greedy randomized Kaczmarz methods for solving large sparse linear systems, *Appl. Math. Lett.* **83** (2018), 21–26.

[3] Z. Z. Bai and W. T. Wu, On greedy randomized coordinate descent methods for solving large linear least-squares problems, *Numer. Linear Algebra Appl.* **26** (2019), no. 4, e2237.

[4] K. Du and H. Gao, A new theoretical estimate for the convergence rate of the maximal weighted residual Kaczmarz algorithm, *Numer. Math. Theory Methods Appl.* **12** (2019), no. 2, 627–639.

[5] X. Guo, X. Li, V. Howle, and J. Pan, An efficient greedy quasi block coordinate descent method for solving linear least-squares problems, *Appl. Math. Lett.* **171** (2025), 109675.

[6] G. X. Huang and S. Y. Zhong, Tensor randomized extended Kaczmarz methods for large inconsistent tensor linear equations with t-product, *Numer. Algorithms* **96** (2024), no. 4, 1755–1778.

[7] X. L. JIANG and K. ZHANG, A randomized block extended Kaczmarz method with hybrid partitions for solving large inconsistent linear systems, *Appl. Math. Lett.* **152** (2024), 109027.

[8] X. L. JIANG, K. ZHANG, and J. F. YIN, Randomized block Kaczmarz methods with k-means clustering for solving large linear systems, *J. Comput. Appl. Math.* **403** (2022), 113828.

[9] M. E. KILMER, K. BRAMAN, N. HAO, and R. C. HOOVER, Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging, *SIAM J. Matrix Anal. Appl.* **34** (2013), no. 1, 148–172.

[10] M. E. KILMER and C. D. MARTIN, Factorization strategies for third-order tensors, *Linear Algebra Appl.* **435** (2011), no. 3, 641–658.

[11] T. G. KOLDA and B. W. BADER, Tensor decompositions and applications, *SIAM Rev.* **51** (2009), no. 3, 455–500.

[12] L. LI, S. R. LIU, and T. LI, A randomized Achlioptas block residual steepest descent method for large sparse overdetermined linear systems, *Appl. Math. Lett.* **174** (2026), 109808.

[13] T. LI, L. LI, and X. F. ZHANG, Randomized block residual steepest descent method with k-means clustering for large sparse linear systems, *Comput. Appl. Math.* **44** (2025), 286.

[14] J. LIU and S. WRIGHT, An accelerated randomized Kaczmarz algorithm, *Math. Comp.* **85** (2016), 153–178.

[15] Y. LIU and C. Q. GU, On greedy randomized block Kaczmarz method for consistent linear systems, *Linear Algebra Appl.* **616** (2021), 178–200.

[16] H. LUO and A. MA, Frontal slice approaches for tensor linear systems, *Numer. Math. Theory Methods Appl.* **18** (2025), no. 2, 353–394.

[17] A. MA and D. MOLITOR, Randomized Kaczmarz for tensor linear systems, *BIT Numer. Math.* **62** (2022), 171–194.

[18] D. NEEDELL and J. A. TROPP, Paved with good intentions: Analysis of a randomized block Kaczmarz method, *Linear Algebra Appl.* **441** (2014), 199–221.

[19] Y. Q. NIU and B. ZHENG, On the randomized block Kaczmarz algorithms for solving matrix equation $AXB = C$, *J. Comput. Appl. Math.* **460** (2025), 116421.

[20] L. RUI, Y. AI-LI, and M. JIAN, Maximal residual coordinate descent method with k-means clustering for solving large linear least-squares problems, *Numer. Algorithms* **101** (2026), 1157–1173.

[21] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, second ed., Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2003.

[22] T. STROHMER and R. VERSHYNIN, A randomized Kaczmarz algorithm for linear systems with exponential convergence, *J. Fourier Anal. Appl.* **15** (2009), 262–278.

[23] L. Z. TAN and X. P. GUO, On multi-step greedy randomized coordinate descent method for solving large linear-squares problems, *Comput. Appl. Math.* **42** (2023), 37.

[24] W. WANG, D. LIU, G. QU, and M. E. HOCHSTENBACH, Deterministic and randomized Kaczmarz methods for $AXB = C$ with applications to color image restoration, *BIT Numer. Math.* **66** (2026), 13.

[25] A. Q. XIAO, J. F. YIN, and N. ZHENG, On fast greedy block Kaczmarz methods for solving large consistent linear systems, *Comput. Appl. Math.* **42** (2023), 119.

[26] X. ZENG, S. LIANG, Y. HONG, and J. CHEN, Distributed computation of linear matrix equations: An optimization perspective, *IEEE Trans. Automat. Control* **64** (2018), no. 5, 1858–1873.

[27] J. ZHANG and J. G. NAGY, An effective alternating direction method of multipliers for color image restoration, *Appl. Numer. Math.* **164** (2021), 43–56.

[28] J. H. ZHANG and J. H. GUO, On relaxed greedy randomized coordinate descent methods for solving large linear least-squares problems, *Appl. Numer. Math.* **157** (2020), 372–384.

[29] K. ZHANG, X. X. CHEN, and X. L. JIANG, A residual-based surrogate hyperplane extended Kaczmarz algorithm for large least squares problems, *Calcolo* **61** (2024), 51.

[30] K. ZHANG, J. Y. DENG, and X. L. JIANG, A quantile-based block Kaczmarz algorithm for solving large consistent linear systems, *Comput. Appl. Math.* **44** (2025), 8.