AIMS Mathematics, 11(4): 8859–8878.
DOI: 10.3934/math.2026365
Received: 06 January 2026
Revised: 07 March 2026
Accepted: 13 March 2026
Published: 01 April 2026

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

*Research article*

# The k-means clustering-based CD method for solving large-scale matrix equation $AX = B$

**Lifang Dai[1], Maolin Liang[1], Qun Li[2,*], Ruijuan Zhao[3] and Yong-hong Shen[4]**

[1] School of Mathematics and Statistics, Tianshui Normal University, Tianshui, 741000, China
[2] School of Data Sciences, Zhejiang University of Finance and Economics, Hangzhou, 310018, China
[3] School of Information Engineering, Lanzhou University of Finance and Economics, Lanzhou, 730101, China
[4] School of Mathematics and Computer Science, Northwest Minzu University, Lanzhou, 730000, China

* **Correspondence:** Email: qunli_2021@163.com.

**Abstract:** In this paper, the coordinate descent (CD) method integrated with k-means clustering is proposed to address the least-squares problem of the large-scale matrix equation $AX = B$, where the coefficient matrix $A$ is assumed to be of full column rank. Rigorous theoretical analysis shows that the iteration sequence generated by this method converges to the unique least-norm least-squares solution of the problem. The performed numerical results demonstrate that the method developed here is feasible and efficient.

**Keywords:** matrix equation; least-squares; k-means clustering; coordinate descent method
**Mathematics Subject Classification:** 15A24, 65F10

## 1. Introduction

In this paper, we consider the large-scale matrix equation

$$AX = B, \tag{1.1}$$

where $A \in \mathbb{R}^{n\times p}$, $B \in \mathbb{R}^{n\times q}$, and the unknown $X \in \mathbb{R}^{p\times q}$.

The matrix Eq (1.1) has important applications in several fields including structural dynamics, automatic control, parameter identification, quantum chromodynamics, and so on [1–5]. Many scholars have investigated the matrix Eq (1.1) over different constrained conditions by means of the singular value decomposition or the generalized inverse of matrices [6] driven by their plentiful

practical applications in engineering, see; e.g., [7–16]. However, the above direct methods entail intensive computations of the singular value decomposition and generalized inverse, rendering them non-trivial to implement in practice. Alternatively, iterative approaches have become the preferred choice, and various iterative methods have been proposed for it. In fact, the design of iterative solvers for this problem has been an active research topic since the early 1980s. When the coefficient matrix $A$ is symmetric and positive definite, the block conjugate gradient method proposed by O'Leary [17], as well as its variants [18, 19] is efficient. After that, Simoncini and Gallopoulos [20] developed the iterative method for nonsymmetric cases, which combines the advantages of Krylov subspace iteration and Richardson acceleration technique. Furthermore, a flexible and adaptive block generalized minimal residual (GMRES) algorithm with the deflated restarting method was provided in [21], which is more stable than the GMRES method. For more information, one can also refer to [22–28].

While effective for solving the matrix Eq (1.1) under specific conditions, the aforementioned iterative methods exhibit critical limitations when applied to large-scale problems. Specifically, block conjugate gradient methods are only applicable to symmetric positive definite matrices A; Krylov subspace-based methods suffer from exorbitant computational complexity owing to repeated orthogonalization operations; and deflated restarting block GMRES entails substantial storage overhead and high computational costs for local subproblem solving. Such drawbacks drive the design of new dedicated methods for large-scale instances of (1.1).

Given the aforementioned limitations of iterative methods for the matrix Eq (1.1), and the practical difficulty in verifying the consistency of that for large-scale cases, we focus on investigating the corresponding least-squares (LS) problem:

$$\min_{X} \|AX - B\|_F, \tag{1.2}$$

in which the coefficient matrix $A$ is assumed to be of full column rank; here and in the sequel, the symbol $\|\cdot\|_F$ stands for the Frobenius norm of a matrix. One way to address this LS problem is to reformulate it as the least-squares problem for the linear system

$$(I_q \otimes A)vec(X) = vec(B),$$

where *vec* is the vectorization operator, which flattens a matrix into a column vector by stacking its columns sequentially, and the symbol $\otimes$ denotes the Kronecker product between two matrices. Nevertheless, the Kronecker product inevitably leads to a dramatic dimensional explosion of the problem, which brings about heavy computational and memory burdens for large-scale cases. The other way is to find the solution $X^*$ by means of the normal equation $A^TAX = A^TB$. Unfortunately, in large-scale scenarios, iterative algorithms applied directly to a normal equation may be typically impractical owing to prohibitive storage and computational demands.

Recently, several elegant randomized iterative methods were proposed for solving large-scale linear systems or their least-square problems; see, e.g., [29–33]. Notably, combining the k-means clustering [34] and the randomized Kaczmarz methods [29], the authors in [35] put forward an efficient algorithm for solving large linear systems, which inherits the advantages of those two methods, and contains the classical randomized Kaczmarz method as the special case. Moreover, taking advantage of the coordinate descent (CD) method [36], the randomized CD method was

proposed for solving linear least-squares problems [37]. For the most recent studies on this topic, one can refer to, e.g., [38–40] and therein.

Motivated by the aforementioned limitations of existing solvers for the large-scale matrix Eq (1.1), we propose a k-means clustering-based coordinate descent method (MCCD(k)) for the LS problem (1.2) in this paper. It will be proved that the iteration sequence generated by our method converges to the unique least-norm solution $X^* = A^\dagger B$ to the problem (1.2) under the hypothesis that the involved coefficient matrix $A$ is of full column rank. For the rank deficiency situation, it can be addressed similarly by adding a regularization item. Our MCCD(k) method addresses these limitations by fusing k-means clustering with the coordinate descent framework, and it exhibits two core advantages for the large-scale case. First, we cluster the columns of $A$ by cosine distance (angular similarity), which enables block-wise coordinate updates instead of single-coordinate updates and avoids the expensive $AA^T$ multiplication required by classical iterative methods such as steepest descent and conjugate gradient, thus significantly reducing the per-iteration computational complexity for large $n$. Second, we incorporate a greedy maximum-residual selection strategy within each k-means cluster, which prioritizes the most impactful directions for residual reduction and accelerates the convergence of the iteration sequence. Additionally, MCCD(k) circumvents the dimensionality explosion issue of the Kronecker product vectorization approach and is free from the strict structural constraints (e.g., symmetric positive definiteness of $A$) of traditional block iterative methods, making it more flexible for general large-scale matrix equation problems. To the best of our knowledge, we have not yet found similar approaches that combine the k-means clustering method with the coordinate descent method for solving the matrix Eq (1.1).

The remaining of this paper is organized as follows: In Section 2, we will briefly review the k-means clustering method and the classical CD method. In Section 3, the k-means clustering-based coordinate descent method will be established, and then its convergence will be analyzed. In Section 4, several numerical experiments will be conducted to demonstrate the feasibility and effectiveness of the proposed method. Finally, we will conclude this paper with some remarks.

## 2. Preliminary knowledge

For ease of expression, throughout this paper, scalars are represented by lowercase letters, e.g., $a$; vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$; and matrices are indicated by capital letters, e.g., $I$, identity matrix with appropriate size. The set comprising all $m \times n$ real matrices is symbolized by $\mathbb{R}^{m\times n}$. For a matrix $A \in \mathbb{R}^{m\times n}$, the symbols $A^T, A^\dagger, \|A\|_2$, $\sigma_{\max}(A)$, $\sigma_{\min}(A)$, and $A_{(j)}$ represent the transpose, the Moore-Penrose inverse, the 2-norm, the maximum singular value, the minimum singular value, and the $j$th column of the matrix $A$, respectively. In addition, the cardinality of a set $S$ is denoted by $|S|$.

Now, we first present a brief review of the k-means clustering method and the coordinate descent (CD) method. In the context of data mining, k-means clustering is one of the most popular learning algorithms for dividing data into different groups [34]. The main idea of this method is outlined in Algorithm 1.

**Algorithm 1:** The k-means clustering method

Step 1: Input a data set $D = \{D_1, D_2, \ldots, D_n\}$, the number of clusters $k$.
Step 2: Randomly select k samples from $D$ as initial cluster centers $C_{\delta_1}, C_{\delta_2}, \ldots, C_{\delta_k}$.
Step 3: For $i = 1 : n$,
Compute the distance $d_{ij}$ between sample $D_i$ and initial cluster center $C_{\delta_j}$,$j = 1 : k$;
Assign $D_i$ to the class associated with the nearest centroid $C_{\delta_j}$;
Compute the new cluster centers $C_{\delta_j} = \dfrac{1}{|\delta_j|} \sum\limits_{d \in \delta_j} d$.
Step 4: Repeat Steps 2-3 until the cluster centers do not change.
Step 5: Output cluster partition $\Delta = \{\delta_1, \delta_2, \ldots, \delta_k\}$.

Additionally, the history of the CD method dates back to the early development of the discipline of optimization [41]; one can refer to the recent surveys [42, 43] and references therein for details. The basic CD framework for solving the unconstrained optimization problem $\min\limits_{\mathbf{x}} f(\mathbf{x})$ with continuous and differential function $f(\mathbf{x})$ defined from $\mathbb{R}^n$ to $\mathbb{R}$, is shown in Algorithm 2, in which $[\nabla f(\mathbf{x})]_i$ represents the $i$th component of the gradient $\nabla f(\mathbf{x})$ at the current point.

**Algorithm 2:** The CD method

Step 1: Input $f(\mathbf{x})$, $\mathbf{x}^{(0)}$ and the tolerance $\epsilon$.
Step 2: For $k = 0, 1, 2, \ldots$,
Choose the index $i_t \in \{1, 2, \ldots, n\}$, and compute
$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha_t [\nabla f(\mathbf{x}^{(t)})]_{i_t} \mathbf{e}_{i_t}$ with $\alpha_t > 0$.
Step 3: Until $\mathbf{x}^{(t+1)}$ satisfies the tolerance $\epsilon$.

For the linear systems $A\mathbf{x} = \mathbf{b}$, the CD method can be described as follows:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \frac{A_{(j)}^T(\mathbf{b} - A\mathbf{x}^{(t)})}{\|A_{(j)}^T\|_2^2}\mathbf{e}_{j_i}, t = 1, 2, \ldots,$$

where $j_i = (i \ mod \ p) + 1$, and the symbol $\mathbf{e}_{j_i}$ denotes the $j_i$th column of the identity matrix with appropriate size.

## 3. The k-means clustering-based CD method for the LS problem (1.2)

In this section, we shall establish the algorithm to solve the large-scale LS problem (1.2), which originates from the fusion of the k-means clustering method and the CD method. To do this, we regard all columns $A_{(j)}$ $(j = 1, 2, \ldots, p)$ of the coefficient matrix $A$ as a set of vectors and then employ the k-means clustering method (i.e., Algorithm 1) to partition them such that columns with small angles are allocated into one block, in which the distance $d_{ij}$ between the column vectors $A_{(i)}$ and $A_{(j)}$ is chosen as the cosine distance, i.e., $d_{ij} = 1 - \cos(A_{(i)}, A_{(j)}) = 1 - \frac{A_{(i)}^T A_{(j)}}{\|A_{(i)}\|_2 \|A_{(j)}\|_2}$. Moreover, as proposed in [31], we also use the greedy strategy, which prioritizes selecting components with maximal residuals in each step. We formulate the k-means clustering-based CD method for solving the matrix Eq (1.1) (shortened by MCCD(k)) as follows, i.e., Algorithm 3.

**Algorithm 3:** The MCCD(k) method

Step 1: Input $A \in \mathbb{R}^{n\times p}$ is of full column rank, $B \in \mathbb{R}^{n\times q}$, $X^{(0)} \in \mathbb{R}^{p\times q}$, the positive integer $k$, and the tolerance $\epsilon$.

Step 2: Apply Algorithm 1 to divide the columns of $A = (A_{(1)}, A_{(2)}, \ldots, A_{(p)})$ into k-parts $A_{S_1}, A_{S_2}, \ldots, A_{S_k}$.

Step 3: For $t = 0, 1, 2, \ldots$, compute

$j_i = arg\max\limits_{j\in S_i} \left\|A_{(j)}^T(B - AX^{(t)})\right\|_2^2$, $i = 1, 2, \ldots, k$;

$S := \{j_1, j_2, \ldots, j_k\}$, $A_S := (A_{(j_1)} A_{(j_2)} \cdots A_{(j_k)})$,

$I_S := (e_{j_1} e_{j_2} \cdots e_{j_k}) \in \mathbb{R}^{p\times k}$, $e_{j_i}$ denotes the $j_i$-th standard basis vector of $\mathbb{R}^p$;

$X^{(t+1)} := X^{(t)} + I_S(A_S^T A_S)^{-1}A_S^T(B - AX^{(t)})$.

Step 4: If $\|B - AX^{(t+1)}\|_F < \epsilon$ (consistent) or $\|A_S^T(B - AX^{(t+1)})\|_F < \epsilon$ (inconsistent), stop.

We have some comments on this algorithm:

• In this algorithm, after selecting indices $j_1, j_2, \ldots, j_k$, we construct the block matrix $A_S$, where each column $A_{(j_i)}$ is the representative of cluster $S_i$. The matrix $I_S$ is a selection matrix, where $e_{j_i}$ denotes the $j_i$-th standard basis vector of $\mathbb{R}^p$. The role of $I_S$ is to map the block update vector (of size $k\times q$) to the full-dimensional $X$ (of size $p\times q$), ensuring that only the columns corresponding to selected indices $j_1, j_2, \ldots, j_k$ are updated. This retains the efficiency of block updates while avoiding redundant computations on less impactful columns.

• Since the least-squares problem (1.2) is equivalent to the solution of the normal equation $A^TAX = A^TB$, applying the k-means clustering method to the row vectors of the coefficient matrix $A^TA$ transforms into directly clustering the column ones of the matrix $A$.

• In each cluster, the angle between any two column vectors of the matrix $A$ is as small as possible. In each iteration, we select the column vector within each cluster that contributes the most to reducing the residual, i.e., the column maximizing $\|A_{(j)}^T(B-AX^{(t)})\|_2^2$. This greedy selection of the most impactful columns for residual reduction helps accelerate the convergence of the iterative sequence.

• If the column index $j_i$ obtained in Algorithm 3 is not unique, we choose $j_i$ as follows:

$$j_i = \min\left\{arg\max_{j\in S_i} \left\|A_{(j)}^T(B - AX^{(t)})\right\|_2^2\right\}.$$

This is numerically stable and does not affect convergence (as proven in Theorem 4.1).

• For the update rule in the Step 3, it is derived from minimizing the residual within the block subspace spanned by columns of $I_S$. For the block update direction, we solve the local least-squares problem $\min\limits_{\Delta X_S\in\mathbb{R}^{k\times q}} \left\|A_S\Delta X_S - (B - AX^{(t)})\right\|$, whose solution is $\Delta X_S = (A_S^TA_S)^{-1}A_S^T(B - AX^{(t)})$. Multiplying by $I_S$ maps $\Delta X_S$ to the full-dimensional space, yielding the update for $X^{(t+1)}$. This block update strategy balances convergence speed (by leveraging multiple impactful directions) and computational cost (by limiting the block size to $k$), outperforming single-coordinate updates in large-scale scenarios. Particularly, the inversion of $A_S^TA_S$ is numerically stable. In practice, one can avoid direct inversion and instead solve the linear system $A_S^TA_SY = A_S^T(B - AX^{(t)})$ using Cholesky decomposition, which is backward stable and robust to moderate conditioning.

• The computational cost of the MCCD(k) method consists of two parts: the initial k-means clustering and the iterative update. Let $T_{mc}$ be the number of k-means iterations; this phase costs

$O(T_{mc}npk)$, which is a one-time overhead. Each iteration of this algorithm primarily involves: computing the residual $B - AX^{(t)}$ (computational amounts $O(npq)$); selecting the index set $S$ (computational amounts $O(npq)$) and solving the least-squares subproblems ($O(nk^2 + nkq + k^3)$). Thus, the per-iteration cost is $O(npq + nk^2 + nkq + k^3)$. Compared to the SD method [44] and the CG method [45], whose per-iteration cost is dominated by $O(npq + n^2p)$, the MCCD(k) method avoids the expensive $n^2p$ term related to the multiplication by $AA^T$. Moreover, the choice of $k$ represents a trade-off: a larger $k$ may improve the convergence rate but also increases the $O(k^3)$ cost per iteration.

• In implementation, we choose the initial iterative matrix $X^{(0)} = O$. In the full column rank case, it does not affect the uniqueness of the solution. However, in the rank deficient case, initializing from zero helps guide the iterative process towards the unique least-norm solution $A^\dagger B$, which is often the desired solution in practical applications due to its stability [6, 29].

## 4. Convergence analysis

In this section, we discuss the convergence of the algorithms proposed in this paper.

We begin with the following lemma. For the sake of completeness, we present its proof, which is slightly different from the original [46].

**Lemma 4.1.** *Let $H \in \mathbb{R}^{p\times m}, G \in \mathbb{R}^{m\times n}$. Then, $\sigma_{\max}(H)\|G\|_F \geq \|HG\|_F \geq \sigma_{\min}(H)\|G\|_F$.*

*Proof.* Suppose that $r = rank(H)$, and the singular value decomposition (SVD) of the matrix $H$ is $H = U\Sigma V^T$, where $U \in \mathbb{R}^{p\times p}$ and $V \in \mathbb{R}^{m\times m}$ are unitary matrices, $\Sigma = \begin{pmatrix} \Gamma & \\ & O \end{pmatrix}$, $\Gamma = diag(\sigma_1, \sigma_2, \ldots, \sigma_r)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. Partition the matrix $G$ into blocks by columns as $G = (G_1\ G_2\ \cdots\ G_n)$, then we have

$$\|HG\|_F^2 = \|U\Sigma V^T G\|_F^2 = \|\Sigma V^T (G_1\ G_2\ \cdots\ G_n)\|_F^2 = \sum_{j=1}^{n} \|\Sigma V^T G_j\|_2^2.$$

Let $C = \Sigma V^T$; using the Rayleigh-Ritz Theorem, we know that

$$\sigma_{min}^2(C)\|G_j\|_2^2 \leq \|\Sigma V^T G_j\|_2^2 = G_j^T C^T C G_j \leq \sigma_{max}^2(C)\|G_j\|_2^2,$$

together with the fact that $C^T C = V \begin{pmatrix} \Gamma^2 & \\ & O \end{pmatrix} V^T$, which yields

$$\sigma_r^2\|G\|_F^2 = \sum_{j=1}^{n} \sigma_r^2\|G_j\|_2^2 \leq \|HG\|_F^2 \leq \sum_{j=1}^{n} \sigma_1^2\|G_j\|_2^2 = \sigma_1^2\|G\|_F^2,$$

which implies that the conclusions hold. □

**Remark 1.** *Let $\sigma_{\min}^+(\cdot)$ represent the minimum positive singular value of a matrix. By Lemma 4.1, we obtain that $\|HG\|_F \geq \sigma_{\min}^+(H)\|G\|_F$ when the matrix $H$ is of full column rank. Otherwise, $\|HG\|_F \geq \sigma_{\min}^+(H)\|G\|_F$ if all the columns of $G$ belong to the range of $H^T$.*

The following well-known inequalities are indispensable for proving the main results.

**Lemma 4.2.** *Let $a_k > 0, k = 1, 2, \ldots, n$. Then*

$$\frac{n}{\sum\limits_{k=1}^{n}\frac{1}{a_k}} \leq \sqrt{\prod_{k=1}^{n} a_k} \leq \frac{\sum\limits_{k=1}^{n} a_k}{n} \leq \sqrt{\frac{\sum\limits_{k=1}^{n} a_k^2}{n}}.$$

Making use of the Algorithm 3 and the lemmas above, we can prove the following main results.

**Theorem 4.1.** *Suppose that $X^*$ is the least-squares solution to the problem* (1.2) *with full column rank A, then the sequence $\{X^{(t)}\}$ generated by Algorithm 3 converges to the point $X^*$, that is,*

$$\|A(X^{(t+1)} - X^*)\|_F^2 \leq (1 - \frac{\gamma k}{\delta p})\|A(X^{(t)} - X^*)\|_F^2, \tag{4.1}$$

*where* $\gamma = \min\limits_{1\leq i\leq k} \sigma_{\min}^2(A_{S_i})$, $\delta = \max\limits_{1\leq j\leq p} \|A_{(j)}\|_2^2$.

*Proof.* Noticing that $A^T B = A^T A X^*$, from Algorithm 3, we obtain that

$$\begin{aligned} A(X^{(t+1)} - X^{(t)}) &= A_S(A_S^T A_S)^{-1} A_S^T (B - AX^{(t)}) \\ &= A_S(A_S^T A_S)^{-1} I_S^T A^T (B - AX^{(t)}) \\ &= -A_S(A_S^T A_S)^{-1} A_S^T A(X^{(t)} - X^*) \\ &= -A_S A_S^\dagger A(X^{(t)} - X^*) \end{aligned} \tag{4.2}$$

and

$$\begin{aligned} A(X^{(t+1)} - X^*) &= A(X^{(t)} - X^*) + A_S(A_S^T A_S)^{-1} A_S^T (B - AX^{(t)}) \\ &= A(X^{(t)} - X^*) - A_S A_S^\dagger A(X^{(t)} - X^*) \\ &= (I - A_S A_S^\dagger) A(X^{(t)} - X^*). \end{aligned} \tag{4.3}$$

Since $A_S A_S^\dagger$ is an orthogonal projector onto the range of $A_S$, then it follows from the Pythagorean theorem that

$$\|A(X^{(t)} - X^*)\|_F^2 = \|A(X^{(t+1)} - X^{(t)})\|_F^2 + \|A(X^{(t+1)} - X^*)\|_F^2.$$

Then, together with the equalities (4.2) and (4.3), using Lemma 4.1, yields

$$\begin{aligned} \|A(X^{(t+1)} - X^*)\|_F^2 &= \|A(X^{(t)} - X^*)\|_F^2 - \|A(X^{(t+1)} - X^{(t)})\|_F^2 \\ &= \|A(X^{(t)} - X^*)\|_F^2 - \|A_S A_S^\dagger A(X^{(t)} - X^*)\|_F^2 \\ &= \|A(X^{(t)} - X^*)\|_F^2 - \|(A_S^\dagger)^T A_S^T A(X^{(t)} - X^*)\|_F^2 \\ &\leq \|A(X^{(t)} - X^*)\|_F^2 - \sigma_{\min}^2(A_S^\dagger)\|A_S^T A(X^{(t)} - X^*)\|_F^2 \\ &= \|A(X^{(t)} - X^*)\|_F^2 - \frac{1}{\sigma_{\max}^2(A_S)}\|A_S^T A(X^{(t)} - X^*)\|_F^2. \end{aligned} \tag{4.4}$$

Furthermore, utilizing the iterate scheme of Algorithm 3 again, we have

$$\begin{aligned} \|A_S^T A(X^{(t)} - X^*)\|_F^2 &= \sum_{i=1}^{k} \|A_{(j_i)}^T A(X^{(t)} - X^*)\|_2^2 \\ &= \sum_{i=1}^{k} \max_{j \in S_i} \|A_{(j)}^T A(X^{(t)} - X^*)\|_2^2 \\ &\geq \sum_{i=1}^{k} \frac{1}{|S_i|} \|A_{S_i}^T A(X^{(t)} - X^*)\|_F^2 \\ &\geq \sum_{i=1}^{k} \frac{1}{|S_i|} \sigma_{\min}^2(A_{S_i}) \|A(X^{(t)} - X^*)\|_F^2. \end{aligned} \tag{4.5}$$

Connecting with the inequalities (4.4) and (4.5), it follows that

$$\begin{aligned} \|A(X^{(t+1)} - X^*)\|_F^2 &\leq \|A(X^{(t)} - X^*)\|_F^2 - \frac{1}{\sigma_{\max}^2(A_S)} \sum_{i=1}^{k} \frac{1}{|S_i|} \sigma_{\min}^2(A_{S_i}) \|A(X^{(t)} - X^*)\|_F^2 \\ &\leq \|A(X^{(t)} - X^*)\|_F^2 - \frac{1}{\|A_S\|_F^2} \sum_{i=1}^{k} \frac{1}{|S_i|} \sigma_{\min}^2(A_{S_i}) \|A(X^{(t)} - X^*)\|_F^2 \\ &\leq (1 - \frac{1}{k\delta} \frac{\gamma k^2}{p}) \|A(X^{(t)} - X^*)\|_F^2 \\ &= (1 - \frac{\gamma k}{\delta p}) \|A(X^{(t)} - X^*)\|_F^2, \end{aligned} \tag{4.6}$$

in which we use the result of Lemma 4.2 and the fact that $\sigma_{\max}(A_S) \leq \|A_S\|_F$, as well as $\sum_{i=1}^{k} |S_i| = p$. The proof is completed. □

**Remark 2.** *It should be emphasized that the conditions $\sigma_{\min}(A_S) > 0$ and $\sigma_{\min}(A_{S_i}) > 0$ are essential in the proof procedures of inequalities* (4.3) *and* (4.4)*. These conditions are automatically satisfied when the coefficient matrix A in problem* (1.2) *is of full column rank. Under these circumstances, the least-squares solution to problem* (1.2) *is unique, specifically given by $X^* = A^\dagger B$. Moreover, Algorithm 3 is guaranteed to converge to this unique least-squares solution regardless of the choice of initial iterative values.*

**Remark 3.** *If the coefficient matrix A is not of full column rank, the least-squares problem* (1.2) *admits multiple solutions. It is well-known that the general solution can be expressed as $X = A^\dagger B + (I_p - A^\dagger A)Y$ where $Y \in \mathbb{R}^{p \times q}$, is an arbitrary matrix. Among these solutions, $A^\dagger B$ is the unique least-norm solution. To handle the case when A is rank-deficient, one may employ a regularization approach. Specifically, the problem* (1.2) *is reformulated as $\min_X \|AX - B\|_F^2 + \lambda \|X\|_F^2$, where $\lambda > 0$ is the regularization parameter. Accordingly, the iterative scheme in Algorithm 3 can be modified as*

$$X^{(t+1)} := X^{(t)} + I_S (A_S^T A_S + \lambda I_k)^{-1} A_S^T (B - AX^{(t)}).$$

*As $\lambda \to 0$, the solution of this regularized problem converges to the least-norm solution.*

From Theorem 4.1, we can obtain the following conclusion:

**Corollary 4.1.** *Under the conditions of Theorem 4.1, it holds that*

$$\|A(X^{(t+1)} - X^*)\|_F^2 \leq (1 - \frac{1}{\kappa^2(A)}\frac{k^2}{p^2})\|A(X^{(t)} - X^*)\|_F^2, \tag{4.7}$$

*where* $\kappa(A) = \|A\|_2\|A^\dagger\|_2$.

*Proof.* In fact, by (4.6), we have

$$\begin{aligned}
\|A(X^{(t+1)} - X^*)\|_F^2 &\leq \left(1 - \frac{1}{\|A_S\|_F^2}\sum_{i=1}^{k}\frac{1}{|S_i|}\sigma_{\min}^2(A_{S_i})\right)\|A(X^{(t)} - X^*)\|_F^2 \\
&\leq \left(1 - \frac{1}{\|A\|_F^2}\frac{k^2}{p}\sigma_{\min}^2(A)\right)\|A(X^{(t)} - X^*)\|_F^2 \\
&= \left(1 - \frac{\sigma_{\min}^2(A)}{\sum_{i=1}^{p}\sigma_i^2(A)}\frac{k^2}{p}\right)\|A(X^{(t)} - X^*)\|_F^2 \\
&\leq \left(1 - \frac{\sigma_{\min}^2(A)}{\sigma_{\max}^2(A)}\frac{k^2}{p^2}\right)\|A(X^{(t)} - X^*)\|_F^2 \\
&= \left(1 - \frac{1}{\kappa^2(A)}\frac{k^2}{p^2}\right)\|A(X^{(t)} - X^*)\|_F^2.
\end{aligned}$$

The proof is completed. □

The inequality (4.7) reveals that the size of the condition number $\kappa(A)$ is inversely proportional to the convergence speed of the algorithm given in present paper. Moreover, noting that $\sigma_{\min}^2(A) \leq \min_{1\leq i\leq k}\sigma_{\min}^2(A_{S_i})$, then we have

**Corollary 4.2.** *Under the conditions of Theorem 4.1 with $k = p$, it holds that*

$$\|A(X^{(t+1)} - X^*)\|_F^2 \leq (1 - \frac{\sigma_{\min}^2(A)}{\delta})\|A(X^{(t)} - X^*)\|_F^2.$$

## 5. Numerical experiments

In this section, several numerical experiments will be given to illustrate the efficiency of the Algorithm 3. All codes here were implemented in MATLAB software (version R2016a) on a personal computer with Intel(R) Core(TM) i7-10510U@1.80GHz and 8.00G memory. The numerical results to be shown later contain the number of iterations (denoted by 'IT'), the elapsed CPU time in seconds (denoted by 'CPU'), and the residual (denoted by 'RES') of the least-squares problem (1.2). $t_{\max}$ denotes the maximum number of iterations.

In the tests, we chose the zero matrix as the initial iteration matrix $X^{(0)}$, and the procedure corresponding to the MCCD(k) method terminates when the residual RES:$=\|B - AX^{(k)}\|_F < 1.0e-03$ for consistent matrix equations and ERR:$=\|A_S^T(B - AX^{(k)})\|_F < 1.0e-03$ for inconsistent ones, or the number of iterations exceeds the maximum $t_{\max} = 10000$, where $X^{(k)}$ denotes the $k$th iteration matrix. In order to illustrate and compare the convergence of the proposed method, we recall the following

two methods, which are very efficient for solving the matrix equation as shown in the literature. The first one is the steepest descent method (denoted by SD for short) proposed in [44], that is,

**Algorithm 4:** The SD method

Step 1: Input $A \in \mathbb{R}^{n\times p}$, $B \in \mathbb{R}^{n\times q}$, $X^{(0)} \in \mathbb{R}^{p\times q}$, $R^{(0)} = B$.

Step 2: For $t = 0, 1, 2, \ldots$, compute

$$\Delta X_t = \alpha_t A^T R^{(t)}, \alpha_t = \frac{\| A^T R^{(t)} \|_F^2}{\| AA^T R^{(t)} \|_F^2};$$

$$X^{(t+1)} = X^{(t)} + \Delta X_t;$$

Step 3: If $\Delta X_t = O$, stop. Otherwise, $R^{(t+1)} = B - AX^{(t+1)}$.

The second one is a slight variant of the CG-type method (denoted by CG for short) proposed in [45], that is,

**Algorithm 5:** The CG method

Step 1: Input $A \in \mathbb{R}^{n\times p}$, $B \in \mathbb{R}^{n\times q}$, $X^{(0)} \in \mathbb{R}^{p\times q}$, $R^{(0)} = B$.

Step 2: Compute $R^{(0)} = B - AX^{(0)}$, $P^{(0)} = A^T R^{(0)}$.

Step 3: For $t = 0, 1, 2, \ldots$, compute

$$X^{(t+1)} = X^{(t)} + \alpha_t P^{(t)}, \alpha_t = \frac{\| R^{(t)} \|_F^2}{\| P^{(t)} \|_F^2};$$

$$R^{(t+1)} = R^{(t)} - \alpha_t A P^{(t)};$$

$$P^{(t+1)} = A^T R^{(t+1)} + \beta_t P^{(t)}, \beta_t = \frac{\|R^{(t+1)}\|_F^2}{\|R^{(t)}\|_F^2};$$

Step 4: If $R^{(t+1)} = 0$, or $R^{(t+1)} \neq 0$, $P^{(t+1)} = 0$, stop.

**Example 1.** *Let the matrices* $A = \texttt{randn}(n, p) \in \mathbb{R}^{n\times p}$*, and* $B \in \mathbb{R}^{n\times q}$ *be chosen such that* $AX^* = B$*, where* $X^* = \texttt{ones}(p, q) * 8 \in \mathbb{R}^{p\times q}$*, where the function* $\texttt{ones}(\cdot)$ *generates the matrix with all elements being 1, and the function* $\texttt{randn}(\cdot)$ *generates the matrix with the entries obeying normal distribution.*

First, we validated the impact of clustering number $k$ on the MCCD(k) method numerically. Let $[n, p] = [5000, 500]$, and the results are shown in Figure 1, reflecting that the convergence of this method varies as varies $k$. The larger the $k$, the better the convergence effect of the algorithm. As analyzed in Section 3, a larger $k$ may improve the convergence rate but also increases the cost. However, considering the storage space and complexity of algorithm operations, for large-scale matrix equations, it is not possible to choose a particularly large value of $k$. In the following numerical experiments, we let $k = 50, 100, 200$ in our method.

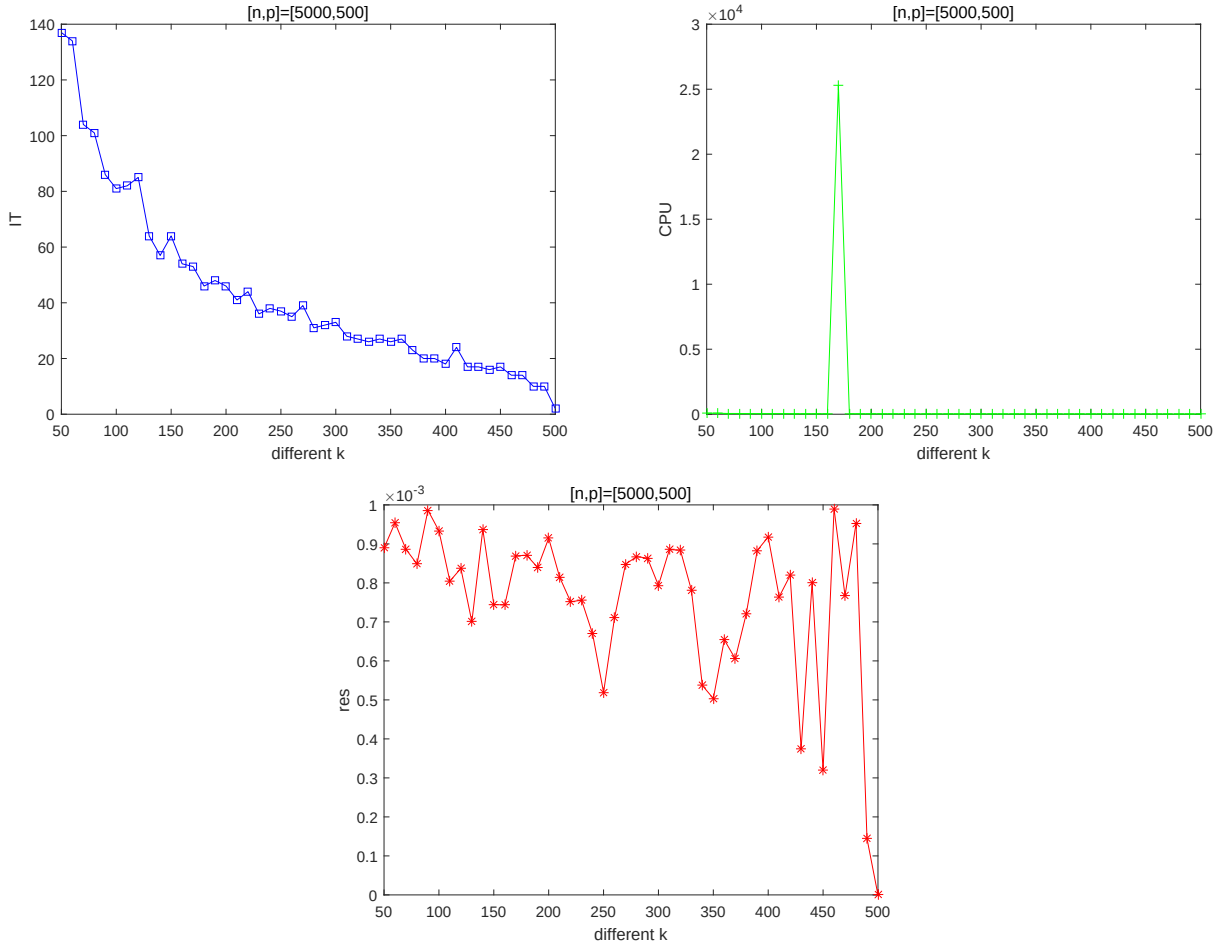


**Figure 1.** The impact of clustering $k$ on the proposed method for the matrix equation with $[n, p] = [5000, 500]$ in Example 1.

For the initial iteration matrix $X^{(0)} = O$, we ran the MCCD(k) method (i.e., Algorithm 3), the CG method [45], and the SD method [44], and the results are reported in Table 1 for different sizes of the known matrices $A$ and $B$. From this table, one can observe that all the methods are feasible; the number of iteration steps of the MCCD(k) method is almost equivalent to the other methods, but the running time is longer, which may be related to the time spent on clustering. In addition, the computational accuracy of the MCCD(k) and SD methods is significantly higher than that of the CG method.

Furthermore, the convergence curves of the three methods are plotted in Figure 2. From these figures, it can be seen that our algorithm is more stable. If a better value of $k$ is chosen, the convergence effect will be better.

**Table 1.** Numerical results for the matrix equations from Example 1.

| Algorithms | [n,p] | [1000, 500] | [5000, 500] | [10000, 500] | [10000, 1000] | [10000, 2000] |
|---|---|---|---|---|---|---|
| CG | IT | 261 | 38 | 26 | 41 | 76 |
| | CPU | 2.9803 | 0.5534 | 0.3713 | 4.3437 | 57.0365 |
| | RES | 3.0685e-02 | 2.3166e-02 | 3.0151e-02 | 2.6054e-02 | 2.7109e-02 |
| SD | IT | 169 | 24 | 16 | 25 | 45 |
| | CPU | 6.5062 | 2.9889 | 2.9902 | 20.5921 | 212.3252 |
| | RES | 9.1815e-04 | 6.2310e-04 | 8.3894e-04 | 8.9310e-04 | 9.1693e-04 |
| MCCD(50) | IT | 626 | 139 | 114 | 292 | 841 |
| | CPU | 48.4509 | 86.0094 | 151.5575 | 1916.7744 | 5253.1784 |
| | RES | 9.7927e-04 | 9.7829e-04 | 8.7995e-04 | 9.6407e-04 | 9.8309e-04 |
| MCCD(100) | IT | 350 | 79 | 64 | 160 | 437 |
| | CPU | 26.7420 | 50.2468 | 85.9783 | 1004.1155 | 3256.7706 |
| | RES | 9.5797e-04 | 9.2189e-04 | 9.4044e-04 | 9.6946e-04 | 9.9299e-04 |
| MCCD(200) | IT | 168 | 45 | 40 | 115 | 235 |
| | CPU | 14.4301 | 30.1226 | 57.1737 | 730.9638 | 2155.7868 |
| | RES | 9.8355e-04 | 7.3107e-04 | 9.6597e-04 | 9.7280e-04 | 9.7003e-04 |

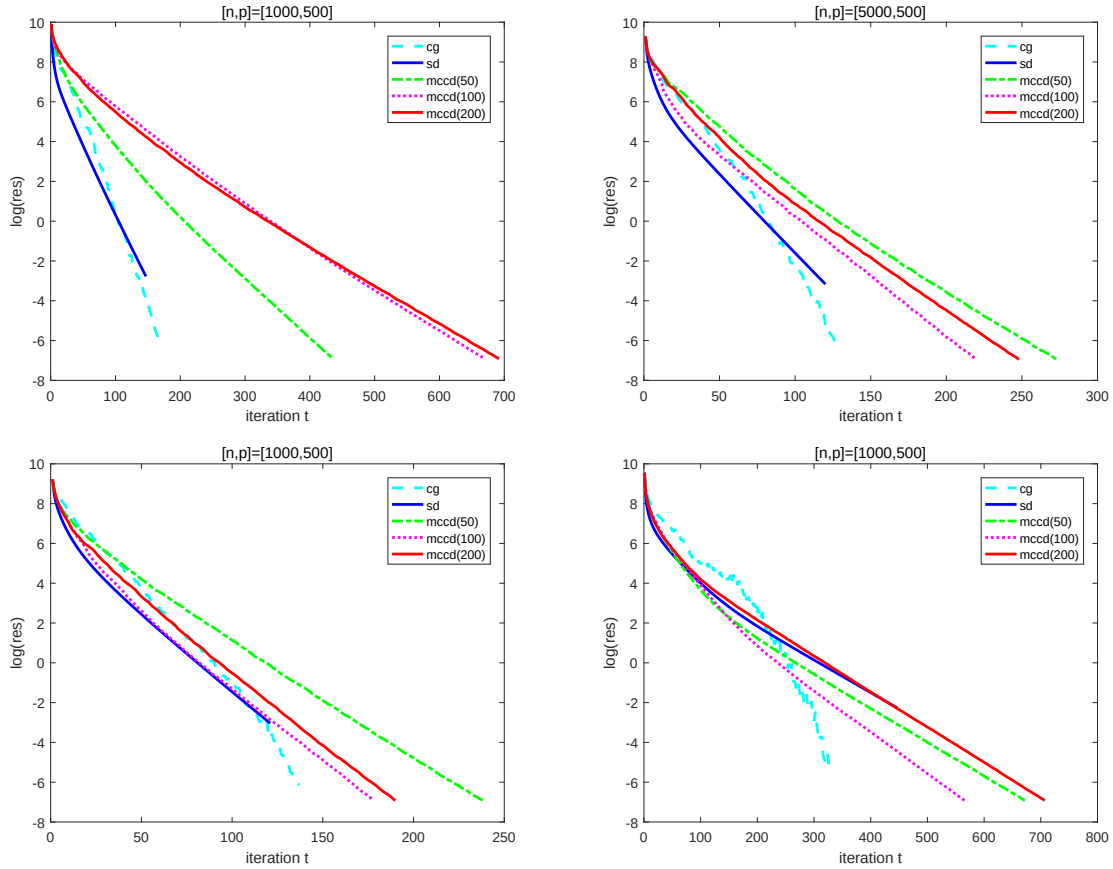


**Figure 2.** Comparison of the three methods for the matrix equations with different sizes in Example 1.

Next, we apply our method to the inconsistent matrix equations.

**Example 2.** *Let the matrices* $A = \texttt{randn}(n, p) \in \mathbb{R}^{n\times p}$, *and* $B = \texttt{randn}(n, q) \in \mathbb{R}^{n\times q}$.

In this case, the matrix Eq (1.1) is possibly inconsistent. We ran the MCCD(k) method, the CG method [45], and the SD method [44], and the results are displayed in Table 2 for different sizes of the known matrices $A$ and $B$. This table reveals that the CPU time of our method is less than that of the CG method, although its number is higher. This phenomenon may derive from the analysis on the complexity of Algorithm 3 as given in Section 3; the MCCD(k) method avoids the expensive $n^2p$ term related to the multiplication by $AA^T$ contained in the SD method and the CG method.

Furthermore, the convergence curves corresponding to the three methods are shown in Figure 3. These figures reflect similar phenomena as observed in Table 2.

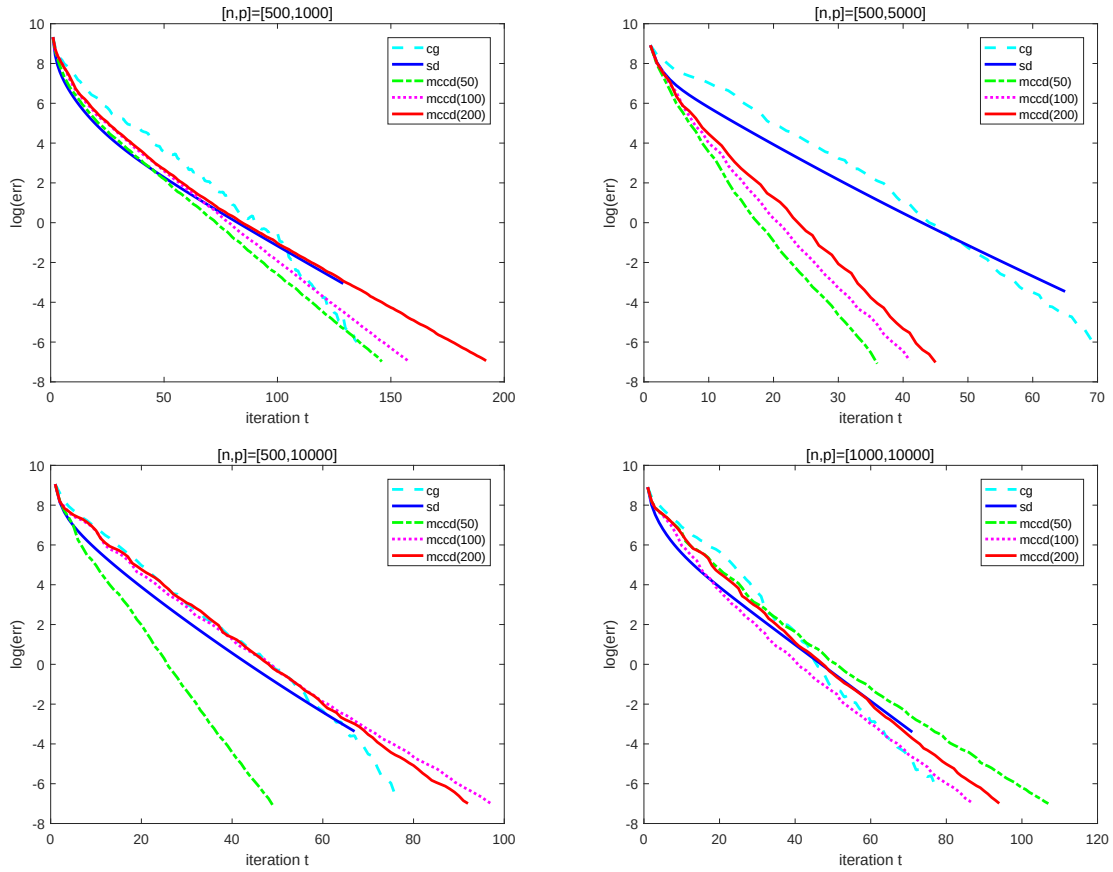


**Figure 3.** Comparison of the three methods for the matrix equations with different sizes in Example 2.

**Table 2.** Numerical results for the matrix equations from Example 2.

| Algorithms | [n,p] | [500, 1000] | [500, 5000] | [500, 10000] | [1000, 5000] | [1000, 10000] |
|---|---|---|---|---|---|---|
| CG | IT | 205 | 29 | 21 | 53 | 36 |
| | CPU | 6.8858 | 103.9580 | 544.0866 | 181.3354 | 495.1660 |
| | ERR | 3.0392e-02 | 2.7814e-02 | 1.9419e-02 | 3.0561e-02 | 2.6500e-02 |
| SD | IT | 98 | 14 | 10 | 24 | 16 |
| | CPU | 5.7671 | 57.1660 | 67696.1249 | 103.5517 | 214.2680 |
| | ERR | 9.4106e-04 | 7.2235e-04 | 4.7447e-04 | 8.9957e-04 | 9.0989e-04 |
| MCCD(50) | IT | 114 | 43 | 42 | 112 | 103 |
| | CPU | 6.7077 | 31.4229 | 139.9928 | 213.7561 | 1081.9577 |
| | ERR | 9.4375e-04 | 9.1518e-04 | 9.8397e-04 | 9.2232e-04 | 9.2540e-04 |
| MCCD(100) | IT | 118 | 43 | 42 | 112 | 103 |
| | CPU | 6.9729 | 31.4949 | 149.1594 | 215.8909 | 1706.7477 |
| | ERR | 9.5361e-04 | 9.2900e-04 | 9.9473e-04 | 9.3919e-04 | 9.8161e-04 |
| MCCD(200) | IT | 116 | 43 | 43 | 112 | 103 |
| | CPU | 6.7860 | 31.2526 | 149.7246 | 2128.1543 | 973.9959 |
| | ERR | 9.3236e-04 | 9.3358e-04 | 7.1454e-04 | 8.7161e-04 | 9.3619e-04 |

To this end, we apply the proposed MCCD(k) method to matrix equations with sparsely structured coefficient matrices.

**Example 3.** *Let the matrix* $A = \texttt{sprand}(n, p, 0.6) \in \mathbb{R}^{n\times p}$*, and the matrix* $B \in \mathbb{R}^{n\times q}$ *be given by* $AX^* = B$*, where* $X^* = \texttt{ones}(p, q) * 8 \in \mathbb{R}^{p\times q}$.

In this setting, the function $\texttt{sprand}(n, p, 0.6)$ generates an $n\times p$ random matrix with sparsity 0.6. For the initial iteration matrix $X^{(0)} = O$, we ran the MCCD(k) method, the CG method and the SD method, and the results are listed in Table 3 for different sizes of the matrices $A$ and $B$. This table demonstrates the feasibility of all three methods, with the SD method yielding the fewest iterations across all test cases. While the CG method involves fewer iterations than the proposed MCCD(k) method, the latter achieves markedly superior solution accuracy. Notably, the number of iterations of MCCD(k) declines substantially as the clustering number $k$ increases, which shows that a suitably selected $k$ directly accelerates the iterative convergence of MCCD(k). Moreover, the convergence curves of the three methods are plotted in Figure 4, from which it can be clearly observed that the proposed MCCD(k) algorithm exhibits more stable convergence behavior. Also, a more optimal choice of $k$ leads to further improved convergence performance. Moreover, the observed trends in Figure 4 are highly consistent with the numerical results presented in Table 3.

**Table 3.** Numerical results for the matrix equations from Example 3.

| Algorithms | [n,p] | [1000, 500] | [5000, 500] | [10000, 500] | [10000, 1000] | [10000, 2000] |
|---|---|---|---|---|---|---|
| CG | IT | 361 | 217 | 123 | 138 | 164 |
| | CPU | 21.1705 | 7.9227 | 7.8969 | 8.3242 | 17.5036 |
| | RES | 3.1220e-02 | 2.7231e-02 | 2.7241e-02 | 3.0649e-02 | 2.5976e-02 |
| SD | IT | 259 | 153 | 83 | 96 | 107 |
| | CPU | 11.6701 | 7.4598 | 5.0070 | 7.9793 | 12.0946 |
| | RES | 9.7513e-04 | 9.4064e-04 | 8.3280e-04 | 9.1506e-04 | 9.4862e-04 |
| MCCD(50) | IT | 2014 | 2944 | 2231 | 1569 | 2433 |
| | CPU | 118.6647 | 175.4897 | 171.8974 | 229.0704 | 370.6094 |
| | RES | 9.9551e-04 | 9.9598e-04 | 9.9571e-04 | 9.9463e-04 | 9.9834e-04 |
| MCCD(100) | IT | 3000 | 915 | 834 | 1615 | 2239 |
| | CPU | 161.0880 | 58.8761 | 67.4942 | 253.2235 | 353.4429 |
| | RES | 2.8254e-04 | 9.8673e-04 | 9.9329e-04 | 9.9603e-04 | 9.9551e-04 |
| MCCD(200) | IT | 357 | 327 | 249 | 421 | 408 |
| | CPU | 22.2875 | 24.7928 | 27.5847 | 71.2899 | 70.3762 |
| | RES | 9.6810e-04 | 9.8285e-04 | 9.3602e-04 | 9.7616e-04 | 9.6656e-04 |

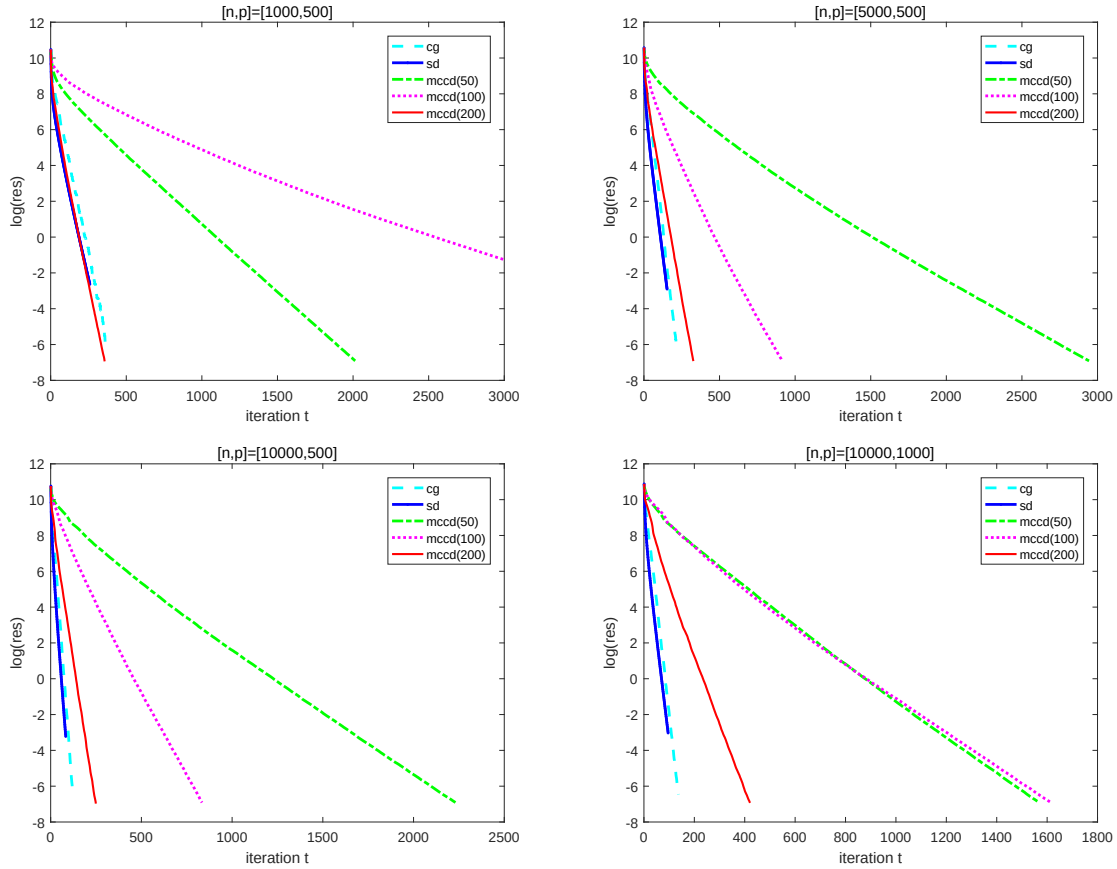


**Figure 4.** Comparison of the three methods for the matrix equations with different sizes in Example 3.

In summary, for the predefined accuracy, the proposed MCCD(k) method incurs longer CPU time than the SD and CG methods, and this is solely attributed to the one-time computational overhead of the cosine distance-based k-means clustering step for partitioning the columns of $A$— a fundamental design step unique to MCCD(k) that is not required by the classic SD and CG algorithms, which lack any clustering mechanism. Notwithstanding this, the MCCD(k) method demonstrates superior numerical performance and practical competitiveness in the vast majority of test cases, and its practical applicability for solving large-scale matrix equations is well validated by three key merits. First, it achieves a residual/error of $1.0e-04$, meeting the preset accuracy requirement of $1.0e-03$ with a significant margin, whereas the CG method only reaches a residual/error of $1.0e-04$ and thus fails to satisfy the predefined accuracy criterion. Second, for the most common practical scenario of inconsistent matrix equations, MCCD(k) consumes much less CPU time than the CG method and is even comparable to the SD method in CPU time for most cases, while maintaining the same high accuracy as SD. Third, as elaborated earlier, MCCD(k) completely avoids the expensive $AA^T$ matrix multiplication required by SD and CG. This overcomes the scalability bottleneck of SD and CG for extremely large $n$, making MCCD(k) far more suitable for real-world large-scale matrix equation problems where $n$ is sufficiently large. These advantages fully confirm its numerical competitiveness and practical applicability for solving the least-squares problem of the matrix equation $AX = B$.

## 6. Conclusions

In this paper, we have proposed a coordinate descent method integrated with k-means clustering for solving the least-squares problem of the matrix equation $AX = B$ with known matrices $A$ and $B$. The convergence of the proposed method is proven under the condition that the coefficient matrix $A$ is of full column rank. For the rank deficient case, we further addressed the corresponding least-squares problem by introducing a regularization strategy. Finally, we provided several numerical experiments to illustrate the feasibility and effectiveness of the proposed method, which reveal that our method has better performance if the appropriate clustering number $k$ is chosen. Notably, the proposed method exhibits high computational efficiency for large-scale problems with a large $n$ owing to its avoidance of the expensive $AA^T$ matrix multiplication, yet its performance is sensitive to the selection of the clustering number $k$. Thus, to explore an effective strategy to determine the optimal $k$ remains a promising research direction for future work. In addition, conducting a rigorous convergence analysis for the regularized version of the MCCD(k) method tailored for rank-deficient $A$ is also an important topic for subsequent research.

## Author contributions

L. Dai proposed the MCCD(k) method and conducted part of its convergence analysis. M. Liang contributed to the convergence analysis and performed some numerical analysis. Q. Li carried out part of the numerical experiments and assisted in revising the manuscript. R. Zhao completed part of numerical analysis. Y. Shen provided suggestions for revising the original version. All authors read and approved the final manuscript.

## Use of Generative-AI tools declaration

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

## Acknowledgments

The authors wish to thank the anonymous reviewers for their insightful comments and constructive suggestions, which have greatly improved the quality of this paper. We are also grateful to the editorial team for their professional guidance and efficient handling of the manuscript throughout the review process. This work was supported by the National Natural Science Foundation of China (Grant Nos. 12361081, 12201267, 12401500), the Innovation Foundation of Education Department of Gansu Province (Grant No. 2024A-125), the Young Doctor Foundation Program of Gansu Province (Grant No. 2026QB-083), the Special Project of Education Science '14th Five Year Plan' 2025 in Gansu Province (Grant No. GS[2025]GHBZX0099), the Zhejiang Provincial Natural Science Foundation of China (Grant No. ZCLQN25A0102), the First Class Discipline of Zhejiang-B (Zhejiang University of Finance and Economics-Statistics), and the Research Project of Tianshui Normal University (Grant No. TDJ2023-02).

## Conflict of interest

The authors declare that they have no conflict of interest.

## References

1. M. J. Corless, A. E. Frazho, *Linear systems and control: an operator perspective*, New York: CRC Press, 2003. https://doi.org/10.1201/9780203911372
2. B. N. Datta, Linear and numerical linear algebra in control theory: some research problems, *Linear Algebra Appl.*, **197–198** (1994), 755–790. https://doi.org/10.1016/0024-3795(94)90512-6
3. V. Simoncini, Computational methods for linear matrix equations, *SIAM Rev.*, **58** (2016), 377–441. https://doi.org/10.1137/130912839
4. F. Uhlig, On the matrix equation AX=B with applications to the generators of controllability matrix, *Linear Algebra Appl.*, **85** (1987), 203–209. https://doi.org/10.1016/0024-3795(87)90217-5
5. T. Sakurai, H. Tadano, Y. Kuramashi, Application of block Krylov subspace algorithms to the Wilson-Dirac equation with multiple right-hand sides in lattice QCD, *Comput. Phys. Commun.*, **181** (2010), 113–117. https://doi.org/10.1016/j.cpc.2009.09.006
6. A. Ben-Israel, T. Greville, *Generalized inverses: theory and applications*, New York: Springer, 2003. https://doi.org/10.1007/b97366
7. A. L. Andrew, Solution of equations involving centrosymmetric matrices, *Technometrics*, **15** (1973), 365–378. https://doi.org/10.1080/00401706.1973.10489049
8. K. E. Chu, Symmetric solutions of linear matrix equations by matrix decompositions, *Linear Algebra Appl.*, **119** (1989), 35–50. https://doi.org/10.1016/0024-3795(89)90067-0

9. Z. Y. Peng, X. Y. Hu, The reflexive and anti-reflexive solutions of the matrix equation $AX = B$, *Linear Algebra Appl.*, **375** (2003), 147–155. https://doi.org/10.1016/S0024-3795(03)00607-4
10. W. F. Trench, Hermitian, Hermitian $R$-symmetric, and Hermitian $R$-skew symmetric Procrustes problems, *Linear Algebra Appl.*, **387** (2004), 83–98. https://doi.org/10.1016/j.laa.2004.01.018
11. Q. F. Xiao, X. Y. Hu, L. Zhang, The symmetric minimal rank solution of the matrix equation $AX = B$ and the optimal approximation, *Electron. J. Linear Al.*, **18** (2009), 264–273. https://doi.org/10.13001/1081-3810.1311
12. M. L. Arias, M. C. Gonzalez, Proper splittings and reduced solutions of matrix equations, *J. Math. Anal. Appl.*, **505** (2022), 125588. https://doi.org/10.1016/j.jmaa.2021.125588
13. S. F. Yuan, Q. W. Wang, X. F. Duan, On solutions of the quaternion matrix equation AX=B and their applications in color image restoration, *Appl. Math. Comput.*, **221** (2013), 10–20. https://doi.org/10.1016/j.amc.2013.05.069
14. Y. Li, F. X. Zhang, W. B. Guo, J. L. Zhao, On solutions of the quaternion matrix equation AX=B and their applications in color image restoration, *Comput. Math. Appl.*, **62** (2011), 1389–1397. https://doi.org/10.1016/j.camwa.2011.04.004
15. Z. S. Yang, B. Zheng, Block minimum perturbation algorithm based on block Arnoldi process for nonsymmetric linear systems with multiple right-hand sides, *Appl. Math. Comput.*, **347** (2019), 741–766. https://doi.org/10.1016/j.amc.2018.11.024
16. W. Ding, Y. Li, D. Wang, Special least squares solutions of the reduced biquaternion matrix equation $AX = B$ with applications, *Comput. Appl. Math.*, **40** (2021), 279. https://doi.org/10.1007/s40314-021-01641-0
17. D. P. O'Leary, The block conjugate gradient algorithm and related methods, *Linear Algebra Appl.*, **29** (1980), 293–322. https://doi.org/10.1016/0024-3795(80)90247-5
18. M. Gutknecht, Block Krylov space methods for linear systems with multiple right-hand sides: an introduction, In: *Modern mathematical models, methods and algorithms for real world systems*, 2007.
19. B. Krasnopolsky, Revisiting performance of BiCGStab methods for solving systems with multiple right-hand sides, *Comput. Math. Appl.*, **79** (2020), 2574–2597. https://doi.org/10.1016/j.camwa.2019.11.025
20. V. Simoncini, E. Gallopoulos, An iterative method for nonsymmetric systems with multiple right-hand sides, *SIAM J. Sci. Comput.*, **16** (1995), 917–933. https://doi.org/10.1137/0916053
21. H. X. Zhong, G. Wu, G. L. Chen, A flexible and adaptive simpler block GMRES with deflated restarting for linear systems with multiple right-hand sides, *J. Comput. Appl. Math.*, **282** (2015), 139–156. https://doi.org/10.1016/j.cam.2014.12.040
22. M. Hached, K. Jbilou, Numerical methods for differential linear matrix equations via Krylov subspace methods, *J. Comput. Appl. Math.*, **370** (2020), 112674. https://doi.org/10.1016/j.cam.2019.112674
23. C. Jelich, M. Karimi, N. Kessissoglou, S. Marburg, Efficient solution of block Toeplitz systems with multiple right-hand sides arising from a periodic boundary element formulation, *Eng. Anal. Bound. Elem.*, **130** (2021), 135–144. https://doi.org/10.1016/j.enganabound.2021.05.003

24. V. Kalantzis, A. C. I. Malossi, C. Bekas, A. Curioni, E. Gallopoulos, A scalable iterative dense linear system solver for multiple right-hand sides in data analytics, *Parallel Comput.*, **74** (2018), 136–153. https://doi.org/10.1016/j.parco.2017.12.005
25. L. J. Zhao, X. Y. Hu, L. Zhang, Least squares solutions to $AX = B$ for bisymmetric matrices under a central principal submatrix constraint and the optimal approximation, *Linear Algebra Appl.*, **428** (2008), 871–880. https://doi.org/10.1016/j.laa.2007.08.019
26. L. Xu, X. Q. Liu, J. Liang, J. Wang, M. Li, S. Shen, Quantum algorithm for solving matrix equations of the form $AX = B$, *Laser Phys. Lett.*, **19** (2022), 055202. https://doi.org/10.1088/1612-202X/ac5b5a
27. Q. W. Wang, L. M. Xie, Z. H. Gao, A survey on solving the matrix equation $AXB = C$ with applications, *Mathematics*, **13** (2025), 450. https://doi.org/10.3390/math13030450
28. S. Hadjiantoni, G. Loizou, Numerical strategies for recursive least squares solutions to the matrix equation $AX = B$, *Int. J. Comput. Math.*, **100** (2023), 497–510. https://doi.org/10.1080/00207160.2022.2123220
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. A. Zouzias, N. M. Freris, Randomized extended Kaczmarz for solving least-squares, *SIAM J. Matrix Anal. Appl.*, **34** (2013), 773–793. https://doi.org/10.1137/120889897
31. 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
32. A. Ma, D. Needell, A. Ramdas, Convergence properties of the randomized extended Gauss-Seidel and Kaczmarz methods, *SIAM J. Matrix Anal. Appl.*, **36** (2015), 1590–1604. https://doi.org/10.1137/15M1014425
33. 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
34. A. K. Jain, Data clustering: 50 years beyond K-means, *Pattern Recogn. Lett.*, **31** (2010), 651–666. https://doi.org/10.1016/j.patrec.2009.09.011
35. 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
36. Y. Nesterov, Efficiency of coordinate descent methods on huge-scale optimization problems, *SIAM J. Optim.*, **22** (2012), 341–362. https://doi.org/10.1137/100802001
37. J. H. Zhang, J. H. Guo, On relaxed greedy randomized coordinate descent methods for solving large linear least-squares problems, *Appl. Numer. Math.*, **157** (2020), 372–384. https://doi.org/10.1016/j.apnum.2020.06.014
38. R. R. Li, H. Liu, Y. Y. Ding, Block Kaczmarz algorithm for solving large overdetermined systems of linear algebraic equations, *J. Comput. Math. Higher Educ. I.*, **43** (2021), 150–160.
39. H. L. Chen, A. L. Yang, R. Liu, Y. Cao, K-means clustering based maximal residual (block) Kaczmarz methods for solving consistent system of linear equations, *Comp. Appl. Math.*, **44** (2025), 307. https://doi.org/10.1007/s40314-025-03265-0

40. L. Rui. Y. Ai-Li, M. Yang, Maximal residual coordinate descent method with K-means clustering for solving large linear least-squares problems, *Numer. Algor.*, **101** (2026), 1157–1173. https://doi.org/10.1007/s11075-025-02036-6

41. D. A. Désopo, A convex programming procedure, *Nav. Res. Log. Quart.*, **6** (1959), 33–42. https://doi.org/10.1002/nav.3800060105

42. H. J. Shi, S. Y. Tu, Y. Y. Xu, W. Yin, A primer on coordinate descent algorithms, 2016. https://doi.org/10.48550/arXiv.1610.00040

43. S. J. Wright, Coordinate descent algorithms, *Math. Program.*, **151** (2015), 3–34. https://doi.org/10.1007/s10107-015-0892-3

44. K. H. Guo, X. Y. Hu, L. Zhang, A new iteration method for the matrix equation, *Appl. Math. Comput.*, **187** (2007), 1434–1441. https://doi.org/10.1016/j.amc.2006.09.059

45. G. X. Huang, F. Yin, K. Guo, An iterative method for the skew-symmetric solution and the optimal approximate solution of the matrix equation $AXB = C$, *J. Comput. Appl. Math.*, **212** (2008), 231–244. https://doi.org/10.1016/j.cam.2006.12.005

46. S. G. Shafiei, M. Hajarian, Developing Kaczmarz method for solving Sylvester matrix equations, *J. Franklin I.*, **359** (2022), 8991–9005. https://doi.org/10.1016/j.jfranklin.2022.09.028


AIMS Press