Electronic Research Archive

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

ERA, 32(11): 5926–5942.
DOI: 10.3934/era.2024274
Received: 21 September 2024
Revised: 21 October 2024
Accepted: 24 October 2024
Published: 04 Novemberr 2024

Research article

A hybrid singular value thresholding algorithm with diagonal-modify for low-rank matrix recovery

Ruiping Wen[1,*], **Liang Zhang**[2] **and Yalei Pei**[2]

[1] Shanxi Key Laboratory for Intelligent Optimization Computing and Block-chain Technology, Taiyuan Normal University, Jinzhong 030619, China

[2] School of Mathematics and Statistics, Taiyuan Normal University, Jinzhong 030619, China

* **Correspondence:** Email: wenrp@163.com; Tel: 03512886656; Fax: 03512886653.

Abstract: In this paper, a new hybrid singular value thresholding with diagonal-modify algorithm based on the augmented Lagrange multiplier (ALM) method was proposed for low-rank matrix recovery, in which only part singular values were treated by a hybrid threshold operator with diagonal-update, and which allowed the algorithm to make use of simple arithmetic operation and keep the computational cost of each iteration low. The new algorithm decreased the complexity of the singular value decomposition and shortened the computing time. The convergence of the new algorithm was discussed. Finally, numerical experiments shown that the new algorithm greatly improved the solving efficiency of a matrix recovery problem and saved the calculation cost, and its effect was obviously better than that of the other algorithms mentioned in experiments.

Keywords: low-rank matrix recovery; augmented Lagrange multiplier; diagonal-modify; hybrid singular value threshold

1. Introduction

The low-rank matrix recovery, also known as robust PCA or sparse and low-rank matrix decomposition, means to find the lowest rank matrices based on fewer linear measurements, arise in many fields such as collaborative filtering [1–3], machine learning [4], picture alignment [5, 6], signal processing [7], quantum state tomography [8] and more.

The problem of a low-rank matrix recovery was first introduced in [9–12], which can be regarded as a matrix-analogue of compressed sensing [13] refers to automatically identify the corrupted elements and recover the original matrix when some elements of the matrix were seriously missing. Wright et al. [14] had proved that the low-rank matrix A can be exactly recovered from matrix D by solving the

following convex optimization problem,

$$\begin{aligned} \min_{A,E} \quad & \|A\|_* + \lambda\|E\|_1, \\ \text{s.t.} \quad & D = A + E, \end{aligned} \tag{1.1}$$

where $\|A\|_* := \sum_{k=1}^{r} \sigma_k(A)$, $\sigma_k(A)$ denotes the k-th largest singular value of $A \in \mathbb{R}^{n_1 \times n_2}$ with $\text{rank}(A) = r$, was called the nuclear norm of A. The sum of the absolute values of the matrix E entries was denoted as $\|E\|_1$, and λ was a positive weighting parameter.

As for the solution of the model (1.1), there are a lot of studies both from theoretical and algorithmic aspects. For example, the iterative threshold (IT) algorithm (see [14]) had been put forward, which used a soft threshold operator and linear Bremgan iteration to solve model (1.1). The de-capitalize iterative hard threshold (IHT) algorithm (see [15]) had been proposed based on the influence of hard threshold operator on compressed sensing. In summary, the accelerated proximal gradient (APG) method (see [16]), the singular value thresholding (SVT) algorithm (see [17]), the dual algorithm (see [18]), the augmented Lagrangian multiplier (ALM) algorithm (see [19]), and the hybrid augmented Lagrange multiplier (HALM) algorithm (see [20]) had been later presented to deal with the convex problem (1.1). Numerous details and derivations on a low-rank matrix recovery problem can be referred to the references given therein.

This paper aims mainly at establishing a new hybrid thresholding with a diagonal-modify iterative algorithm based on the ALM algorithm for recovering a low-rank matrix. By using a diagonal-modify technique, the sequence matrices generated by the new algorithm were approximated in the true solution well, which saves significant computational cost.

The rest of the paper was organized as follows: In Section 2, the notations used throughout this paper and related preliminaries studies were introduced. In Section 3, we presented the proposed matrix recovery algorithm in detail. The convergence analysis of the new algorithm was given in Section 4. Then, numerical experiments were shown in Section 5. Finally, we ended the paper with the concluding remarks in Section 6.

2. Notations and preliminaries

In this position, we provide some basic notations and preliminaries that were used in our analysis. $\mathbb{R}^{n_1 \times n_2}$ denotes the set of $n_1 \times n_2$ real matrices; A^T is used to express the transpose of a matrix A. $\text{diag}(d_{11}, d_{22}, \ldots, d_{nn})$ stands for a diagonal matrix with the diagonal elements $d_{11}, d_{22}, \ldots, d_{nn}$. $\|A\|_1$ denotes the sum of the absolute values of matrix A entries. $\|A\|_2$ represents the spectral norm, the square root of the maximum eigenvalue of $A^T A$, and the Frobenius norm $\|A\|_F = \sqrt{\sum_{j=1}^{n_2} \sum_{i=1}^{n_1} a_{ij}^2}$. The $\text{tr}(A)$ represents the trace of A, and the standard inner product between two matrices is denoted by $\langle X, Y \rangle = \text{tr}(X^T Y)$.

Definition 1. (Singular Value Decomposition (SVD) [21]) The singular value decomposition (SVD) of a r-rank matrix $A \in \mathbb{R}^{n_1 \times n_2}$ is defined by

$$A = U \Sigma_r V^T, \qquad \Sigma_r = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r),$$

where $U \in \mathbb{R}^{n_1 \times r}$ and $V \in \mathbb{R}^{n_2 \times r}$ are both column orthogonal matrices, and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ are all positive singular values of A. By the way, $\|A\|_* := \sum_{k=1}^{r} \sigma_k(A)$ denotes the nuclear norm of A.

Definition 2. For each $\tau \geq 0$, the soft thresholding (shrinkage) operator (see [17]) S_τ was defined by

$$S_\tau[\sigma] = \begin{cases} \sigma - \tau, & if\ |\sigma| \geq \tau \\ 0, & if\ |\sigma| < \tau \end{cases},$$

the hard thresholding operator (see [22]) η_τ was defined by

$$\eta_\tau[\sigma] = \begin{cases} \sigma, & if\ |\sigma| \geq \tau \\ 0, & if\ |\sigma| < \tau \end{cases},$$

where $\sigma \in \mathbb{R}$, and τ is called a thresholding.

Definition 3. (Hybrid Singular Value Thresholding Operator [20]) Let $X = U\Sigma_r V^T \in \mathbb{R}^{n_1 \times n_2}$ be the SVD of X mentioned above. For each $\tau \geq 0, z > 1$, the hybrid thresholding operator $\mathcal{H}_{\tau,z}$ is defined by

$$\mathcal{H}_{\tau,z}(X) := U\mathcal{H}_{\tau,z}(\Sigma)V^T, \qquad \mathcal{H}_{\tau,z}(\Sigma) = \begin{cases} \eta_\tau[\sigma_i], & \sigma_i \geq z\tau \\ S_\tau[\sigma_i], & \sigma_i < z\tau \end{cases}.$$

Moreover, $S_\tau(A) := US_\tau(\Sigma)V^T$ and $\eta_\tau(A) := U\eta_\tau(\Sigma)V^T$ based on definition 2.

3. Description of algorithms

In this section, a new fast algorithm was proposed after introducing a related algorithm for solving the problem (1.1).

3.1. The augmented Lagrange multipliers (ALM) algorithm

It is well-known that the partial augmented Lagrange function of (1.1) is

$$\mathcal{L}(A, E, Y, \mu) = \|A\|_* + \lambda\|E\|_1 + \langle Y, D - A - E\rangle + \frac{\mu}{2}\|D - A - E\|_F^2,$$

where $A, E, Y \in \mathbb{R}^{n_1 \times n_2}$, $\mu > 0$ is the penalty parameter. It is reported that the algorithm of augmented Lagrange multipliers has been applied to the low-rank matrix recovery problem. It is of much better numerical behavior, and it is also of much higher accuracy. Then the augmented Lagrange multipliers algorithm is summarized in the following Algorithm 3.1.

For convenience, $[U_k, \Sigma_k, V_k] = \text{svd}(\cdot)$ denotes the SVD of the corresponding matrix.

Algorithm 3.1. (the ALM algorithm [19])

Step 0: Given a sampled matrix $D = A + E$, parameters $\mu_0 > 0$, $\rho > 1$. Given also two initial matrices $Y_0 = 0$, $E_0 = 0$, $k := 0$;

Step 1: Solve $A_k = \arg\min_A \mathcal{L}(A, E_k, Y_k, \mu_k)$, compute the SVD of the matrix $(D - E_k + \mu_k^{-1} Y_k)$,

$$[U_k, \Sigma_k, V_k] = \text{svd}(D - E_k + \mu_k^{-1} Y_k);$$

Step 2: Set

$$A_{k+1} = U_k S_{\mu_k^{-1}}(\Sigma_k) V_k^T.$$

Solves $E_{k+1} = \arg\min \mathcal{L}(A_{k+1}, E, Y_k, \mu_k)$,

$$E_{k+1} = S_{\mu_k^{-1}}(D - A_{k+1} + \mu_k^{-1} Y_k);$$

Step 3: If $\|D - A_{k+1} - E_{k+1}\|_F / \|D\|_F < \epsilon_1$ and $\mu_k \|E_{k+1} - E_k\|_F / \|D\|_F < \epsilon_2$, stop; otherwise, go to next Step;

Step 4: Set $Y_{k+1} = Y_k + \mu_k (D - A_{k+1} - E_{k+1})$,
If $\mu_k \|E_{k+1} - E_k\|_F / \|D\|_F < \epsilon_2$, set $\mu_{k+1} = \rho \mu_k$; otherwise, go to Step 1.

3.2. The hybrid augmented Lagrange multiplier algorithm with diagonal-modify

Because the soft thresholding operator compressed the thresholding in a constant way and maybe lost some effective large thresholdings, while the hard thresholding operator was discontinuous, which reduced the smoothness of the matrix. To complement the advantages of the two operators, a hybrid singular value threshold operator had been designed. Based on the augmented Lagrange multiplier algorithm, the new algorithm employs the hybrid singular threshold operator with diagonal-modify $\mathcal{H}$ to deal with the singular value in each iteration.

The details of the so-called hybrid singular value threshold operator: keep some large singular values unchanged, compress some middle size singular values in a constant way, and take some small singular values as 0. This operator can make up for the deficiency of a single soft or hard threshold operator, and improve the accuracy of the recovered matrix partially. Further, a diagonal-modify $\mathcal{W}(A) = AW$ was used to improve its recovery efficiency at the kth iteration, where the diagonal matrix $W_k = \text{diag}(w_{(k)}^{11}, w_{(k)}^{22}, \ldots, w_{(k)}^{nn}) \in \mathbb{R}^{n \times n}$ can be obtained by

$$W_k = \arg\min \|A - A_k W_k\|_F. \tag{3.1}$$

In fact, Eq (3.1) is easy to compute since it is so simple just some arithmetic operation required, without extra cost. In fact, the exact solution of (3.1) is given by

$$w_{(k)}^{jj} = \frac{\langle A(:, j), A_k(:, j) \rangle}{\langle A_k(:, j), A_k(:, j) \rangle}, \quad j = 1, \ldots, n.$$

The new algorithm can be designed as follows:

Algorithm 3.2. (Hybrid augmented Lagrange multiplier algorithm with diagonal-modify, denoted by **D-HALM**)

Step 0: Given a sampled matrix $D = A + E$. Set parameters $k > 0, \rho > 1, \epsilon_1, \epsilon_2$, the initial matrices $Y_0 = 0, E_0 = 0, k := 0$;

Step 1: Solve $A_k = \arg\min_A \mathcal{L}(A, E_k, Y_k, \mu_k)$, compute the SVD of the matrix $(D - E_k + \mu_k^{-1} Y_k)$,

$$[U_k, \Sigma_k, V_k] = \text{svd}(D - E_k + \mu_k^{-1} Y_k);$$

Step 2: Set

$$\tilde{A}_{k+1} = U_k \mathcal{H}_{\mu_k^{-1}, z}(\Sigma_k) V_k^T, \ W_k = \arg\min \|A - \tilde{A}_{k+1} W_k\|_F$$

Step 3: Set

$$A^{k+1} = A_{k+1} W_k.$$

Solves $E_{k+1} = \arg\min \mathcal{L}(A_{k+1}, E, Y_k, \mu_k)$,

$$E_{k+1} = \mathcal{H}_{\mu_k^{-1}, z}(D - A_{k+1} + \mu_k^{-1} Y_k);$$

Step 4: If $\|D - A_{k+1} - E_{k+1}\|_F / \|D\|_F < \epsilon_1$ and $\mu_k \|E_{k+1} - E_k\|_F / \|D\|_F < \epsilon_2$, stop, otherwise, go to next step;

Step 5: Set $Y_{k+1} = Y_k + \mu_k (D - A_{k+1} - E_{k+1})$, if $\mu_k \|E_{k+1} - E_k\|_F / \|D\|_F < \epsilon_2$, set $\mu_{k+1} = \rho \mu_k$,
otherwise, go to Step 1.

The difference with the classical ALM algorithm focuses on Steps 2 and 3. Moreover, Agorithm 3.2 includes the HALM algorithm in [20] as a special case when $W = I$.

4. Convergence analysis

In this section, we presented briefly the convergence of Algorithm 3.2 by analyzing the properties of the sequences $\{A_k\}, \{E_k\}$, and $\{Y_k\}$. Since the D-HALM algorithm was a progression of the classical ALM algorithm, its proof is a trivial generalization. For details of proofs and techniques, one can refer to [19] and references given therein.

Lemma 4.1. (see [17]) Let $A \in \mathbb{R}^{n_1 \times n_2}$ be a matrix and $U\Sigma V^T$ be its SVD. Then the subgradients set of the nuclear norm of A is given by

$$\partial \|A_k\|_* = \{UV^T + W : W \in \mathbb{R}^{n_1 \times n_2}, U^T W = 0, WV = 0, \|W\|_2 \le 1\}.$$

Lemma 4.2. (see [19])

$$\begin{aligned}
&\|E_{k+1} - E^*\|_F^2 + \mu_k^{-2} \|Y_{k+1} - Y^*\|_F^2 \\
=&\|E_k - E^*\|_F^2 + \mu_k^{-2} \|Y_k - Y^*\|_F^2 - \|E_{k+1} - E_k\|_F^2 - \mu_k^{-2} \|Y_{k+1} - Y_k\|_F^2 \\
&-2\mu_k^{-1} (\langle Y_{k+1} - Y_k, E_{k+1} - E_k \rangle + \langle A_{k+1} - A^*, \bar{Y}_{k+1} - Y^* \rangle + \langle E_{k+1} - E^*, Y_{k+1} - Y^* \rangle),
\end{aligned}$$

where (A^*, E^*) and Y^* are the optimal solutions to the problem (1) and its dual problem, respectively.

Lemma 4.3. Let $\overline{Y}_k = Y_{k-1} + \mu(D - A_k - E_{k-1})$, then the sequences $\{\overline{Y}_k\}$ and $\{Y_k\}$ are bounded.

Proof. Let $\overline{A}_{k+1} = U_k \mathcal{D}_{\mu_k^{-1}}(\Sigma) V_k^T$, we have

$$\begin{aligned}
\partial_A L(A_{k+1}, E_k, Y_k, \mu_k) &= \partial\|A_{k+1}\|_* - Y_k - \mu_k(D - A_{k+1} - E_k) \\
&= \partial\|\overline{A}_{k+1}\|_* - Y_k - \mu_k(D - \overline{A}_{k+1} - E_k) + \mu_k(A_{k+1} - \overline{A}_{k+1}) \\
&= \mu_k(A_{k+1} - \overline{A}_{k+1}) \\
&= \tau\mu_k \overline{U}_k \overline{V}_k^T \\
&= \overline{U}_k \overline{V}_k^T.
\end{aligned}$$

Thus,

$$\overline{Y}_{k+1} = \partial\|A_{k+1}\|_* - \partial_A L(A_{k+1}, E_k, Y_k, \mu_k).$$

Now we can obtain the following result:

$$\|\overline{Y}_{k+1}\|_2 \le \|\partial\|A_{k+1}\|_*\|_2 - \|\partial_A L(A_{k+1}, E_k, Y_k, \mu_k)\|_2 \le 2.$$

From E_{k+1}, as shown in Algorithm 3.2, we can obtain

$$\partial_E L(A_{k+1}, E_{k+1}, Y_k, \mu_k) = \partial(\|\lambda E_{k+1}\|_1) - Y_k - \mu_k(D - A_{k+1} - E_{k+1}) = 0.$$

Hence,

$$Y_{k+1} \in \partial(\|\lambda E_{k+1}\|_1).$$

The boundness of the sequences $\{\overline{Y}_k\}$ and $\{Y_k\}$ is obtained.

Lemma 4.4. (see [19]) The subgradient of a convex function is a monotone operator. Namely,

$$\langle x_1 - x_2, y_1 - y_2\rangle \ge 0, \quad \forall y_i \in \partial f(x_i),\ i = 1, 2$$

under f is a convex function.

Lemma 4.5. (see [19]) The terms of the series

$$\sum_{k=1}^{\infty} \mu_k^{-1}(\langle Y_{k+1} - Y_k, E_{k+1} - E_k\rangle + \langle A_{k+1} - A^*, \bar{Y}_{k+1} - Y^*\rangle + \langle E_{k+1} - E^*, Y_{k+1} - Y^*\rangle)$$

is nonnegative, and the series is convergent provided that μ_k is nondecreasing.

In Algorithm 3.2, we do not have to solve the sub-problem

$$(A_k, E_k) = \arg\min_{A,E} \mathcal{L}(A, E_k, Y_k, \mu_k)$$

exactly. Rather, updating them once when solving this sub-problem is sufficient for A_k and E_k to convergence the local optimal solution of the problem (1). And this leads to an inexact ALM with diagonal-modify, similar to the IALM introduced in [19]. After the analysis on A_k, E_k, and Y_k via these Lemmas above, the validity and optimality of Algorithm 3.2 are guaranteed by the following theorem (see [19]).

Theorem 4.1 Suppose that μ_k is nondecreasing and the series $\sum_{i=1}^{\infty} \mu_k^{-1}$ diverges, then the sequence $\{(A_k, E_k)\}$ converges to the optimal solution (A^*, E^*) of problem (1.1).

Proof. Because of $Y^* \in \partial\|A^*\|_*$, $Y^* \in \partial(\|\lambda E^*\|_1)$, $Y_{k+1} \in \partial(\|\lambda E_{k+1}\|_1)$, and $\overline{Y}_{k+1} = \partial\|A_{k+1}\|_* - \partial_A L(A_{k+1}, E_k, Y_k, \mu_k)$, we have

$$\langle Y_{k+1} - Y^*, E_{k+1} - E^* \rangle \geq 0,$$

$$\langle E_{k+1} - E_k, Y_{k+1} - Y_k \rangle \geq 0.$$

By analysis, we obtain

$$\begin{aligned}\langle A_{k+1} - A^*, \overline{Y}_{k+1} - Y^* \rangle &= \langle A_{k+1} - A^*, \partial\|A_{k+1}\|_* - \partial\|A^*\|_* + \overline{U}_k \overline{V}_k^T \rangle \\ &= \langle A_{k+1} - A^*, \partial\|A_{k+1}\|_* - \partial\|A^*\|_* \rangle + \langle A_{k+1} - A^*, \overline{U}_k \overline{V}_k^T \rangle.\end{aligned}$$

Obviously, we can obtain

$$\langle A_{k+1} - A^*, \partial\|A_{k+1}\|_* - \partial\|A^*\|_* \rangle \geq 0,$$

$$\langle A_{k+1} - A^*, \overline{U}_k \overline{V}_k^T \rangle \geq 0.$$

Thus, $\|E_k - E^*\|^2 + \mu_k^{-2}\|Y_k - Y^*\|^2$ is non-increasing, as shown in [19]. From the Theorem 2 in [19], it holds

$$\lim_{k\to\infty} A_k = A^*, \ \lim_{k\to\infty} E_k = E^*,$$

we obtain that (A^*, E^*) is the solution of the low-rank matrix recovery problem (1.1), which completes the proof.

5. Numerical experiments

In this section, some random data and video sequences were used to demonstrate the proposed algorithm was feasible and effective to solve a low-rank matrix recovery problem (1.1). All the experiments are performed under Windows 11 and MATLAB R2019a running on a DELL laptop.

5.1. Random data

Some original results of four algorithms (APG, ALM, HALM, and D-HALM) were provided for the $n \times n$ matrices with different ranks for the problem (1.1). We conduct numerical experiments on the same workstation. By analyzing and comparing iteration numbers (denoted by IT), computing time in seconds (denoted by CPU(s)), and deviation (Error 1, Error 2), defined by

$$\text{Error 1} = \frac{\|\overline{A} - A^*\|_F}{\|A^*\|_F}, \quad \text{Error 2} = \frac{\|\overline{E} - E^*\|_F}{\|E^*\|_F}.$$

In the experiments, $p = \frac{m}{n^2}$ represents the sampling density, where m is the number of observed entries. We denote the optimal solution by the ordered pair (A^*, E^*), and the output by $(\overline{A}, \overline{E})$. For

Algorithm 3.2 (D-HALM), we choose a fixed weighting parameter $\lambda = \frac{1}{\sqrt{n}}$, set $Y_0 = \frac{D}{J(D)}(J(D) = \max(\|D\|_2, \lambda^{-1}\|D\|_\infty)$ [19], and we empirically set the parameter $\mu_0 = \frac{1.25}{\|D\|_2}, \tau_0 = 0.5\|D\|_2, \epsilon_1 = 1 \times 10^{-8}, \epsilon_2 = 1 \times 10^{-7}, z = 1.4$. By the way, the ALM, HALM, and D-HALM algorithms share the same parameters.

Tests were conducted for $n_1 = n_2, p \in \{0.1, 0.2, 0.3, 0.4\}$, and the rank of the underlying matrix to be reconstructed, $r = 0.005n$. The specific comparison results were shown in Tables 1–4, and the compared curve of the computing time for different algorithms was mapped in Figures 1 and 2.

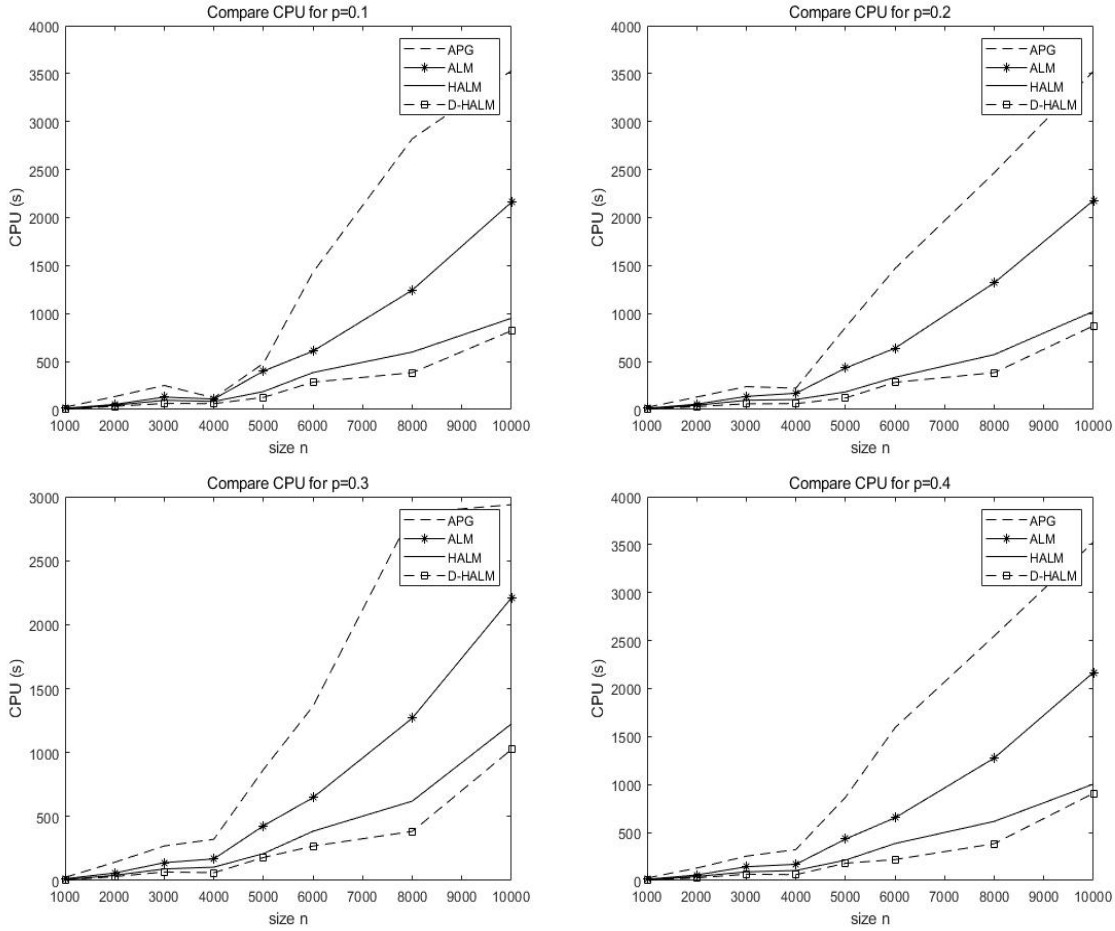

Figure 1. Compared curve of CPU(s) time of four algorithms when $n \leq 10^4$.

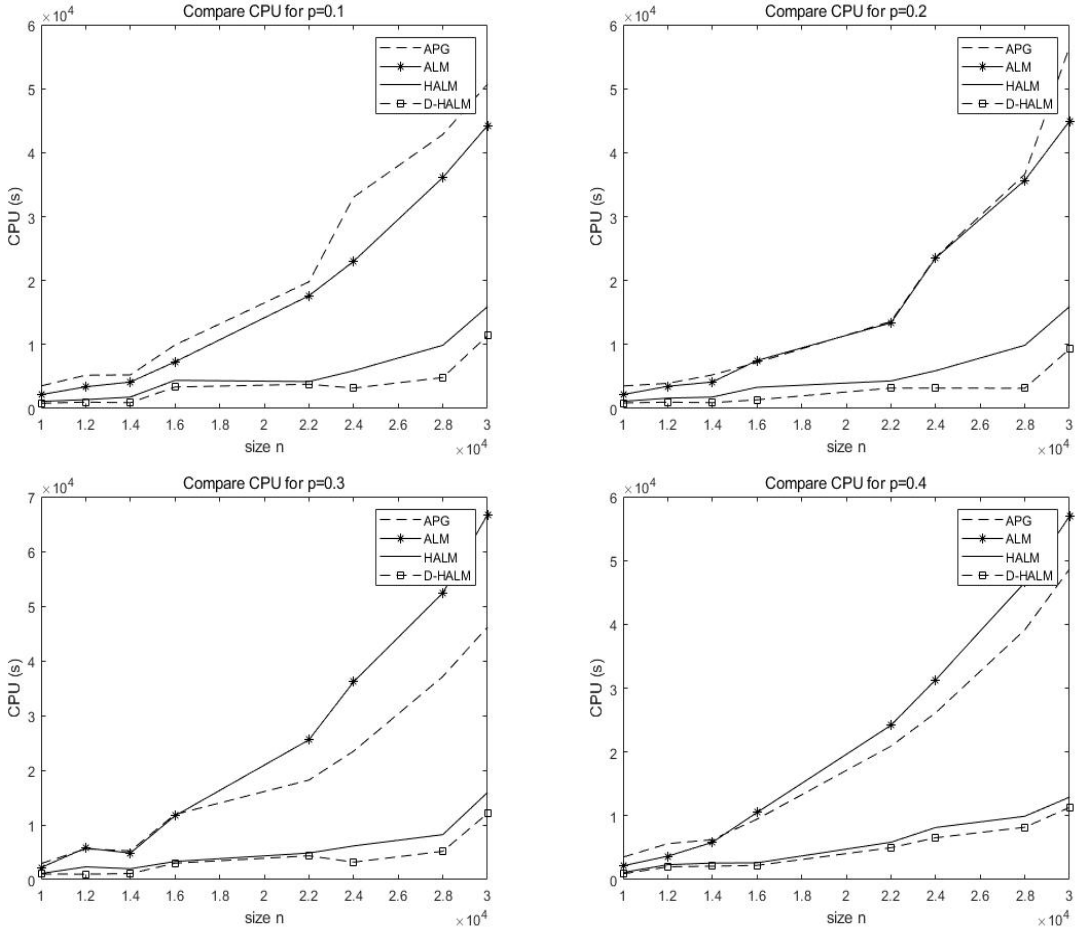

Figure 2. Compared curve of CPU(s) time of four algorithms when $n \geq 10^4$.

Table 1. Compared results of four algorithms for $p = 0.1$.

Size n	r(A)	Algorithm	IT	Error 1	Error 2	CPU(s)	SP
		APG	120	4.91e-7	2.03e-5	135.06	3.9
2000	10	ALM	33	8.57e-10	1.25e-7	53.56	1.5
		HALM	33	2.01e-9	1.99e-7	44.10	1.3
		D-HALM	32	1.52e-9	1.91e-7	34.90	1
		APG	120	5.24e-7	2.63e-5	250.30	4.0
3000	15	ALM	33	1.20e-9	1.90e-7	129.75	2.1
		HALM	32	2.09e-9	1.85e-7	96.51	1.5
		D-HALM	32	1.13e-9	1.79e-7	62.96	1
		APG	119	5.32e-7	3.76e-5	1431.28	3.7
6000	30	ALM	33	1.15e-9	2.63e-7	608.93	1.6
		HALM	34	1.13e-9	2.39e-7	420.77	1.1
		D-HALM	31	1.11e-9	2.51e-7	385.50	1
		APG	120	6.56e-7	6.11e-5	3530.34	4.3
10,000	50	ALM	33	1.23e-9	3.34e-7	2160.80	2.6
		HALM	33	1.32e-9	3.92e-9	951.68	1.2
		D-HALM	31	1.02e-9	2.98e-7	821.62	1
		APG	119	5.32e-7	6.19e-5	9932.01	3.0
16,000	80	ALM	34	1.52e-9	5.68e-7	7293.78	2.2
		HALM	34	1.49e-9	5.56e-7	4404.62	1.3
		D-HALM	32	1.09e-9	4.96e-7	3363.61	1
		APG	117	6.83e-7	9.43e-5	19,800.99	5.2
22,000	110	ALM	35	9.22e-10	4.04e-7	17,623.04	4.6
		HALM	34	8.70e-10	3.63e-7	4212.63	1.1
		D-HALM	33	8.97e-10	3.93e-7	3795.17	1
		APG	117	6.81e-7	1.10e-4	50,699.09	4.5
30,000	150	ALM	34	9.51e-10	4.86e-7	44,212.62	3.9
		HALM	33	8.76e-10	4.49e-7	15,920.68	1.4
		D-HALM	32	6.76e-10	3.69e-7	11,382.12	1

Table 2. Compared results of four algorithms for $p = 0.2$.

Size n	r(A)	Algorithm	IT	Error 1	Error 2	CPU(s)	SP
		APG	122	4.26e-7	1.60e-5	129.65	4.2
2000	10	ALM	33	1.30e-9	8.41e-8	55.65	1.8
		HALM	32	1.39e-9	1.55e-7	44.15	1.4
		D-HALM	31	1.46e-9	1.21e-7	30.57	1
		APG	123	3.84e-7	1.75e-5	238.86	4.0
3000	15	ALM	33	1.69e-9	1.74e-7	135.97	2.3
		HALM	33	1.62e-9	1.33e-7	96.55	1.6
		D-HALM	30	1.47e-9	1.58e-7	59.22	1
		APG	122	4.26e-7	2.75e-5	1469.07	5.0
6000	30	ALM	34	9.77e-10	1.45e-7	639.15	2.3
		HALM	33	8.99e-10	1.99e-7	335.54	1.2
		D-HALM	32	1.01e-9	1.52e-7	283.67	1
		APG	120	4.46e-7	4.13e-5	3525.37	4
10,000	50	ALM	34	1.09e-9	2.10e-7	2178.51	2.5
		HALM	34	1.14e-9	2.20e-7	1022.68	1.2
		D-HALM	31	1.04e-9	2.11e-7	872.03	1
		APG	124	3.80e-7	3.60e-5	11,914.50	4.0
16,000	80	ALM	34	1.75e-9	3.16e-7	11,673.95	3.9
		HALM	34	1.28e-9	3.12e-7	3316.21	1.1
		D-HALM	32	1.03e-9	2.02e-7	3001.70	1
		APG	120	5.48e-7	6.88e-5	17,883.42	5.6
22,000	110	ALM	36	1.37e-9	3.91e-7	18,003.14	5.7
		HALM	35	1.55e-9	3.12e-7	4312.69	1.4
		D-HALM	33	1.07e-9	2.41e-7	3172.79	1
		APG	120	5.50e-7	8.07e-5	56,540.07	6.1
30,000	150	ALM	35	1.50e-9	5.00e-7	44,925.23	4.8
		HALM	35	7.43e-10	2.48e-7	15,920.68	1.7
		D-HALM	31	2.43e-10	1.08e-7	9287.39	1

Table 3. Compared results of four algorithms for $p = 0.3$.

Size n	r(A)	Algorithm	IT	Error 1	Error 2	CPU(s)	SP
		APG	124	3.71e-7	1.26e-5	142.08	4.6
2000	10	ALM	34	9.14e-10	5.39e-8	58.00	1.9
		HALM	34	8.22e-10	5.66e-7	42.1	1.4
		D-HALM	31	2.06e-10	3.77e-8	30.90	1
		APG	124	3.75e-7	1.56e-5	269.70	4.0
3000	15	ALM	34	1.20e-9	9.53e-8	139.49	2.1
		HALM	34	2.11e-9	9.77e-8	90.36	1.4
		D-HALM	31	1.09e-9	6.18e-8	66.81	1
		APG	124	3.75e-7	2.22e-5	1363.39	5.1
6000	30	ALM	34	1.41e-9	1.55e-7	648.55	2.4
		HALM	33	2.33e-9	3.65e-7	385.87	1.4
		D-HALM	33	1.07e-9	1.10e-7	269.28	1
		APG	124	4.27e-7	3.29e-5	2937.07	2.9
10,000	50	ALM	34	1.54e-9	2.19e-7	2209.08	2.1
		HALM	34	1.32e-9	2.18e-7	1224.68	1.2
		D-HALM	31	1.02e-9	2.11e-7	1024.10	1
		APG	124	3.80e-7	3.60e-5	11,914.50	4.0
16,000	80	ALM	34	1.75e-9	3.16e-7	11,673.95	3.9
		HALM	33	1.75e-9	3.12e-7	3316.5	1.1
		D-HALM	32	1.03e-9	2.02e-7	3001.70	1
		APG	122	4.91e-7	5.61e-5	18,187.18	4.2
22,000	110	ALM	34	1.02e-9	2.15e-7	25,549.05	5.9
		HALM	34	1.11e-9	2.20e-7	4855.6	1.1
		D-HALM	30	1.02e-9	1.15e-7	4354.47	1
		APG	122	4.71e-7	6.53e-5	46,154.80	3.8
30,000	150	ALM	34	1.10e-9	2.73e-7	66,690.46	5.5
		HALM	34	8.55e-10	2.31e-7	15,920.68	1.3
		D-HALM	31	5.98e-10	2.08e-7	12,095.16	1

Table 4. Compared results of four algorithms for $p = 0.4$.

Size n	r(A)	Algorithm	IT	Error 1	Error 2	CPU(s)	SP
		APG	124	3.36e-7	1.04e-5	129.32	4.7
2000	10	ALM	33	1.14e-9	5.29e-8	57.46	2.1
		HALM	33	1.39e-9	4.88e-7	42.1	1.6
		D-HALM	32	1.02e-9	2.81e-8	27.83	1
		APG	126	3.36e-7	1.29e-5	253.08	3.9
3000	15	ALM	34	1.44e-9	8.37e-8	143.15	2.2
		HALM	33	1.48e-9	8.12e-8	90.40	1.4
		D-HALM	33	1.57e-9	7.58e-8	65.77	1
		APG	126	3.38e-7	1.81e-5	1593.19	7.3
6000	30	ALM	34	1.83e-9	1.57e-7	654.36	3.0
		HALM	33	1.77e-9	1.98e-7	385.45	1.8
		D-HALM	31	1.09e-9	1.54e-7	218.02	1
		APG	124	4.21e-7	2.96e-5	3527.02	3.9
10,000	50	ALM	34	2.04e-9	2.24e-7	2168.58	2.4
		HALM	34	2.08e-9	2.24e-7	1004.68	1.1
		D-HALM	32	2.06e-9	2.16e-7	908.74	1
		APG	126	3.38e-7	2.97e-5	9472.75	4.2
16,000	80	ALM	35	1.24e-9	1.72e-7	10,515.76	4.7
		HALM	35	1.25e-9	1.74e-7	2636.23	1.2
		D-HALM	33	1.25e-9	1.74e-7	2236.69	1
		APG	123	4.96e-7	5.05e-5	20,905.17	4.2
22,000	110	ALM	39	1.37e-9	2.24e-7	24,175.33	4.8
		HALM	38	1.38e-9	2.36e-7	5855.87	1.2
		D-HALM	34	1.36e-9	2.23e-7	5000.01	1
		APG	124	4.38e-7	5.32e-5	48,635.34	4.3
30,000	150	ALM	35	1.46e-9	2.79e-7	56,912.09	5.0
		HALM	35	1.43e-9	2.63e-7	12,920.68	1.2
		D-HALM	35	1.30e-9	2.50e-7	11,302.13	1

From Figures 1 and 2, it can be seen that the time of the D-HALM algorithm was always less than other algorithms. The larger the matrix scale, the more obvious the effect.

As can be shown in Tables 1–4, the proposed algorithm, ALM, and HALM algorithms achieve almost the same iterations and relative errors. However, we can see the CPU of our algorithm was obviously less than that of APG, ALM, and HALM algorithms.

By introducing the speed-up (denoted by SP) as follows,

$$\text{SP} = \frac{\text{time of the other algorithm}}{\text{time of the D-HALM algorithm}},$$

it is noted that the maximum SP values of APG, ALM and HALM algorithms were approximately 7.3, 2.6, and 1.6 when the matrix size $n \leq 10000$; however, those of them were approximately 6.1, 5.9 and 1.7. The computational cost of the proposed algorithm was significantly lower than the other

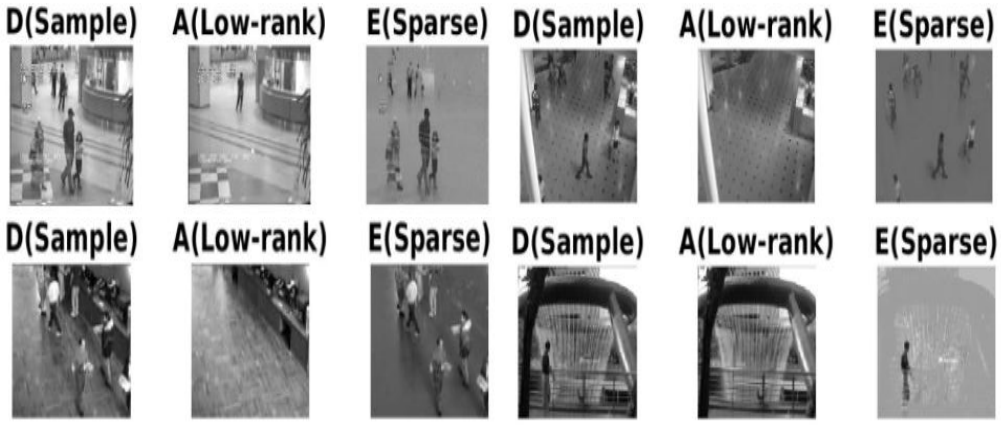

Figure 3. Experiment results of ALM method on videos 1-4, respectively.

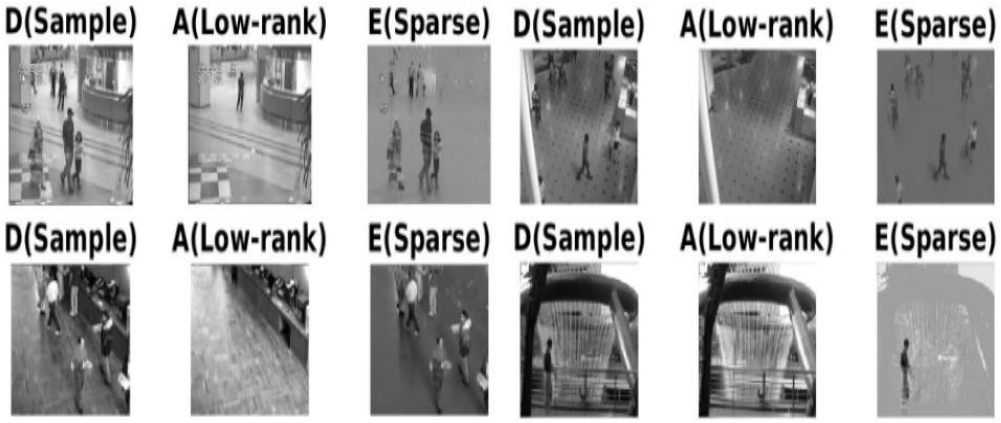

Figure 4. Experiment results of HALM algorithm on videos 1–4, respectively.

algorithms.

5.2. Video sequences

In this subsection, we use four surveillance video sequences (video 1: Hall of a business building; video 2: Mall; video 3: Bootstrap; video 4: Fountain) with different resolutions, respectively, to compare the separated effects of three ADMM algorithms (ALM, HALM, and D-HALM). One of 200 images of four video sequences was randomly selected for testing. Figures 3–5 shown the results of foreground and background separation of four surveillance video sequences under the $D = A + E$ model. The error and running time were reported in Table 5, where

$$\text{Error} = \frac{\|A + E - D\|_F}{\|D\|_F}.$$

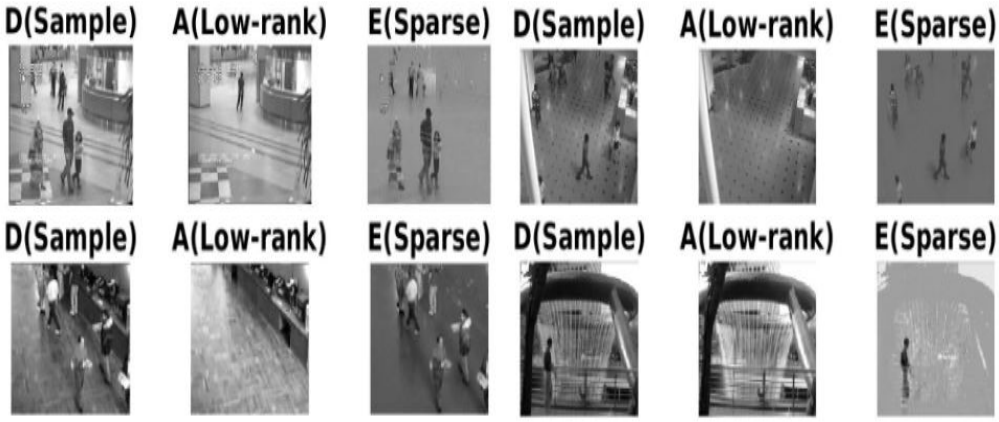

Figure 5. Experiment results of D-HALM algorithm on videos 1–4, respectively.

Table 5. Background foreground separation experiment randomly extracts of a graph.

Video	Resolution	Item	ALM	HALM	D-HALM
Hall	144×176	Error	8.83e-8	6.90e-8	6.42e-8
		CPU(s)	334.35	198.23	108.85
Mall	256×320	Error	8.52e-8	8.23e-8	8.07e-8
		CPU(s)	1038.13	231.88	191.09
Bootstrap	120×160	Error	9.17e-8	8.06e-8	7.31e-8
		CPU(s)	275.68	112.36	98.57
Fountain	128×160	Error	8.58e-8	7.05e-8	6.45e-8
		CPU(s)	304.14	135.33	99.30

From Table 5, it can be seen that three algorithms shared similar separation accuracy while the running time of D-HALM was cost-effective.

6. Conclusions

To solve the low-rank matrix recovery problem, we have proposed a hybrid singular value thresholding algorithm with diagonal-modify (D-HALM) based on the classical ALM algorithm. In the proposed algorithm, a hybrid singular value thresholding was used and a diagonal-modify was carried out in each iteration. The iterative sequence generated by Algorithm 3.2 for solving the optimization models converges to the optimal solution, which can be guaranteed by the traditional convergence theory. In experiments, compared with APG, ALM, and HALM algorithms introduced in the past three years. The experiments based on randomly numerical simulation data and video sequence separation have shown both that the proposed algorithm in this paper takes less time and gets more precision for solving the low-rank matrix recovery problem. It was found that the D-HALM algorithm was more efficient for solving low-rank matrix recovery problems than the other algorithms. In addition, it is worth mentioning that the new algorithm can further develop to solve the tensor completion problems.

Use of AI tools declaration

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

Acknowledgments

The authors are very much indebted to the anonymous referees for their helpful comments and suggestions, which greatly improved the original manuscript of this paper. The authors are so thankful for the support from the NSF of China (12371381), the special fund for science and technology innovation teams of Shanxi province (202204051002018), and the Shanxi higher education science and technology innovation project (2023L243).

Conflict of interest

The authors declare that they have no conflict of interests.

References

1. A. Mongia, A. Majumdar, Matrix completion on multiple graphs: Application in collaborative filter- ing, *Signal Process.*, **165** (2019), 144–148. https://doi.org/10.1016/j.sigpro.2019.07.002
2. S. Daei, F. Haddadi, A. Amini, Exploitation prior information in block sprase signal, *IEEE Trans. Signal Process.*, **99** (2018). https://doi.org/10.1109/TSP.2019.2931209
3. Z. Zhang, K. Zhao, H. Zha, Inducible regularization for low-rank matrix factorizations for collaborative filtering, *Neurocomputing*, **97** (2012), 52–62. https://doi.org/10.1016/j.neucom.2012.05.010
4. S. Li, Q. Li, Z. Zhu, G. Tang, M. B. Wakin, The global geometry of centralized and distributed low-rank matrix recovery without regularization, *IEEE Signal Process. Lett.*, **27** (2020), 1400–1404. https://doi.org/10.1109/LSP.2020.3008876
5. B. J. Han, J. Y. Simg, Glass reflection removal using co-saliency based image alignment and low- rank matrix completion in gradient domain, *IEEE Trans. Image Process.*, **27** (2018), 1–1. https://doi.org/10.1109/TIP.2018.2849880.
6. L. I. Jie, F. Chen, Y. Wang, J. I. Fei, Y. U. Hua, Spatial spectrum estimation of incoherently dis tributed sources based on low-rank matrix recovery, *IEEE Trans. Veh. Technol.*, **99** (2020). https://doi.org/10.1109/TVT.2020.2986783
7. Z. H. Zhu, Q. W. Tang, G. G. Wakin, B. Michael, Global optimality in low-rank matrix optimization, *IEEE Trans. Signal Process.*, **66** (2018), 3614–3628. https://doi.org/10.1109/TSP.2018.2835403
8. M. Kech, M. Wolf, Constrained quantum tomography of semi-algebraic sets with applications to low-rank matrix recovery, *Mathematics*, **6** (2017), 171–195. https://doi.org/10.1093/imaiai/iaw019
9. E. J. Candès, B. Recht, Exact low-rank matrix completion via convex optimization, in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, (2009), 806–812. https://doi.org/10.1109/ALLERTON.2008.4797640

10. E. J. Candès, T. Tao, The power of convex relaxation: Near-optimal matrix completion, *IEEE Trans. Inf. Theory*, **56** (2010), 2053–2080. https://doi.org/10.1109/TIT.2010.2044061
11. B. Recht, A simpler approach to matrix completion, *J. Mach. Learn. Res.*, **12** (2009), 3413–3430. https://doi.org/10.1177/0959651810397461
12. B. Recht, M. Fazel, P. A. Parrilo, Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization, *SIAM Rev.*, **52** (2010), 471–501. https://doi.org/10.1137/070697835
13. D. Gross, Recovering low-rank matrices from few coefficients in any basis, *IEEE Trans Inf. Theory*, **57** (2011), 1548–1566. https://doi.org/10.1109/TIT.2011.2104999
14. J. Wright, A. Ganesh, S. Rao, Y. Ma, Robust principal component analysis: exact recovery of corrupted low-rank matrices, in *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop*, 2009. https://doi.org/10.1109/NNSP.2000.889420
15. R. Meka, P. Jain, I. Dhillon, Guaranteed rank minimization via singular value projection, preprint, arXiv:0909.5457. https://doi.org/10.48550/arXiv.0909.5457
16. A. Ganesh, Z. Lin, J. Wright, L. Wu, M. Chen, Y. Ma, Fast algorithms for recovering a corrupted low-rank matrix, in *2009 3rd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2009. https://doi.org/10.1109/CAMSAP.2009.5413299.
17. J. F. Cai, E. J. Candès, Z. Shen, A singular value thresholding algorithm for matrix completion, *SIAM J. Optim.*, **20** (2010), 1956–1982. https://doi.org/10.1137/080738970
18. Z. Lin, A. Ganesh, J. Wright, L. Wu, M. Chen, Y. Ma, Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix, *Coordinated Science Laboratory Report no. UILU-ENG-09-2214, DC-246*, 2009. https://doi.org/10.1017/S0025315400020749
19. Z. Lin, M. Chen, Y. Ma, The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices, preprint, arXiv:1009.5055. https://doi.org/10.48550/arXiv.1009.5055
20. J. Guo, R. P. Wen, Hybrid augmented lagrange multiplier algorithm for low-rank matrix completion, *Numer. Math.: Theory, Methods Appl.*, **44** (2022), 187–201.
21. G. H. Golub, C. F. Van Loan, *Matrix Computations*, 4^{th} edition, Johns Hopkins Studies in Mathematical Sciences, 2013.
22. D. Goldfarb, S. Ma, Convergence of fixed-point continuation algorithms for matrix rank minimization, *Found. Comput. Math.*, **11** (2011), 183–210. https://doi.org/10.1007/s10208-011-9084-6