


# Accelerated and Inexact Soft-Impute for Large-Scale Matrix and Tensor Completion

Quanming Yao , *Member, IEEE* and James T. Kwok, *Fellow, IEEE*

**Abstract**—Matrix and tensor completion aim to recover a low-rank matrix / tensor from limited observations and have been commonly used in applications such as recommender systems and multi-relational data mining. A state-of-the-art matrix completion algorithm is Soft-Impute, which exploits the special “sparse plus low-rank” structure of the matrix iterates to allow efficient SVD in each iteration. Though Soft-Impute is a proximal algorithm, it is generally believed that acceleration destroys the special structure and is thus not useful. In this paper, we show that Soft-Impute can indeed be accelerated without comprising this structure. To further reduce the iteration time complexity, we propose an approximate singular value thresholding scheme based on the power method. Theoretical analysis shows that the proposed algorithm still enjoys the fast  $O(1/T^2)$  convergence rate of accelerated proximal algorithms. We also extend the proposed algorithm to tensor completion with the scaled latent nuclear norm regularizer. We show that a similar “sparse plus low-rank” structure also exists, leading to low iteration complexity and fast  $O(1/T^2)$  convergence rate. Besides, the proposed algorithm can be further extended to nonconvex low-rank regularizers, which have better empirical performance than the convex nuclear norm regularizer. Extensive experiments demonstrate that the proposed algorithm is much faster than Soft-Impute and other state-of-the-art matrix and tensor completion algorithms.

**Index Terms**—Matrix completion, tensor completion, collaborative filtering, link prediction, proximal algorithms

## 1 INTRODUCTION

MATRICES are common place in data mining applications. For example, in recommender systems, the ratings data can be represented as a sparsely observed user-item matrix [1]. In social networks, user interactions can be modeled by an adjacency matrix [2], [3]. Matrices also appear in applications such as image processing [4], [5], [6], question answering [7] and large scale classification [8].

Due to limited feedback from users, these matrices are usually not fully observed. For example, users may only give opinions on very few items in a recommender system. As the rows/columns are usually related to each other, the low-rank matrix assumption is particularly useful to capture such relatedness, and low-rank matrix completion has become a powerful tool to predict missing values in these matrices. Sound recovery guarantee [9] and good empirical performance [1] have been obtained.

However, directly minimizing the matrix rank is NP-hard [9]. To alleviate this problem, the nuclear norm (which is the sum of singular values) is often used instead. It is known that the nuclear norm is the tightest convex lower bound of the rank [9]. Specifically, consider an  $m \times n$  matrix  $O$  (without loss of generality, we assume that  $m \geq n$ ), with positions

of the observed entries indicated by  $\Omega \in \{0, 1\}^{m \times n}$ , where  $\Omega_{ij} = 1$  if  $O_{ij}$  is observed, and 0 otherwise. The matrix completion tries to find a low-rank matrix  $X$  by solving following optimization problem:

$$\min_X \frac{1}{2} \|P_\Omega(X - O)\|_F^2 + \lambda \|X\|_*, \quad (1)$$

where  $[P_\Omega(A)]_{ij} = A_{ij}$  if  $\Omega_{ij} = 1$ , and 0 otherwise; and  $\|\cdot\|_*$  is the nuclear norm. Though the nuclear norm is only a surrogate of the matrix rank, there are theoretical guarantees that the underlying matrix can be exactly recovered [9].

Computationally, though the nuclear norm is nonsmooth, problem (1) can be solved by various optimization tools. An early attempt is based on reformulating (1) as a semidefinite program (SDP) [9]. However, SDP solvers have large time and space complexities, and are only suitable for small data sets. For large-scale matrix completion, singular value thresholding (SVT) algorithm [10] pioneered the use of first-order methods. However, a singular value decomposition (SVD) is required in each SVT iteration. This takes  $O(mn^2)$  time and can be computationally expensive. In [11], this is reduced to a partial SVD by computing only the leading singular values/vectors using PROPACK (a variant of the Lanczos algorithm) [12]. Another major breakthrough is made by the Soft-Impute algorithm [13], which utilizes a special “sparse plus low-rank” structure associated with the SVT to efficiently compute the SVD. Empirically, this allows Soft-Impute to perform matrix completion on the entire *Netflix* data set. The SVT algorithm can also be viewed as a proximal algorithm [14]. Hence, it converges with a  $O(1/T)$  rate, where  $T$  is the number of iterations [15]. Later, this is further “accelerated”, and the convergence rate is improved to  $O(1/T^2)$  [11], [16]. However, Tibshirani [14] suggested that this is not useful, as the special “sparse plus low-rank” structure crucial to the efficiency of

- Q. Yao is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, and 4Paradigm Inc, Beijing, China. E-mail: qyaoaa@cse.ust.hk.
- J.T. Kwok is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. E-mail: jamesk@cse.ust.hk.

Manuscript received 16 Mar. 2017; revised 20 Aug. 2018; accepted 24 Aug. 2018. Date of publication 28 Aug. 2018; date of current version 2 Aug. 2019. (Corresponding author: Quanming Yao.)

Recommended for acceptance by S. Yan.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.

Digital Object Identifier no. 10.1109/TKDE.2018.2867533

Soft-Impute no longer exist. In other words, the gain in convergence rate is more than compensated by the increase in iteration time complexity.

In this paper, we show that accelerating Soft-Impute is indeed possible while still preserving the “sparse plus low-rank” structure. To further reduce the iteration time complexity, instead of computing SVT exactly using PROPACK [11], [13], we propose an approximate SVT scheme based on the power method [17]. Though the SVT obtained in each iteration is only approximate, we show that convergence can still be as fast as performing exact SVT. Hence, the resultant algorithm has low iteration complexity and fast  $O(1/T^2)$  convergence rate. To further boost performance, we extend the post-processing procedure in [13] to any smooth convex loss function. The proposed algorithm is also extended for nonconvex low-rank regularizers, such as the truncated nuclear norm [18] and log-sum-penalty [9]. which can give better Empirically, these nonconvex low-rank regularizers have better performance than the convex nuclear norm regularizer.

Besides matrices, tensors have also been commonly used to describe the linear and multilinear relationships in the data [4], [19], [20], [21]. Analogous to matrix completion, tensor completion can also be solved by convex optimization algorithms. However, multiple expensive SVDs on large dense matrices are required [4], [20]. To alleviate this problem, we demonstrate that a similar “sparse plus low-rank” structure also exists when the scaled latent nuclear norm [20], [22] is used as the regularizer. We extend the proposed matrix-based algorithm to this tensor scenario. The resulting algorithm has low iteration cost and fast  $O(1/T^2)$  convergence rate. Experiments on matrix/tensor completion problems with both synthetic and real-world data sets show that the proposed algorithm outperforms state-of-the-art algorithms.

Preliminary results of this paper have been reported in a shorter conference version [23]. While only the square loss is used in [23], here we consider more general smooth convex loss functions. Moreover, we extend the proposed algorithm to tensor completion and nonconvex low-rank regularization. Besides, post-processing is proposed to boost the recovery performance for matrix/tensor completion with nuclear norm regularization. All proofs can be found in Appendix A, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TKDE.2018.2867533>.

**Notation.** In the sequel, the transpose of vector/matrix is denoted by the superscript  $\cdot^\top$ , and tensors are denoted by bold-face Euler. For a vector  $x$ ,  $\|x\|_1 = \sum_i |x_i|$  is its  $\ell_1$ -norm, and  $\|x\| = \sqrt{\sum_i x_i^2}$  its  $\ell_2$ -norm. For a matrix  $X$ ,  $\sigma_1(X) \geq \sigma_2(X) \geq \dots \geq \sigma_m(X)$  are its singular values,  $\text{tr}(X) = \sum_i X_{ii}$  is its trace,  $\|X\|_1 = \sum_{i,j} |X_{ij}|$ ,  $\|X\|_\infty$  is its maximum singular value, and  $\|X\|_F = \sqrt{\text{tr}(X^\top X)}$  the Frobenius norm,  $\|X\|_* = \sum_i \sigma_i(X)$  the nuclear norm, and  $\text{span}(X)$  is the column span of  $X$ . Moreover,  $I$  denotes the identity matrix.

For tensors, we follow the notations in [19]. For a  $D$ -order tensor  $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_D}$ , its  $(i_1, i_2, \dots, i_D)$ th entry is  $x_{i_1 i_2 \dots i_D}$ . Let  $I_{D \setminus d} = \prod_{j=1, j \neq d}^D I_j$ , the mode- $d$  matricizations  $\mathcal{X}_{(d)}$  of  $\mathcal{X}$  is a  $I_d \times I_{D \setminus d}$  matrix with  $(\mathcal{X}_{(d)})_{idj} = x_{i i_2 \dots i_D}$ , and  $j = 1 + \sum_{l=1, l \neq d}^D (i_l - 1) \prod_{m=1, m \neq d}^{l-1} I_m$ . Given a matrix  $A$ , its mode- $d$  tensorization  $A^{(d)}$  is a tensor  $\mathcal{A}$  with elements  $x_{i_1 i_2 \dots i_D} = a_{i_d j}$ , and  $j$  is as defined above. The inner product of two tensors  $\mathcal{X}$  and  $\mathcal{Y}$  is  $\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \dots \sum_{i_D=1}^{I_D} x_{i_1 i_2 \dots i_D} y_{i_1 i_2 \dots i_D}$ , and the Frobenius norm of  $\mathcal{X}$  is  $\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}$ .

For a convex but nonsmooth function  $f$ , the subgradient is  $g \in \partial f(x)$  where  $\partial f(x) = \{u : f(y) \geq f(x) + u^\top(y - x), \forall y\}$  is its subdifferential. When  $f$  is differentiable, we use  $\nabla f$  for its gradient.

## 2 RELATED WORK

### 2.1 Proximal Algorithms

Consider minimizing composite functions of the form

$$F(x) \equiv f(x) + g(x), \quad (2)$$

where  $f, g$  are convex, and  $f$  is smooth but  $g$  is possibly non-smooth. The proximal algorithm [24] generates a sequence of estimates  $\{x_t\}$  as

$$x_{t+1} = \text{prox}_{\mu g}(z_t) \equiv \arg \min_x \frac{1}{2} \|x - z_t\|_2^2 + \mu g(x),$$

where

$$z_t = x_t - \mu \nabla f(x_t), \quad (3)$$

and  $\text{prox}_{\mu g}(\cdot)$  is the proximal operator. When  $f$  is  $\rho$ -Lipschitz smooth (i.e.,  $\|\nabla f(x_1) - \nabla f(x_2)\| \leq \rho \|x_1 - x_2\|$ ) and a fixed stepsize

$$\mu \leq 1/\rho, \quad (4)$$

is used, the proximal algorithm converges to the optimal solution with a rate of  $O(1/T)$ , where  $T$  is the number of iterations [24]. By replacing the update in (3) with

$$y_t = (1 + \theta_t)x_t - \theta_t x_{t-1}, \quad (5)$$

$$z_t = y_t - \mu \nabla f(y_t), \quad (6)$$

where  $\theta_{t+1} = \frac{t-1}{t+2}$ , it can be accelerated to a convergence rate of  $O(1/T^2)$  [15].

Often,  $g$  is “simple” in the sense that  $\text{prox}_{\mu g}(\cdot)$  can be easily obtained. However, in more complicated problems such as overlapping group lasso [25],  $\text{prox}_{\mu g}(\cdot)$  may be expensive to compute. To alleviate this problem, inexact proximal algorithm is proposed which allows two types of errors in standard/accelerated proximal algorithms [26]: (i) an error  $e_t$  in computing  $\nabla f(\cdot)$ , and (ii) an error  $\varepsilon_t$  in the proximal step, i.e.,

$$h_{\mu g}(x_{t+1}; z_t) \leq \varepsilon_t + h_{\mu g}(\text{prox}_{\mu g}(z_t); z_t), \quad (7)$$

where

$$h_{\mu g}(x; z_t) \equiv \frac{1}{2} \|x - z_t\|_2^2 + \mu g(x), \quad (8)$$

is the proximal step’s objective. Let the dual problem of  $\min_x h_{\mu g}(x; z_t)$  be  $\max_w \mathcal{D}_{\mu g}(w)$  where  $w$  is the dual variable. The the duality gap is defined as  $\vartheta_t \equiv h_{\mu g}(x_{t+1}; z_t) - \mathcal{D}(w_{t+1})$  where  $w_{t+1}$  is the corresponding dual variable of  $x_{t+1}$ . Then  $\varepsilon_t$  is upper-bounded by the duality gap  $\vartheta_t$ . Thus, (7) can be ensured by monitoring  $\vartheta_t$ . The following Proposition shows that by decreasing  $e_t$  and  $\varepsilon_t$  sufficiently fast, the convergence rate remains at  $O(1/T^2)$ .

**Proposition 2.1 ([26]).** If  $\|e_t\|$  and  $\sqrt{\varepsilon_t}$  decrease as  $O(1/t^{2+\delta})$  for some  $\delta > 0$ , the inexact accelerated proximal gradient algorithm converges with a rate of  $O(1/T^2)$ .

In the sequel, as our focus is on matrix completion, the variable  $x$  in (2) will be a matrix  $X$ .

## 2.2 Soft-Impute

Soft-Impute [13] is a state-of-the-art algorithm for matrix completion. At iteration  $t$ , let the current iterate be  $X_t$ . The missing values in  $O$  are filled in as

$$Z_t = P_\Omega(O) + P_{\Omega^\perp}(X_t) = P_\Omega(O - X_t) + X_t, \quad (9)$$

where  $\Omega_{ij}^\perp = 1 - \Omega_{ij}$  is the complement of  $\Omega$ . The next estimate  $X_{t+1}$  is then generated by the singular value thresholding operator [10]

$$X_{t+1} = \text{SVT}_\lambda(Z_t) \equiv \arg \min_X \frac{1}{2} \|X - Z_t\|_F^2 + \lambda \|X\|_*, \quad (10)$$

which can be computed as follows.

**Lemma 2.2 ([10]).** *Let the SVD of a matrix  $Z_t$  be  $U\Sigma V^\top$ . Then,  $\text{SVT}_\lambda(Z_t) \equiv U(\Sigma - \lambda I)_+ V^\top$  where  $[(A)_+]_{ij} = \max(A_{ij}, 0)$ .*

Let  $\bar{k}_t$  be the number of singular values in  $Z_t$  that are larger than  $\lambda$ . From Lemma 2.2, a rank- $k_t$  SVD, where  $k_t \geq \bar{k}_t$ , is sufficient for computing  $X_{t+1}$  in (10). In [13], this rank- $k_t$  SVD is obtained by the PROPACK algorithm [12]. The most expensive steps in computing the SVD are matrix-vector multiplications of the form  $Zu$  and  $v^\top Z$ , where  $u \in \mathbb{R}^n$  and  $v \in \mathbb{R}^m$ . In general, the above multiplications take  $O(mn)$  time and rank- $k_t$  SVD on  $Z_t$  takes  $O(mnk_t)$  time.

However, to make Soft-Impute efficient, an important observation in [13] is that  $Z_t$  in (9) has a special “sparse plus low-rank” structure, namely that  $P_\Omega(O - X_t)$  is sparse and  $X_t$  is low-rank. Multiplications of the form  $Z_t u$  and  $v^\top Z_t$  can then be efficiently performed as follows. Let the rank of  $X_t$  be  $r_t$ , and its SVD be  $U_t \Sigma_t V_t^\top$ .  $Z_t v$  can be computed as

$$Z_t v = P_\Omega(O - X_t)v + U_t \Sigma_t (V_t^\top v). \quad (11)$$

Constructing  $P_\Omega(O - X_t)$  takes  $O(r_t \|\Omega\|_1)$  time, while computing the products  $P_\Omega(O - X_t)u$  and  $U_t \Sigma_t (V_t^\top u)$  take  $O(\|\Omega\|_1)$  and  $O(mr_t)$  time, respectively. Similarly,  $u^\top Z_t$  can be computed as  $u^\top P_\Omega(O - X_t) + (u^\top U_t) \Sigma_t V_t^\top$ . Thus, to obtain the rank- $k$  SVD of  $Z_t$ , Soft-Impute needs only

$$O(k_t \|\Omega\|_1 + r_t k_t m), \quad (12)$$

time, and one iteration costs

$$O((r_t + k_t) \|\Omega\|_1 + r_t k_t m), \quad (13)$$

time. Since the solution is low-rank,  $k_t, r_t \ll m$ , and (13) is much faster than the  $O(mnk_t)$  time for direct rank- $k_t$  SVD.

## 3 ACCELERATED INEXACT SOFT-IMPUTE

In this section, we describe the proposed matrix completion algorithm. Tibshirani [14] suggested that acceleration is not useful for Soft-Impute, as it destroys the essential “sparse plus low-rank” structure. However, we will show that it can indeed be preserved with acceleration. We also show that further speedup can be achieved by using approximate SVT.

### 3.1 Soft-Impute as a Proximal Algorithm

In (1), let

$$f(X) = \frac{1}{2} \|P_\Omega(X - O)\|_F^2 = \sum_{(i,j) \in \Omega} \ell(X_{ij}, O_{ij}), \quad (14)$$

where  $\ell$  is the loss function, and  $g(X) = \lambda \|X\|_*$ . The proximal step in the (unaccelerated) proximal algorithm is

$$X_{t+1} = \text{prox}_{\mu g}(Z_t) \equiv \arg \min_x \frac{1}{2} \|X - Z_t\|_F^2 + \mu \lambda \|X\|_*,$$

where  $Z_t = X_t - \mu P_\Omega(X_t - O)$ . Note that the square loss  $\ell(X_{ij}, O_{ij}) \equiv \frac{1}{2} (X_{ij} - O_{ij})^2$  in (1) is 1-Lipschitz smooth. The following shows that  $f$  in (14) is also 1-Lipschitz smooth.

**Proposition 3.1.** *If  $\ell$  is  $\rho$ -Lipschitz smooth,  $f$  in (14) is also  $\rho$ -Lipschitz smooth.*

From (4), one can thus simply set  $\mu = 1$  for (1). We then have  $X_{t+1} = \text{prox}_g(Z_t) = \text{SVT}_\lambda(Z_t)$  which is the same as (10). Hence, interestingly, Soft-Impute is a proximal algorithm [14], and thus converges at a rate of  $O(1/T)$  [13].

### 3.2 Accelerating Soft-Impute

Since Soft-Impute is a proximal algorithm, it is natural to use acceleration (Section 2.1). Recall that the efficiency of Soft-Impute hinges on the “sparse plus low-rank” structure of  $Z_t$ , which allows matrix-vector multiplications of the form  $Z_t u$  and  $v^\top Z_t$  to be computed inexpensively. To accelerate Soft-Impute, from (5) and (6), we have to compute

$$\text{prox}_g(\check{Z}_t) = \text{SVT}_\lambda(\check{Z}_t) = \arg \min_X \frac{1}{2} \|X - \check{Z}_t\|_F^2 + \lambda \|X\|_*, \quad (15)$$

where  $Y_t = (1 + \theta_t)X_t - \theta_t X_{t-1}$ , and

$$\check{Z}_t = P_\Omega(O - Y_t) + (1 + \theta_t)X_t - \theta_t X_{t-1}. \quad (16)$$

In the following, we show that  $\check{Z}_t$  also has a similar “sparse plus low-rank” structure.

Assume that  $X_t$  and  $X_{t-1}$  have ranks  $r_t$  and  $r_{t-1}$ , and their SVDs are  $U_t \Sigma_t V_t^\top$  and  $U_{t-1} \Sigma_{t-1} V_{t-1}^\top$ , respectively. Note that  $P_\Omega(O - Y_t)$  is sparse, and  $(1 + \theta_t)X_t - \theta_t X_{t-1}$  has rank at most  $r_t + r_{t-1}$ . Similar to (11), for any  $v \in \mathbb{R}^n$ , we have

$$\check{Z}_t v = P_\Omega(O - Y_t)v + (1 + \theta_t)U_t \Sigma_t (V_t^\top v) - \theta_t U_{t-1} \Sigma_{t-1} (V_{t-1}^\top v).$$

The first term takes  $O(\|\Omega\|_1)$  time while the last two terms take  $O((r_{t-1} + r_t)m)$  time, thus a total of  $O(\|\Omega\|_1 + (r_{t-1} + r_t)m)$  time. Similarly, for any  $u \in \mathbb{R}^m$ , we have

$$u^\top \check{Z}_t = u^\top P_\Omega(O - Y_t) + (1 + \theta_t)(u^\top U_t) \Sigma_t V_t^\top - \theta_t (u^\top U_{t-1}) \Sigma_{t-1} V_{t-1}^\top.$$

This takes  $O(\|\Omega\|_1 + (r_{t-1} + r_t)m)$  time. The rank- $k_t$  SVD of  $\check{Z}_t$  can be obtained using PROPACK in

$$O(k_t \|\Omega\|_1 + (r_{t-1} + r_t)k_t m), \quad (17)$$

time. As the target matrix is low-rank,  $r_{t-1}$  and  $r_t$  are much smaller than  $n$ . Hence, (17) is much faster than the  $O(mnk_t)$  time required for a direct rank- $k_t$  SVD.

The accelerated algorithm has a slightly higher iteration complexity than the unaccelerated one in (12). However, this is more than compensated by improvement in the convergence rate (from  $O(1/T)$  to  $O(1/T^2)$ ), as will be empirically demonstrated in Section 5.1.

### 3.3 Approximating the SVT

Though acceleration preserves the “sparse plus low-rank” structure, the proposed algorithm (and Soft-Impute) can still be computationally expensive as the SVT in each

iteration uses exact SVD. In this section, we show that further speedup is possible by using inexact SVD.

As SVT in (10) can be seen as a proximal step, one might want to perform inexact SVT by monitoring the duality gap as in Section 2.1. It can be shown that the dual of (15) is

$$\max_{\|W\|_\infty \leq 1} \text{tr}(W^\top \check{Z}_t) - \frac{\lambda}{2} \|W\|_F^2, \quad (18)$$

where  $W \in \mathbb{R}^{m \times n}$  is the dual variable.

**Proposition 3.2 ([24]).** *Let the SVD of matrix  $\check{Z}_t$  be  $U\Sigma V^\top$ . The optimal solution of (18) is  $W_* = U \min(\Sigma, \lambda I) V^\top$ , where  $[\min(A, B)]_{ij} = \min(A_{ij}, B_{ij})$ .*

Proposition 3.2 shows that a full SVD is required. This takes  $O(m^2n)$  time and is even more expensive than directly using SVT ( $O(mnk_t)$  time). Instead, the proposed approximation is motivated by the following Proposition.

**Proposition 3.3.** *Let  $\check{k}_t$  be the number of singular values in  $\check{Z}_t \in \mathbb{R}^{m \times n}$  larger than  $\lambda$ , and  $Q \in \mathbb{R}^{m \times k_t}$ , where  $k_t \geq \check{k}_t$ , be orthogonal and contains the subspace spanned by the top  $\check{k}_t$  left singular vectors of  $\check{Z}_t$ . Then,  $\text{SVT}_\lambda(\check{Z}_t) = Q\text{SVT}_\lambda(Q^\top \check{Z}_t)$ .*

Since a low-rank solution is desired,  $k_t$  can be much smaller than  $m$  [13]. Thus, once we identify the span of  $\check{Z}_t$ 's top left singular vectors, we only need to perform SVT on the much smaller  $Q^\top \check{Z}_t \in \mathbb{R}^{k_t \times n}$  (instead of  $\check{Z} \in \mathbb{R}^{m \times n}$ ). The question is how to find  $Q$ . We adopt the power method (Algorithm 1) [17], which is more efficient than PROPACK [27]. Matrix  $R_t$  in Algorithm 1 is for warm-start.

---

**Algorithm 1.** PowerMethod( $\check{Z}_t, R_t, J$ ) [17]

---

**Require:**  $\check{Z}_t \in \mathbb{R}^{m \times n}$ ,  $R_t \in \mathbb{R}^{n \times k_t}$ , and the number of iterations  $J$ ;

- 1: initialize  $Q_0 = \text{QR}(\check{Z}_t R_t)$ ; //  $\text{QR}(\cdot)$  is QR factorization
- 2: **for**  $j = 1, 2, \dots, J$  **do**
- 3:  $Q_j = \text{QR}(\check{Z}_t (\check{Z}_t^\top Q_{j-1}))$ ;
- 4: **end for**
- 5: **return**  $Q_j$ .

---

Algorithm 2 shows the approximate SVT procedure. Step 1 approximates the top  $k_t$  left singular vectors of  $\check{Z}_t$  with  $Q$ . In steps 2 to 5, a much smaller and less expensive (exact) SVT is performed on  $Q^\top \check{Z}_t$ . Finally,  $\text{SVT}_\lambda(\check{Z}_t)$  is recovered as  $\tilde{X} = (QU)\Sigma V^\top$  using Proposition 3.3.

### 3.4 The Proposed Algorithm

We extend problem (1) by allowing the loss  $\ell$  to be  $\rho$ -Lipschitz smooth (e.g., logistic loss and squared hinge loss)

$$\min_X F(X) \equiv \sum_{(i,j) \in \Omega} \ell(X_{ij}, O_{ij}) + \lambda \|X\|_*. \quad (19)$$

Using (6),  $\check{Z}_t = Y_t - \mu \nabla f(Y_t) = Y_t - \mu S_t$ , where  $S_t$  is a sparse matrix with

$$[S_t]_{ij} = \begin{cases} \frac{d\ell(Y_t, O_{ij})}{d(Y_t)_{ij}} & \text{if } (i, j) \in \Omega \\ 0 & \text{otherwise.} \end{cases} \quad (20)$$

Using Proposition 3.1 and (4), the stepsize  $\mu$  can be set as  $1/\rho$ . The whole procedure is shown in Algorithm 3. The core steps are 6-8, which performs approximate SVT. As in [28],  $R_t$  is warm-started as  $\text{QR}([V_t, V_{t-1}])$  at step 7. Moreover, as in [29], we restart the algorithm if  $F(X)$  starts to increase

(step 10). For further speedup,  $\lambda$  is dynamically reduced (step 3) by a continuation strategy [11], [13].

---

**Algorithm 2.** Approximating the SVT of  $\check{Z}_t$ :  
Approx-SVT( $\check{Z}_t, R_t, \lambda, J$ )

---

**Require:**  $\check{Z}_t \in \mathbb{R}^{m \times n}$ ,  $R_t \in \mathbb{R}^{n \times k_t}$  and  $\lambda \geq 0$ ;

- 1:  $Q = \text{PowerMethod}(\check{Z}_t, R_t, J)$ ;
- 2:  $[U, \Sigma, V] = \text{SVD}(Q^\top \check{Z}_t)$ ;
- 3:  $U = \{u_i \mid \sigma_i > \lambda\}$ ;
- 4:  $V = \{v_i \mid \sigma_i > \lambda\}$ ;
- 5:  $\Sigma = (\Sigma - \lambda I)_+$ ;
- 6: **return**  $QU, \Sigma$  and  $V$ . //  $\tilde{X} = (QU)\Sigma V$

---



---

**Algorithm 3.** Accelerated Inexact Soft-Impute (AIS-Impute)

---

**Require:** partially observed matrix  $O$ , parameter  $\lambda$ .

- 1: initialize  $c = 1$ ,  $X_0 = X_1 = 0$ , stepsize  $\mu = 1/\rho$ ,  $\hat{\lambda} > \lambda$  and  $v \in (0, 1)$ ;
- 2: **for**  $t = 1, 2, \dots, T$  **do**
- 3:  $\lambda_t = (\hat{\lambda} - \lambda)v^{t-1} + \lambda$ ;
- 4:  $Y_t = X_t + \theta_t(X_t - X_{t-1})$ , where  $\theta_t = \frac{c-1}{c+2}$ ;
- 5:  $\check{Z}_t = Y_t - \mu S_t$ , with  $S_t$  in (20);
- 6:  $V_{t-1} = V_{t-1} - V_t(V_t^\top V_{t-1})$ , remove zero columns;
- 7:  $R_t = \text{QR}([V_t, V_{t-1}])$ ;
- 8:  $[U_{t+1}, \Sigma_{t+1}, V_{t+1}] = \text{approx-SVT}(\check{Z}_t, R_t, \mu \lambda_t, J)$ ;  
//  $X_{t+1} = U_{t+1} \Sigma_{t+1} V_{t+1}^\top$
- 9: **if**  $F(X_{t+1}) > F(X_t)$  **then**  $c = 1$ ;
- 10: **else**  $c = c + 1$ ; **end if**
- 11: **end for**
- 12: **return**  $U_{T+1}, \Sigma_{T+1}$  and  $V_{T+1}$ .

---

### 3.5 Convergence and Time Complexity

In the following, we will show that the proposed algorithm has a convergence rate of  $O(1/T^2)$ . Let  $X_{t+1} = U_{t+1} \Sigma_{t+1} V_{t+1}^\top$  be the output of approx-SVT at step 8. Since it only approximates  $\text{SVT}_{\mu\lambda}(\check{Z}_t)$ , there is a difference ( $\varepsilon_t$  in (7)) between the proximal objectives  $h_{\mu\lambda, \|\cdot\|_*}(X_{t+1}; \check{Z}_t)$  and  $h_{\mu\lambda, \|\cdot\|_*}(\text{SVT}_{\mu\lambda}(\check{Z}_t); \check{Z}_t)$  after performing step 8, where  $h_{\mu\lambda, \|\cdot\|_*}(\cdot; \cdot)$  is as defined in (8). The following shows that  $\varepsilon_t$  decreases at a linear rate.

**Proposition 3.4.** *Assume that (i)  $k_t \geq \check{k}_t$  for all  $t$  and  $J = t^2$ ; (ii)  $\{F(X_t)\}$  is upper-bounded. Then  $\varepsilon_t$  decreases to zero linearly.*

Using Propositions 2.1 and 3.4, convergence of the proposed algorithm is provided by the following Theorem.

**Theorem 3.5.** *The sequence  $\{X_t\}$  generated from Algorithm 3 converges to the optimal solution with a  $O(1/T^2)$  rate.*

The basic operations in the power method are multiplications of the form  $\check{Z}_t u$  and  $v^\top \check{Z}_t$ . The tricks in Section 3.2 can again be used for acceleration, and computing the approximate SVT using Algorithm 2 takes only

$$O(k_t \|\Omega\|_1 + (r_{t-1} + r_t) k_t m), \quad (21)$$

time. This is slightly more expensive than (12), the time for performing exact SVD in Soft-Impute. However, Soft-Impute is not accelerated and has slower convergence than Algorithm 3 (Theorem 3.5). The complexity in (21) is also the same as (17). However, as will be demonstrated in Section 5.1, approximate SVT is empirically much faster. The cost of

1. In practice, we simply set  $J = 3$  as in [28].

TABLE 1  
Iteration Time Complexity of Algorithm 3

steps	complexity
5 (construct $S_t$ )	$O(r_t \ \Omega\ _1)$
6,7 (warm-start)	$O(nk_t^2)$
8 (approximate SVT)	$O(k_t \ \Omega\ _1 + (r_{t-1} + r_t)k_t m)$
total	$O((r_t + k_t) \ \Omega\ _1 + (r_{t-1} + r_t + k_t)k_t m)$

one AIS-Impute iteration is summarized in Table 1. This is only slightly more expensive than (13) for Soft-Impute.

Table 2 compares Algorithm 3 with some existing algorithms that will be empirically compared in Section 5.2. Overall, Algorithm 3 enjoys fast convergence and low iteration complexity.

### 3.6 Post-Processing

The nuclear norm penalizes all singular values equally. This may over-penalize the more important leading singular values. To alleviate this problem, we post-process the solution as in [13]. Note that only the square loss is considered in [13]. Here, any smooth convex loss can be used.

Let the rank- $k$  matrix obtained from Algorithm 3 be  $X = U\Sigma V^\top$ , where  $U = [u_i]$  and  $V = [v_i]$ . Let  $A(\theta) = U \text{Diag}(\theta) V^\top$ . We undo part of the shrinkage on the singular values by replacing  $X$  with  $A(\theta_*)$ , where

$$\theta_* = \arg \min_{\theta} \phi(\theta) \equiv \sum_{(i,j) \in \Omega} \ell(A(\theta)_{ij}, O_{ij}). \quad (22)$$

When  $\ell$  is the square loss, (22) has a closed-form solution [13]. However, for smooth convex  $\ell$  in general, this is not possible and we optimize (22) using L-BFGS. The most expensive step in each L-BFGS iteration is the computation of the gradient  $\nabla \phi(\theta) \in \mathbb{R}^k$ , where  $[\nabla \phi(\theta)]_i = u_i^\top B v_i$ ,  $B_{ij} = \frac{d\ell(A(\theta)_{ij}, O_{ij})}{dA(\theta)_{ij}}$  if  $(i, j) \in \Omega$ , and 0 otherwise. As  $S$  is sparse, computing  $\nabla \phi(\theta)$  only takes  $O(k \|\Omega\|_1)$  time where  $k \ll n$ . Thus, one iteration of L-BFGS takes  $O(k \|\Omega\|_1)$  time, which is not significant compared to the  $O((r_t + k_t) \|\Omega\|_1 + (r_{t-1} + r_t + k_t)k_t m)$  time in each AIS-Impute iteration (Table 1). Moreover, L-BFGS has superlinear convergence [38]. Empirically, it converges in fewer than 10 iterations. These make post-processing very efficient.

### 3.7 Nonconvex Regularization

While post-processing alleviates the problem of over-penalizing singular values, recently nonconvex regularizers have been proposed to address this problem in a more direct manner. In this section, we first show that the proposed algorithm can be extended for truncated nuclear norm regularization (TNNR) [18], which is a popular nonconvex variant of the nuclear norm. Then we show that it can be further extended for more general nonconvex regularizers.

*Truncated Nuclear Norm.* The optimization problem for TNNR [18] can be written as

$$\min_X \frac{1}{2} \|P_\Omega(X - O)\|_F^2 + \lambda \sum_{i=r}^m \sigma_i(X), \quad (23)$$

where  $r \in \{1, \dots, m\}$ . Using DC programming [39], this is rewritten as

$$\begin{aligned} \min_X \max_{A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{n \times r}} \frac{1}{2} \|P_\Omega(X - O)\|_F^2 + \lambda \|X\|_* - \lambda \text{tr}(A^\top X B) \\ \text{s.t. } A^\top A = I, B^\top B = I. \end{aligned} \quad (24)$$

$A, B$  and  $X$  are then obtained using alternative minimization as

$$(A_{\tau+1}, B_{\tau+1}) = \min_{A^\top A = I, B^\top B = I} \text{tr}(A^\top X_\tau B), \quad (25)$$

$$\begin{aligned} X_{\tau+1} = \min_X \frac{1}{2} \|P_\Omega(X - O)\|_F^2 + \lambda \|X\|_* \\ - \lambda \text{tr}(A_{\tau+1}^\top X B_{\tau+1}). \end{aligned} \quad (26)$$

Subproblem (25) has the closed-form solution  $A_{\tau+1} = U_\tau$  and  $B_{\tau+1} = V_\tau$  [18], where  $U_\tau \Sigma_\tau V_\tau^\top$  is the rank- $r$  SVD of  $X_\tau$ . Subproblem (26) involves convex optimization with the nuclear norm regularizer, and is solved by the accelerated proximal gradient (APG) algorithm [15] in [18].

The proposed AIS-Impute can be used to solve (26) more efficiently. Let  $X_{t-1}$  and  $X_t$  be two consecutive iterates from AIS-Impute. As in Section 3.2, in order to generate  $X_{t+1}$ , we compute

$$\begin{aligned} Y_t &= (1 + \theta_t)X_t - \theta_t X_{t-1}, \\ \check{Z}_t &= Y_t + \mu \lambda A_{\tau+1} B_{\tau+1}^\top - \mu P_\Omega(Y_t - O). \end{aligned} \quad (27)$$

Note that  $Y_t + \mu \lambda A_{\tau+1} B_{\tau+1}^\top$  is low-rank and  $\mu P_\Omega(Y_t - O)$  is sparse. Thus,  $\check{Z}_t$  again has the special ‘‘sparse plus low-rank’’ structure which is key to AIS-Impute. Each AIS-Impute

TABLE 2  
Comparison of AIS-Impute (Algorithm 3) with Other Algorithms

	algorithm	iteration complexity	rate
matrix factorization	LMaFit [32]	$O(\ \Omega\ _1 k_t + m k_t)$	—
	ASD [33]	$O(\ \Omega\ _1 k_t + m k_t)$	—
	RIMP [34]	$O(\ \Omega\ _1 + m k_t^2)$	$O(c_1^T)$
nuclear norm minimization	active subspace selection [28]	$O(\ \Omega\ _1 k_t^2 T_a)$	$O(c_2^{T-T_s})$
	boost [31]	$O(\ \Omega\ _1 t^2 T_a)$	$O(1/T)$
	Sketchy [35]	$O(\ \Omega\ _1 + m k_t^2)$	$O(1/T)$
	TR [30]	$O(\ \Omega\ _1 t^2 T_a)$	—
	ALT-Impute [36]	$O(\ \Omega\ _1 k_t + m k_t^2)$	$O(1/T)$
	SSGD [37]	$O(m k_t^2)$	$O(1/\sqrt{T})$
	AIS-Impute	$O((r_t + k_t) \ \Omega\ _1 + (r_{t-1} + r_t + k_t) m k_t)$	$O(1/T^2)$

The algorithms in [28], [30], [31] involve solving some optimization subproblems iteratively, and  $T_a$  is the number of iterations used. Moreover, integer  $T_s$  and  $c_1, c_2 \in (0, 1)$  are some constants.

iteration then takes  $O((r_t + k_t)\|\Omega\|_1 + (r_{t-1} + r_t + k_t)k_t m)$  time, which is much cheaper than the  $O(mnk)$  time for APG.

*Other Nonconvex Low-rank Regularizers.* Assume that the regularizer is of the form  $r(X) = \sum_{i=1}^m \bar{r}(\sigma_i(X))$ , where  $\bar{r}(\alpha)$  is a concave and nondecreasing function on  $\alpha \geq 0$ . This assumption is satisfied by the log-sum-penalty [40], minimax concave penalty [41], and capped- $\ell_1$  norm [42]. The corresponding optimization problem is  $\min_X \frac{1}{2}\|P_\Omega(X - O)\|_F^2 + \lambda r(X)$ . As in [18], using DC programming, we obtain

$$X_{\tau+1} = \min_X \frac{1}{2}\|P_\Omega(X - O)\|_F^2 + \lambda \sum_{i=1}^m (w_{\tau+1})_i \sigma_i(X), \quad (28)$$

$$[w_{\tau+1}]_i = \hat{\partial} \bar{r}(\sigma_i(X_\tau)), \quad i = 1, \dots, m, \quad (29)$$

where  $\hat{\partial} \bar{r}$  is the super-gradient [43] of  $\bar{r}$ . Subproblem (29) can be easily computed in  $O(m)$  time. As for (28), its regularizer is a weighted nuclear norm. As  $\bar{r}$  is non-decreasing and concave,  $(w_{\tau+1})_1 \leq (w_{\tau+1})_2 \leq \dots \leq (w_{\tau+1})_m$  [6]. The following Lemma shows that the proximal step in (28) has a closed-form solution.

**Lemma 3.6 ([5], [6]).** *Let the SVD of  $Z$  be  $U\Sigma V^T$  and  $0 \leq w_1 \leq w_2 \leq \dots \leq w_m$ . The solution of the proximal step  $\min_X \frac{1}{2}\|X - Z\|_F^2 + \lambda \sum_{i=1}^m w_i \sigma_i(X)$  is given by  $U[\Sigma - \lambda \text{Diag}(w_1, \dots, w_m)]_+ V^T$ .*

Similar to the truncated nuclear norm, we have  $\check{Z}_t = P_\Omega(O - Y_t) + Y_t$ , where  $P_\Omega(O - Y_t)$  is sparse and  $Y_t$  (defined in (27)) is low-rank. Hence, we again have the special ‘‘sparse plus low-rank’’ structure. AIS-Impute algorithm can still be used and one iteration takes  $O((r_t + k_t)\|\Omega\|_1 + (r_{t-1} + r_t + k_t)k_t m)$  time.

## 4 TENSOR COMPLETION

Complicated data objects can often be arranged as tensors. In this section, we extend the proposed Algorithm 3 in Section 3 from matrices to tensors.

### 4.1 Tensor Model

The nuclear norm can be defined on tensors in various ways. The following two are the most popular.

**Definition 4.1 ([20]).** *For a  $D$ -order tensor  $\mathcal{X}$ , the overlapped nuclear norm is  $\|\mathcal{X}\|_{\text{overlap}} = \sum_{d=1}^D \lambda_d \|\mathcal{X}_{(d)}\|_*$ , and the scaled latent nuclear norm is  $\|\mathcal{X}\|_{\text{scaled}} = \min_{\mathcal{X}^1, \dots, \mathcal{X}^D} \sum_{d=1}^D \mathcal{X}^d = \mathcal{X} \sum_{d=1}^D \lambda_d \|\mathcal{X}_{(d)}\|_*$ . Here,  $\lambda_d \geq 0$ 's are hyperparameters.*

The overlapped nuclear norm regularizer penalizes nuclear norms on all modes. When only several modes are low-rank, decomposition with the scaled latent nuclear norm has better generalization [20], [22]. In this paper, we focus on the scaled latent nuclear norm regularizer.

Given a partially observed tensor  $\mathcal{O} \in \mathbb{R}^{I_1 \times \dots \times I_D}$ , with the observed entries indicated by  $\Omega \in \{0, 1\}^{I_1 \times \dots \times I_D}$ . The tensor completion problem can be formulated as

$$\begin{aligned} & \min_{\mathcal{X}^1, \dots, \mathcal{X}^D} F([\mathcal{X}^1, \dots, \mathcal{X}^D]) \\ & \equiv \sum_{(i_1, \dots, i_D) \in \Omega} \ell \left( \sum_{d=1}^D \mathcal{X}_{i_1 \dots i_D}^d, \mathcal{O}_{i_1 \dots i_D} \right) + \sum_{d=1}^D \lambda_d \|\mathcal{X}_{(d)}^d\|_* \end{aligned} \quad (30)$$

The recovered tensor is  $\mathcal{X} = \sum_{d=1}^D \mathcal{X}^d$ . In [4], [20], problem (30) is solved using ADMM [44]. However, the ADMM update involves SVD in each iteration, which takes  $O(\prod_{d=1}^D I_d \sum_{d=1}^D I_d)$  time and can be expensive.

### 4.2 Generalizing SVT

In (30), let

$$f([\mathcal{X}^1, \dots, \mathcal{X}^D]) = \sum_{(i_1, \dots, i_D) \in \Omega} \ell \left( \sum_{d=1}^D \mathcal{X}_{i_1 \dots i_D}^d, \mathcal{O}_{i_1 \dots i_D} \right), \quad (31)$$

$$g([\mathcal{X}^1, \dots, \mathcal{X}^D]) = \sum_{d=1}^D \lambda_d \|\mathcal{X}_{(d)}^d\|_*. \quad (32)$$

The iterates in Algorithm 3 are generated by SVT. As there are multiple nuclear norms in (32), the following extends SVT for this case.

As  $g$  in (32) is separable w.r.t.  $\mathcal{X}^i$ 's, one can compute the proximal step for each  $\mathcal{X}^i$  separately [24]. Updates (5), (6) in the APG become

$$\begin{aligned} \mathcal{Y}_t^d &= (1 + \theta_t) \mathcal{X}_t^d - \theta_t \mathcal{X}_{t-1}^d, \\ \check{\mathcal{Z}}_t^d &= \mathcal{Y}_t^d - \mu \mathcal{S}_t = (1 + \theta_t) \mathcal{X}_t^d - \theta_t \mathcal{X}_{t-1}^d - \mu \mathcal{S}_t, \end{aligned} \quad (33)$$

for  $d = 1, \dots, D$ , where  $\mathcal{S}_t$  is a sparse tensor with

$$(\mathcal{S}_t)_{i_1 \dots i_D} = \begin{cases} \frac{d\ell(\hat{\mathcal{Y}}_t)_{i_1 \dots i_D} \mathcal{O}_{i_1 \dots i_D}}{d(\hat{\mathcal{Y}}_t)_{i_1 \dots i_D}} & \text{if } (i_1, \dots, i_D) \in \Omega \\ 0 & \text{otherwise,} \end{cases} \quad (34)$$

and  $\hat{\mathcal{Y}}_t = \sum_{d=1}^D \mathcal{Y}_t^d$ . Lemma 2.2 is also extended to  $[\mathcal{X}_{t+1}^1, \dots, \mathcal{X}_{t+1}^D] = \text{prox}_{\mu g}([\check{\mathcal{Z}}_t^1, \dots, \check{\mathcal{Z}}_t^D])$  as follows.

**Proposition 4.1.**  $(\mathcal{X}_{t+1}^d)_{(d)} = \text{SVT}_{\mu \lambda_d \|\cdot\|_*}((\check{\mathcal{Z}}_t^d)_{(d)})$ .

The stepsize rule in (4) depends on the modulus of Lipschitz smoothness of  $f$ , which is given by the following.

**Proposition 4.2.** *If  $\ell$  is  $\rho$ -Lipschitz smooth,  $f$  in (31) is  $\sqrt{D}\rho$ -Lipschitz smooth.*

Proposition 3.3 can be used to reduce the size of  $(\check{\mathcal{Z}}_t^d)_{(d)}$  in Proposition 4.1, and Algorithm 1 can be used to approximate the underlying SVD. However, this is still not fast enough. Assume that  $k_t^d$  singular values in  $(\check{\mathcal{Z}}_t^d)_{(d)}$  are larger than  $\mu \lambda_d$ , and rank- $k_t^d$  SVD, where  $k_t^d \geq k_t^d$ , is performed. SVT on  $(\check{\mathcal{Z}}_t^d)_{(d)}$  takes  $O(k_t^d \prod_{d=1}^D I_d)$  time. As SVT has to be performed on each mode, one iteration of APG takes  $O(\prod_{d=1}^D I_d \sum_{d=1}^D k_t^d)$  time, which is expensive.

### 4.3 Fast Approximate SVT with Special Structure

In Section 3.2, the special ‘‘sparse plus low-rank’’ structure can greatly reduce the time complexity of matrix multiplications. As  $\mathcal{X}_{t-1}^d, \mathcal{X}_t^d$  are low-rank tensors and  $\mathcal{S}_t$  is sparse,  $\check{\mathcal{Z}}_t^d$  in (33) also has the ‘‘sparse plus low-rank’’ structure. However, to generate  $\mathcal{X}_{t+1}^d$  using Proposition 4.1, we need to perform matrix multiplications of the form  $(\check{\mathcal{Z}}_t^d)_{(d)} v$ , where  $v \in \mathbb{R}^{I_D \times d}$ , and  $u^T (\check{\mathcal{Z}}_t^d)_{(d)}$ , where  $u \in \mathbb{R}^{I_d}$ . Unfolding  $\check{\mathcal{Z}}_t^d$  takes  $O(\prod_{d=1}^D I_d)$  time and can be expensive. In the following, we show how this can be avoided.

To generate  $(\mathcal{X}_{t+1}^d)_{(d)}$ , it can be seen from Proposition 4.1 and (33) that  $\mathcal{X}_t^d$  and  $\mathcal{X}_{t-1}^d$  only need to be unfolded along their  $d$ th modes. Hence, instead of storing them as tensors,

we store  $(\mathcal{X}_t^d)_{(d)}$  as its rank- $r_t^d$  SVD  $U_t^d \Sigma_t^d V_t^{d\top}$ , and  $(\mathcal{X}_{t-1}^d)_{(d)}$  as its rank- $r_{t-1}^d$  SVD  $U_{t-1}^d \Sigma_{t-1}^d V_{t-1}^{d\top}$ . For any  $v \in \mathbb{R}^{I_{D \setminus d}}$ ,

$$\begin{aligned} (\check{\mathcal{Z}}_t^d)_{(d)} v &= (1 + \theta_t) U_t^d \Sigma_t^d (V_t^{d\top} v) \\ &\quad - \theta_t U_{t-1}^d \Sigma_{t-1}^d (V_{t-1}^{d\top} v) - \mu (\mathcal{S}_t)_{(d)} v. \end{aligned}$$

The first two terms can be computed in  $O((I_d + I_{D \setminus d})(r_t^d + r_{t-1}^d))$  time. As  $\mathcal{S}_t$  is sparse, computing the last term takes  $O(\|\Omega\|_1)$  time. Thus,  $(\check{\mathcal{Z}}_t^d)_{(d)} v$  can be obtained in  $O(\|\Omega\|_1 + (I_d + I_{D \setminus d})(r_t^d + r_{t-1}^d))$  time. Similarly, for any  $u \in \mathbb{R}^{I_d}$ ,  $u^\top (\check{\mathcal{Z}}_t^d)_{(d)}$  can be computed in  $O(\|\Omega\|_1 + (I_d + I_{D \setminus d})(r_t^d + r_{t-1}^d))$  time. Thus, performing approximate SVT on  $(\check{\mathcal{Z}}_t^d)_{(d)}$ , with rank  $k_t^d \geq \check{k}_t^d$ , using Algorithm 2 takes  $O(k_t^d \|\Omega\|_1 + k_t^d (I_d + I_{D \setminus d})(r_t^d + r_{t-1}^d))$  time. Using Proposition 4.1, solving the proximal step  $\text{prox}_{\mu g}([\check{\mathcal{Z}}_t^1, \dots, \check{\mathcal{Z}}_t^D])$  takes a total of

$$O\left(\sum_{d=1}^D k_t^d \|\Omega\|_1 + k_t^d (I_d + I_{D \setminus d})(r_t^d + r_{t-1}^d)\right), \quad (35)$$

time. As the target tensor is low-rank,  $r_t^d, k_t^d \ll I_d$  for  $d = 1, \dots, D$ . Hence, (35) is much faster than directly using Proposition 4.1 ( $O(\prod_{d=1}^D I_d \sum_{d=1}^D k_t^d)$  time).

#### 4.4 The Proposed Algorithm

The whole procedure is shown in Algorithm 4. Unlike, Algorithm 3,  $D$  SVTs have to be computed (steps 5-11) in each iteration.

Analogous to Theorem 3.5, we have the following.

**Theorem 4.3.** Assume that (i)  $k_t^d \geq \check{k}_t^d$  for  $d = 1, \dots, D$ , all  $t$  and  $J = t$ ; (ii)  $F([\mathcal{X}_t^1, \dots, \mathcal{X}_t^D])$  is upper bounded. The sequence  $\{\mathcal{X}_t^1, \dots, \mathcal{X}_t^D\}$  generated from Algorithm 4 converges to the optimal solution with a  $O(1/T^2)$  rate.

#### 4.5 Post-Processing

As in Section 3.6, the nuclear norm regularizer in (30) may over-penalize top singular values. To undo such shrinkage and boost recovery performance, we also adopt post-processing here. Let the tensor output from Algorithm 4 be  $\mathcal{X} = \sum_{d=1}^D \mathcal{X}^d$ , where  $\mathcal{X}_{(d)}^d = U^d \Sigma^d (V^d)^\top$  has rank  $k^d$ . Define  $\mathcal{A}(\theta^1, \dots, \theta^D) = \sum_{d=1}^D (U^d \text{Diag}(\theta^d) (V^d)^\top)_{(d)}$ . As in (22), we replace  $\mathcal{X}$  with  $\mathcal{A}(\theta_*^1, \dots, \theta_*^D)$ , where

$$[(\theta_*^1)^\top, \dots, (\theta_*^D)^\top]^\top = \arg \min_{\theta^1, \dots, \theta^D} \phi(\theta^1, \dots, \theta^D), \quad (36)$$

and

$$\phi(\theta^1, \dots, \theta^D) = \sum_{(i_1, \dots, i_D) \in \Omega} \ell(\mathcal{A}(\theta^1, \dots, \theta^D)_{i_1 \dots i_D}, \mathcal{O}_{i_1 \dots i_D}).$$

As (36) is a smooth convex problem, L-BFGS is used for optimization. Let  $U^d = [u^d]$  and  $V^d = [v^d]$ . Then,  $\nabla \phi(\theta^1, \dots, \theta^D) = [(w^1)^\top, \dots, (w^D)^\top]^\top$  where  $w^d = [w_i^d] \in k^d$ ,  $w_i^d = (u_i^d)^\top \mathcal{B}_{(d)} v_i^d$ , and  $\mathcal{B}_{i_1 \dots i_D} = \frac{d\ell(\mathcal{A}(\theta^1, \dots, \theta^D)_{i_1 \dots i_D}, \mathcal{O}_{i_1 \dots i_D})}{d\mathcal{A}(\theta^1, \dots, \theta^D)_{i_1 \dots i_D}}$  if  $(i_1, \dots, i_D) \in \Omega$  and 0 otherwise. Computation of  $\nabla \phi(\theta^1, \dots, \theta^D)$  takes  $O(\sum_{d=1}^D k^d \|\Omega\|_1)$  time, which is comparable to the per-iteration complexity of AIS-Impute in (35) and is very efficient. Thus, each L-BFGS iteration is inexpensive. As for

the matrix case, empirically, L-BFGS converges in fewer than 10 iterations. These make post-processing very efficient.

---

#### Algorithm 4. AIS-Impute (Tensor Case)

---

**Require:** partially observed tensor  $\mathcal{O}$ , parameter  $\lambda$ ;

- 1: initialize  $c = 1$ ,  $\mathcal{X}_0^1 = \dots = \mathcal{X}_0^D = 0$ ,  $\mathcal{X}_1^1 = \dots = \mathcal{X}_1^D = 0$ , step-size  $\mu = 1/(\sqrt{D}\rho)$ ,  $\hat{\lambda} > \max_{d=1, \dots, D} \lambda_d$  and  $\nu \in (0, 1)$ ;
  - 2: **for**  $t = 1, 2, \dots, T$  **do**
  - 3:  $\theta_t = (c - 1)/(c + 2)$ ;
  - 4: construct the sparse observed tensor  $\mathcal{S}_t$  from (34);
  - 5: **for**  $d = 1, \dots, D$  **do**
  - 6:  $(\lambda_d)_t = (\hat{\lambda} - \lambda_d) \nu^{t-1} + \lambda_d$ ;
  - 7:  $\check{\mathcal{Z}}_t^d = (1 + \theta_t) \mathcal{X}_t^d - \theta_t \mathcal{X}_{t-1}^d + \mu \mathcal{S}_t$
  - 8:  $V_{t-1}^d = V_{t-1}^d - V_{t-1}^d ((V_{t-1}^d)^\top V_{t-1}^d)^{-1} V_{t-1}^d$ , remove zero columns;
  - 9:  $R_t^d = \text{QR}([V_t^d, V_{t-1}^d])$ ;
  - 10:  $[U_{t+1}^d, \Sigma_{t+1}^d, V_{t+1}^d]$   
 $= \text{approx-SVT}((\check{\mathcal{Z}}_t^d)_{(d)}, R_t^d, \mu(\lambda_d)_t, J)$ ;  
 $// \mathcal{X}_{(d)}^d = U_{t+1}^d \Sigma_{t+1}^d (V_{t+1}^d)^\top$
  - 11: **end for**
  - 12: **if**  $F([\mathcal{X}_{t+1}^1, \dots, \mathcal{X}_{t+1}^D]) > F([\mathcal{X}_t^1, \dots, \mathcal{X}_t^D])$  **then**  $c = 1$ ;
  - 13: **else**  $c = c + 1$ ; **end if**
  - 14: **end for**
  - 15: **return**  $U_{t+1}^d, \Sigma_{t+1}^d, V_{t+1}^d$  where  $d = 1, \dots, D$ .
- 

## 5 EXPERIMENTS

In this section, we perform experiments on matrix completion (Sections 5.1, 5.2, 5.3, 5.4, and 5.5) and tensor completion (Sections 5.6 and 5.7). Experiments are performed on a PC with Intel Xeon E5-2695 CPU and 256 GB RAM.

### 5.1 Synthetic Data

In this section, we perform matrix completion experiments with synthetic data. The ground-truth matrix has a rank of 5, and is generated as  $O = UV \in \mathbb{R}^{m \times m}$ , where the entries of  $U \in \mathbb{R}^{m \times 5}$  and  $V \in \mathbb{R}^{5 \times m}$  are sampled i.i.d. from the standard normal distribution  $\mathcal{N}(0, 1)$ . Noise, sampled from  $\mathcal{N}(0, 0.05)$ , is then added. We randomly choose  $15m \log(m)$  of the entries in  $O$  as observed. Half of them are used for training, and the other half as validation set for parameter tuning. Testing is performed on the unobserved (missing) entries. We vary  $m$  in the range  $\{250, 1000, 4000\}$ .

The following proximal algorithms are compared: (i) accelerated proximal gradient algorithm (denoted ‘‘APG’’) [11]: It uses PROPACK to obtain singular values that are larger than  $\lambda$ ; (ii) Soft-Impute [13]; (iii) AIS-Impute (the proposed Algorithm 3); and (iv) AIS-Impute (exact): This is a variant of the proposed algorithm with exact SVT step (computed using PROPACK).

Let  $X$  be the recovered matrix. For performance evaluation, we use the (i) normalized mean squared error  $\text{NMSE} = \|P_{\Omega^\perp}(X - UV)\|_F / \|P_{\Omega^\perp}(UV)\|_F$ , and (ii) rank of  $X$ . To reduce statistical variability, experimental results are averaged over 5 repetitions.

Results<sup>2</sup> are shown in Table 3. As can be seen, all algorithms have similar NMSE performance, with Soft-Impute being slightly worse. The plots of objective value versus time and iterations are shown in Fig. 1. In terms of the number

2. The lowest and comparable results (according to the pairwise t-test with 95 percent confidence) are highlighted.

TABLE 3  
Matrix Completion Results on the Synthetic Data

		NMSE		rank	post-processing time
		without post-processing	with post-processing		
$m = 250$ (sparsity: 33.1%)	APG	<b>0.0167 ± 0.0007</b>	<b>0.0098 ± 0.0001</b>	5	0.01
	Soft-Impute	<b>0.0166 ± 0.0007</b>	0.0099 ± 0.0001	5	0.01
	AIS-Impute (exact)	<b>0.0165 ± 0.0007</b>	<b>0.0098 ± 0.0001</b>	5	0.01
	AIS-Impute	<b>0.0165 ± 0.0007</b>	<b>0.0098 ± 0.0001</b>	5	0.01
$m = 1000$ (sparsity: 10.4%)	APG	<b>0.0165 ± 0.0001</b>	<b>0.0090 ± 0.0001</b>	5	0.01
	Soft-Impute	0.0170 ± 0.0005	0.0097 ± 0.0001	5	0.03
	AIS-Impute (exact)	<b>0.0166 ± 0.0001</b>	0.0093 ± 0.0001	5	0.02
	AIS-Impute	<b>0.0166 ± 0.0001</b>	0.0092 ± 0.0001	5	0.02
$m = 4000$ (sparsity: 3.1%)	APG	<b>0.0142 ± 0.0002</b>	0.0080 ± 0.0001	5	0.05
	Soft-Impute	<b>0.0143 ± 0.0003</b>	0.0082 ± 0.0002	5	0.18
	AIS-Impute (exact)	<b>0.0142 ± 0.0002</b>	<b>0.0080 ± 0.0001</b>	5	0.11
	AIS-Impute	<b>0.0142 ± 0.0002</b>	<b>0.0080 ± 0.0001</b>	5	0.13

Here, sparsity is the proportion of observed entries, and post-processing time is in seconds.

of iterations, the accelerated algorithms (APG, AIS-Impute (exact) and AIS-Impute) are very similar and converge much faster than Soft-Impute (which only has a  $O(1/T)$  convergence rate). However, in terms of time, both APG and Soft-Impute are slow, as APG does not utilize the “sparse plus low-rank” structure and Soft-Impute has slow convergence. AIS-Impute (exact) is consistently faster than APG and Soft-Impute, as both acceleration and “sparse plus low-rank” structure are utilized. However, AIS-Impute is the fastest as it further allows inexact updates of the proximal step. This also verifies our motivation of using the approximate SVT in Section 3.3.

Table 3 also shows the NMSE results with post-processing (Section 3.6). Compared to the time used by the main algorithm (Fig. 1), the post-processing time is small and can be ignored. Thus, post-processing is always performed in the sequel.

## 5.2 Recommender System

In this section, experiments are performed on two well-known benchmark data sets, *MovieLens* and *Netflix*.

*MovieLens*. The *MovieLens* data set (Table 4) contains ratings ( $\{1, 2, 3, 4, 5\}$ ) of different users on movies. It has been commonly used in matrix completion experiments [13], [28]. We randomly use 50 percent of the observed ratings for training, 25 percent for validation and the rest for testing.

We compare AIS-Impute with the two most popular low-rank matrix learning approaches [1], [9], namely, factorization-based and nuclear-norm minimization methods. The factorization-based methods include (i) large scale matrix fit (“LMaFit”) [32], which uses alternative minimization with over-relaxation; (ii) alternative steepest descent (“ASD”) [33], which uses alternating steepest descent; (iii) rank-one matrix pursuit (“R1MP”) [34], which greedily pursues a rank-one basis in each iteration. The nuclear-norm minimization methods include (i) active subspace selection (“active”) [28], which uses the power method in each iteration to identify the active row and column subspaces; (ii) a boosting approach (“boost”) [31], which uses a variant of the Frank-Wolfe (FW) algorithm [45], with local optimization in each iteration using L-BFGS; (iii) sketchy decisions (“Sketchy”) [35], which is also a FW variant, and uses random matrix projection [17] to reduce the space and per-iteration time complexities; (iv) second-order trust-region algorithm (“TR”) [30], which alternates fixed-rank optimization and rank-one updates; (v) stochastic gradient descent (“SSGD”) [37], which is a stochastic gradient descent algorithm; and (v) matrix completion based on fast alternating least squares

TABLE 4  
*MovieLens* Data Sets Used in the Experiments

	#users	#movies	# observed ratings
100K	943	1,682	100,000
1M	6,040	3,449	999,714
10M	69,878	10,677	10,000,054

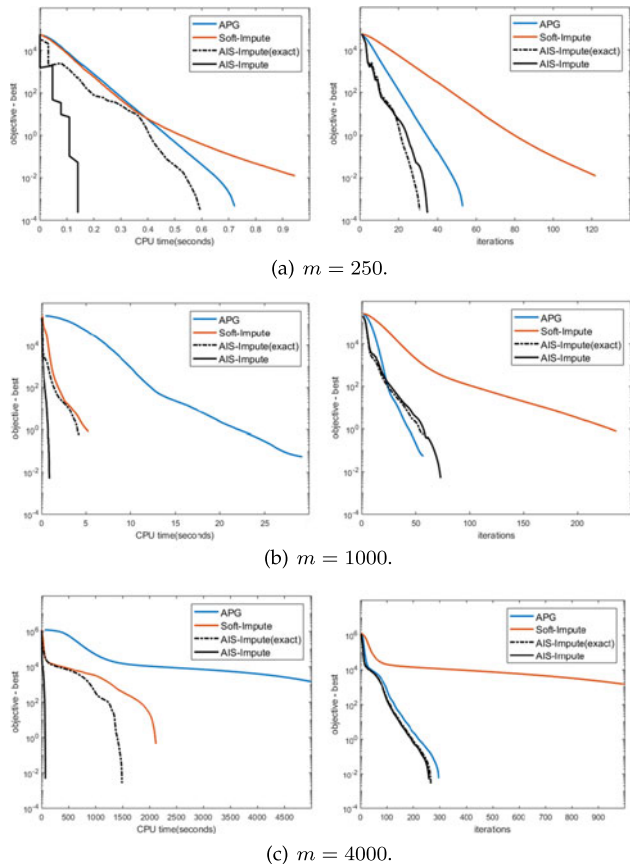


Fig. 1. Convergence of objective value on the synthetic data. Left: versus CPU time (in seconds); Right: versus number of iterations (note that AIS-Impute (exact) and AIS-Impute overlap with each other).



TABLE 5  
Results on the *MovieLens* Data Sets

		100K		1M		10M	
		RMSE	rank	RMSE	rank	RMSE	rank
factorization	LMaFit	$0.896 \pm 0.011$	3	$0.827 \pm 0.002$	6	$0.819 \pm 0.001$	12
	ASD	$0.905 \pm 0.055$	3	$0.826 \pm 0.004$	6	$0.816 \pm 0.002$	12
	R1MP	$0.938 \pm 0.016$	10	$0.857 \pm 0.001$	19	$0.853 \pm 0.002$	27
nuclear norm minimization	active	<b><math>0.880 \pm 0.003</math></b>	8	<b><math>0.821 \pm 0.001</math></b>	16	<b><math>0.803 \pm 0.001</math></b>	72
	boost	<b><math>0.881 \pm 0.003</math></b>	8	<b><math>0.821 \pm 0.001</math></b>	16	$0.814 \pm 0.001$	15
	Sketchy	$0.889 \pm 0.003$	8	<b><math>0.821 \pm 0.001</math></b>	48	$0.826 \pm 0.001$	60
	TR	$0.884 \pm 0.002$	8	<b><math>0.820 \pm 0.001</math></b>	20	—	—
	SSGD	$0.886 \pm 0.011$	8	$0.849 \pm 0.006$	16	$0.858 \pm 0.014$	45
	APG	<b><math>0.880 \pm 0.003</math></b>	8	<b><math>0.820 \pm 0.001</math></b>	16	—	—
	Soft-Impute	<b><math>0.881 \pm 0.003</math></b>	8	<b><math>0.821 \pm 0.001</math></b>	16	<b><math>0.803 \pm 0.001</math></b>	72
	ALT-Impute	<b><math>0.882 \pm 0.003</math></b>	8	$0.823 \pm 0.001$	16	$0.805 \pm 0.001$	45
	AIS-Impute	<b><math>0.880 \pm 0.003</math></b>	8	<b><math>0.820 \pm 0.001</math></b>	16	<b><math>0.802 \pm 0.001</math></b>	72

Note that TR and APG cannot converge in  $10^4$  seconds on the 10M data set.

(“ALT-Impute”) [36], which is a fast variant of Soft-Impute [13] that avoids SVD by alternating least squares. For all algorithms, parameters are tuned using the validation set. The algorithm is stopped when the relative change in objectives between consecutive iterations is smaller than  $10^{-4}$ .

For performance evaluation, as in [13], [28], we use (i) the root mean squared error on the test set:  $\text{RMSE} = \|P_{\hat{\Omega}}(X - \hat{O})\|_F / (\|\hat{\Omega}\|_1)^{\frac{1}{2}}$ , where  $X$  is the recovered matrix, and the testing ratings  $\{\hat{O}_{ij}\}$  is indexed by the set  $\hat{\Omega}$ ; and (ii) rank of  $X$ . The experiment is repeated 5 times and the average performance is reported.

Results are shown in Table 5. As can be seen, AIS-Impute is consistently the fastest and has the lowest RMSE. On *MovieLens-10M*, TR and APG are not run as they are too slow. Fig. 2 shows the testing RMSE with CPU time. As can be seen, Boost, TR, SSGD and APG are all very slow. Boost and TR need to solve an expensive subproblem in each iteration; SSGD has slow convergence; while APG requires SVD and does not utilize the “sparse plus low-rank” structure for fast matrix multiplication. ALT-Impute and LMaFit do not need SVT, and are faster than Soft-Impute. However, their nonconvex formulations have slow convergence, and are thus slower than AIS-Impute. Overall, AIS-Impute is the fastest, as it combines inexpensive iteration with fast convergence.

*Netflix*. The *Netflix* data set contains ratings of 480,189 users on 17,770 movies. 1 percent of the ratings matrix are observed. We randomly sample 50 percent of the observed ratings for training, and the rest for testing.

We only compare with active subspace selection, ALT-Impute and Soft-Impute; while methods including boost,

TR, SSGD, APG are slow and not compared. LMaFit solves a different optimization problem based on matrix factorization, and has worse recovery performance than AIS-Impute. Thus, it is also not compared. As in [13], several choices of the regularization parameter  $\lambda$  are experimented.

Results are shown in Table 6. As in previous experiments, the RMSEs and ranks obtained by the various algorithms are similar. Fig. 3 shows the plot of testing RMSE versus CPU time. As can be seen, AIS-Impute is again much faster.

### 5.3 Grayscale Images

In this section, we perform experiments on images from [18] (Figs. 4a, 4b, and 4c). The pixels are normalized to zero mean and unit variance. Gaussian noise from  $\mathcal{N}(0, 0.05)$  is then added. In each image, 50 percent of the pixels are randomly sampled as observations (half for training and another half for validation). The task is to fill in the remaining 80 percent of the pixels. The experiment is repeated five times.

Table 7 shows the testing RMSE and recovered rank. As can be seen, nuclear norm minimization is better in terms of RMSE (in particular, AIS-Impute, ALT-Impute, APG and boost are the best), though they require the use of higher ranks. Fig. 5 shows the running time. As can be seen, AIS-Impute is consistently the fastest.

Fig. 6 compares the difference between recovered images from all algorithms and the clean one on image *tree*. As can be seen, the difference on SSGD is the largest. Besides, LMaFit, ASD, and R1MP and SSGD also have larger errors than

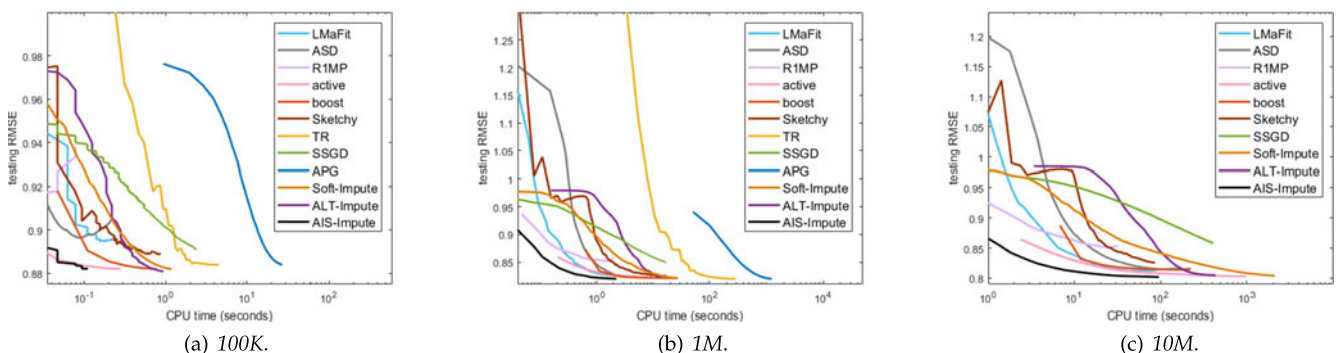


Fig. 2. Testing RMSE versus CPU time (in seconds) on *MovieLens* data sets.

TABLE 6  
Results on the *Netflix* Data Set

		RMSE	rank
$c = 10$	active	$0.894 \pm 0.001$	3
	ALT-Impute	$0.900 \pm 0.006$	3
	Soft-Impute	<b><math>0.893 \pm 0.001</math></b>	3
	AIS-Impute	<b><math>0.893 \pm 0.001</math></b>	3
$c = 20$	active	<b><math>0.847 \pm 0.001</math></b>	14
	ALT-Impute	$0.850 \pm 0.001$	14
	Soft-Impute	<b><math>0.847 \pm 0.001</math></b>	14
	AIS-Impute	<b><math>0.847 \pm 0.001</math></b>	14
$c = 30$	active	<b><math>0.820 \pm 0.001</math></b>	116
	ALT-Impute	$0.825 \pm 0.001$	116
	AIS-Impute	<b><math>0.820 \pm 0.001</math></b>	116

The regularization parameter  $\lambda$  in (1) is set as  $\lambda_0/c$ , where  $\lambda_0 = \|P_\Omega(O)\|_F$ . Soft-Impute with  $c = 30$  is not run as it is very slow.

the rest. The observations on *rice* and *wall* are similar, however, due to space limitation, we do not show them here.

#### 5.4 Nonconvex Regularization

In the following, we first perform experiments on (i) synthetic data, using the setup in Section 5.1 (with  $m = 250$  and 1000); and (ii) the recommender data set *MovieLens-100K*, using the setup in Section 5.2. Three nonconvex low-rank regularizers are considered, namely, truncated nuclear norm (TNN) [18], capped- $\ell_1$  norm [42] and log-sum-penalty (LSP) [40].

For TNN, we compare three solvers: (i) TNNR(APG): the solver used in [18]; (ii) IRNN [6], which is a more recent proximal algorithm for optimization with nonconvex low-rank



(a) *rice* ( $854 \times 960$ ). (b) *tree* ( $800 \times 800$ ). (c) *wall* ( $841 \times 850$ ).

Fig. 4. Grayscale images used for matrix completion. Their sizes are shown in the bracket.

matrix regularizers (including the TNN); and (iii) the proposed AIS-Impute extension (denoted DC(AIS-Impute)), which replaces the original APG solver in [18] for the subproblem in TNNR with AIS-Impute. For capped- $\ell_1$  and LSP, two solvers are considered: (i) IRNN and (ii) the proposed AIS-Impute extension. As a further baseline, we also compare with (convex) nuclear norm regularization with the AIS-Impute solver. Experiments are repeated five times.

Results are shown in Table 8 and Fig. 7. As can be seen, the errors obtained by nonconvex regularization (i.e., TNN, capped- $\ell_1$  and LSP) are much lower than those from convex nuclear norm regularization, illustrating the advantage of using nonconvex regularization. The performance obtained by the different nonconvex regularizers are comparable.

#### 5.5 Link Prediction

Given a graph with  $m$  nodes and an incomplete adjacency matrix  $O \in \{\pm 1\}^{m \times m}$ , link prediction aims to recover a low-rank matrix  $X \in \mathbb{R}^{m \times m}$  such that the signs of  $X_{ij}$ 's and  $O_{ij}$ 's agree on most of the observed entries. This is a binary

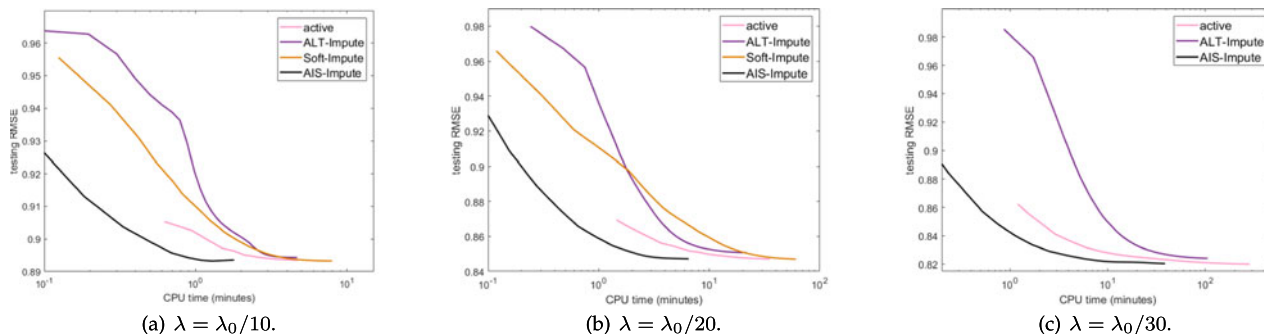


Fig. 3. Testing RMSE versus CPU time (in minutes) on the *Netflix* data set, with various values for the regularization parameter  $\lambda$ .

TABLE 7  
Matrix Completion Results on Grayscale Images

		<i>rice</i>		<i>tree</i>		<i>wall</i>	
		RMSE	rank	RMSE	rank	RMSE	rank
factorization	LMaFit	$0.189 \pm 0.002$	45	$0.174 \pm 0.013$	25	$0.238 \pm 0.004$	50
	ASD	$0.194 \pm 0.020$	45	$0.142 \pm 0.004$	25	$0.189 \pm 0.012$	50
	R1MP	$0.207 \pm 0.001$	54	$0.159 \pm 0.002$	53	$0.175 \pm 0.001$	58
nuclear norm minimization	active	<b><math>0.176 \pm 0.002</math></b>	100	<b><math>0.130 \pm 0.002</math></b>	71	<b><math>0.150 \pm 0.002</math></b>	101
	boost	<b><math>0.176 \pm 0.004</math></b>	94	<b><math>0.130 \pm 0.002</math></b>	60	<b><math>0.149 \pm 0.002</math></b>	93
	Sketchy	$0.186 \pm 0.007$	89	$0.134 \pm 0.002$	41	$0.157 \pm 0.008$	88
	TR	<b><math>0.179 \pm 0.001</math></b>	150	<b><math>0.131 \pm 0.002</math></b>	103	<b><math>0.151 \pm 0.001</math></b>	149
	SSGD	$0.447 \pm 0.058$	96	$0.424 \pm 0.037$	60	$0.463 \pm 0.023$	96
	APG	<b><math>0.176 \pm 0.001</math></b>	96	<b><math>0.130 \pm 0.002</math></b>	60	<b><math>0.151 \pm 0.001</math></b>	96
	Soft-Impute	<b><math>0.176 \pm 0.001</math></b>	113	<b><math>0.131 \pm 0.004</math></b>	71	<b><math>0.151 \pm 0.002</math></b>	112
	AIS-Impute	<b><math>0.176 \pm 0.001</math></b>	96	<b><math>0.130 \pm 0.004</math></b>	71	<b><math>0.150 \pm 0.001</math></b>	95
	AIS-Impute	<b><math>0.176 \pm 0.001</math></b>	96	<b><math>0.219 \pm 0.002</math></b>	70	<b><math>0.150 \pm 0.001</math></b>	95

CPU time is in seconds.

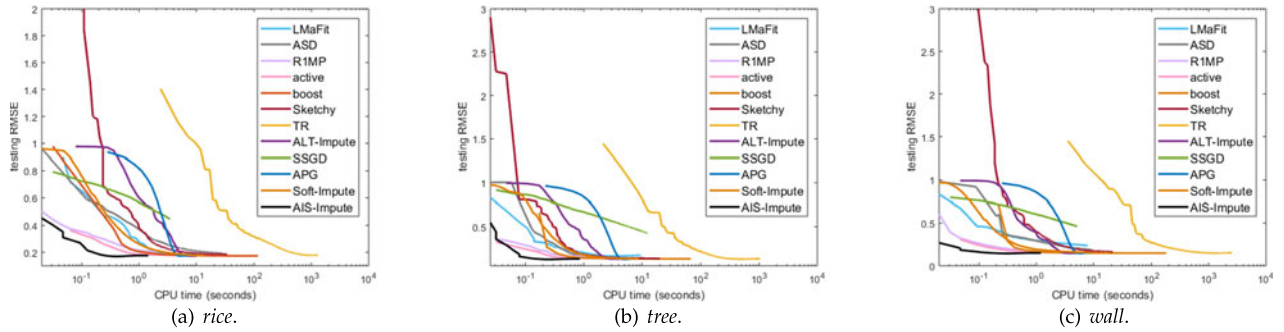
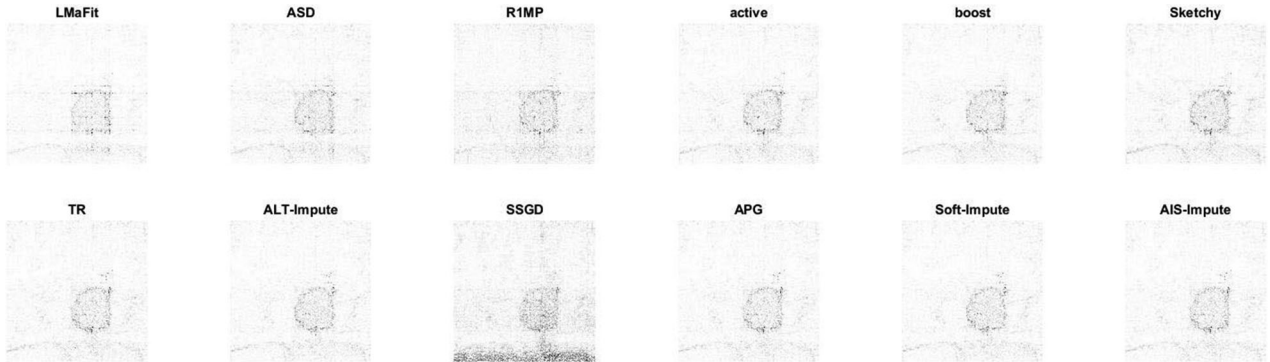
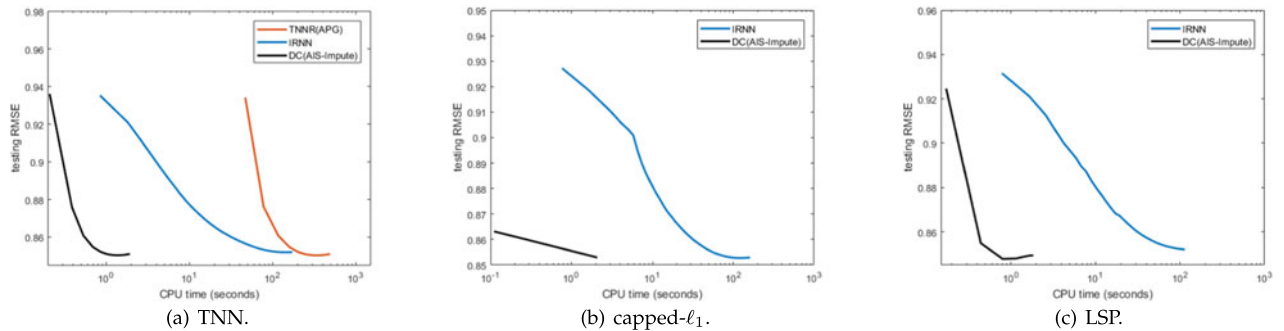


Fig. 5. Testing RMSE versus CPU time (in seconds) on grayscale images.


 Fig. 6. Comparison on the difference between reconstructed images and the clean one on image *tree*.

 Fig. 7. Convergence of testing RMSE versus CPU time (in seconds) on the *MovieLens-100K* data set.

matrix completion problem [3], and we use the logistic loss  $\ell(X_{ij}, O_{ij}) \equiv \log(1 + \exp(-X_{ij}O_{ij}))$  in (19).

Experiments are performed on the *Epinions* and *Slashdot* data sets [3] (Table 9). Each row/column of the matrix  $O$  corresponds to a user (users with fewer than two observations are removed). For *Epinions*,  $O_{ij} = 1$  if user  $i$  trusts user  $j$ , and  $-1$  otherwise. Similarly for *Slashdot*,  $O_{ij} = 1$  if user  $i$  tags user  $j$  as friend, and  $-1$  otherwise. As can be seen from previous sections, Boost, TR, SSGD, APG and Soft-Impute are all slow, and thus they are not considered here. Besides, LMaFit and ALT-Impute are designed for the square loss. Thus, comparison is performed with (i) active subspace selection; (ii) AIS-Impute; and (iii) AltMin: the alternative minimization approach used in [3]. We use 80 percent of the ratings for training, 10 percent for validation and the rest for testing. Let  $X$  be the recovered matrix, and the test set  $\{\hat{O}_{ij}\}$  be indexed by the set  $\hat{\Omega}$ . For performance evaluation, we use the (i) testing accuracy  $\frac{1}{\|\hat{\Omega}\|_1} \sum_{(i,j) \in \hat{\Omega}} I(\text{sign}(X_{ij}) = \hat{O}_{ij})$ , where  $I(\cdot)$  is the indicator function; and (ii) the rank of  $X$ . To reduce statistical variability, experimental results are averaged over 5 repetitions.

Results are shown in Table 10 and Fig. 8 shows the testing accuracy with CPU time. As can be seen, active and AIS-Impute have slightly better accuracies than AltMin, and AIS-Impute is the fastest.

## 5.6 Tensor Completion: Synthetic Data

In this section, we perform tensor completion experiments with synthetic data. The ground-truth data tensor (of size  $m \times m \times 3$ ) is generated as  $\mathcal{O} = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3$ , where the elements of  $A_1 \in \mathbb{R}^{m \times 3}$ ,  $A_2 \in \mathbb{R}^{m \times 3}$ ,  $A_3 \in \mathbb{R}^{3 \times 3}$  and the core tensor  $\mathcal{C} \in \mathbb{R}^{3 \times 3 \times 3}$  are all sampled i.i.d. from the standard normal distribution  $\mathcal{N}(0, 1)$ , and  $\times_k$  is the  $k$ -mode product.<sup>3</sup> Thus,  $\mathcal{O}$  is low-rank for the first two mode but not for the third. Noise  $\mathcal{G}$ , with elements sampled i.i.d. from the normal distribution  $\mathcal{N}(0, 0.05)$ , is then added. A total number of  $\Omega = 45m \log(m)$  random elements in  $\mathcal{O}$  are observed. Half of them are used for training, and the other half for validation. On testing, we perform evaluation on the unobserved

3. The  $k$ -mode product of a tensor  $\mathcal{X}$  and a matrix  $A$  is defined as  $\mathcal{X} \times_k A = (\mathcal{X}^{(k)} A)_{(k)}$  [19].

TABLE 8  
Comparison of Nuclear Norm Regularization with Various Nonconvex Regularizations

		NMSE		RMSE
		synthetic ( $m = 250$ )	synthetic ( $m = 1000$ )	<i>MovieLens-100K</i>
nuclear norm	AIS-Impute	0.0098 ± 0.0004	0.0092 ± 0.0002	0.883 ± 0.005
	TNNR(APG)	<b>0.0081 ± 0.0004</b>	<b>0.0073 ± 0.0001</b>	<b>0.851 ± 0.002</b>
	IRNN	<b>0.0081 ± 0.0004</b>	<b>0.0073 ± 0.0001</b>	0.853 ± 0.004
capped- $\ell_1$	DC(AIS-Impute)	<b>0.0081 ± 0.0004</b>	<b>0.0073 ± 0.0002</b>	<b>0.851 ± 0.002</b>
	IRNN	0.0089 ± 0.0005	<b>0.0074 ± 0.0001</b>	0.853 ± 0.002
LSP	DC(AIS-Impute)	<b>0.0081 ± 0.0004</b>	<b>0.0073 ± 0.0002</b>	0.852 ± 0.005
	IRNN	0.0083 ± 0.0004	0.0076 ± 0.0001	0.852 ± 0.006
	DC(AIS-Impute)	<b>0.0081 ± 0.0004</b>	<b>0.0073 ± 0.0002</b>	<b>0.850 ± 0.002</b>

entries and use the same criteria as in Section 5.1, i.e., NMSE and recovered rank in each mode.

Similar to Section 5.1, we compare the following algorithms: (i) APG; (ii) extension of Soft-Impute to tensor completion, which is based on Section 4.2; (iii) the proposed algorithm with exact SVD (AIS-Impute(exact)); and (iv) the proposed algorithm which uses power method to approximate SVT (AIS-Impute).

As suggested in [22], we set  $(\lambda_1, \lambda_2, \lambda_3)$  in the scaled latent nuclear norm to  $(1, 1, \frac{\sqrt{m}}{\sqrt{3}})\lambda$ . Thus, the only tunable parameter is  $\lambda$ , which is obtained by grid search using the validation set. We also vary  $m$  in  $\{125, 500, 2000\}$ . Experimental results are averaged over 5 repetitions.

Results on NMSE and rank are shown in Table 11. As can be seen, APG, Soft-Impute, AIS-Impute(exact) and AIS-Impute have comparable performance. The plots of objective value versus time and iterations are shown in Fig. 9. In terms of iterations, APG, AIS-Impute(exact) and AIS-Impute have similar behavior as they all have  $O(1/T^2)$  convergence rate. These also agree with the matrix case in Section 5.1. In terms of time, as APG does not utilize the “sparse plus low-rank” structure, it is slower than AIS-Impute(exact) and AIS-Impute. AIS-Impute is the fastest, as it has both fast  $O(1/T^2)$  convergence rate and low per-iteration complexity.

Performance with post-processing in Section 4.5 is shown in Table 11. As can be seen, it is very efficient and improves NMSE. Thus, we always perform post-processing in the sequel.

TABLE 9  
Data Sets for Link Prediction

	#rows	#columns	#signs
<i>Epinions</i>	84,601	48,091	505,074
<i>Slashdot</i>	70,284	32,188	324,745

TABLE 10  
Performance on Link Prediction

		accuracy	rank
<i>Epinions</i>	active	<b>0.939 ± 0.002</b>	12
	AltMin	0.936 ± 0.002	41
	AIS-Impute	<b>0.940 ± 0.001</b>	12
<i>Slashdot</i>	active	<b>0.844 ± 0.001</b>	16
	AltMin	0.839 ± 0.002	39
	AIS-Impute	<b>0.843 ± 0.001</b>	16

## 5.7 Multi-Relational Link Prediction

In this section, we perform experiments on the *YouTube* data set [46]. It contains 15,088 users, and describes five types of user interactions: contact, number of shared friends, number of shared subscriptions, number of shared subscribers, and the number of shared favorite videos. Thus, it forms a  $15,088 \times 15,088 \times 5$  tensor, with a total of 27,257,790 non-zero elements. Following [3], we formulate multi-relational link prediction as a tensor completion problem. As the observations are real-valued, we use the square loss in (30). Besides AIS-Impute (Algorithm 4), we also compare with the following state-of-the-art non-proximal-based tensor completion algorithms: (i) geometric nonlinear CG for tensor completion (denoted “GeomCG”) [47]: a gradient descent approach with gradients restricted on the Riemannian manifold; (ii) An alternating direction method of multipliers approach (denoted “ADMM(overlap)”) [20], which solves the overlapping nuclear norm regularized tensor completion problem; (iii) fast low rank tensor completion (denoted “FaLRTC”) [4]: It smooths the overlapping nuclear norm and then solves the relaxed problem with accelerated gradient descent; and (iv) tensor completion by parallel matrix factorization (denoted “TMac”) [48]: An extension of LMaFit [32] to tensor completion, which performs simultaneous low-rank matrix factorizations to all mode matricizations.

*YouTube Subset.* First, we perform experiments on a small *YouTube* subset, obtained by random selecting 1,000 users (leading to 12,101 observations). We use 50 percent of the observations for training, another 25 percent for validation and the remaining for testing. Let  $\mathcal{X}$  be the recovered tensor, and the testing ratings  $\hat{O}_{ij}$  be indexed by the set  $\hat{\Omega}$ . For performance evaluation, we use (i) the testing root mean squared error  $\text{RMSE} = \sqrt{\|P_{\hat{\Omega}}(\mathcal{X} - \hat{\mathcal{O}})\|_F^2 / \|\hat{\Omega}\|_1}$ ; and

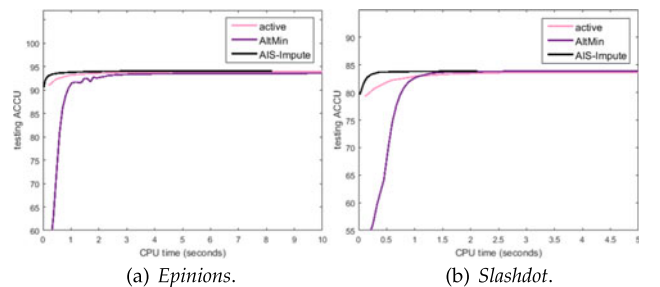


Fig. 8. Testing accuracy versus CPU time (in seconds) on the *Epinions* and *Slashdot* data sets.

TABLE 11  
Tensor Completion Results on the Synthetic Data

		NMSE		rank	post-processing time
		no post-processing	with post-processing		
$m = 125$ (sparsity: 62.4%)	APG	$0.0162 \pm 0.0015$	$0.0100 \pm 0.0006$	3,3,0	0.1
	Soft-Impute	$0.0162 \pm 0.0014$	$0.0100 \pm 0.0005$	3,3,0	0.1
	AIS-Impute(exact)	$0.0161 \pm 0.0015$	$0.0100 \pm 0.0005$	3,3,0	0.1
	AIS-Impute	$0.0159 \pm 0.0011$	$0.0099 \pm 0.0004$	3,3,0	0.1
$m = 500$ (sparsity: 16.0%)	APG	$0.0166 \pm 0.0007$	$0.0105 \pm 0.0004$	3,3,0	0.1
	Soft-Impute	$0.0168 \pm 0.0007$	$0.0106 \pm 0.0004$	3,3,0	0.1
	AIS-Impute(exact)	$0.0167 \pm 0.0006$	$0.0104 \pm 0.0003$	3,3,0	0.1
	AIS-Impute	$0.0167 \pm 0.0007$	$0.0105 \pm 0.0003$	3,3,0	0.1
$m = 2000$ (sparsity: 3.9%)	APG	$0.0162 \pm 0.0013$	$0.0105 \pm 0.0006$	3,3,0	0.5
	Soft-Impute	$0.0168 \pm 0.0016$	$0.0109 \pm 0.0011$	3,3,0	0.4
	AIS-Impute(exact)	$0.0161 \pm 0.0012$	$0.0104 \pm 0.0007$	3,3,0	0.4
	AIS-Impute	$0.0161 \pm 0.0012$	$0.0104 \pm 0.0007$	3,3,0	0.1

Here, sparsity is the proportion of observed entries, and post-processing time is in seconds.

(ii) rank of the unfolded matrix in each mode. The experiments are repeated five times.

Performance is shown in Table 12 and Fig. 10a shows the time comparison. ADMM(overlap) and FaLRTC have similar recovery performance, but are all very slow due to usage of the SVD. As the overlapping nuclear norm is smoothed in FaLRTC, its cannot exactly recover a low-rank tensor. TMac is fast, but has the worst recovery performance. AIS-Impute enjoys fast speed and good recovery performance.

Full YouTube Data. Next, we perform experiments on the full YouTube data set with the same setup. As ADMM (overlap) and FaLRTC are too slow, we only compare with GeomCG, TMac and AIS-Impute. Experiments are repeated five times.

Results are shown in Table 13, and Fig. 10b shows the time. TMac has much worse performance than GeomCG and AIS-Impute. GeomCG is based on the (nonconvex) Turker decomposition, and its convergence rate is unknown. Moreover, its iteration time complexity has a

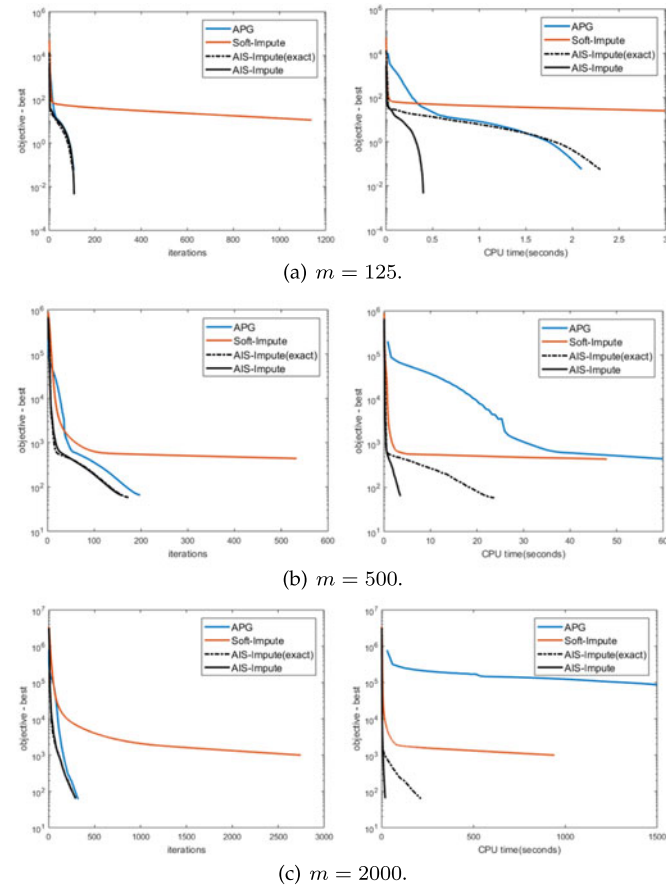


Fig. 9. Convergence of objective value on the synthetic tensor data. Left: versus number of iterations. Right: versus CPU time (in seconds).

TABLE 12  
Results on the YouTube Subset

	RMSE	rank
GeomCG	$0.672 \pm 0.050$	7, 7, 5
ADMM(overlap)	$0.690 \pm 0.030$	142, 142, 5
FaLRTC	$0.672 \pm 0.032$	1000, 1000, 5
TMac	$0.786 \pm 0.027$	4, 4, 0
AIS-Impute	$0.616 \pm 0.029$	33, 33, 0

The rank is for each mode.

TABLE 13  
Results on the Full YouTube Dataset

	RMSE	rank
GeomCG	$0.388 \pm 0.001$	51, 51, 5
TMac	$0.611 \pm 0.007$	10, 10, 0
AIS-Impute	$0.369 \pm 0.006$	70, 70, 0

The rank is for each mode.

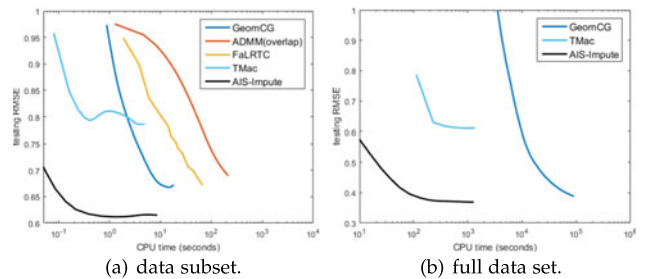


Fig. 10. Testing RMSE versus CPU time on the Youtube data set.

worse dependency on the tensor rank than AIS-Impute ( $\prod_{i=1}^D r_t^d$  versus  $\sum_{i=1}^D r_t^d$ ), and thus GeomCG becomes very slow when the tensor rank is large. Overall, AIS-Impute has fast speed and good recovery performance.

## 6 CONCLUSION

In this paper, we show that Soft-Impute, as a proximal algorithm, can be accelerated without losing the “sparse plus low-rank” structure crucial to its efficiency. To further reduce the per-iteration time complexity, we proposed an approximate-SVT scheme based on the power method. Theoretical analysis shows that the proposed algorithm still enjoys the fast  $O(1/T^2)$  convergence rate. We also extend the proposed algorithm for low-rank tensor completion with the scaled latent nuclear norm regularizer. Again, the “sparse plus low-rank” structure can be preserved and a convergence rate of  $O(1/T^2)$  can be obtained. The proposed algorithm can be further extended to nonconvex low-rank regularizers, which have better empirical performance than the convex nuclear norm regularizer. Extensive experiments on both synthetic and real-world data sets show that the proposed algorithm is much faster than the state-of-the-art.

## ACKNOWLEDGMENTS

This research was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region (Grant 614513). The first author is especially thankful for support from 4Paradigm Inc.

## REFERENCES

- [1] Y. Koren, “Factorization meets the neighborhood: A multifaceted collaborative filtering model,” in *Proc. 14th Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 426–434.
- [2] M. Kim and J. Leskovec, “The network completion problem: Inferring missing nodes and edges in networks,” in *Proc. 11st Int. Conf. Data Mining*, 2011, pp. 47–58.
- [3] K.-Y. Chiang, C.-J. Hsieh, N. Natarajan, I. Dhillon, and A. Tewari, “Prediction and clustering in signed networks: A local to global perspective,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1177–1213, 2014.
- [4] J. Liu, P. Musialski, P. Wonka, and J. Ye, “Tensor completion for estimating missing values in visual data,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, Jan. 2013.
- [5] S. Gu, Q. Xie, D. Meng, W. Zuo, X. Feng, and L. Zhang, “Weighted nuclear norm minimization and its applications to low level vision,” *Int. J. Comput. Vis.*, vol. 121, no. 2, pp. 183–208, 2017.
- [6] C. Lu, J. Tang, S. Yan, and Z. Lin, “Nonconvex nonsmooth low rank minimization via iteratively reweighted nuclear norm,” *IEEE Trans. Image Process.*, vol. 25, no. 2, pp. 829–839, Feb. 2016.
- [7] Z. Zhao, L. Zhang, X. He, and W. Ng, “Expert finding for question answering via graph regularized matrix completion,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 4, pp. 993–1004, Apr. 2015.
- [8] J. Fan, Z. Tian, M. Zhao, and T. Chow, “Accelerated low-rank representation for subspace clustering and semi-supervised classification on large-scale data,” *Neural Netw.*, vol. 100, pp. 39–48, 2018.
- [9] E. Candès and B. Recht, “Exact matrix completion via convex optimization,” *Found. Comput. Math.*, vol. 9, no. 6, pp. 717–772, 2009.
- [10] J.-F. Cai, E. Candès, and Z. Shen, “A singular value thresholding algorithm for matrix completion,” *SIAM J. Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.
- [11] K.-C. Toh and S. Yun, “An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems,” *Pacific J. Optimization*, vol. 6, no. 615–640, 2010, Art. no. 15.
- [12] R. Larsen, “Lanczos bidiagonalization with partial reorthogonalization,” Department of Computer Science, Aarhus University, DAIMI PB-357, 1998.
- [13] R. Mazumder, T. Hastie, and R. Tibshirani, “Spectral regularization algorithms for learning large incomplete matrices,” *J. Mach. Learn. Res.*, vol. 11, pp. 2287–2322, 2010.
- [14] R. Tibshirani, “Proximal gradient descent and acceleration,” Lecture Notes, 2010, [Online]. Available: <http://www.stat.cmu.edu/~ryantibs/convexopt-S15/lectures/08-prox-grad.pdf>
- [15] A. Beck and M. Teboulle, “A fast iterative shrinkage-thresholding algorithm for linear inverse problems,” *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.
- [16] S. Ji and J. Ye, “An accelerated gradient method for trace norm minimization,” in *Proc. 26th Int. Conf. Mach. Learn.*, 2009, pp. 457–464.
- [17] N. Halko, P.-G. Martinsson, and J. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions,” *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.
- [18] Y. Hu, D. Zhang, J. Ye, X. Li, and X. He, “Fast and accurate matrix completion via truncated nuclear norm regularization,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 9, pp. 2117–2130, Sep. 2013.
- [19] T. Kolda and B. Bader, “Tensor decompositions and applications,” *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009.
- [20] R. Tomioka, K. Hayashi, and H. Kashima, “Estimation of low-rank tensors via convex optimization,” arXiv:1010.0789, University of Tokyo, Tokyo, Japan, 2010.
- [21] E. Acar, D. Dunlavy, T. Kolda, and M. Mørup, “Scalable tensor factorizations for incomplete data,” *Chemometrics Intell. Laboratory Syst.*, vol. 106, no. 1, pp. 41–56, 2011.
- [22] K. Wimalawarne, M. Sugiyama, and R. Tomioka, “Multitask learning meets tensor factorization: Task imputation via convex optimization,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2825–2833.
- [23] Q. Yao and J. T. Kwok, “Accelerated inexact soft-impute for fast large-scale matrix completion,” in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 4002–4008.
- [24] N. Parikh and S. Boyd, “Proximal algorithms,” *Found. Trends Optimization*, vol. 1, no. 3, pp. 127–239, 2014.
- [25] L. Jacob, G. Obozinski, and J.-P. Vert, “Group lasso with overlap and graph lasso,” in *Proc. 26th Int. Conf. Mach. Learn.*, 2009, pp. 433–440.
- [26] M. Schmidt, N. Roux, and F. Bach, “Convergence rates of inexact proximal-gradient methods for convex optimization,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 1458–1466.
- [27] K. Wu and H. Simon, “Thick-restart Lanczos method for large symmetric eigenvalue problems,” *SIAM J. Matrix Anal. Appl.*, vol. 22, no. 2, pp. 602–616, 2000.
- [28] C.-J. Hsieh and P. Olsen, “Nuclear norm minimization via active subspace selection,” in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 575–583.
- [29] B. O’Donoghue and E. Candès, “Adaptive restart for accelerated gradient schemes,” *Found. Comput. Math.*, vol. 15, pp. 1–18, 2012.
- [30] B. Mishra, G. Meyer, F. Bach, and R. Sepulchre, “Low-rank optimization with trace norm penalty,” *SIAM J. Optimization*, vol. 23, no. 4, pp. 2124–2149, 2013.
- [31] X. Zhang, D. Schuurmans, and Y.-L. Yu, “Accelerated training for matrix-norm regularization: A boosting approach,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 2906–2914.
- [32] Z. Wen, W. Yin, and Y. Zhang, “Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm,” *Math. Program. Comput.*, vol. 4, no. 4, pp. 333–361, 2012.
- [33] J. Tanner and K. Wei, “Low rank matrix completion by alternating steepest descent methods,” *Appl. Comput. Harmonic Anal.*, vol. 40, no. 2, pp. 417–429, 2016.
- [34] Z. Wang, M. Lai, Z. Lu, W. Fan, H. Davulcu, and J. Ye, “Orthogonal rank-one matrix pursuit for low rank matrix completion,” *SIAM J. Sci. Comput.*, vol. 37, no. 1, pp. A488–A514, 2015.
- [35] A. Yurtsever, M. Udell, J. Tropp, and V. Cevher, “Sketchy decisions: Convex low-rank matrix optimization with optimal storage,” in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1188–1196.
- [36] T. Hastie, R. Mazumder, J. Lee, and R. Zadeh, “Matrix completion and low-rank SVD via fast alternating least squares,” *J. Mach. Learn. Res.*, vol. 16, pp. 3367–3402, 2015.
- [37] H. Avron, S. Kale, V. Sindhvani, and S. Kasiviswanathan, “Efficient and practical stochastic subgradient descent for nuclear norm regularization,” in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1231–1238.

- [38] J. Nocedal and S. Wright, *Numerical Optimization*. Berlin, Germany: Springer, 1999.
- [39] L. An and P. Tao, "The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems," *Ann. Operations Res.*, vol. 133, no. 1/4, pp. 23–46, 2005.
- [40] E. Candès, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted  $\ell_1$  minimization," *J. Fourier Anal. Appl.*, vol. 14, no. 5/6, pp. 877–905, 2008.
- [41] C. Zhang, "Nearly unbiased variable selection under minimax concave penalty," *sAnnals Statist.*, vol. 38, no. 2, pp. 894–942, 2010.
- [42] T. Zhang, "Analysis of multi-stage convex relaxation for sparse regularization," *J. Mach. Learn. Res.*, vol. 11, pp. 1081–1107, 2010.
- [43] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, 2009.
- [44] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, pp. 1–122, 2011.
- [45] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Res. Logistics*, vol. 3, no. 1/2, pp. 95–110, 1956.
- [46] T. Lei, X. Wang, and H. Liu, "Uncovering groups via heterogeneous interaction analysis," in *Proc. IEEE Int. Conf. Data Mining*, 2009, pp. 503–512.
- [47] D. Kressner, M. Steinlechner, and B. Vandereycken, "Low-rank tensor completion by Riemannian optimization," *BIT Numerical Math.*, vol. 54, no. 2, pp. 447–468, 2014.
- [48] Y. Xu, R. Hao, W. Yin, and Z. Su, "Parallel matrix factorization for low-rank tensor completion," *Inverse Problems Imag.*, vol. 9, no. 2, 2013, pp. 601–624.



HKUST in 2015, and a Google PhD fellowship (machine learning) in 2016. He is a member of the IEEE.

**Quanming Yao** received the bachelor's degree in electronic and information engineering from the Huazhong University of Science and Technology (HUST), in 2013, and the PhD degree from Computer Science and Engineer Department, Hong Kong University of Science and Technology (HKUST), in 2018. His research interests focus on machine learning. Currently, he is a research scientist in 4Paradigm Inc., Beijing, China. He was awarded as Qiming Star of HUST in 2012, the Tse Cheuk Ng Tai Research Excellence Prize from



Outstanding Paper Award, and the Second Class Award in Natural Sciences by the Ministry of Education, Peoples Republic of China, in 2008. He has been a program cochair for a number of international conferences, and served as an associate editor for the *IEEE Transactions on Neural Networks and Learning Systems* and the *Neurocomputing Journal*. He is a fellow of the IEEE.

**James T. Kwok** received the PhD degree in computer science from the Hong Kong University of Science and Technology, in 1996. He was with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, as an assistant professor. He is currently a professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include kernel methods, machine learning, example recognition, and artificial neural networks. He received the 2004 IEEE

▷ **For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).**