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

- Q. Yao is with 4Paradigm Inc, Beijing, China. E-mail: yaoquan-ming@4Paradigm.com
- James 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.

X by solving following optimization problem:

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

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 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.

Notation. In the sequel, the transpose of vector/ matrix is denoted by the superscript $\cdot^\top$, and tensors are denoted by boldface Euler. For a vector x, $\|x\|_1 = \sum_i |x_i|$ is its ℓ_1-norm, and $\|x\| = \sqrt{\sum_i x_i^2}$ its ℓ_2-norm. For a matrix X, $\sigma_1(X) \geq \sigma_2(X) \geq \ldots \sigma_m(X)$ 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 $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$, its $(i_1, i_2, \ldots, i_D)$th entry is $x_{i_1 i_2 \ldots i_D}$. Let $I_{D \backslash d} = \prod_{j=1, j \neq d}^D I_j$, the mode-$d$ matricizations $\boldsymbol{\mathcal{X}}_{\langle d \rangle}$ of $\boldsymbol{\mathcal{X}}$ is a $I_d \times I_{D \backslash d}$ matrix with $(\boldsymbol{\mathcal{X}}_{\langle d \rangle})_{i_d j} = x_{i_1 i_2 \cdots i_D}$, 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 $A^{\langle d \rangle}$ is a tensor $\boldsymbol{\mathcal{X}}$ with elements $x_{i_1 i_2 \cdots i_D} = a_{i_d j}$, and j is

as defined above. The inner product of two tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ is $\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \rangle = \sum_{i_1=1}^{I_1} \cdots \sum_{i_D=1}^{I_D} x_{i_1 i_2 \ldots i_D} y_{i_1 i_2 \ldots i_D}$, and the Frobenius norm of $\boldsymbol{\mathcal{X}}$ is $\|\boldsymbol{\mathcal{X}}\|_F = \sqrt{\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}} \rangle}$.

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 ∇f for its gradient.

2 RELATED WORK

2.1 Proximal Algorithms

Consider minimizing composite functions of the form:

$$F(x) \equiv f(x) + g(x), \tag{2}$$

where f, g are convex, and f is smooth but g is possibly nonsmooth. 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

$$z_t = x_t - \mu \nabla f(x_t), \tag{3}$$

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

$$\mu \leq 1/\rho \tag{4}$$

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

$$\begin{align} y_t &= (1 + \theta_t)x_t - \theta_t x_{t-1}, \tag{5} \\ z_t &= y_t - \mu \nabla f(y_t), \tag{6} \end{align}$$

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 ε_t in the proximal step, i.e.,

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

where

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

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 ε_t is upper-bounded by the duality gap ϑ_t. Thus, (7) can be ensured by monitoring ϑ_t. The following Proposition shows that by decreasing e_t and ε_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 Ω. The next estimate X_{t+1} is then generated by the singular value thresholding (SVT) 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, $SVT_\lambda(Z_t) \equiv U(\Sigma - \lambda I)_+ V^\top$ where $[(A)_+]_{ij} = \max(A_{ij}, 0)$.*

Let $\bar{k}_t$ be the number of singular values in Z_t that are larger than λ. 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 ℓ 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 ℓ is ρ-Lipschitz smooth, f in (14) is also ρ-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

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

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

Proposition 3.2 ([24]). *Let the SVD of matrix $\breve{Z}_t$ 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 $O(m^2 n)$ 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 $\breve{k}_t$ be the number of singular values in $\breve{Z}_t \in \mathbb{R}^{m \times n}$ larger than λ, and $Q \in \mathbb{R}^{m \times \breve{k}_t}$, where $k_t \geq \breve{k}_t$, be orthogonal and contains the subspace spanned by the top $\breve{k}_t$ left singular vectors of $\breve{Z}_t$. Then, $SVT_\lambda(\breve{Z}_t) = QSVT_\lambda(Q^\top \breve{Z}_t)$.*

Since a low-rank solution is desired, k_t can be much smaller than m [13]. Thus, once we identify the span of $\breve{Z}_t$'s top left singular vectors, we only need to perform SVT on the much smaller $Q^\top \breve{Z}_t \in \mathbb{R}^{k_t \times n}$ (instead of $\breve{Z} \in \mathbb{R}^{m \times n}$). 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($\breve{Z}_t, R_t, J$) [17]

Require: $\breve{Z}_t \in \mathbb{R}^{m \times n}$, $R_t \in \mathbb{R}^{n \times k_t}$, and the number of iterations J;
1: initialize $Q_0 = \text{QR}(\breve{Z}_t R_t)$; // QR($\cdot$) is QR factorization
2: **for** $j = 1, 2, \ldots, J$ **do**
3:    $Q_j = \text{QR}(\breve{Z}_t(\breve{Z}_t^\top Q_{j-1}))$;
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 $\breve{Z}_t$ with Q. In steps 2 to 5, a much smaller and less expensive (exact) SVT is performed on $Q^\top \breve{Z}_t$. Finally, $SVT_\lambda(\breve{Z}_t)$ is recovered as $\tilde{X} = (QU)\Sigma V^\top$ using Proposition 3.3.

Algorithm 2 Approximating the SVT of $\breve{Z}_t$: approx-SVT($\breve{Z}_t, R_t, \lambda, J$)

Require: $\breve{Z}_t \in \mathbb{R}^{m \times n}$, $R_t \in \mathbb{R}^{n \times k_t}$ and $\lambda \geq 0$;
1: $Q = \text{PowerMethod}(\breve{Z}_t, R_t, J)$;
2: $[U, \Sigma, V] = \text{SVD}(Q^\top \breve{Z}_t)$;
3: $U = \{u_i \mid \sigma_i > \lambda\}$;
4: $V = \{v_i \mid \sigma_i > \lambda\}$;
5: $\Sigma = (\Sigma - \lambda I)_+$;
6: **return** QU, Σ and V. // $\tilde{X} = (QU)\Sigma V$

3.4 The Proposed Algorithm

We extend problem (1) by allowing the loss ℓ to be ρ-Lipschitz smooth (e.g., logistic loss and squared hinge loss):

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

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

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

Using Proposition 3.1 and (4), the stepsize μ 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 QR($[V_t, V_{t-1}]$) at step 7. Moroever, as in [29], we restart the algorithm if $F(X)$ starts to increase (step 10). For further speedup, λ is dynamically reduced (step 3) by a continuation strategy [11], [13].

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

Require: partially observed matrix O, parameter λ.
1: initialize $c = 1$, $X_0 = X_1 = 0$, stepsize $\mu = 1/\rho$, $\hat{\lambda} > \lambda$ and $\nu \in (0,1)$;
2: **for** $t = 1, 2, \ldots, T$ **do**
3:   $\lambda_t = (\hat{\lambda} - \lambda)\nu^{t-1} + \lambda$;
4:   $Y_t = X_t + \theta_t(X_t - X_{t-1})$, where $\theta_t = \frac{c-1}{c+2}$;
5:   $\breve{Z}_t = Y_t - \mu S_t$, 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:   $[U_{t+1}, \Sigma_{t+1}, V_{t+1}] = \text{approx-SVT}(\breve{Z}_t, R_t, \mu\lambda_t, J)$;
   // $X_{t+1} = U_{t+1}\Sigma_{t+1}V_{t+1}^\top$
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}, Σ_{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 $SVT_{\mu\lambda}(\breve{Z}_t)$, there is a difference (ε_t in (7)) between the proximal objectives $h_{\mu\lambda\|\cdot\|_*}(X_{t+1}; \breve{Z}_t)$ and $h_{\mu\lambda\|\cdot\|_*}(SVT_{\mu\lambda}(\breve{Z}_t); \breve{Z}_t)$ after performing step 8, where $h_{\mu\lambda\|\cdot\|_*}(\cdot; \cdot)$ is as defined in (8). The following shows that ε_t decreases at a linear rate.

Proposition 3.4. *Assume that (i) $k_t \geq \breve{k}_t$ for all t and $J = t$;[1] (ii) $\{F(X_t)\}$ is upper-bounded. Then ε_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 $\breve{Z}_t u$ and $v^\top \breve{Z}_t$. The tricks in Section 3.2

---

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

can again be used for acceleration, and computing the approximate SVT using Algorithm 2 takes only

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

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

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

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

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

When ℓ is the square loss, (22) has a closed-form solution [13]. However, for smooth convex ℓ 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

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

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

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

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

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

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

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_τ. 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

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

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 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-ℓ_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

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

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

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, $(w_{\tau+1})_1 \leq (w_{\tau+1})_2 \leq \cdots \leq (w_{\tau+1})_m$ [6]. The following Lemma shows that the proximal step in (28) has a closed-form solution.

TABLE 2
Comparison of AIS-Impute (Algorithm 3) with other algorithms. 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.

	algorithm	iteration complexity	rate
matrix factorization	LMaFit [32]	$O(\|\Omega\|_1 k_t + mk_t)$	—
	ASD [33]	$O(\|\Omega\|_1 k_t + mk_t)$	—
	R1MP [34]	$O(\|\Omega\|_1 + mk_t^2)$	$O(c_1^T)$
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]	$O(\|\Omega\|_1 + mk_t^2)$	$O(1/T)$
	TR [30]	$O(\|\Omega\|_1 t^2 T_a)$	—
	ALT-Impute [36]	$O(\|\Omega\|_1 k_t + mk_t^2)$	$O(1/T)$
	SSGD [37]	$O(mk_t^2)$	$O(1/\sqrt{T})$
	AIS-Impute	$O((r_t + k_t)\|\Omega\|_1 + (r_{t-1} + r_t + k_t)mk_t)$	$O(1/T^2)$

Lemma 3.6 ([5], [6]). *Let the SVD of Z be $U\Sigma V^\top$ and $0 \leq w_1 \leq w_2 \leq \cdots \leq w_m$. 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 $U [\Sigma - \lambda Diag(w_1, \ldots, w_m)]_+ V^\top$.*

Similar to the truncated nuclear norm, we have $\breve{Z}_t = P_\Omega(O - Y_t) + Y_t$, 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 $\|\mathcal{X}\|_{overlap} = \sum_{d=1}^D \lambda_d \|\mathcal{X}_{\langle d \rangle}\|_*$, and the scaled latent nuclear norm is $\|\mathcal{X}\|_{scaled} = \min_{\mathcal{X}^1, \ldots, \mathcal{X}^D : \sum_{d=1}^D \mathcal{X}^d = \mathcal{X}} \sum_{d=1}^D \lambda_d \|\mathcal{X}_{\langle d \rangle}^d\|_*$. 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 $\mathcal{O} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$, with the observed entries indicated by $\Omega \in \{0, 1\}^{I_1 \times \cdots \times I_D}$. The tensor completion problem can be formulated as

$$\min_{\mathcal{X}^1, \ldots, \mathcal{X}^D} F([\mathcal{X}_t^1, \ldots, \mathcal{X}_t^D]) \tag{30}$$

$$\equiv \sum_{(i_1, \ldots, i_D) \in \Omega} \ell(\sum_{d=1}^D \mathcal{X}_{i_1 \ldots i_D}^d, \mathcal{O}_{i_1 \ldots i_D}) + \sum_{d=1}^D \lambda_d \|\mathcal{X}_{\langle d \rangle}^d\|_*.$$

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

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

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

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

$$\mathcal{Y}_t^d = (1 + \theta_t)\mathcal{X}_t^d - \theta_t \mathcal{X}_{t-1}^d,$$
$$\breve{\mathcal{Z}}_t^d = \mathcal{Y}_t^d - \mu\mathcal{S}_t = (1 + \theta_t)\mathcal{X}_t^d - \theta_t \mathcal{X}_{t-1}^d - \mu\mathcal{S}_t, \tag{33}$$

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

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

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

Proposition 4.1. $(\mathcal{X}_{t+1}^d)_{\langle d \rangle} = SVT_{\mu\lambda_d \|\cdot\|_*}((\breve{\mathcal{Z}}_t^d)_{\langle d \rangle})$.

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

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

Proposition 3.3 can be used to reduce the size of $(\breve{\mathcal{Z}}_t^d)_{\langle d \rangle}$ in Proposition 4.1, and Algorithm 1 can be used to approximate the underlying SVD. However, this is still not fast enough. Assume that $\breve{k}_t^d$ singular values in $(\breve{\mathcal{Z}}_t^d)_{\langle d \rangle}$ are larger than $\mu\lambda_d$, and rank-k_t^d SVD, where $k_t^d \geq \breve{k}_t^d$, is performed. SVT on $(\mathcal{Z}_t^d)_{\langle d \rangle}$ 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, $\breve{\mathcal{Z}}_t^d$ in (33) also has the "sparse plus low-rank" structure.

However, to generate $\mathfrak{X}_{t+1}^d$ using Proposition 4.1, we need to perform matrix multiplications of the form $(\check{\mathfrak{z}}_t^d)_{\langle d\rangle}v$, where $v \in \mathbb{R}^{I_{D\setminus d}}$, and $u^\top(\check{\mathfrak{z}}_t)_{\langle d\rangle}$, where $u \in \mathbb{R}^{I_d}$. Unfolding $\check{\mathfrak{z}}_t$ 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 $(\mathfrak{X}_{t+1}^d)_{\langle d\rangle}$, it can be seen from Proposition 4.1 and (33) that $\mathfrak{X}_t^d$ and $\mathfrak{X}_{t-1}^d$ only need to be unfolded along their dth modes. Hence, instead of storing them as tensors, we store $(\mathfrak{X}_t^d)_{\langle d\rangle}$ as its rank-r_t^d SVD $U_t^d\Sigma_t^d V_t^{d\top}$, and $(\mathfrak{X}_{t-1}^d)_{\langle d\rangle}$ as its rank-r_{t-1}^d SVD $U_{t-1}^d\Sigma_{t-1}^d V_{t-1}^{d\;\top}$. For any $v \in \mathbb{R}^{I_{D\setminus d}}$,

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

The first two terms can be computed in $O((I_d+I_{D\setminus d})(r_t^d+r_{t-1}^d))$ time. As $\mathfrak{S}_t$ is sparse, computing the last term takes $O(\|\Omega\|_1)$ time. Thus, $(\check{\mathfrak{z}}_t^d)_{\langle d\rangle}v$ can be obtained in $O(\|\Omega\|_1 + (I_d+I_{D\setminus d})(r_t^d+r_{t-1}^d))$ time. Similarly, for any $u \in \mathbb{R}^{I_d}$, $u^\top(\check{\mathfrak{z}}_t)_{\langle d\rangle}$ 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 $(\check{\mathfrak{z}}_t^d)_{\langle d\rangle}$, 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 $\text{prox}_{\mu g}([\check{\mathfrak{z}}_t^1,\dots,\check{\mathfrak{z}}_t^D])$ takes a total of

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

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) $F([\mathfrak{X}_t^1,\dots,\mathfrak{X}_t^D])$ is upper bounded. The sequence $\{[\mathfrak{X}_t^1,\dots,\mathfrak{X}_t^D]\}$ 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 $\mathfrak{X} = \sum_{d=1}^D \mathfrak{X}^d$, where $\mathfrak{X}_{\langle d\rangle}^d = U^d\Sigma^d(V^d)^\top$ has rank k^d. Define $\mathcal{A}(\theta^1,\dots,\theta^D) = \sum_{d=1}^D (U^d\text{Diag}(\theta^d)(V^d)^\top)_{\langle d\rangle}$. As in (22), we replace $\mathfrak{X}$ with $\mathcal{A}(\theta_*^1,\dots,\theta_*^D)$, where

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

and

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

Algorithm 4 AIS-Impute (tensor case).

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

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 $w^d = [w_i^d] \in k^d$, $w_i^d = (u_i^d)^\top\mathfrak{B}_{\langle d\rangle}v_i^d$, and $\mathfrak{B}_{i_1\dots i_D} = \frac{d\ell(\mathcal{A}([\theta^1,\dots,\theta^D])_{i_1\dots i_D}, \mathcal{O}_{i_1\dots i_D})}{d\mathcal{A}([\theta^1,\dots,\theta^D])_{i_1\dots i_D}}$ 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.

5 EXPERIMENTS

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

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 λ; (ii) Soft-Impute [13]; (iii) AIS-Impute (the proposed Algorithm 3); and (iv) AIS-Impute (exact): This

TABLE 3
Matrix completion results on the synthetic data. Here, sparsity is the proportion of observed entries, and post-processing time is in seconds.

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

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 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[2] 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 vs time and iterations are shown in Figure 1. In terms of the number 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 (Figure 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% of the observed ratings for training, 25% for validation and the rest for testing.

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

We compare AIS-Impute with the two most popular low-rank matrix learning approaches [1], [9], namely,

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

(a) $m = 250$.

(b) $m = 1000$.

(c) $m = 4000$.

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

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

TABLE 5
Results on the *MovieLens* data sets. Note that TR and APG cannot converge in 10^4 seconds on the *10M* data set.

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

(a) *100K*. (b) *1M*. (c) *10M*.

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

and per-iteration time complexities; (iv) second-order trust-region algorithm ("TR") [30], which alternates fixed-rank optimization and rank-one updates; (vi) stochastic gradient descent ("SSGD") [37], which is a stochastic gradient descent algorithm; and (v) matrix completion based on fast alternating least squares ("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: 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. Figure 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% of the ratings matrix are observed. We randomly sample 50% 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 λ are experimented.

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

TABLE 6
Results on the *Netflix* data set. The regularization parameter λ in (1) is set as λ_0/c, where $\lambda_0 = \|P_\Omega(O)\|_F$. Soft-Impute with $c = 30$ is not run as it is very slow.

		RMSE	rank
$c = 10$	active	0.894±0.001	3
	ALT-Impute	0.900±0.006	3
	Soft-Impute	**0.893±0.001**	3
	AIS-Impute	**0.893±0.001**	3
$c = 20$	active	**0.847±0.001**	14
	ALT-Impute	0.850±0.001	14
	Soft-Impute	**0.847±0.001**	14
	AIS-Impute	**0.847±0.001**	14
$c = 30$	active	**0.820±0.001**	116
	ALT-Impute	0.825±0.001	116
	AIS-Impute	**0.820±0.001**	116

5.3 Grayscale Images

In this section, we perform experiments on images from [18] (Figures 4(a)-4(c)). 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% 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% 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

(a) $\lambda = \lambda_0/10$. (b) $\lambda = \lambda_0/20$. (c) $\lambda = \lambda_0/30$.

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

(a) *rice* (854×960).   (b) *tree* (800×800).   (c) *wall* (841×850).

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

boost are the best), though they require the use of higher ranks. Figure 5 shows the running time. As can be seen, AIS-Impute is consistently the fastest.

Figure 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 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-ℓ_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 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-ℓ_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. As can be seen, the errors obtained by nonconvex regularization (i.e., TNN, capped-ℓ_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 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% of the ratings for training, 10% 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 Figure 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 $\mathcal{C} \in \mathbb{R}^{3 \times 3}$ are all sampled i.i.d. from the standard normal distribution $\mathcal{N}(0, 1)$, and $\times_k$ is the k-mode product[3]. 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 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

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

(a) *rice.* (b) *tree.* (c) *wall.*

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

TABLE 7
Matrix completion results on grayscale images. CPU time is in seconds.

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

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

TABLE 8
Comparison of nuclear norm regularization with various nonconvex regularizations.

		NMSE		RMSE
		synthetic ($m = 250$)	synthetic ($m = 1000$)	*MovieLens-100K*
nuclear norm	AIS-Impute	0.0098±0.0004	0.0092±0.0002	0.883±0.005
TNN	TNNR(APG)	**0.0081±0.0004**	**0.0073±0.0001**	**0.851±0.002**
	IRNN	**0.0081±0.0004**	**0.0073±0.0001**	0.853±0.004
	DC(AIS-Impute)	**0.0081±0.0004**	**0.0073±0.0002**	**0.851±0.002**
capped-ℓ_1	IRNN	0.0089±0.0005	**0.0074±0.0001**	0.853±0.002
	DC(AIS-Impute)	**0.0081±0.0004**	**0.0073±0.0002**	0.852±0.005
LSP	IRNN	0.0083±0.0004	0.0076±0.0001	0.852±0.006
	DC(AIS-Impute)	**0.0081±0.0004**	**0.0073±0.0002**	**0.850±0.002**

TABLE 9
Data sets for link prediction.

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

TABLE 10
Performance on link prediction.

		accuracy	rank
Epinions	active	**0.939±0.002**	12
	AltMin	0.936±0.002	41
	AIS-Impute	**0.940±0.001**	12
Slashdot	active	**0.844±0.001**	16
	AltMin	0.839±0.002	39
	AIS-Impute	**0.843±0.001**	16

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 λ, 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 vs time and iterations are shown in Figure 9. In terms of iterations, APG, AIS-Impute(exact) and AIS-Impute

(a) TNN.   (b) capped-ℓ_1.   (c) LSP.

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

TABLE 11
Tensor completion results on the synthetic data. Here, sparsity is the proportion of observed entries, and post-processing time is in seconds.

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

(a) *Epinions*.   (b) *Slashdot*.

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

(a) $m = 125$.

(b) $m = 500$.

(c) $m = 2000$.

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

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.

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 $15088 \times 15088 \times 5$ tensor, with a total of $27,257,790$ nonzero 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 1000 users (leading to $12,101$ observations). We use 50% of the observations for training, another 25% 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 RMSE $= \sqrt{\|P_{\hat{\Omega}}(\mathcal{X} - \hat{\mathcal{O}})\|_F^2 / \|\hat{\Omega}\|_1}$; and (ii) rank of the unfolded matrix in each mode. The experiments are repeated five times.

Performance is shown in Table 12 and Figure 10(a) 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.

TABLE 12
Results on the *YouTube* subset. The rank is for each mode.

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

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 Figure 10(b) 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 worse dependency on the tensor rank than AIS-Impute ($\prod_{i=1}^{D} r_t^d$ vs $\sum_{i=1}^{D} r_t^d$), and thus GeomCG becomes very slow when the tensor rank is large. Overall, AIS-Impute has fast speed and good recovery performance.

TABLE 13
Results on the full *YouTube* dataset. The rank is for each mode.

	RMSE	rank
GeomCG	0.388±0.001	51, 51, 5
TMac	0.611±0.007	10, 10, 0
AIS-Impute	**0.369±0.006**	70, 70, 0

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

(a) data subset.   (b) full data set.

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

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.

REFERENCES

[1] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 426–434.

[2] M. Kim and J. Leskovec, "The network completion problem: inferring missing nodes and edges in networks," in *Proceedings of the 11st International Conference on 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," *Journal of Machine Learning Research*, 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 Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 208–220, 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," *International Journal of Computer Vision*, 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 Transactions on Image Processing*, vol. 25, no. 2, pp. 829–839, 2016.

[7] Z. Zhao, L. Zhang, X. He, and W. Ng, "Expert finding for question answering via graph regularized matrix completion," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 993–1004, 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 Networks*, vol. 100, pp. 39–48, 2018.

[9] E. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, 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 Journal on 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 Journal of Optimization*, vol. 6, no. 615-640, p. 15, 2010.

[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," *Journal of Machine Learning Research*, vol. 11, pp. 2287–2322, 2010.

[14] R. Tibshirani, "Proximal gradient descent and acceleration," Lecture Notes, 2010, 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 Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[16] S. Ji and J. Ye, "An accelerated gradient method for trace norm minimization," in *Proceedings of the 26th International Conference on Machine Learning*, 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 Review*, 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 Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 9, pp. 2117–2130, 2013.

[19] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[20] R. Tomioka, K. Hayashi, and H. Kashima, "Estimation of low-rank tensors via convex optimization," Department of Mathematical Informatics, University of Tokyo, Tech. Rep. arXiv:1010.0789, 2010.

[21] E. Acar, D. Dunlavy, T. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, 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 *Advances in Neural Information Processing Systems*, 2014, pp. 2825–2833.

[23] Q. Yao and J. T. Kwok, "Accelerated inexact soft-impute for fast large-scale matrix completion," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015, pp. 4002–4008.

[24] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in 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 *Proceedings of the 26th International Conference on Machine Learning*, 2009, pp. 433–440.

[26] M. Schmidt, N. Roux, and F. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 1458–1466.

[27] K. Wu and H. Simon, "Thick-restart Lanczos method for large symmetric eigenvalue problems," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 2, pp. 602–616, 2000.

[28] C.-J. Hsieh and P. Olsen, "Nuclear norm minimization via active subspace selection," in *Proceedings of the 31st International Conference on Machine Learning*, 2014, pp. 575–583.

[29] B. O'Donoghue and E. Candès, "Adaptive restart for accelerated gradient schemes," *Foundations of Computational Mathematics*, pp. 1–18, 2012.

[30] B. Mishra, G. Meyer, F. Bach, and R. Sepulchre, "Low-rank optimization with trace norm penalty," *SIAM Journal on 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 *Advances in Neural Information Processing Systems*, 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," *Mathematical Programming Computation*, vol. 4, no. 4, pp. 333–361, 2012.

[33] J. Tanner and K. Wei, "Low rank matrix completion by alternating steepest descent methods," *Applied and Computational Harmonic Analysis*, 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 Journal on Scientific Computing*, 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 *Artificial Intelligence and Statistics*, 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," *Journal of Machine Learning Research*, vol. 16, pp. 3367–3402, 2015.

[37] H. Avron, S. Kale, V. Sindhwani, and S. Kasiviswanathan, "Efficient and practical stochastic subgradient descent for nuclear norm regularization," in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 1231–1238.

[38] J. Nocedal and S. Wright, *Numerical optimization*. 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," *Annals of operations research*, vol. 133, no. 1-4, pp. 23–46, 2005.

[40] E. Candès, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 877–905, 2008.

[41] C. Zhang, "Nearly unbiased variable selection under minimax concave penalty," *sAnnals of statistics*, vol. 38, no. 2, pp. 894–942, 2010.

[42] T. Zhang, "Analysis of multi-stage convex relaxation for sparse regularization," *Journal of Machine Learning Research*, vol. 11, pp. 1081–1107, 2010.

[43] S. Boyd and L. Vandenberghe, *Convex optimization*. 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," *Foundations and Trends in Machine Learning*, pp. 1–122, 2011.

[45] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Research 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 *IEEE International Conference on Data Mining*, 2009, pp. 503–512.

[47] D. Kressner, M. Steinlechner, and B. Vandereycken, "Low-rank tensor completion by Riemannian optimization," *BIT Numerical Mathematics*, 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 & Imaging*, vol. 9, no. 2, 2013.

[49] G. Golub and C. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 2012.

Quanming Yao obtained his Ph.D from Computer Science and Engineer Department of Hong Kong University of Science and Technology (HKUST) in 2018, and bachelor degree in Electronic and Information Engineering from the Huazhong University of Science and Technology (HUST) in 2013. 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, Tse Cheuk Ng Tai research excellence prize from HKUST in 2015 and Google PhD fellowship (machine learning) in 2016.

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 in 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 IEEE Outstanding 2004 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 Neurocomputing journal.

APPENDIX A
PROOFS

A.1 Proposition 3.1

Proof. For any $X, Y \in \mathbb{R}^{m \times n}$,

$$\|\nabla f(X) - \nabla f(Y)\|_F^2$$
$$= \sum_{(i,j) \in \Omega} \left[\frac{d\ell(X_{ij}, O_{ij})}{dX_{ij}} - \frac{d\ell(Y_{ij}, O_{ij})}{dY_{ij}} \right]^2$$
$$\leq \sum_{(i,j) \in \Omega} \rho^2 (X_{ij} - Y_{ij})^2 \qquad (37)$$
$$\leq \rho^2 \|X - Y\|_F^2,$$

where (37) follows from the fact that ℓ is ρ-Lipschitz smooth. Thus, $f(X)$ is ρ-Lipschitz smooth. $\qquad \square$

A.2 Proposition 3.3

Proof. First, we introduce the following theorem.

Theorem A.1 (Separation theorem [49]). *Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{m \times r}$ with $B^\top B = I$. Then*

$$\sigma_i \left(B^\top A \right) \leq \sigma_i(A), \text{ for } i = 1, \dots, \min(r, n).$$

Let the SVD of Z be $U\Sigma V^\top$. Z can then be rewritten as

$$Z = \left[U_{\breve{k}_t}; U_\perp \right] \begin{bmatrix} \Sigma_{\breve{k}_t} & \\ & \Sigma_\perp \end{bmatrix} \left[V_{\breve{k}_t}; V_\perp \right]^\top, \qquad (38)$$

where $U_{\breve{k}_t}$ contains the $\breve{k}_t$ leading columns of U, and $U_\perp$ the remaining columns. Similarly, $\Sigma_{\breve{k}_t}$ (resp. $V_{\breve{k}_t}$) contains the $\breve{k}_t$ leading eigenvalues (resp. columns) of Σ (resp. V). Then, let

$$\tilde{u}_i = Q^\top u_i \quad \text{and} \quad \tilde{v}_i = v_i, \qquad (39)$$

where u_i (resp. v_i) is the ith column of U (resp. V). For $i = 1, \cdots, \breve{k}_t$, we have

$$\tilde{u}_i^\top \left(Q^\top Z \right) \tilde{v}_i = U_i^\top \left(QQ^\top \right) ZV_i$$
$$= U_i^\top ZV_i \qquad (40)$$
$$= \sigma_i(Z), \qquad (41)$$

where (40) is due to $\text{span}(U_{\breve{k}_t}) \subseteq \text{span}(Q)$. Hence,

$$\sigma_i \left(Q^\top Z \right) = \sigma_i(Z), \text{ for } i = 1, \cdots, \breve{k}_t \qquad (42)$$

From Theorem A.1, by substituting $Q = B$ and $A = Z$, we have $\sigma_i(Q^\top Z) \leq \sigma_i(Z)$. Combining with (42), we obtain that the rank-$\breve{k}_t$ SVD of $Q^\top Z$ is $(Q^\top U_{\breve{k}_t})\Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top$, with the corresponding left and right singular vectors contained in $Q^\top U_{\breve{k}_t}$ and $V_{\breve{k}_t}$, respectively.

Again, by Theorem A.1, we have

$$\sigma_{\breve{k}_t+1} \left(Q^\top Z \right) \leq \sigma_{\breve{k}_t+1}(Z) \leq \mu.$$

Besides, using (38),

$$\sigma_i \left(Q^\top Z \right) = \max_{u,v} u \left(Q^\top U_{\breve{k}_t} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top + Q^\top U_\perp \Sigma_\perp V_\perp^\top \right) v.$$

Since the first $\breve{k}_t$ singular values are from the term $Q^\top U_{\breve{k}_t} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top$, then

$$\sigma_{\breve{k}_t+1} \left(Q^\top Z \right) = \max_{u,v} u^\top \left(Q^\top U_\perp \Sigma_\perp V_\perp^\top \right) v \leq \mu. \qquad (43)$$

Then,

$$\text{SVT}_{\mu r}(Q^\top Z)$$
$$= \text{SVT}_{\mu r} \left(Q^\top U_{\breve{k}_t} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top + Q^\top U_\perp \Sigma_\perp V_\perp^\top \right)$$
$$= \text{SVT}_{\mu r} \left(Q^\top U_{\breve{k}_t} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top \right) + \text{SVT}_{\mu r} \left(Q^\top U_\perp \Sigma_\perp V_\perp^\top \right) \quad (44)$$
$$= \text{SVT}_{\mu r} \left(Q^\top U_{\breve{k}_t} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top \right). \qquad (45)$$

where (44) follows from that $Q^\top U_{\breve{k}_t}$ (resp. $V_{\breve{k}_t}$) is orthogonal to $QU_\perp$ (resp. $V_\perp$). (43) shows that there are only $\breve{k}_t$ singular values in $Q^\top Z$ larger than μ. Thus, $\text{SVT}_{\mu r} Q^\top U_\perp \Sigma_\perp V_\perp^\top = 0$ and we get (45). Finally,

$$Q\text{SVT}_{\mu r} \left(Q^\top Z \right) = Q \left(Q^\top U_{\breve{k}_t} \text{SVT}_{\mu r} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top \right)$$
$$= U_{\breve{k}_t} \text{SVT}_{\mu r} \left(\Sigma_{\breve{k}_t} \right) V_{\breve{k}_t}^\top \qquad (46)$$
$$= \text{SVT}_{\mu r}(Z), \qquad (47)$$

where (46) comes from $\text{span}(U_{\breve{k}_t}) \subseteq \text{span}(Q)$; (47) comes from that rank-$\breve{k}_t$ SVD of Z is $U_{\breve{k}_t} \Sigma_{\breve{k}_t} V_{\breve{k}_t}^\top$ and Z only has $\breve{k}_t$ singular values larger than μ. $\qquad \square$

A.3 Proposition 3.4

Before proof of Proposition 3.4, we first introduce some Lemmas (Lemma A.2, A.4, A.3 and A.7) and Propositions (Proposition A.5 and A.6).

Lemma A.2 ([10]). *For any matrices A and B, $\|SVT_\lambda(A) - SVT_\lambda(B)\|_F \leq \|A - B\|_F$.*

Let $Z_t^* \equiv \text{SVT}_{\mu\lambda}(\breve{Z}_t)$, $\beta_t \equiv \|\breve{Z}_t\|_F$ and $\eta_t = \frac{\sigma_{k+1}(\breve{Z}_t)}{\sigma_k(\breve{Z}_t)}$.

Lemma A.3 ([17]). *Let the input to Algorithm 1 be $\breve{Z}_t$, and its top k left singular vectors be contained in U_k. Then, for $j = 0, 1, 2, \dots$,*

$$\|Q_j Q_j^\top - U_k U_k^\top\|_F \leq \eta_t^j \alpha_t,$$

where $\alpha_t = \|Q_0 Q_0^\top - U_k U_k^\top\|_F$ and Q_0 is the span of $\breve{Z}_t R_t$.

Lemma A.4. *For output $\tilde{X} = (QU)\Sigma V^\top$ from Algorithm 2, we have $\|\tilde{X} - Z_t^*\|_F \leq \|U_k U_k^\top - QQ^\top\|_F \beta_t$.*

Proof. From Proposition 3.3,

$$Z_t^* - \tilde{X} = \text{SVT}_{\mu\lambda}(\breve{Z}_t) - Q\text{SVT}_{\mu\lambda}(Q^\top \breve{Z}_t)$$
$$= \text{SVT}_{\mu\lambda}(U_k U_k^\top \breve{Z}_t) - \text{SVT}_{\mu\lambda}(QQ^\top \breve{Z}_t).$$

Using Lemma A.2 and the Cauchy's inequality,

$$\|\tilde{X} - Z_*\|_F = \|\text{SVT}_{\mu\lambda}(U_k U_k^\top \breve{Z}_t) - \text{SVT}_{\mu\lambda}(QQ^\top \breve{Z}_t)\|_F$$
$$\leq \|(U_k U_k^\top - QQ^\top)\breve{Z}_t\|_F$$
$$\leq \|U_k U_k^\top - QQ^\top\|_F \beta_t,$$

and result follows. $\qquad \square$

Proposition A.5. *Let $G_t \in \partial h_{\mu\lambda\|\cdot\|_*}(\tilde{X}; \breve{Z}_t)$, then $\|G_t\|_F$ is upper-bounded by a constant γ_t.*

Proof. Let the reduced SVD of $\tilde{X}$ be $U\Sigma V^\top$ (only positive singular values are contained). By the definition of subgradient of the nuclear norm [9],

$$\partial h_{\mu\lambda\|\cdot\|_*}(\tilde{X}; \breve{Z}_t) = \tilde{X} - \breve{Z}_t + \mu\lambda(UV^\top + W),$$

where

$$W^\top U = 0, \; WV = 0, \; \text{and} \; \|W\|_\infty \le 1. \tag{48}$$

Thus,

$$\begin{aligned}
\|G_t\|_F &= \|\tilde{X} - \check{Z}_t + \mu\lambda(UV^\top + W)\|_F \\
&\le \|\tilde{X} - \check{Z}_t\|_F + \mu\lambda\|UV^\top + W\|_F. \tag{49}
\end{aligned}$$

For the first term in (49),

$$\begin{aligned}
&\|\tilde{X} - \check{Z}_t\|_F \\
&= \|\tilde{X} - Z_t^* + Z_t^* - \check{Z}_t\|_F \\
&\le \|\tilde{X} - Z_t^*\|_F + \|Z_t^* - \check{Z}_t\|_F \\
&= \|Z_t^* - \check{Z}_t\|_F + \|Z_t^* - Q\mathrm{SVT}_{\mu\lambda}(Q^\top\check{Z}_t)\|_F \\
&= \|Z_t^* - \check{Z}_t\|_F + \|Z_t^* - \tilde{X}\|_F \\
&\le \|Z_t^* - \check{Z}_t\|_F + \|U_k U_k^\top - QQ^\top\|_F \beta_t \tag{50} \\
&\le \|Z_t^* - \check{Z}_t\|_F + \alpha_t\beta_t. \tag{51}
\end{aligned}$$

Here, (50) follows from Lemma A.4, and (51) from Lemma A.3. As $\|W\|_\infty \le 1$ from (48), thus

$$\|W\|_F = \sqrt{\sum_{i=1}^m \sigma_i^2(W)} \le \sqrt{m}.$$

For the second term in (49), then

$$\begin{aligned}
\|UV^\top + W\|_F &\le \sqrt{\mathrm{tr}(U^\top U V^\top V)} + \|W\|_F \\
&\le \sqrt{k_t} + \sqrt{m} \le 2\sqrt{m}. \tag{52}
\end{aligned}$$

Combining (51) and (52), by Lemma A.3:

$$\|G_t\|_F \le 2\mu\lambda\sqrt{m} + \|Z_t^* - \check{Z}_t\|_F + \alpha_t\beta_t. \tag{53}$$

Since Z_t^* is independent of $\tilde{X}$, $\|Z_t^* - \check{Z}_t\|_F$ is a constant. Hence, $\|G_t\|_F$ is upper bounded by

$$\gamma_t = 2\mu\lambda\sqrt{m} + \|Z_t^* - \check{Z}_t\|_F + \alpha_t\beta_t,$$

which a constant. $\qquad\square$

Proposition A.6. *Assume that $k_t \ge \check{k}_t$. Let $h_{\mu\lambda\|\cdot\|_*}(\tilde{X}; \check{Z}_t)$ be as defined in (8). Then, for Algorithm 2, we have*

$$h_{\mu\lambda\|\cdot\|_*}(\tilde{X}; \check{Z}_t) \le h_{\mu\lambda\|\cdot\|_*}(Z_t^*; \check{Z}_t) + \alpha_t\beta_t\gamma_t\eta_t^J.$$

Proof. As h is convex,

$$h_{\mu\lambda\|\cdot\|_*}(\tilde{X}; \check{Z}_t) \le h_{\mu\lambda\|\cdot\|_*}(Z_t^*; \check{Z}_t) + \mathrm{tr}((\tilde{X} - Z_t^*)^\top G_t) \tag{54}$$

where $G_t \in \partial h_{\mu\lambda\|\cdot\|_*}(\tilde{X}; \check{Z}_t)$. Next, we bound the second term on the r.h.s. of (54).

$$\begin{aligned}
\mathrm{tr}((\tilde{X} - Z_t^*)^\top G_t) &\le \|\tilde{X} - Z_t^*\|_F \|G_t\|_F \\
&\le \gamma_t \|\tilde{X} - Z_t^*\|_F \tag{55} \\
&\le \gamma_t\beta_t \|QQ^\top - U_k U_k^\top\|_F \tag{56} \\
&\le \eta_t^J(\alpha_t\beta_t\gamma_t). \tag{57}
\end{aligned}$$

Here, (55) follows from Proposition A.5; (56) from Lemma A.4; and (57) from Lemma A.3. Result follows on combining (54) and (57). $\qquad\square$

Lemma A.7. *If $\{F(X_t)\}$ is upper-bounded where F is the objective at (19), then $\|X_t\|_F$ from Algorithm 3 is upper-bounded.*

Proof. As $\{F(X_t)\}$ is upper bounded and note that

$$F(X) \to +\infty \Leftrightarrow \|X\|_F \to +\infty.$$

for (19), then $\{\|X_t\|_F\}$ is also upper bounded. $\qquad\square$

Now, we are ready to prove Proposition 3.4. As α_t, β_t and γ_t only depend on X_t, from Lemma A.7, they are all upper bounded. Let $q = \sup_t \alpha_t\beta_t\gamma_t$, and $q < \infty$ is a constant. Then by Proposition A.6, and note that Algorithm 2 is run for t iterations at tth loop of Algorithm 3. Let $\eta = \max_t \eta_t \in (0,1)$, we have

$$h_{\mu\lambda\|\cdot\|_*}(X_{t+1}; \check{Z}_t) \le h_{\mu\lambda\|\cdot\|_*}(Z_t^*; \check{Z}_t) + \varepsilon_t.$$

Hence, $\varepsilon_t = q\eta^t$ decays at a linear rate. $\qquad\square$

A.4 Theorem 3.5

Proof. From Proposition 3.4, ε_t decays at a linear rate. Moreover, there is no error on the computation of gradient. Thus, conditions in Proposition 2.1 are satisfied, and Algorithm 3 converges with a rate of $O(1/T^2)$. $\qquad\square$

A.5 Proposition 4.1

Proof. Note that

$$\min_{\mathcal{X}^1,\dots,\mathcal{X}^D} \frac{1}{2}\|[\mathcal{X}^1,\dots,\mathcal{X}^D] - [\check{\mathcal{Z}}_t^1,\dots,\check{\mathcal{Z}}_t^D]\|_F^2 + \mu\sum_{d=1}^D \lambda_d\|\mathcal{X}_{\langle d\rangle}^d\|_*$$

$$= \sum_{d=1}^D \min_{\mathcal{X}^d} \frac{1}{2}\|\mathcal{X}^d - \check{\mathcal{Z}}_t^d\|_F^2 + \mu\lambda_d\|\mathcal{X}_{\langle d\rangle}^d\|_*,$$

$$= \sum_{d=1}^D \min_{\mathcal{X}^d} \frac{1}{2}\|\mathcal{X}_{\langle d\rangle}^d - (\check{\mathcal{Z}}_t^d)_{\langle d\rangle}\|_F^2 + \mu\lambda_d\|\mathcal{X}_{\langle d\rangle}^d\|_*. \tag{58}$$

The $\mathcal{X}^d$'s in (58) are independent of each other, and

$$(\mathrm{SVT}_{\mu\lambda_d}(\check{\mathcal{Z}}_{\langle d\rangle}^d))_{\langle d\rangle} = \arg\min_{\mathcal{X}^d} \frac{1}{2}\|\mathcal{X}_{\langle d\rangle}^d - \check{\mathcal{Z}}_{\langle d\rangle}^d\|_F^2 + \mu\lambda_d\|\mathcal{X}_{\langle d\rangle}^d\|_*.$$

and thus result follows. $\qquad\square$

A.6 Proposition 4.2

Proof. For any $\mathcal{X}^1,\dots,\mathcal{X}^D$, $\mathcal{Y}^1,\cdots,\mathcal{Y}^D$, and let $\tilde{\mathcal{X}} = \sum_{d=1}^D \mathcal{X}^d$ and $\tilde{\mathcal{Y}} = \sum_{d=1}^D \mathcal{Y}^d$.

$$\|\nabla f([\mathcal{X}^1,\dots,\mathcal{X}^D]) - \nabla f([\mathcal{Y}^1,\cdots,\mathcal{Y}^D])\|_F^2$$

$$= \sum_{(i_1,\dots,i_D)\in\Omega} \left[\frac{d\ell(\tilde{\mathcal{X}}_{i_1\dots i_D}, \mathcal{O}_{i_1\dots i_D})}{d\tilde{\mathcal{X}}_{i_1\dots i_D}} - \frac{d\ell(\tilde{\mathcal{Y}}_{i_1\dots i_D}, \mathcal{O}_{i_1\dots i_D})}{d\tilde{\mathcal{Y}}_{i_1\dots i_D}}\right]^2$$

$$\le \sum_{(i_1,\dots,i_D)\in\Omega} \rho^2\left(\tilde{\mathcal{X}}_{i_1\dots i_D} - \tilde{\mathcal{Y}}_{i_1\dots i_D}\right)^2 \le \rho^2\left\|\tilde{\mathcal{X}} - \tilde{\mathcal{Y}}\right\|_F^2,$$

where the first inequality comes from the ρ-Lipschitz smoothness of ℓ. Note that

$$\begin{aligned}
\left\|\tilde{\mathcal{X}} - \tilde{\mathcal{Y}}\right\|_F^2 &\le D\sum_{d=1}^D \|\mathcal{X}^d - \mathcal{Y}^d\|_F^2 \\
&= D\|[\mathcal{X}^1,\dots,\mathcal{X}^D] - [\mathcal{Y}^1,\dots,\mathcal{Y}^D]\|_F^2.
\end{aligned}$$

We have

$$\begin{aligned}
&\|\nabla f([\mathcal{X}^1,\dots,\mathcal{X}^D]) - \nabla f([\mathcal{Y}^1,\dots,\mathcal{Y}^D])\|_F \\
&\le \sqrt{D}\rho\|[\mathcal{X}^1,\dots,\mathcal{X}^D] - [\mathcal{Y}^1,\dots,\mathcal{Y}^D]\|_F,
\end{aligned}$$

and thus f is $\sqrt{D}\rho$-Lipschitz smooth. $\qquad\square$

A.7 Theorem 4.3

Proof. From the definition of h in (8),

$$
h_{\mu g}\left([\mathcal{X}_{t+1}^1,\ldots,\mathcal{X}_{t+1}^D];[\check{\mathcal{Z}}_t^1,\ldots,\check{\mathcal{Z}}_t^D]\right)
$$

$$
= \sum_{d=1}^D \frac{1}{2}\left\|(\mathcal{X}_{t+1}^d)_{\langle d\rangle} - (\check{\mathcal{Z}}_t)_{\langle d\rangle}^d\right\|_F^2 + \mu\lambda_d\|(\mathcal{X}_{t+1}^d)_{\langle d\rangle}\|_*,
$$

$$
= \sum_{d=1}^D h_{\mu\lambda_d\|\cdot\|_*}\left((\mathcal{X}_{t+1}^d)_{\langle d\rangle};(\check{\mathcal{Z}}_t)_{\langle d\rangle}^d\right). \tag{59}
$$

As proximal step is inexact in Algorithm 4, using Proposition A.6 on (59),

$$
h_{\mu\lambda_d\|\cdot\|_*}\left((\mathcal{X}_{t+1}^d)_{\langle d\rangle};(\check{\mathcal{Z}}_t)_{\langle d\rangle}^d\right)
$$

$$
\leq h_{\mu\lambda_d\|\cdot\|_*}\left((\mathcal{W}_*^d)_{\langle d\rangle};(\check{\mathcal{Z}}_t)_{\langle d\rangle}^d\right) + (\alpha_d)_t(\beta_d)_t(\gamma_d)_t(\eta_d)_t^J,
$$

where $(\mathcal{W}_*^d)_{\langle d\rangle} = \text{SVT}_{\mu\lambda_d\|\cdot\|_*}\left((\check{\mathcal{Z}}_t)_{\langle d\rangle}^d\right)$, and α_d, β_d, γ_d, η_d are constants depending on $(\check{\mathcal{Z}}_t)_{\langle d\rangle}^d$. Let $(c_d)_t = (\alpha_d)_t(\beta_d)_t(\gamma_d)_t$. As $J = t$,

$$
h_{\mu g}\left([\mathcal{X}_{t+1}^1,\cdots,\mathcal{X}_{t+1}^D];[\check{\mathcal{Z}}_t^1,\cdots,\check{\mathcal{Z}}_t^D]\right) \tag{60}
$$

$$
\leq h_{\mu g}\left([\mathcal{W}_*^1,\cdots,\mathcal{W}_*^D];[\check{\mathcal{Z}}_t^1,\cdots,\check{\mathcal{Z}}_t^D]\right) + \sum_{d=1}^D (c_d)_t(\eta_d)_t^t.
$$

As $F([\mathcal{X}_t^1,\ldots,\mathcal{X}_t^D])$ is upper-bounded and

$$
\lim_{\|\mathcal{X}^d\|_F\to\infty} F([\mathcal{X}_t^1,\ldots,\mathcal{X}_t^D]) = \infty,
$$

for any $d = 1,\ldots,D$. Then, $\|\mathcal{X}_t^d\|_F$ for $d = 1,\ldots,D$ are also upper-bounded. Thus,

$$
q = \sup_t \sum_{d=1}^D (c_d)_t < \infty.
$$

Let $\eta = \max_{d,t}((\eta_d)_t) < 1$. Together with (60), we have

$$
h_{\mu g}\left([\mathcal{X}_{t+1}^1,\ldots,\mathcal{X}_{t+1}^D];[\check{\mathcal{Z}}_t^1,\ldots,\check{\mathcal{Z}}_t^D]\right)
$$

$$
\leq h_{\mu g}\left([\mathcal{W}_*^1,\ldots,\mathcal{W}_*^D];[\check{\mathcal{Z}}_t^1,\ldots,\check{\mathcal{Z}}_t^D]\right) + \varepsilon_t,
$$

and the approximation error $\varepsilon_t = q\eta^t$ decays at a linear rate. Moreover, there is no error on the computation of gradient. Thus, the conditions in Proposition 2.1 are satisfied, and Algorithm 4 converges with a rate of $O(1/T^2)$. $\square$