Accelerated Inexact Soft-Impute for Fast Large-Scale Matrix Completion

Quanming Yao  James T. Kwok
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
\{qyaoaa, jamesk\}@cse.ust.hk

Abstract
Matrix factorization tries to recover a low-rank matrix from limited observations. A state-of-the-art algorithm is the Soft-Impute, which exploits a special “sparse plus low-rank” structure of the matrix to allow efficient SVD in each iteration. Though Soft-Impute is also a proximal gradient algorithm, it is generally believed that acceleration techniques are not useful and will destroy the special structure. In this paper, we show that Soft-Impute can indeed be accelerated without compromising the “sparse plus low-rank” structure. To further reduce the per-iteration time complexity, we propose an approximate singular value thresholding scheme based on the power method. Theoretical analysis shows that the proposed algorithm enjoys the fast $O(1/T^2)$ convergence rate of accelerated proximal gradient algorithms. Extensive experiments on both synthetic and large recommendation data sets show that the proposed algorithm is much faster than Soft-Impute and other state-of-the-art matrix completion algorithms.

1 Introduction
In many applications, the data are stored in a matrix. Given a partially observed matrix $O$, matrix factorization attempts to recover a low-rank matrix $X$ that best approximates $O$ on the observed entries [Candès and Recht, 2009]. Matrix factorization has been widely used in a variety of domains. Examples include collaborative filtering for recommender systems [Mazumder et al., 2010], link prediction for social networks [Kim and Leskovec, 2011], image inpainting [Liu et al., 2013], and multilabel learning [Cabral et al., 2011].

However, direct minimization of the matrix rank is NP-hard, and the nuclear norm (which is the sum of singular values) is often used instead. Mathematically, let $O \in \mathbb{R}^{m \times n}$ (where, without loss of generality, we assume that $m \geq n$), and the positions of the observed entries be indicated by $\Omega \in \{0, 1\}^{m \times n}$, such that $\Omega_{ij} = 1$ if $O_{ij}$ is observed, and 0 otherwise. Matrix factorization is formulated as the following optimization problem

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

where $[P_\Omega (A)]_{ij} = A_{ij}$ if $\Omega_{ij} = 1$, and 0 otherwise; and $\|X\|_*$ is the nuclear norm of $X$. It is known that the nuclear norm is the tightest convex lower bound of the rank [Recht et al., 2010]. Besides, though the nuclear norm is only a surrogate, there are theoretical guarantees that the underlying matrix can be recovered [Candès and Recht, 2009].

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) [Candès and Recht, 2009]. However, SDP solvers have large time and space complexities, and are only suitable for small data sets. For large-scale matrix completion, Cai et al. [2010] pioneered the use of first-order methods and proposed the singular value thresholding (SVT) algorithm. However, a singular value decomposition (SVD) is required in each SVT iteration. This takes $O(mn^2)$ time and can be computationally expensive. In [Toh and Yun, 2010], this is reduced to a partial SVD by computing only the leading singular values/vectors using PROPACK (a variant of the Lanczos algorithm) [Larsen, 1998]. Another major breakthrough is made by the Soft-Impute algorithm [Mazumder et al., 2010], 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 with a reasonable time.

The SVT algorithm can also be viewed as a proximal gradient algorithm [Tibshirani, 2010]. Hence, it converges with a $O(1/T)$ rate, where $T$ is the number of iterations [Beck and Teboulle, 2009; Nesterov, 2013]. Later, this is further “accelerated”, and the convergence rate is improved to $O(1/T^2)$ [Ji and Ye, 2009; Toh and Yun, 2010]. However, Tibshirani [2010] 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 per-iteration time complexity.

In this paper, we show that accelerating Soft-Impute is indeed possible while simultaneously preserving the “sparse plus low-rank” structure. Moreover, instead of using PROPACK to compute the (exact) partial SVD as in [Toh and Yun, 2010], we propose to use the power method [Halko et al., 2011] to obtain only an approximation of the dominant singular subspace. This is more efficient than PROPACK.
and also allows warm-start, which is particularly useful because of the iterative nature of Soft-Impute. Since the SVT obtained is then only approximate, we use recent results in proximal gradient algorithms [Schmidt et al., 2011] to show that convergence can still be as fast as performing exact SVT in each iteration. Hence, the resultant algorithm has low per-iteration complexity and fast $O(1/T^2)$ convergence rate.

The rest of the paper is organized as follows. Section 2 provides a brief review on the related work. The proposed accelerated inexact Soft-Impute algorithm is described in Section 3. Experimental results are presented in Section 4, and the last section gives some concluding remarks.

**Notation**

In the sequel, the transpose of vector/matrix is denoted by the superscript $\top$. For a vector $x$, $\|x\|_1 = \sum_i |x_i|$ is its $\ell_1$-norm, and $\|x\| = \sqrt{\sum_i x_i^2}$ its $\ell_2$-norm. For a matrix $X$, $\sigma_1 \geq \ldots \sigma_r$ are its singular values, $\text{tr}(X) = \sum_i X_{ii}$ is its trace, $\|X\|_1 = \sum_{i,j} |X_{ij}|$, $\|X\|_F = \text{tr}(X^\top X)$ is the Frobenius norm, and $\|X\|_* = \sum_i \sigma_i$ the nuclear norm. Moreover, $I$ denotes the identity matrix. For a function $f$, we let $f'$ be a subgradient in the subdifferential $\partial f(x) = \{g | f(y) \geq f(x) + g^T(y-x), \forall y\}$. When $f$ is differentiable, we use $\nabla f$ for its gradient. Finally, we use $\Omega^\perp$ to denote the complement of a set $\Omega$. For a matrix $A$, $[P_{t\Omega}(A)]_{ij} = \begin{cases} A_{ij} & \Omega_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$, and $[P^\perp_{t\Omega}(A)]_{ij} = \begin{cases} 0 & \Omega_{ij} = 1 \\ A_{ij} & \text{otherwise} \end{cases}$.

## 2 Related Work

### 2.1 Proximal Gradient Algorithms

Consider minimizing composite functions of the form:

$$F(x) = f(x) + g(x),$$

where $f, g$ are convex, $f$ is smooth but $g$ is possibly nonsmooth. The proximal gradient algorithm [Beck and Teboulle, 2009; Nesterov, 2013; Parikh and Boyd, 2014] generates a sequence of estimates $\{x_t\}$ as

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

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

$$\text{prox}_{\mu g}(\cdot)$$

is called the proximal operator. When $f$ has $\rho$-Lipschitz continuous gradient (i.e., $\|\nabla f(x_1) - \nabla f(x_2)\| \leq \rho \|x_1 - x_2\|$) and a fixed stepsize $\mu \leq 1/\rho$ is used, this algorithm converges with rate $O(1/T)$, where $T$ is the number of iterations [Parikh and Boyd, 2014]. Moreover, it can be accelerated by replacing (4) with

$$y_t = (1 + \theta_t)x_t - \theta_t x_{t-1}, \quad z_t = y_t - \mu \nabla f(y_t).$$

Here, several choices for $\theta_t$ can be used. The resultant accelerated proximal gradient (APG) algorithm converges with the optimal $O(1/T^2)$ rate [Nesterov, 2013].

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

### 2.2 Soft-Impute

Soft-Impute is a state-of-the-art algorithm for large-scale matrix completion [Mazumder et al., 2010]. At iteration $t$, the missing values in $O$ are filled in as

$$Z_t = P_{t\Omega}(O) + P^\perp_{t\Omega}(X_t) = P_{t\Omega}(O - X_t) + X_t,$$

where $X_t$ is the current iterate. $X_{t+1}$ is then generated as

$$X_{t+1} = \text{SVT}_\lambda(Z_t),$$

where $\text{SVT}_\lambda(\cdot)$ is the singular value thresholding (SVT) operator defined as follows.

**Lemma 2.1 ([Cai et al., 2010]).** Let the SVD of a matrix $Z$ be $U\Sigma V^\top$. The solution to the optimization problem

$$\min_{X} \frac{1}{2}\|X - Z\|_F^2 + \lambda \|X\|_*$$

is a low-rank matrix given by

$$\text{SVT}_\lambda(Z) = U(\Sigma - \lambda I)_+ V^\top,$$

where $((A)_+)_{ij} = \max(A_{ij}, 0)$.

To compute $X_{t+1}$ in (7), we thus need to first perform SVD on $Z_t$. In general, obtaining the rank-$k$ SVD of a $m \times n$ matrix $Z$ takes $O(mnk)$ time. Its most expensive steps are on computing matrix-vector multiplications of the form $Zu$ and $v^\top Z$, where $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^m$.

To make Soft-Impute efficient, an important observation by Mazumder et al. is that $Z_t$ in (6) has a special “sparse plus low-rank” structure, namely that $P_{t\Omega}(O - X_t)$ is sparse and $X_t$ is low-rank. Let the rank of $X_t$ be $r$, and its SVD be $U_t \Sigma_t V_t^\top$. For any $u \in \mathbb{R}^n$, $Z_t u$ can then be computed as

$$Z_t u = P_{t\Omega}(O - X_t) u + U_t \Sigma_t (V_t^\top u).$$

Constructing $P_{t\Omega}(O - X_t)$ takes $O(r\|\Omega\|_1)$ time while computing $U_t \Sigma_t (V_t^\top u)$ takes $O(\|\Omega\|_1)$ time, and computing $U_t \Sigma_t (V_t^\top u)$ takes $O((m + n)r)$ time. Thus, to obtain the rank-$k$ SVD of $Z_t$, Soft-Impute needs only $O((r + k)\|\Omega\|_1 + rk(m + n))$ time. Moreover, its convergence rate is $O(1/T)$.

In (1), take $f(X) = \frac{1}{2}\|P_{t\Omega}(X - O)\|_F^2$ and $g(X) = \lambda \|X\|_*$. Note that $\nabla f(X) = P_{t\Omega}(X - O)$, and $\|\nabla f(X) - \nabla f(X_2)\|_F^2 = \|P_{t\Omega}(X_1 - X_2)\|_F^2 \leq \|X_1 - X_2\|_F^2$. Hence, $f$ is convex and has 1-Lipschitz-continuous gradient (i.e., $\rho = 1$); while $g$ is also convex but nonsmooth. It is obvious that $Z_t$ in (4) is the same as that in (6). Using $\mu = 1/\rho = 1$, it can be easily shown that $\text{prox}_{\mu g}(Z_t) = \text{prox}_{g}(Z_t)$ in (3) is indeed equal to $\text{SVT}_\lambda(Z_t)$. Hence, interestingly, Soft-Impute is also a proximal gradient algorithm [Tibshirani, 2010].

### 3 Accelerated Inexact Soft-Impute

Since Soft-Impute is a proximal gradient algorithm, it is natural to consider accelerating it with the schemes in Section 2.1. However, Tibshirani [2010] suggested that this is not useful, as the special “sparse plus low-rank” structure crucial to the efficiency of Soft-Impute will be lost. In this Section, we show that this structure can indeed be preserved, and thus acceleration is possible.
3.1 Accelerating Soft-Impute
To accelerate Soft-Impute, recall from Sections 2.1 and 2.2 that we have to compute \( \text{prox}_{\rho}(\tilde{Z}_t) = \text{SVT}_\lambda(\tilde{Z}_t) \), where

\[
\tilde{Z}_t = Y_t - \nabla f(Y_t) = P_O(O - Y_t) + Y_t \\
= P_O(O - Y_t) + (1 + \theta_t)X_t - \theta_tX_{t-1}.
\]  
(10)

The efficiency of Soft-Impute hinges on the “sparse plus low-rank” structure of \( \tilde{Z}_t \) in (6), which allows matrix-vector multiplications of the form \( Z \tilde{u} \) and \( v^\top \tilde{Z}_t \) to be computed inexpensively. In the following, we show that \( \tilde{Z}_t \) also has a similar 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. Using these SVDs, the sparse \( P_O(O - Y_t) \) can be constructed in \( O(r_t + r_{t-1})\|\Omega\|_1 \) time. Using (10), for any \( u \in \mathbb{R}^n \), \( P_O(O - Y_t)u \) can be obtained in \( O(\|\Omega\|_1) \) time; and \((1 + \theta_t)X_tu - \theta_tX_{t-1}u = (1 + \theta_t)U_t \Sigma_t V_t^\top u - \theta_tU_{t-1} \Sigma_{t-1} V_{t-1}^\top u \) in \( O((m + n)(r_{t-1} + r_t)) \) time. The same applies to the computation of \( v^\top \tilde{Z}_t \) for any \( v \in \mathbb{R}^m \).

Thus, the rank-\( k \) SVD of \( \tilde{Z}_t \) can be obtained in \( O((r_t + r_{t-1} + k)\|\Omega\|_1 + (r_{t-1} + r_t)(m + n)) \) time, which is only about twice that of Soft-Impute.

3.2 Approximating the SVD
Note from (8) that \( \text{SVT}_\lambda(\tilde{Z}_t) \) only involves the leading singular vectors of \( \tilde{Z}_t \) (whose singular values are greater than or equal to \( \lambda \)), and thus only a partial SVD is needed. Let \( k \) be the number of such singular values. The power method (Algorithm 1) [Halko et al., 2011] is a fast and accurate algorithm for obtaining an approximation \( Q \in \mathbb{R}^{m \times k} \) (where \( k \geq \hat{k} \)) of such a subspace. Besides the power method, algorithms such as PROPACK [Larsen, 1998] have also been used [Toh and Yun, 2010]. However, the power method is more efficient than PROPACK [Wu and Simon, 2000], and it also allows warm-start, which is particularly useful because of the iterative nature of Soft-Impute.

Algorithm 1 PowerMethod(\( \tilde{Z}, R, \varepsilon \)). In step 3, QR(\( \cdot \)) is the QR factorization.

Require: \( \tilde{Z} \in \mathbb{R}^{m \times n}, \) initial \( R \in \mathbb{R}^{k \times n} \) for warm-start, tolerance \( \varepsilon \);  
1: Initialize \( Y_1 \leftarrow \tilde{Z}R; \)
2: for \( \tau = 1, 2, \ldots \) do
3: \( Q_{\tau+1} = \text{QR}(Y_\tau); \)
4: \( Y_{\tau+1} = \tilde{Z}(\tilde{Z}^\top Q_{\tau+1}); \)
5: if \( \|Q_{\tau+1}Q_{\tau+1} - Q_{\tau}Q_{\tau}^\top\|_F \leq \varepsilon \) then
6: break;
7: end if
8: end for
9: return \( Q_{\tau+1}. \)

With this \( Q \), \( \text{SVT}(\tilde{Z}_t) \) can be obtained from a much smaller SVD as follows.

Proposition 3.1. Let \( Q \in \mathbb{R}^{m \times k} \) where \( k \geq \hat{k} \), be orthogonal and contains the subspace spanned by the top \( k \) left singular vectors of \( \tilde{Z}_t \). Then, \( \text{SVT}_\lambda(\tilde{Z}_t) = Q \text{SVT}_\lambda(Q^\top \tilde{Z}_t). \)

Algorithm 2 shows how to approximate \( \text{SVT}_\lambda(\tilde{Z}_t) \). Let the target (exact) rank-\( k \) SVD of \( \tilde{Z}_t \) be \( \hat{U}_t \Sigma_t V_t^\top \). Step 1 first approximates \( \hat{U}_k \) by the power method. In steps 2 to 5, a much smaller, and thus less expensive, (exact) \( \text{SVT}_\lambda(Q^\top \tilde{Z}_t) \) is obtained from (8). Finally, \( \text{SVT}_\lambda(\tilde{Z}_t) \) is recovered using Proposition 3.1.

Algorithm 2 Algorithm to approximate the SVD of \( \tilde{Z}_t \) (approx-SVT(\( \tilde{Z}_t, R, \lambda, \varepsilon \))).

Require: \( \tilde{Z}_t \in \mathbb{R}^{m \times n}, R \in \mathbb{R}^{n \times k}, \) thresholds \( \lambda \) and \( \varepsilon \);  
1: \( Q = \text{PowerMethod}(\tilde{Z}_t, R, \varepsilon); \)
2: \( [U, \Sigma, V] = \text{SVD}(Q^\top \tilde{Z}_t); \)
3: \( U = \{u_i \mid \sigma_i > \lambda\}; \)
4: \( V = \{v_i \mid \sigma_i > \lambda\}; \)
5: \( \Sigma = (\Sigma - \lambda I)_+, \)
6: return \( QU, \Sigma \) and \( V. \)

The main computation is on the power method (step 1), whose most expensive operations, in turn, are on matrix multiplications (steps 2 and 4 of Algorithm 1). By using the special structure of \( \tilde{Z}_t \) in (10) as discussed in Section 3.1, the time complexity of Algorithm 2 can be reduced from \( O(mnk) \) to \( O((r_t + r_{t-1} + k)\|\Omega\|_1 + (r_{t-1} + r_t)(m + n)) \).

The following Proposition guarantees the quality of \( X_t = (QU)\Sigma V^\top \) returned from Algorithm 2.

Proposition 3.2. Let \( \sigma_{t,k} \) be the \( k \)th singular value of \( \tilde{Z}_t \), \( \eta_t = \sigma_{t,k+1}/\sigma_{t,k} < 1 \); \( \alpha_t, \beta_t, \gamma_t \) be constants depending on \( \tilde{Z}_t \), and that the power method terminates after \( \tau \) iterations. Assume that \( k \geq \hat{k} \), and \( \varepsilon \geq \alpha_t\eta_t^6 \sqrt{1 + \eta_t^4}. \) Then,

\[
h_{\lambda (\|Q\|_2)}(\tilde{X}_t; \tilde{Z}_t) \leq h_{\lambda (\|S\|_2)}(\text{SVT}_\lambda(\tilde{Z}_t); \tilde{Z}_t) + \frac{\eta_t}{1 - \eta_t} \beta_t \gamma_t, \quad (11)
\]

where \( h_{\lambda (\|Q\|_2)}(X; \tilde{Z}_t) \equiv \frac{1}{2} \|X - \tilde{Z}_t\|^2 + \lambda \|X\|_\ast \) is the objective in the proximal step.

3.3 Proposed Algorithm
The whole procedure is shown in Algorithm 3. The core steps are 8–11, where an approximate SVT is performed. Similar to [Lin et al., 2010; Toh and Yun, 2010], steps 9–10 use the column space of the last two iterations \( (V_t \text{ and } V_{t-1}) \) to warm-start the power method. For faster speedup, at step 5, we employ a continuation strategy as in [Toh and Yun, 2010; Mazumder et al., 2010], in which \( \lambda_t \) is initialized to a large value and then decreased gradually. Moreover, as in IO’Donoghue and Candès, 2012; Nesterov, 2013, if \( F(X) \) starts to increase.

3.4 Convergence of the Inexact APG Algorithm
Since \( X_t \) only approximates \( \text{SVT}(\tilde{Z}_t) \), the proximal objective \( h_{\lambda (\|Q\|_2)}(\cdot; \tilde{Z}_t) \) is not exactly minimized. The convergence of such inexact APG algorithms has been recently studied in [Schmidt et al., 2011]. In general, for problem (2), an inexact APG algorithm may commit two types of errors: (i) an error \( \varepsilon_t \) in the calculation of \( \nabla f(\cdot) \), and (ii) an error \( \varepsilon_t \) in the proximal objective achieved by \( x_{t+1} \), i.e.,

\[
h_{\mu g}(x_{t+1}; z_t) \leq \varepsilon_t + h_{\mu g}(\text{prox}_{\mu g}(z_t); z_t). \quad (12)
\]
Algorithm 3 Accelerated Inexact Soft-Impute (AIS-Impute).

Require: partially observed matrix $O$, parameter $\lambda$, decay parameter $\nu \in (0, 1)$, threshold $\epsilon$;
1: $[\hat{U}_0, \hat{\lambda}_0, \hat{V}_0] = \text{rank-1 SVD}(P_1(O))$;
2: initialize $c = 1$, $\hat{e}_0 = \|P_1(O)\|_F$, $X_0 = X_1 = \lambda_0 U_0 V_0^T$;
3: for $t = 1, 2, \ldots$ do
4: \hspace{0.5cm} $\hat{e}_t = \nu^t \hat{e}_0$;
5: \hspace{0.5cm} $\lambda_t = \nu^t (\lambda_0 - \lambda) + \lambda$;
6: \hspace{0.5cm} $\theta_t = (c - 1)/\left((c + 2)\right)$;
7: \hspace{0.5cm} $Y_t = X_t + \theta_t (X_t - X_{t-1})$;
8: \hspace{0.5cm} $\tilde{Z}_t = Y_t + P_1(O - Y_t)$;
9: \hspace{0.5cm} $V_{t-1} = V_{t-1} - V_t (V_t^T V_t)^{-1}$, remove zero columns;
10: \hspace{0.5cm} $R_t = QR([V_t, V_{t-1}])$;
11: $[U_{t+1}, \Sigma_{t+1}, V_{t+1}] = \text{approx-SVT}(\tilde{Z}_t, R_t, \lambda_t, \hat{e}_t)$;
12: $F(U_{t+1} \Sigma_{t+1} V_{t+1}^T) > F(U_t \Sigma_t V_t^T)$ then
13: \hspace{0.5cm} $c = 1$;
14: else
15: \hspace{0.5cm} $c = c + 1$;
16: end if
17: $F(U_{t+1} \Sigma_{t+1} V_{t+1}^T) - F(U_t \Sigma_t V_t^T) \leq \epsilon$ then
18: \hspace{0.5cm} break;
19: end if
20: end for
21: return $X_{t+1} = U_{t+1} \Sigma_{t+1} V_{t+1}^T$.

The following Theorem shows that the convergence rate remains unchanged if the errors decrease at appropriate rates.

Theorem 3.3 ([Schmidt et al., 2011]). Assume that (i) $f$ is convex and has Lipschitz-continuous gradient; and (ii) $g$ is convex and possibly nonsmooth. If $\|e_t\|$ and $\sqrt{e_t}$ decrease as $O(1/t^{2+\delta})$ for some $\delta > 0$, the inexact APG algorithm still converges with rate $O(1/T^2)$.

In the proposed Algorithm 3, there is no error in $\nabla f(\cdot)$ and so $e_t = 0$. As $e_t$, we have from Proposition 3.2 that $\hat{e}_t \leq \frac{n}{1-m} \hat{e}_0 \beta_t^\gamma_t$. From step 4, $\hat{e}_t = O(\nu^t)$ where $\nu \in (0, 1)$. Thus, $\hat{e}_t$ converges to zero at a linear rate and is faster than the $O(1/t^{2+\delta})$ rate required in Theorem 3.3. This is formally stated in the following Theorem.

Theorem 3.4. Assume the conditions in Proposition 3.2. Algorithm 3 converges to the optimal solution of (1) with a rate of $O(1/T^2)$.

4 Experiments

In this section, we perform experiments on both synthetic data (Section 4.1) and the entire MovieLens and Netflix recommendation data sets (Section 4.2).

4.1 Synthetic Data

We generate a $m \times m$ data matrix $O = UV + G$, where the elements of $U \in \mathbb{R}^{m \times 5}$, $V \in \mathbb{R}^{5 \times m}$ are sampled i.i.d. from the normal distribution $\mathcal{N}(0, 1)$, and elements of $G$ sampled from $\mathcal{N}(0, 0.05)$. A total of $\|O\|_1 = 15m \log(m)$ random elements in $O$ are 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) elements.

The following variants of proximal gradient algorithms are compared:

- accelerated proximal gradient algorithm (denoted “APG”);\(^1\) [Ji and Ye, 2009; Toh and Yun, 2010];
- Soft-Impute\(^2\) [Mazumder et al., 2010];
- the proposed accelerated inexact Soft-Impute algorithm (denoted “AIS-Impute”) in Algorithm 3.

APG and AIS-Impute are in Matlab, while Soft-Impute is in R (and is called from Matlab). For performance evaluation, we use (i) the normalized mean squared error

$$\text{NMSE} = \frac{\|P_{10}^T(X - UV)\|_F^2}{\|P_{10}^T(UV)\|_F^2},$$

where $X$ is the recovered matrix; and (ii) the rank of $X$. We vary $m$ in $\{500, 1000, 1500, 2000\}$. Each experiment is repeated five times. Experiments are performed on a PC with Intel i7 CPU and 16GB RAM.

Results are shown in Table 1. As can be seen, all algorithms are equally good at recovering the missing matrix entries, but AIS-Impute is much faster. A more detailed timing comparison is in Figure 1. APG and AIS-Impute converge much faster than Soft-Impute w.r.t. the number of iterations (Figure 1(a)), as their convergence rates are $O(1/T^2)$ rather than $O(1/T)$. Because of its inexact proximal step, AIS-Impute has a slightly higher objective than APG. However, when measured against time (Figure 1(b)), APG is the slowest as it does not utilize the “sparse plus low-rank” structure. Overall, AIS-Impute is the fastest, as it has both low per-iteration complexity and fast $O(1/T^2)$ convergence rate.

![Figure 1](http://grouplens.org/datasets/movielens/)

(a) objective vs #iterations.  (b) objective vs time.

4.2 Recommendation Data

MovieLens Data

The MovieLens data set\(^3\) (Table 2) contains ratings (from 1 to 5) of different users on movies, and has been commonly used in matrix completion experiments [Mazumder et al., 2010; Hsieh and Olsen, 2014]. Following [Wang et al., 2014], we use 50% of the observed ratings for training, 25% for validation and the rest for testing. Besides the proximal gradient algorithms in Section 4.1, we also compare with other state-of-the-art non-proximal-gradient-based matrix completion algorithms, including

---

\(^1\)http://www.math.nus.edu.sg/~mattohkc/NLNS.html
\(^2\)http://cran.r-project.org/web/packages/softImpute/index.html
\(^3\)http://grouplens.org/datasets/movielens/
active subspace selection (denoted “active”) algorithm [Hsieh and Olsen, 2014]: In each iteration, this algorithm uses the power method to identify the active row and column subspaces, and then reduces the nuclear norm optimization problem to a smaller problem;

• a Frank-Wolfe based algorithm with local acceleration (denoted “boost”) [Zhang et al., 2012];

• a recent variant of Soft-Impute, which replaces the SVD in the soft-thresholding step by alternating least squares (denoted as “ALT-Impute”) [Hastie et al., 2014] and

• a second-order trust-region algorithm (denoted “TR”) that alternates between fixed-rank optimization and rank-one updates [Mishra et al., 2013].

For performance evaluation, as in [Hsieh and Olsen, 2014; Mazumder et al., 2010], we use (i) the root mean squared error RMSE = \sqrt{\parallel P_3(X - O) \parallel_F^2 / \parallel O \parallel_1}; and (ii) the rank of X. The experiment is repeated five times.

Results are shown in Table 3, and convergence of the objective and testing RMSE w.r.t. the running time are shown in Figure 4. As can be seen, all algorithms can recover the same RMSE and rank. However, ALT-Impute does not involve with SVD and has a better O(1/T^2) rate. Thus, it is again the fastest.

5 Conclusion

In this paper, we show that Soft-Impute, as a proximal gradient algorithm, can be accelerated without losing the “sparse plus low-rank” structure crucial to the efficiency of Soft-Impute. To further reduce the per-iteration time complexity, we proposed an approximate-SVT scheme based on the power method. Theoretical analysis shows that the proposed algorithm still enjoys the fast O(1/T^2) convergence rate. Extensive experiments on both synthetic and large recommendation data sets show that the proposed algorithm is much faster than the state-of-the-art.

Acknowledgment

This research was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region (Grant No. 614012).

References


Table 1: Results on the synthetic data (time is in seconds). Here, number in brackets is the data sparsity.

<table>
<thead>
<tr>
<th></th>
<th>m = 500 (18.64%)</th>
<th>m = 1000 (10.36%)</th>
<th>m = 1500 (7.31%)</th>
<th>m = 2000 (5.70%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APG</td>
<td>0.0183 5 5.1</td>
<td>0.0223 5 45.5</td>
<td>0.0251 5 172.7</td>
<td>0.0273 5 483.9</td>
</tr>
<tr>
<td>Soft-Impute</td>
<td>0.0183 5 1.3</td>
<td>0.0223 5 4.4</td>
<td>0.0251 5 13.3</td>
<td>0.0273 5 18.7</td>
</tr>
<tr>
<td>AIS-Impute</td>
<td>0.0183 5 0.3</td>
<td>0.0223 5 1.1</td>
<td>0.0251 5 2.0</td>
<td>0.0273 5 2.9</td>
</tr>
</tbody>
</table>

Table 2: Recommendation data sets used in the experiments.

<table>
<thead>
<tr>
<th></th>
<th>#users</th>
<th>#movies</th>
<th>#ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens-100K</td>
<td>943</td>
<td>1.682</td>
<td>100,000</td>
</tr>
<tr>
<td>MovieLens-1M</td>
<td>6,040</td>
<td>3,449</td>
<td>999,714</td>
</tr>
<tr>
<td>MovieLens-10M</td>
<td>69,878</td>
<td>10,677</td>
<td>10,000,054</td>
</tr>
</tbody>
</table>

Netflix Data

In this paper, we demonstrate the speedup of AIS-Impute over ALT-Impute and Soft-Impute on the Netflix data set.
Table 3: Results on the MovieLens data sets. Note that many algorithms fail to converge in $10^6$ seconds.

<table>
<thead>
<tr>
<th></th>
<th>MovieLens-100K</th>
<th></th>
<th>MovieLens-1M</th>
<th></th>
<th>MovieLens-10M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE rank time</td>
<td>RMSE rank time</td>
<td>RMSE rank time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>active</td>
<td>1.037 70 59.5</td>
<td>0.925 180 1431.4</td>
<td>0.918 217 29681.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>boost</td>
<td>1.038 71 19.5</td>
<td>0.925 178 616.3</td>
<td>0.917 216 13873.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALT-Impute</td>
<td>1.037 70 29.1</td>
<td>0.925 179 797.1</td>
<td>0.919 215 17337.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR</td>
<td>1.037 71 1911.4</td>
<td>—</td>
<td>—</td>
<td>&gt; $10^6$</td>
<td></td>
</tr>
<tr>
<td>APG</td>
<td>1.037 70 83.4</td>
<td>0.925 180 2060.3</td>
<td>—</td>
<td>—</td>
<td>&gt; $10^6$</td>
</tr>
<tr>
<td>Soft-Impute</td>
<td>1.037 70 537.6</td>
<td>0.925 180 8821.0</td>
<td>—</td>
<td>—</td>
<td>&gt; $10^6$</td>
</tr>
<tr>
<td>AIS-Impute</td>
<td>1.037 70 5.8</td>
<td>0.925 179 129.7</td>
<td>0.916 215 2817.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) MovieLens-100K. (b) MovieLens-1M. (c) MovieLens-10M.

Figure 2: Convergence of the objective (top) and testing RMSE (bottom) w.r.t. time on the MovieLens data sets.
Table 4: Results on the Netflix data set. Here, $\lambda_0 = \| P_\Omega(O) \|_F$, and time is in hours.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\lambda = \lambda_0/50$</th>
<th>$\lambda = \lambda_0/100$</th>
<th>$\lambda = \lambda_0/150$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE rank time</td>
<td>RMSE rank time</td>
<td>RMSE rank time</td>
</tr>
<tr>
<td>ALT-Impute</td>
<td>1.480 3 1.02</td>
<td>1.213 5 2.65</td>
<td>1.099 21 9.02</td>
</tr>
<tr>
<td>Soft-Impute</td>
<td>1.480 3 1.78</td>
<td>1.213 5 5.75</td>
<td>1.099 21 24.47</td>
</tr>
<tr>
<td>AIS-Impute</td>
<td>1.480 3 0.13</td>
<td>1.213 5 0.21</td>
<td>1.099 21 0.79</td>
</tr>
</tbody>
</table>

Figure 4: Convergence of the objective (top) and testing RMSE (bottom) w.r.t. time on the Netflix data set.