Fast Singular Value Thresholding without Singular Value Decomposition

Jian-Feng Cai * Stanley Osher †

Abstract

Singular value thresholding (SVT) is a basic subroutine in many popular numerical schemes for solving nuclear norm minimization that arises from low-rank matrix recovery problems such as matrix completion. The conventional approach for SVT is first to find the singular value decomposition (SVD) and then to shrink the singular values. However, such an approach is time-consuming under some circumstances, especially when the rank of the resulting matrix is not significantly low compared to its dimension. In this paper, we propose a fast algorithm for directly computing SVT for general dense matrices without using SVDs. Our algorithm is based on matrix Newton iteration for matrix functions, and the convergence is theoretically guaranteed. Numerical experiments show that our proposed algorithm is more efficient than the SVD-based approaches for general dense matrices.

1 Introduction

Singular value thresholding (SVT) introduced in [7] is a key subroutine in many popular numerical schemes (e.g. [7, 12, 13, 52, 54, 66]) for solving nuclear norm minimization that arises from low-rank matrix recovery problems such as matrix completion [13–15, 60]. Let $\boldsymbol{Y} \in \mathbb{R}^{m\times n}$ be a given matrix, and $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be its singular value decomposition (SVD), where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices and $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_s)$ is the diagonal matrix with diagonals being the singular values of $\boldsymbol{Y}$. Then, the SVT of $\boldsymbol{Y}$ is defined as

$$\mathcal{D}_\tau(\boldsymbol{Y}) := \boldsymbol{U}\begin{bmatrix}(\sigma_1-\tau)_+ & & \\ & \ddots & \\ & & (\sigma_s-\tau)_+\end{bmatrix}\boldsymbol{V}^T, \quad \text{where} \quad (\sigma_i-\tau)_+ = \begin{cases}\sigma_i-\tau, & \text{if } \sigma_i-\tau>0,\\ 0, & \text{otherwise.}\end{cases} \tag{1}$$

In other words, in $\mathcal{D}_\tau(\boldsymbol{Y})$, the singular vectors of $\boldsymbol{Y}$ are kept and the singular values are shrunk by the soft-thresholding [25]. In this paper, we aim at developing fast numerical algorithms for computing the SVT of general dense matrices.

This topic is strongly motivated by the rapidly growing interest in the recovery of an unknown low-rank or approximately low-rank matrix from very limited information. The problem of low-rank matrix recovery has many different settings in a variety of applications. A large source of low-rank matrix recovery problems is in machine learning [5, 63]. The solution techniques developed in the machine learning community can often be reformulated as methods for solving low-rank recovery problems. For example, the principle component analysis (PCA) [40] is a fundamental tool for data analysis and dimension reduction; though it is derived from a statistical point of view, it can be understood as a low-rank matrix recovery from noisy data. Furthermore, variants of PCA, for instances, sparse PCA [22,78] and robust PCA [12], can be formulated as low-rank matrix recovery under different settings. More examples of low-rank matrix recovery techniques in machine learning include clustering [24,37,43,69], multi-task learning [2,59], metric learning [74], collaborative filtering [16,55], and etc. Low-rank matrix recovery problems arise from many other areas in applied science and engineering

*Department of Mathematics, University of Iowa, Iowa City, IA 52242. Email: jianfeng-cai@uiowa.edu

†Department of Mathematics, University of California, Los Angeles, CA 90095. Email: sjo@math.ucla.edu

as well, e.g., signal processing [45,46], computer algebra [41], computer vision [19,67], imaging [29,48], image and video processing [38,39], control [27,28], and bioinformatics [1,44].

We start with an illustrative example of low-rank matrix recovery — matrix completion — and its numerical solutions. Matrix completion refers to recovering a matrix from a sampling of its entries. This routinely comes up whenever one collects partially filled out surveys, and one would like to infer the many missing entries. The issue is of course that the matrix completion problem is extraordinarily ill posed since with fewer samples than entries, we have infinitely many completions. Therefore, it is apparently impossible to identify which of these candidate solutions is indeed the "correct" one without some additional information. In many instances, however, the matrix we wish to recover is of low rank or approximately low rank. The premise that the unknown has (approximately) low rank radically changes the problem, making the search for solutions feasible since the lowest-rank solution now tends to be the right one. Let $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ be a low rank matrix whose rank is r satisfying $r \ll \min\{m,n\}$, and $\Omega \subset \{1,2,\ldots,m\}\times\{1,2,\ldots,n\}$ be the set of indices of its sampled entries. The authors in [14,15] showed that most low rank matrices $\boldsymbol{M}$ can be perfectly recovered by solving the optimization problem

$$\begin{array}{ll} \min_{\boldsymbol{X}} & \|\boldsymbol{X}\|_* \\ \text{s.t.} & X_{ij} = M_{ij}, \quad (i,j)\in\Omega, \end{array} \tag{2}$$

provided that the number of samples is great enough. Here $\|\cdot\|_*$ stands for the nuclear norm, i.e., the summation of all singular values. The minimization problem (2) is convex and can be recast as a semidefinite programming [28]. Therefore, (2) can be solved by conventional semidefinite programming solvers such as SDPT3 [65] and SeDeMi [64]. However, such solvers are usually based on interior-point methods, and can not deal with large matrices because they need to solve huge systems of linear equations to compute the Newton direction. Usually, they can only solve problems of size at most hundreds by hundreds on a moderate PC. Interested readers are referred to [53] for some recent progress on interior-point methods concerning some special nuclear norm-minimization problems.

People then turn to customized algorithms for solving (2). One class of popular methods are based on the SVT $\mathcal{D}_\tau$ in (1), which was first introduced in [7]. In these algorithms, the SVT operator $\mathcal{D}_\tau$ serves as a basic and key tool that forces the iteration converges to a low-rank matrix, and it is required to be computed in each iteration. The major computational cost is on the application of the SVT operator. Here we give several examples of such SVT-based algorithms for solving (2) which are popular and recently developed.

- The first example is the SVT algorithm in [7]. In the SVT algorithm, the minimization (2) is first approximated by

  $$\begin{array}{ll} \min_{\boldsymbol{X}} & \tau\|\boldsymbol{X}\|_* + \frac{1}{2}\|\boldsymbol{X}\|_F^2 \\ \text{s.t.} & X_{ij} = M_{ij}, \quad (i,j)\in\Omega, \end{array}$$

  with a large parameter τ, and then we use a gradient ascent algorithm applied to its dual problem. It has been shown in [47,76] that the SVT algorithm with a finite τ can get the perfect matrix completion as (2) does. The SVT algorithm is reformulated as Uzawa's algorithm [4] or linearized Bregman iteration [9,10,58,73]. The iteration is

  $$\begin{cases} \boldsymbol{X}_k = \mathcal{D}_\tau(\boldsymbol{Y}_{k-1}), \\ \boldsymbol{Y}_k = \boldsymbol{Y}_{k-1} + \delta_k \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}_k), \end{cases} \tag{3}$$

  where $\mathcal{D}_\tau$ is the SVT operator defined in (1). The SVT algorithm was shown to be an efficient algorithm for matrix completion, especially for huge low rank matrices.

- The second example is the FPCA algorithm in [54], which combines the fixed point continuation [32] (also known as proximal forward-backward splitting [20]) with Bregman iteration [57]. The iteration is

  $$\begin{cases} \text{Iterate on } i \text{ to get } \boldsymbol{X}_k & \begin{cases} \boldsymbol{X}_i = \mathcal{D}_\tau(\boldsymbol{Y}_{i-1}), \\ \boldsymbol{Y}_i = \boldsymbol{X}_{i-1} + \delta_i \mathcal{P}_\Omega(\boldsymbol{M} + \boldsymbol{Z}_{k-1} - \boldsymbol{X}_i), \end{cases} \\ \boldsymbol{Z}_k = \boldsymbol{Z}_{k-1} + \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}_k) \end{cases} \tag{4}$$

Here again $\mathcal{D}_\tau$ is the SVT operator. The FPCA algorithm is in fact a gradient ascent algorithm applied to an augmented Lagrangian of (2).

- The third example is the augmented Lagrangian method (ALM) in [50]. The problem (2) is first reformulated into

$$\min_{\boldsymbol{X}} \|\boldsymbol{X}\|_* \qquad \text{s.t.} \quad \boldsymbol{X}+\boldsymbol{E}=\mathcal{P}_\Omega(\boldsymbol{M}), \quad \mathcal{P}_\Omega(\boldsymbol{E})=\boldsymbol{0},$$

where $\boldsymbol{E}$ is an auxiliary variable representing the error. Then the corresponding (partial) augmented Lagrangian (ALM) function is

$$\mathcal{L}(\boldsymbol{X},\boldsymbol{E},\boldsymbol{Y},\mu)=\|\boldsymbol{X}\|_*+\langle \boldsymbol{Y},\mathcal{P}_\Omega(\boldsymbol{M})-\boldsymbol{X}-\boldsymbol{E}\rangle+\frac{\mu}{2}\|\mathcal{P}_\Omega(\boldsymbol{M})-\boldsymbol{X}-\boldsymbol{E}\|_F^2, \qquad \text{with} \quad \mathcal{P}_\Omega(\boldsymbol{E})=\boldsymbol{0}.$$

An inexact gradient ascent algorithm applying to the ALM leads to

$$\begin{cases} \boldsymbol{X}_k=\mathcal{D}_{\mu_k^{-1}}(\mathcal{P}_\Omega(\boldsymbol{M})-\boldsymbol{E}_{k-1}+\mu_k^{-1}\boldsymbol{Y}_{k-1}),\\ \boldsymbol{E}_k=\mathcal{P}_{\Omega^c}(-\boldsymbol{X}_k+\mu_k^{-1}\boldsymbol{Y}_k),\\ \boldsymbol{Y}_k=\boldsymbol{Y}_{k-1}+\mu_k(\mathcal{P}_\Omega\boldsymbol{M}-\boldsymbol{X}_k-\boldsymbol{E}_k). \end{cases} \tag{5}$$

Once again here $\mathcal{D}_{\mu_k^{-1}}$ is the SVT operator, and it is the key to make the algorithm converge to low rank matrices. This algorithm is also known as the split Bregman method [11, 30] in the imaging community, and it is extended to the decomposition of a matrix into a low-rank matrix plus a sparse matrix in [12, 50, 75].

There are many other low-rank matrix recovery problems that can be solved via nuclear norm minimization, and SVT could play a fundamental role in the resulting numerical algorithms. In [60], the authors considered the problem of recovering a low-rank matrix from its linear samples. It was shown that the low-rank matrix can be recovered exactly by solving nuclear norm minimization if the sampling operator satisfies a restricted isometry property. In robust PCA [12], a minimization involving both the nuclear norm and the ℓ_1-norm is used to separate a low-rank matrix and a sparse one. Other examples that use nuclear norm minimization to recover low-rank matrices can be found in, e.g., [2, 51, 59, 74]. SVT is a crucial tool in numerical algorithms for solving these resulting nuclear norm minimization problems, due to the fact that $\mathcal{D}_\tau$ is the proximal operator [36] of the nuclear norm function, i.e.,

$$\mathcal{D}_\tau(\boldsymbol{Y})=\arg\min_{\boldsymbol{X}\in\mathbb{R}^{m\times n}} \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{X}\|_F^2+\tau\|\boldsymbol{X}\|_*, \tag{6}$$

where $\|\cdot\|_F$ is the matrix Frobenius norm. The proximal operator has its origins in convex optimization theory, and it has been widely used for non-smooth convex optimization problems. Recently, proximal algorithms received much attention in solving ℓ_1-norm minimization problems arising from compressed sensing [17, 26] and related areas. It is well known that the proximal operator of the ℓ_1-norm is the soft-thresholding operator, and soft-thresholding based algorithms for solving ℓ_1 norm minimization problems include iterative thresholding [23], proximal forward-backward splitting (PFBS) [20], split Bregman method [11,30], linearized Bregman method [9, 10, 58, 73], Bregmanized operator splitting [77], accelerated proximal gradient [3, 68], alternating-direction method of multipliers [72], and so on. These algorithms are efficient and very popular in solving ℓ_1 norm minimization problems. Due to the analogous between the ℓ_1 and nuclear norms, it is natural to extend the soft-thresholding algorithms to solving nuclear norm minimizations for low-rank matrix recovery, and we expect them to have excellent performances too. Indeed, since $\mathcal{D}_\tau$ is the proximal operator of the nuclear norm, every soft-thresholding based algorithm for ℓ_1 norm minimization can be extended to nuclear norm minimizations without too much difficulty by replacing the soft-thresholding operator by $\mathcal{D}_\tau$.

Therefore, finding $\mathcal{D}_\tau(\boldsymbol{Y})$ efficiently for a given matrix $\boldsymbol{Y}$ is crucial in algorithms for solving nuclear norm minimization arising from low-rank matrix recovery problems. One natural way is to use the definition (1). We first compute SVD of $\boldsymbol{Y}$ and then obtain $\mathcal{D}_\tau(\boldsymbol{Y})$ according to (1). This approach is very popular in the literature (c.f. [7, 50, 52, 66]), and it depends highly on the SVD package used. When the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is

extremely low compared to its dimension, one can use partial SVD algorithms [62] that are extremely efficient. However, as the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ increases, partial SVD algorithms becomes less and less efficient. In fact, it was observed in [50] that when we want to compute more than $0.2 \cdot \min\{m, n\}$ singular vectors/values, using PROPACK [49], one of the most popular and efficient partial SVD packages, is often slower than computing the full SVD. This finally slows down the algorithm. Note that $\boldsymbol{Y}$ is some intermediate iterate matrix and $\mathcal{D}_\tau(\boldsymbol{Y})$ is not necessarily of low rank in general.

We aim at developing fast algorithms for computing $\mathcal{D}_\tau(\boldsymbol{Y})$ when $\boldsymbol{Y}$ is a general dense matrix and partial SVDs have no advantage over the full SVD. Our idea is to compute $\mathcal{D}_\tau(\boldsymbol{Y})$ as a single matrix instead of using SVD, by employing methods developed for computing matrix functions (see [35] for a survey). In particular, we will use matrix Newton iteration, which has been used extensively in numerical linear algebra for, e.g., matrix inversions [8, 70] and polar decomposition [33, 34]. Throughout the paper, we assume that the matrix size $m \times n$ satisfies $m \geq n$, and the case $m < n$ can be done completely analogously since $\mathcal{D}_\tau(\boldsymbol{Y}) = (\mathcal{D}_\tau(\boldsymbol{Y}^T))^T$. The outline of our proposed algorithm is as follows.

1. Compute the polar decomposition [31] $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ by the method in [33–35]. Here $\boldsymbol{W}$ is a unitary matrix and $\boldsymbol{Z}$ is a symmetric nonnegative definite matrix.
2. Project $\boldsymbol{Z}$ into the 2-norm ball, i.e., compute $\mathcal{P}_\tau(\boldsymbol{Z}) := \arg\min_{\|\boldsymbol{X}\|_2 \leq \tau} \|\boldsymbol{X} - \boldsymbol{Z}\|_F$, by a matrix Newton iteration.
3. Then $\mathcal{D}_\tau(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{W}\mathcal{P}_\tau(\boldsymbol{Z})$.

There are only matrix inversions and additions are involved in our proposed iterations. Both these operations can be done by basic linear algebra subroutine (BLAS), which are highly optimized on computers to achieve the best performance. Furthermore, we will show that the iterations involved in our algorithm all converge quadratically. Therefore, the proposed algorithm is efficient in finding $\mathcal{D}_\tau(\boldsymbol{Y})$. Numerical experiments shows that our algorithm is generally a few times faster than the algorithm via the full SVD.

We remark that, for matrix completion, there are some algorithms available that do not use the expensive SVDs. These algorithms usually use non-convex formulations instead of the convex nuclear norm minimization (2), and they can be fast and have comparable recoverability to those based on nuclear norm minimization in practice; see, e.g., [6, 21, 42, 56, 71].

The rest of the paper is organized as follows. In Section 2, we give the proposed algorithm. We first describe our algorithm for scalars in order to see the idea clearly, and the matrix version is a natural extension of the scalar case. Numerical experiments are shown in Section 3. Then, finally, we conclude our paper in Section 4 and give some discussions on possible extensions of our algorithm.

2 Algorithms

In this section, we propose our algorithm for finding $\mathcal{D}_\tau(\boldsymbol{Y})$. Instead of computing $\mathcal{D}_\tau(\boldsymbol{Y})$ directly, we compute the projection of $\boldsymbol{Y}$ into the 2-norm balls in the matrix space. This procedure can be seen as the solution to the dual problem of (6). Then, by using relations between primal and dual variables, we can easily recover $\mathcal{D}_\tau(\boldsymbol{Y})$. This is done in Section 2.1. Then, we propose our algorithm for computing the projection. For better understanding of our algorithm, we first describe our algorithm for scalars in Section 2.2, and then the scalar iteration is naturally extended to the matrix iteration for the projection in Section 2.3.

2.1 Primal-dual reformulation of $\mathcal{D}_\tau(\boldsymbol{Y})$

We do not compute $\mathcal{D}_\tau(\boldsymbol{Y})$ directly. Instead, we compute the projection $\mathcal{P}_\tau(\boldsymbol{Y})$ of $\boldsymbol{Y}$ into the 2-norm ball, i.e.,

$$\mathcal{P}_\tau(\boldsymbol{Y}) = \arg\min_{\|\boldsymbol{X}\|_2 \leq \tau} \|\boldsymbol{X} - \boldsymbol{Y}\|_F, \tag{7}$$

where $\|\cdot\|_2$ is the 2-norm (the maximum singular value) of a matrix. Since the 2-norm and the Frobenius norm involved in (7) are all invariant under any unitary transformation, there is an explicit expression of $\mathcal{P}_\tau(\boldsymbol{Y})$ as follows

$$\mathcal{P}_\tau(\boldsymbol{Y}) = \boldsymbol{U} \begin{bmatrix} \min(\sigma_1, \tau) & & \\ & \ddots & \\ & & \min(\sigma_s, \tau) \end{bmatrix} \boldsymbol{V}^T. \tag{8}$$

In views of (1) and (8), we have the following relation between $\mathcal{D}_\tau(\boldsymbol{Y})$ and $\mathcal{P}_\tau(\boldsymbol{Y})$

$$\boldsymbol{Y} = \mathcal{D}_\tau(\boldsymbol{Y}) + \mathcal{P}_\tau(\boldsymbol{Y}). \tag{9}$$

Therefore, if we can find $\mathcal{P}_\tau(\boldsymbol{Y})$, then $\mathcal{D}_\tau(\boldsymbol{Y})$ can be obtained by (9). In other words, the problem of finding the SVT $\mathcal{D}_\tau(\boldsymbol{Y})$ is transferred to the problem of finding the projection $\mathcal{P}_\tau(\boldsymbol{Y})$.

Note that (7) is the dual problem of (6) and (9) is the relation between the primal and dual variables. Let us derive all these from the primal-dual perspective. Recall that $\|\cdot\|_*$ and $\|\cdot\|_2$ are the 1-norm and the ∞-norm of the vector of singular values. Similar to the vector 1-norm and ∞-norm which are dual to each other, $\|\cdot\|_*$ and $\|\cdot\|_2$ are dual to each other under the inner product in matrix space. In particular, we can write the nuclear norm into an equivalent form

$$\|\boldsymbol{X}\|_* = \max_{\|\boldsymbol{Z}\|_2 \le 1} \langle \boldsymbol{X}, \boldsymbol{Z} \rangle, \tag{10}$$

where $\langle \boldsymbol{X}, \boldsymbol{Z} \rangle := \mathrm{trace}(\boldsymbol{X}^T \boldsymbol{Z})$ is the inner product in the Hilbert space of matrices. By (6), $\mathcal{D}_\tau(\boldsymbol{Y})$ is a solution of $\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_F^2 + \tau\|\boldsymbol{X}\|_*$. Substituting (10) into this equation, we have that $\mathcal{D}_\tau(\boldsymbol{Y})$ is a solution of the primal problem

$$\min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} \left(\max_{\|\boldsymbol{Z}\|_2 \le 1} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_F^2 + \tau \langle \boldsymbol{X}, \boldsymbol{Z} \rangle \right). \tag{11}$$

The dual problem is obtained by interchanging the min-max

$$\max_{\|\boldsymbol{Z}\|_2 \le 1} \left(\min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_F^2 + \tau \langle \boldsymbol{X}, \boldsymbol{Z} \rangle \right), \tag{12}$$

which is equivalent to

$$\max_{\|\boldsymbol{Z}\|_2 \le 1} -\frac{1}{2}\|\boldsymbol{Y} - \tau\boldsymbol{Z}\|_F^2.$$

It is obvious that $\frac{1}{\tau}\mathcal{P}_\tau(\boldsymbol{Y})$ is a solution of the dual problem. The relation between the solutions of the primal and dual problems is obtained by solving either the inner maximization problem in (11) or the inner minimization problem in (12). This is exactly (9).

In summary, instead of solving the primal problem (6) directly, we first solve its dual problem (7) and then use the primal-dual relation (9) to get the SVT $\mathcal{D}_\tau(\boldsymbol{Y})$. This strategy was also used in [18] for the Rudin-Osher-Fatemi model [61] in total variation based image denoising. There are more general theory for the decomposition (9). Indeed, $\mathcal{D}_\tau(\boldsymbol{Y})$ is also known as the Moreau-Yosida proximal operator [36] of the nuclear norm, and (9) is the Moreau's decomposition of $\boldsymbol{Y}$ with respect to the nuclear norm and its conjugate.

2.2 Algorithm for scalars

Now we present our algorithm for finding $\mathcal{P}_\tau(\boldsymbol{Y})$. To see our idea more clearly, we start from the simplest case when $n = m = 1$, i.e., matrices are scalars. In this case, the problem of finding $\mathcal{P}_\tau(\boldsymbol{Y})$ becomes: given a number y, we want to find

$$\mathcal{P}_\tau(y) = \mathrm{sign}(y) \cdot \min\{|y|, \tau\}. \tag{13}$$

Since our final algorithm is for matrices where only addition, multiplication, and inversion are available, only those operations are allowed in our algorithm for scalars. We deal with the two factors $\mathrm{sign}(y)$ and $\min\{|y|, \tau\}$ in (13) separately. Namely, our algorithm is divided into two steps as follows.

1. Find $w := \text{sign}(y)$ and $z := |y|$.

2. Find $p := \mathcal{P}_\tau(z) = \min\{z, \tau\}$ and set $\mathcal{P}_\tau(y) = w \cdot p$.

Both these two steps are derived from Newton's method for solving quadratic equations. We assume that $y \neq 0$.

For the first step, notice that w takes the value either -1 or 1 which are one solution of the equation $w^2 = 1$. Applying Newton's method to solving $w^2 = 1$ yields

$$w_{k+1} = \frac{1}{2}(w_k + w_k^{-1}). \tag{14}$$

We have to find a proper initial guess w_0 so that w_k converges to the correct sign of y. The following lemma ensures that (14) with $w_0 = y$ always converges to the correct solution and the convergence rate is quadratic.

Lemma 1 *Assume that $y \neq 0$. Let $w = \text{sign}(y)$ and $w_0 = y$. Then, w_k generated by (14) is well-defined and satisfies*

$$|w_{k+1} - w| \leq \min\left\{\frac{1}{2}|w_k - w|^2, \frac{1}{2}|w_k - w|\right\}.$$

Proof. We show the lemma by two cases, namely, $y > 0$ and $y < 0$. If $y > 0$, then $w = 1$. Since $w_0 = y \neq 0$, w_1 is well-defined and $w_1 = \frac{1}{2}(w_0 + w_0^{-1}) \geq \frac{1}{2}(2\sqrt{\cdot w_0 \cdot w_0^{-1}}) = 1$. Therefore, $w_1 \neq 0$. Consequently, w_2 is well-defined, $w_2 \geq 1$, and $w_2 \neq 0$. Repeating this argument, we conclude that w_k is well-defined and $w_k \geq 1$ for all k. Moreover, by (14),

$$|w_{k+1} - 1| = \left|\frac{1}{2}(w_k + w_k^{-1}) - 1\right| = \frac{1}{|2w_k|}|w_k - 1|^2 \leq \frac{1}{2}|w_k - 1|^2,$$

and

$$|w_{k+1} - 1| = \frac{1}{|2w_k|}|w_k - 1|^2 = \frac{1}{2}(1 - \frac{1}{w_k})|w_k - 1| \leq \frac{1}{2}|w_k - 1|.$$

The case of $y < 0$ is proved analogously. □

In the second step, we want to find $p = \min\{z, \tau\}$. We notice that p must be a solution of the following quadratic equation

$$(p - z)(p - \tau) = 0.$$

Again, we use Newton's iteration to solve the above equation and obtain the following iteration:

$$p_{k+1} = \frac{p_k^2 - \tau z}{2p_k - z - \tau}. \tag{15}$$

With the initial guess $p_0 = 0$, we can show that p_k always converges to the desired solution $p = \min\{z, \tau\}$. Moreover, the convergence rate is quadratic if $z \neq \tau$ and linear if $z = \tau$. The results are summarized in the next lemma.

Lemma 2 *Let $z = |y|$ and $p = \min\{z, \tau\}$. Set $p_0 = 0$. Then, p_k generated by (15) is well-defined and satisfies*

- *When $z \neq \tau$:*
$$|p_{k+1} - p| \leq \min\left\{\frac{1}{|\tau - z|}|p_k - p|^2, \frac{1}{2}|p_k - p|\right\}, \tag{16}$$
- *When $z = \tau$:*
$$|p_{k+1} - p| \leq \frac{1}{2}|p_k - p|. \tag{17}$$

Proof. We first prove by induction that $0 \le p_k < p$ for all k, and therefore, $2p_k - \tau - z < 0$ and p_{k+1} is well defined. For this purpose, it is obvious that $0 \le p_0 < p$. Assume that $p_k \in [0, p)$. Now we show $p_{k+1} \in [0, p)$. Since $p_k \in [0, p)$, both the denominator and the nominator in (15) are negative. So $p_{k+1} \ge 0$. The upper bound of p_{k+1} is derived as follows. If $z > \tau$, then $p = \tau$ and, therefore,

$$p_{k+1} - p = \frac{p_k^2 - \tau z}{2p_k - z - \tau} - \tau = \frac{(p_k - \tau)^2}{2p_k - z - \tau} = \frac{(p_k - p)^2}{2p_k - z - \tau} < 0. \tag{18}$$

If $z < \tau$, then $p = z$ and, therefore,

$$p_{k+1} - p = \frac{p_k^2 - \tau z}{2p_k - z - \tau} - z = \frac{(p_k - z)^2}{2p_k - z - \tau} = \frac{(p_k - p)^2}{2p_k - z - \tau} < 0. \tag{19}$$

Consequently, $p_{k+1} < p$ and $p_{k+1} \in [0, p)$.

It remains to show (16) and (17). By (18) and (19), we have

$$|p_{k+1} - p| = \frac{-1}{2p_k - z - \tau}|p_k - p|^2 \le \frac{-1}{2p - z - \tau}|p_k - p|^2 = \frac{1}{|z - \tau|}|p_k - p|^2$$

and

$$|p_{k+1} - p| = \frac{1}{2}\frac{p_k - p}{p_k - (z + \tau)/2}|p_k - p| \le \frac{1}{2}|p_k - p|.$$

□

2.3 Algorithm for matrices

Now we derive our algorithm for finding $\mathcal{P}_\tau(\boldsymbol{Y})$. The algorithm for matrices is more complicated than that for scalars. But the outline is the same. Similarly, our algorithm is divided into two steps. In the first step, we factorize the matrix $\boldsymbol{Y}$ into the product of its "signum" and "absolute value"; then, in the second step, we project the "absolute value" onto the 2-norm ball. The matrix correspondence of the "signum" and "absolute value" factorization is called the polar decomposition [31,33]. It factorizes a given matrix into the product of a unitary matrix, which is the "signum", and a symmetric nonnegative definite matrix, which is the "absolute value". More precisely, for the matrix $\boldsymbol{Y}$, we factorize it into

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}, \quad \text{where} \quad \boldsymbol{W} \text{ is unitary, } \boldsymbol{Z} \text{ is symmetric nonnegative definite.} \tag{20}$$

With the polar decomposition, now we transfer the problem of finding $\mathcal{P}_\tau(\boldsymbol{Y})$ defined in (7) to a problem of finding the projection of $\boldsymbol{Z}$ onto the 2-norm ball. Since all the norms involved in (7) are unitary invariant, (7) is equivalent to

$$\mathcal{P}_\tau(\boldsymbol{Y}) = \boldsymbol{W} \cdot \left(\arg\min_{\|\boldsymbol{X}\|_2 \le \tau} \|\boldsymbol{Z} - \boldsymbol{X}\|_F^2 \right) = \boldsymbol{W} \cdot \mathcal{P}_\tau(\boldsymbol{Z}), \qquad \text{where} \qquad \boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}. \tag{21}$$

Therefore, finding $\mathcal{P}_\tau(\boldsymbol{Y})$ is identical to finding $\boldsymbol{W} \cdot \mathcal{P}_\tau(\boldsymbol{Z})$. We use the relation (9) to get the SVT $\mathcal{D}_\tau(\boldsymbol{Y})$. We write the outline of our algorithm for the SVT $\mathcal{D}_\tau(\boldsymbol{Y})$ as follows. The details of the two subroutines are discussed in the following two subsections.

Algorithm 1: Algorithm for SVT without SVD.
Input: $\boldsymbol{Y}$
Output: $\mathcal{D}_\tau(\boldsymbol{Y})$
(1) Compute the polar decomposition $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ defined in (20).
(2) Compute the projection $\mathcal{P}_\tau(\boldsymbol{Z}) = \arg\min_{\|\boldsymbol{X}\|_2 \le \tau} \|\boldsymbol{Z} - \boldsymbol{X}\|_F$.
(3) Set $\mathcal{D}_\tau(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{W}\mathcal{P}_\tau(\boldsymbol{Z})$.

2.3.1 Computing the Polar Decomposition

In this subsection, we give the algorithm to compute the polar decomposition (20). The algorithm is from [33, 34]. The polar decomposition has an explicit expression. Let $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the SVD of $\boldsymbol{Y}$. Then, $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$, where $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T \in \mathbb{R}^{m\times n}$ and $\boldsymbol{Z} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^T \in \mathbb{R}^{n\times n}$, is the polar decomposition of $\boldsymbol{Y}$.

We temporarily assume that the matrix $\boldsymbol{Y}$ is nonsingular and square. We use an iteration which is a natural extension of (14) to compute the polar decomposition of $\boldsymbol{Y}$. The iteration is

$$\boldsymbol{W}_{k+1} = \frac{1}{2}\left(\boldsymbol{W}_k + \boldsymbol{W}_k^{-T}\right), \quad k = 0, 1, \ldots, \qquad \boldsymbol{W}_0 = \boldsymbol{Y}, \tag{22}$$

where $\boldsymbol{W}_k^{-T}$ stands for the inverse and transpose of $\boldsymbol{W}_k$. This algorithm is essentially the algorithm proposed in [33, 34]. It was also shown there that the iteration always converges quadratically to the polar factor. Here we give a very brief proof of the theorem for completeness.

Theorem 1 *Assume that $\boldsymbol{Y}$ is square and nonsingular. Let $\boldsymbol{W}$ be the polar factor of $\boldsymbol{Y}$ in* (20)*. Then $\boldsymbol{W}_k$ generated by* (22) *is well-defined and satisfies*

$$\|\boldsymbol{W}_{k+1} - \boldsymbol{W}\|_2 \le \min\left\{\frac{1}{2}\|\boldsymbol{W}_k - \boldsymbol{W}\|_2^2, \frac{1}{2}\|\boldsymbol{W}_k - \boldsymbol{W}\|_2\right\}.$$

Proof. Let $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the SVD of $\boldsymbol{Y}$. Then $\boldsymbol{W}_0 = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ has the left singular vectors $\boldsymbol{U}$ and the right singular vectors $\boldsymbol{V}$. Consequently, $\boldsymbol{W}_1 = \frac{1}{2}(\boldsymbol{W}_0 + \boldsymbol{W}_0^{-T}) = \boldsymbol{U}(\frac{1}{2}(\boldsymbol{\Sigma} + \boldsymbol{\Sigma}^{-1}))\boldsymbol{V}^T$. Therefore, $\boldsymbol{W}_1$ has the same singular vectors as $\boldsymbol{Y}$. Repeating this argument, we find that $\boldsymbol{W}_k$ for any k has the same left and right singular vectors as $\boldsymbol{Y}$. As a result, (22) changes only the singular values which are governed by (14). The lemma follows immediately from Lemma 1. □

In order to further accelerate the convergence of (22), the matrix is scaled in each iteration, as done in [33,34]. When the matrix $\boldsymbol{Y}$ is singular or rectangular, the iteration (22) is not available since $\boldsymbol{W}_0 = \boldsymbol{Y}$ is not invertible. Following [33], we reduce $\boldsymbol{Y}$ to a nonsingular square matrix by using a complete orthogonal decomposition (COD). More specifically, given an arbitrary matrix $\boldsymbol{Y} \in \mathbb{R}^{m\times n}$, we can decompose it into

$$\boldsymbol{Y} = \boldsymbol{O}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{Q}^T, \tag{23}$$

where $\boldsymbol{O} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and $\boldsymbol{R} \in \mathbb{R}^{s\times s}$ is an invertible upper triangular matrix. The COD can be done by, e.g., the QR decomposition; see [34] for details. Once we obtain $\boldsymbol{R}$, we have $\mathcal{D}_\tau(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{O}\begin{bmatrix}\mathcal{P}_\tau(\boldsymbol{R}) & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{Q}^T$. Therefore, we only need to find the projection $\mathcal{P}_\tau(\boldsymbol{R})$ for a nonsingular square matrix. So, we apply the iteration (22) by replacing $\boldsymbol{Y}$ by $\boldsymbol{R}$. Of course, in addition to that, we need to change Step 3 in Algorithm 1 accordingly in order to get $\mathcal{D}_\tau(\boldsymbol{Y})$. We omit the details here. The final algorithm to compute the polar decomposition is described in Algorithm 2.

Algorithm 2: Algorithm for the polar decomposition $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ [33, 34].
Input: Matrix $\boldsymbol{Y}$
Output: the polar factor $\boldsymbol{W}$, the symmetric nonnegative matrix $\boldsymbol{Z}$
(1) If necessary, compute COD of $\boldsymbol{Y}$ in (23) and set $\boldsymbol{W}_0 = \boldsymbol{R}$; otherwise, set $\boldsymbol{W}_0 = \boldsymbol{Y}$.
(2) **for** $k = 0$ **to** maximum number of iteration
(3) Compute $\boldsymbol{W}_k^{-T}$
(4) Set $\gamma_k = \left(\frac{\|W_k^{-1}\|_1\|W_k^{-1}\|_\infty}{\|W_k\|_1\|W_k\|_\infty}\right)^{1/4}$
(5) Set $\boldsymbol{W}_{k+1} = \frac{1}{2}(\gamma_k\boldsymbol{W}_k + \gamma_k^{-1}\boldsymbol{W}_k^{-T})$
(6) **if** $\|\boldsymbol{W}_{k+1} - \boldsymbol{W}_k\|_F \le \epsilon\|\boldsymbol{Y}\|_F$
(7) **return** $\boldsymbol{W} = \boldsymbol{W}_{k+1}$ and $\boldsymbol{Z} = \boldsymbol{W}^T\boldsymbol{Y}$

2.3.2 Computing the projection $\mathcal{P}_\tau(\boldsymbol{Z})$

To find $\mathcal{P}_\tau(\boldsymbol{Z}) = \arg\min_{\|\boldsymbol{X}\|_2 \le \tau} \|\boldsymbol{Z} - \boldsymbol{X}\|_F$, we extend (15) to the matrix case. The iteration is

$$\boldsymbol{P}_{k+1} = (2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}(\boldsymbol{P}_k^2 - \tau\boldsymbol{Z}), \quad k = 0, 1, 2, \ldots, \qquad \boldsymbol{P}_0 = \boldsymbol{0}. \tag{24}$$

Similar to the scalar case, the iteration (24) always converges to $\mathcal{P}_\tau(\boldsymbol{Z})$. Furthermore, when $\boldsymbol{Z} - \tau\boldsymbol{I}$ is invertible, the convergence rate is quadratic; while when $\boldsymbol{Z} - \tau\boldsymbol{I}$ is not invertible, the convergence rate is linear. We summarize the results in the following theorem and give an outline of the proof.

Theorem 2 *Let $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ be the polar decomposition and $\boldsymbol{P} = \mathcal{P}_\tau(\boldsymbol{Z})$. Then $\boldsymbol{P}_k$ generated by (24) is well defined and satisfies*

- *When $\boldsymbol{Z} - \tau\boldsymbol{I}$ is invertible:*

$$\|\boldsymbol{P}_{k+1} - \boldsymbol{P}\|_2 \le \min\left\{ \|(\boldsymbol{Z} - \tau\boldsymbol{I})^{-1}\|_2 \cdot \|\boldsymbol{P}_k - \boldsymbol{P}\|_2^2, \frac{1}{2}\|\boldsymbol{P}_k - \boldsymbol{P}\|_2 \right\},$$

- *When $\boldsymbol{Z} - \tau\boldsymbol{I}$ is not invertible:*

$$\|\boldsymbol{P}_{k+1} - \boldsymbol{P}\|_2 \le \frac{1}{2}\|\boldsymbol{P}_k - \boldsymbol{P}\|_2.$$

Proof. Since $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ is the polar decomposition of $\boldsymbol{Y}$, the matrix $\boldsymbol{Z}$ is a symmetric nonnegative definite matrix. Let $\boldsymbol{Z} = \tilde{\boldsymbol{V}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^T$ be the eigen-decomposition of $\boldsymbol{Z}$. Then, $\tilde{\boldsymbol{\Sigma}}$ is a diagonal matrix with nonnegative diagonals. So, $\boldsymbol{Y} = (\boldsymbol{W}\tilde{\boldsymbol{V}})\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^T$ is an SVD of $\boldsymbol{Y}$. This implies that the eigenvalues of $\boldsymbol{Z}$ are the singular values of $\boldsymbol{Y}$, and the eigenvectors are the right singular values of $\boldsymbol{Y}$. Recall that the SVD of $\boldsymbol{Y}$ is $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. In order to save notations, we write $\boldsymbol{Z} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^T$.

By induction on (24), one can easily see that $\boldsymbol{P}_k$ for any k is a symmetric matrix whose eigenvalues are the same as $\boldsymbol{Z}$. Therefore, in (24), we keep the eigenvectors and change only the eigenvalues of $\boldsymbol{P}_k$ as k varies. Moreover, the changing of the eigenvalues is governed by (15). The theorem follows immediately from this observation. □

The above theorem indicates that the iteration converges very fast to $\mathcal{P}_\tau(\boldsymbol{Z})$ due to the quadratic convergence rate. In the following, we discuss several issues to further accelerate the convergence, to reduce the computational cost per iteration, and to enhance the numerical stability.

First of all, in each iteration of (24), we need one matrix-matrix product $\boldsymbol{P}_k^2$ and one inversion $(2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}$, which both are computed in $O(n^3)$ operations. The remaining operations are matrix additions and subtractions, whose computational costs are only $O(n^2)$. Therefore, the main computations of (24) are the matrix-matrix product and the inversion. By carefully checking the iteration, we find that the matrix-matrix product is not necessary in each iteration. In fact, we can rewrite the iteration in (24) into an equivalent formulation

$$\boldsymbol{P}_{k+1} = \frac{1}{2}\boldsymbol{P}_k + \frac{1}{4}\boldsymbol{Z} + \frac{3}{4}\boldsymbol{I} - (2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}(\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3}{4}\boldsymbol{I}). \tag{25}$$

In the above formulation, the only matrix-matrix product $\boldsymbol{Z}^2$ is a constant during the whole iteration and we only need to compute it once at the beginning of the iteration. Therefore, the inversion $(2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}$ is the only $O(n^3)$ operation that needs to be computed in each step. By this trick, the computational cost is reduced significantly compared to (24).

Secondly, notice that all matrices involved in the iteration (25) are symmetric. We can take this advantage to further reduce the computational cost per step. More specifically, since the matrices $(2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}$ and $\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3}{4}\boldsymbol{I}$ are all symmetric and have the same eigenvectors as shown in the proof of Theorem 2, their product $(2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}(\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3}{4}\boldsymbol{I})$ shares the same eigenvectors with them and, therefore, is symmetric. This, in turn, implies that only half of the entries are required to be computed in the inversion. This helps us further reduce half of the computational cost per iteration.

Finally, we use a "deflation" technique [62] to accelerate the convergence of (24) (or equivalently (25)) and enhance its numerical stability. As shown in Theorem 2, the convergence speed of (24) depends on

$\|(\boldsymbol{Z}-\tau\boldsymbol{I})^{-1}\|_2$, i.e., the reciprocal of the gap between the threshold τ and the singular values of $\boldsymbol{Y}$. The smaller the gap is, the slower the algorithm converges. In the extreme case where τ is a singular value of $\boldsymbol{Y}$, the convergence rate degenerate from quadratic to linear as stated in Theorem 2. In order to get a faster convergence speed of (24), we need to enlarge the gap between the threshold and the singular values. Another reason that we have to enlarge this gap is for the numerical stability. As seen in (24), we need to invert the matrix $2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I}$ in each iteration. Recall that $\boldsymbol{P}_k$ converges to $\boldsymbol{P}$, whose eigenvalues are $2p(\sigma_i) - \sigma_i - \tau$. If the gap between the threshold and the singular values is very small, then there exists an i such that $p(\sigma_i)$, σ_i and τ are very close to each others. Therefore, the matrix $2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I}$ becomes more and more close to the singular as the iteration goes on. As a result, the error contained in $\boldsymbol{P}_k^2 - \tau\boldsymbol{Z}$ is amplified by the inversion of $2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I}$. This, in turn, makes the computation of $\boldsymbol{P}_{k+1}$ numerically unstable when it is close to $\boldsymbol{P}$. Therefore, for a faster convergence and a more stable numerical scheme, we require that the gap between the threshold τ and the singular values of $\boldsymbol{Y}$ is not small.

In order to enlarge the gap between the threshold and the singular values, here we use a technique similar to the deflation in eigenvalue computations. The idea is to remove those singular values around the threshold τ. From the definition of $\mathcal{P}_\tau(\boldsymbol{Z})$, we see that $\mathcal{P}_\tau(\boldsymbol{Z})$ is separable according to its eigenvectors. More precisely, $\mathcal{P}_\tau(\boldsymbol{Z})$ has the following property. Let the eigen decomposition of $\boldsymbol{Z}$ be $\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^T$, and we partition $\boldsymbol{V}$ into $\boldsymbol{V} = [\boldsymbol{V}_1\ \boldsymbol{V}_2]$ and $\boldsymbol{\Sigma}$ into $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2)$. So we have

$$\boldsymbol{Z} = \boldsymbol{V}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T + \boldsymbol{V}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^T. \tag{26}$$

With this decomposition, we have

$$\mathcal{P}_\tau(\boldsymbol{Z}) = \mathcal{P}_\tau(\boldsymbol{V}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T) + \mathcal{P}_\tau(\boldsymbol{V}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^T). \tag{27}$$

Based on this property, our deflation technique is as follows. Once we obtained the matrix "absolute value" $\boldsymbol{Z}$, we computes its eigenvalues around the threshold τ and their corresponding eigenvectors, and denote the eigenvalues be the diagonals of $\boldsymbol{\Sigma}_1$ and the eigenvectors be $\boldsymbol{V}_1$. Then, we can write $\boldsymbol{Z}$ into (26), where $\boldsymbol{\Sigma}_2$ and $\boldsymbol{V}_2$ are the eigenvalues far away from the threshold τ and there associated eigenvectors. According to (27), we can compute the projection of these two parts separately. The computation of $\mathcal{P}_\tau(\boldsymbol{V}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T)$ is straightforward. In order to get $\mathcal{P}_\tau(\boldsymbol{V}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^T)$, we use the iterative algorithm (25). Since $\boldsymbol{\Sigma}_2$ contains only eigenvalues which are far from the threshold, the iteration converges quadratically which is very rapid due to Theorem 2. Moreover, the iteration is numerically more stable since the matrices to be inverted is well conditioned.

Combining all together, the algorithm for computing $\mathcal{P}_\tau(\boldsymbol{Z})$ is summarized in the following algorithm.

Algorithm 3: Algorithm for computing $\mathcal{P}_\tau(\boldsymbol{Z})$
Input: a symmetric nonnegative definite matrix $\boldsymbol{Z}$, a positive number δ
Output: the projection $\boldsymbol{P} = \mathcal{P}_\tau(\boldsymbol{Z})$

(1) Compute the eigenvalues $\boldsymbol{\Sigma_1}$ of $\boldsymbol{Z}$ in the interval $[\tau(1-\delta), \tau(1+\delta)]$, and their associated eigenvectors $\boldsymbol{V}_1$.
(2) Set $\boldsymbol{Z} := \boldsymbol{Z} - \boldsymbol{V}_1\boldsymbol{\Sigma_1}\boldsymbol{V}_1^T$, and $\boldsymbol{P}_k = \boldsymbol{0}$.
(3) **for** $k = 0$ **to** maximum number of iterations
(4) Compute $\boldsymbol{P}_{k+1} = \frac{1}{2}\boldsymbol{P}_k + \frac{1}{4}\boldsymbol{Z} + \frac{3}{4}\boldsymbol{I} - (2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}(\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3}{4}\boldsymbol{I})$
(5) **if** $\|\boldsymbol{P}_{k+1} - \boldsymbol{P}_k\|_F \le \epsilon\|\boldsymbol{Z}\|_F$
(6) **return** $\boldsymbol{P} = \boldsymbol{P}_{k+1} + \boldsymbol{V}_1\mathcal{P}_\tau(\boldsymbol{\Sigma_1})\boldsymbol{V}_1^T$

3 Numerical Experiments

In this section, we give some numerical results to show that our proposed algorithm is very efficient to compute $\mathcal{D}_\tau(\boldsymbol{Y})$ when $\boldsymbol{Y}$ is dense and the full SVD is required. The algorithm is implemented in matlab using mex programming. The full SVD for comparison is computed by the matlab build-in function "svd". The computer is with an Intel Core i5 CPU at 2.50GHz (4 cores) and 6.00GB of memory, and the matlab

Figure 1: Distribution of singular values of a 500×500 Gaussian random matrix

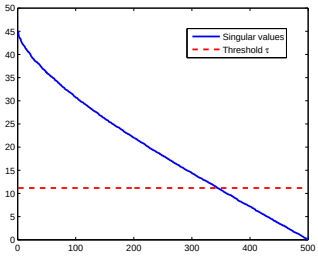

Table 1: Computational results for nonsingular square matrices. All results are averages of 10 runs.

n	Our algorithm (Alg. 1)				Full SVD
	# iter's in Alg. 2	# eig's removed	# iter's in Alg. 3	total time (s)	total time (s)
500	7	9.5	9	0.310	0.677
1000	7	18.8	9	2.02	10.4
1500	7	27.2	9	6.45	37.1
2000	7	37.2	9	14.9	91.0
2500	7	46.1	9	26.4	181
3000	7	55.4	9	47.3	337

version is 7.6.0(R2008a). Our numerical examples show that our proposed algorithm is generally a few times faster than SVT via the full SVD for general dense matrices.

First, we test our algorithm for square matrices. The test matrices are random Gaussian matrices, whose entries are randomly drawn from the standard Gaussian distribution. As predicted by Theorem 2, the computational speed of our algorithm depends on the distribution of singular values, in particular, on the gap between the threshold τ and the singular values. Therefore, we plot the singular values of test matrices and the threshold τ in Figure 1. We choose $\tau = \sqrt{n}/2$. We see that there is no obvious gap between τ and the singular values, so the SVT for Gaussian random matrices are not trivial examples for our algorithm to solve. Even though, our algorithm still performs very well. In Table 1, we list the number of iterations required for both the two steps in Algorithm 1, where we stop the iterations whenever the relative change of two successive steps is less than 10^{-6}. We remark that, though 10^{-6} is used in the stopping criteria, the relative error between the final result and $\mathcal{P}_\tau(\boldsymbol{Y})$ is of precision of order 10^{-10} for the tested examples. From Table 1, we see that both steps of our algorithm converges very fast: they need only 7 and 9 iterations respectively to converge, and the number of iterations keeps constant as the matrix size increases. The numbers of eigen pairs removed in the deflation step, where $\delta = 0.03$, in Algorithm 3 are also listed in Table 1. To compare our algorithm with the method via the full SVD, we report in Table 1 the computational time of both these two algorithms. We see that our algorithm is generally a few times faster than the method via the full SVD. For example, when $n = 2000$, the computational time for our algorithm is 47.3 seconds, and that for SVT via the full SVD is 337 seconds.

Next, we test our algorithm for rectangular and singular matrices respectively. The results are shown in Table 2. We use rectangular Gaussian random matrices as test rectangular matrices, and we choose $\tau = \sqrt{\max\{m,n\}}/2$. For singular test matrices, we generate them by $\boldsymbol{Y} = \boldsymbol{M}_L\boldsymbol{M}_R$, where $\boldsymbol{M}_L$ and $\boldsymbol{M}_R$ are Gaussian random matrices of size $n \times r$ and $r \times n$ respectively. We choose $r = 0.9n$ and $\tau = n/2$. Due to the distribution of singular values of Gaussian random matrices, these test problems are not trivial examples for our algorithm to solve as well. From Table 2, we see again that our algorithm takes a small number of iterations to converge and is much faster than the method via the full SVD.

Finally, we use one example to illustrate that our algorithm can accelerate low-rank matrix reconstruc-

Table 2: Computational results for rectangular and singular matrices. All results are averages of 10 runs.

size	Our algorithm (Alg. 1)				Full SVD
	# iter's in Alg. 2	# eig's removed	# iter's in Alg. 3	total time (s)	total time (s)
1000×500	5	12.4	9	0.338	0.925
2000×1000	5	25	9	2.39	13.7
3000×1500	5	37.9	9	7.54	49.7
1000×1000	7	15.9	9	1.70	7.88
2000×2000	7	32	9	12.6	71.1
3000×3000	7	44.7	9	39.7	288

Table 3: Computational results of matrix completion using (5) with different SVT subroutines.

Unknown Matrix			Alg. 1 + PROPACK			Full SVD + PROPACK		
n	rank	sample ratio	time (s)	# Alg. 1 called	# PROPACK called	time (s)	# full SVD called	# PROPACK called
500	25	0.39	4.7	4	39	5.7	2	41
1000	50		24.6	4	39	44.8	2	41
1500	75		70.8	4	39	144	2	41
2000	100		146	4	39	340	2	41
2500	125		278	4	39	712	2	41
3000	150		447	4	39	1181	2	41

tion algorithms. In particular, we show that Algorithm 1 can improve the computational speed of matrix completion via the inexact augmented Lagrangian method (5). We generate the $n \times n$ underlying low-rank matrix by $\boldsymbol{M} = \boldsymbol{M}_L \boldsymbol{M}_R$ where $\boldsymbol{M}_L \in \mathbb{R}^{n\times r}$ and $\boldsymbol{M}_R \in \mathbb{R}^{r\times n}$ with $r = 0.05n$ are Gaussian random matrices. Then we use (5) to reconstruct $\boldsymbol{M}$ from its randomly sampled entries (the sample ratio is 39% which is 4 times of the degree of freedoms). We choose $\mu_k = 5/n$ in (5). Since Algorithm 1 is developed for accelerating SVT only when full SVD is necessary, it is called only when the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is great enough. When the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is small, we use partial SVD package PROPACK [49], as done in, e.g., [7, 50]. According to our test, when the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is greater than $0.1n$, Algorithm 1 is faster than PROPACK. Therefore, we use Algorithm 1 to compute $\mathcal{D}_\tau(\boldsymbol{Y})$ if its rank is larger than $0.1n$, and PROPACK otherwise. The results are reported in Table 3 and referred to "Algorithm 1 + PROPACK". To see the performance improvement by Algorithm 1, we compare the results of "Algorithm 1 + PROPACK" with "full SVD + PROPACK", where the SVT is done by either matlab build-in function "svd" or PROPACK, whichever is faster. According to our experiments, matlab build-in function "svd" is faster than PROPACK when more than $0.25n$ leading singular values/vectores are computed. Therefore, matlab build-in function "svd" is used if the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ exceeds $0.25n$ and PROPACK otherwise in "full SVD + PROPACK". From Table 3, we see that "Algorithm 1 + PROPACK" uses much less computational time than "full SVD + PROPACK". In other words, Algorithm 1 does accelerate the computational speed of low-rank matrix reconstruction algorithms.

4 Conclusion and Discussion

In this paper, we proposed an algorithm to compute the singular value thresholding (SVT) for a given dense matrix. The algorithm is in two steps, namely, the polar decomposition step and the projection step, and both steps are done by matrix Newton iteration. Numerical experiments show that our algorithm is much faster than the method via the full singular value decomposition (SVD). Our algorithm is generally a few

times faster than the method via the full SVD.

For the future research, we may develop fast algorithms for $\mathcal{D}_\tau(\boldsymbol{Y})$ when $\boldsymbol{Y}$ is structured (e.g. sparse) and/or partial SVDs are more efficient than the full SVD. One possible way is to use the Krylov subspace method. More precisely, we can first project the matrix $\boldsymbol{Y}$ into the Krylov subspace via the Lanczos bidiagonalization procedure to get $\boldsymbol{Y} \approx \boldsymbol{S}\boldsymbol{B}\boldsymbol{T}^T$, where $\boldsymbol{S}$ and $\boldsymbol{T}$ are "tall" and "thin" orthonormal matrices and $\boldsymbol{B}$ is a bidiagonal matrix; then we apply the algorithm in this paper to compute $\mathcal{D}_\tau(\boldsymbol{B})$; finally we have $\mathcal{D}_\tau(\boldsymbol{Y}) \approx \boldsymbol{S}\mathcal{D}_\tau(\boldsymbol{B})\boldsymbol{T}^T$. The difficulty is how to determine the dimension of the Krylov subspace. If the dimension is too high, the computational speed will be slow, and if the dimension is too low, the approximate precision will be not good.

References

[1] O. Alter, P. Brown, and D. Botstein, *Singular value decomposition for genome-wide expression data processing and modeling*, Proceedings of the National Academy of Sciences, 97 (2000), pp. 10101–10106.

[2] A. Argyriou, T. Evgeniou, and M. Pontil, *Convex multi-task feature learning*, Machine Learning, 73 (2008), pp. 243–272.

[3] A. Beck and M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, SIAM Journal on Imaging Sciences, 2 (2009), pp. 183–202.

[4] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, 1999.

[5] C. Bishop et al., *Pattern recognition and machine learning*, vol. 4, Springer New York, 2006.

[6] N. Boumal and P. Absil, *Rtrmc: A riemannian trust-region method for low-rank matrix completion*, Advances in Neural Information Processing Systems, 24 (2011), pp. 406–414.

[7] J.-F. Cai, E. J. Candès, and Z. Shen, *A singular value thresholding algorithm for matrix completion*, SIAM J. Optimiz., 20 (2010), pp. 1956–1982.

[8] J.-f. Cai, M. K. Ng, and Y.-m. Wei, *Modified Newton's algorithm for computing the group inverses of singular Toeplitz matrices*, J. Comput. Math., 24 (2006), pp. 647–656.

[9] J.-F. Cai, S. Osher, and Z. Shen, *Convergence of the linearized Bregman iteration for ℓ_1-norm minimization*, Math. Comp., 78 (2009), pp. 2127–2136.

[10] J.-F. Cai, S. Osher, and Z. Shen, *Linearized Bregman iterations for compressed sensing*, Math. Comp., 78 (2009), pp. 1515–1536.

[11] J.-F. Cai, S. Osher, and Z. Shen, *Split Bregman methods and frame based image restoration*, Multiscale Modeling & Simulation, 8 (2009), pp. 337–369.

[12] E. Candes, X. Li, Y. Ma, and J. Wright, *Robust principal component analysis?*, Journal of ACM, (2011), pp. 1–37.

[13] E. Candes and Y. Plan, *Matrix completion with noise*, Proceedings of the IEEE, (2009).

[14] E. Candes and B. Recht, *Exact matrix completion via convex optimization*, Foundations of Computational Mathematics, 9 (2009), pp. 717–772.

[15] E. Candès and T. Tao, *The power of convex relaxation: Near-optimal matrix completion*, IEEE Transactions on Information Theory, 56 (2010), pp. 2053–2080.

[16] E. J. Candès and J. Romberg, *Quantitative robust uncertainty principles and optimally sparse decompositions*, Found. Comput. Math., 6 (2006), pp. 227–254.

[17] E. J. Candès, J. Romberg, and T. Tao, *Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information*, IEEE Trans. Inform. Theory, 52 (2006), pp. 489–509.

[18] A. Chambolle, *An algorithm for total variation minimization and applications*, J. Math. Imaging Vision, 20 (2004), pp. 89–97. Special issue on mathematics and image analysis.

[19] P. Chen and D. Suter, *Recovering the missing components in a large noisy low-rank matrix: Application to SFM*, IEEE Trans. Pattern Anal. Mach. Intell., 26 (2004), pp. 1051–1063.

[20] P. L. Combettes and V. R. Wajs, *Signal recovery by proximal forward-backward splitting*, Multiscale Model. Simul., 4 (2005), pp. 1168–1200 (electronic).

[21] W. Dai, E. Kerman, and O. Milenkovic, *A geometric approach to low-rank matrix completion*, IEEE Transactions on Information Theory, 58 (2012), pp. 237–247.

[22] A. d'Aspremont, L. El Ghaoui, M. Jordan, and G. Lanckriet, *A direct formulation for sparse pca using semidefinite programming*, SIAM review, 49 (2007), pp. 434–448.

[23] I. Daubechies, M. Defrise, and C. De Mol, *An iterative thresholding algorithm for linear inverse problems with a sparsity constraint*, Comm. Pure Appl. Math., 57 (2004), pp. 1413–1457.

[24] C. Ding and X. He, *K-means clustering via principal component analysis*, in Proceedings of the twenty-first International Conference on Machine Learning, ACM, 2004, p. 29.

[25] D. L. Donoho, *De-noising by soft-thresholding*, IEEE Trans. Inform. Theory, 41 (1995), pp. 613–627.

[26] D. L. Donoho, *Compressed sensing*, IEEE Trans. Inform. Theory, 52 (2006), pp. 1289–1306.

[27] M. Fazel, H. Hindi, and S. Boyd, *A rank minimization heuristic with application to minimum order system approximation*, in American Control Conference, 2001. Proceedings of the 2001, vol. 6, IEEE, 2001, pp. 4734–4739.

[28] M. Fazel, H. Hindi, and S. Boyd, *Log-det heuristic for matrix rank minimization with applications to hankel and euclidean distance matrices*, in American Control Conference, 2003. Proceedings of the 2003, vol. 3, 2003.

[29] H. Gao, J.-F. Cai, Z. Shen, and H. Zhao, *Robust principle component analysis based four-dimensional computed tomography*, Physics in Medicine and Biology, 56 (2011), pp. 3181–3198.

[30] T. Goldstein and S. Osher, *The split Bregman method for L1 regularized problems*, SIAM Journal on Imaging Sciences, 2 (2009), pp. 323–343.

[31] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[32] E. Hale, W. Yin, and Y. Zhang, *Fixed-point continuation for ℓ_1-minimization: Methodology and convergence*, SIAM Journal on Optimization, 19 (2008), pp. 1107–1130.

[33] N. Higham, *Computing the polar decomposition – with applications*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 1160–1174.

[34] N. Higham and R. Schreiber, *Fast polar decomposition of an arbitrary matrix*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 648–655.

[35] N. J. Higham, *Functions of matrices*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008. Theory and computation.

[36] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex analysis and minimization algorithms. I*, vol. 305 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Springer-Verlag, Berlin, 1993. Fundamentals.

[37] A. Jalali and N. Srebro, *Clustering using max-norm constrained optimization*, arXiv preprint arXiv:1202.5598, (2012).

[38] H. Ji, S. Huang, Z. Shen, and Y. Xu, *Robust video restoration by joint sparse and low rank matrix approximation*, SIAM J. Imaging Sci., 4 (2011), pp. 1122–1142.

[39] H. Ji, C. Liu, Z. Shen, and Y. Xu, *Robust video denoising using low rank matrix completion*, in 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2010, pp. 1791–1798.

[40] I. Jolliffe, *Principal component analysis*, Wiley Online Library, 2005.

[41] N. Karmarkar and Y. Lakshman, *On approximate gcds of univariate polynomials*, Journal of Symbolic Computation, 26 (1998), pp. 653–666.

[42] R. Keshavan, A. Montanari, and S. Oh, *Matrix completion from a few entries*, IEEE Transactions on Information Theory, 56 (2010), pp. 2980–2998.

[43] H. Kiers, *Setting up alternating least squares and iterative majorization algorithms for solving various matrix optimization problems*, Computational Statistics & Data Analysis, 41 (2002), pp. 157–170.

[44] H. Kim and H. Park, *Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis*, Bioinformatics, 23 (2007), pp. 1495–1502.

[45] H. Krim and M. Viberg, *Two decades of array signal processing research: the parametric approach*, IEEE Signal Processing Magazine, 13 (1996), pp. 67–94.

[46] R. Kumaresan and D. Tufts, *Estimating the angles of arrival of multiple plane waves*, IEEE Transactions on Aerospace and Electronic Systems, (1983), pp. 134–139.

[47] M.-J. Lai and W. Yin, *Augmented l1 and nuclear-norm models with a globally linearly convergent algorithm*, 2012. Rice CAAM Technical Report 12-02.

[48] D. Lanman, M. Hirsch, Y. Kim, and R. Raskar, *Content-adaptive parallax barriers: optimizing dual-layer 3d displays using low-rank light field factorization*, in ACM Transactions on Graphics (TOG), vol. 29, ACM, 2010, pp. 163:1–10.

[49] R. Larsen, *PROPACK: Software for large and sparse SVD calculations*, http://soi.stanford.edu/rmunk/PROPACK.

[50] Z. Lin, M. Chen, L. Wu, and Y. Ma, *The augmented Lagrange multiplier method for exact recovery of a corrupted low-rank matrices*, Mathematical Programming, submitted, (2009).

[51] G. Liu, Z. Lin, and Y. Yu, *Robust subspace segmentation by low-rank representation*, in Proceedings of the 26th International Conference on Machine Learning (ICML), 2010.

[52] Y. Liu, D. Sun, and K. Toh, *An implementable proximal point algorithmic framework for nuclear norm minimization*, Preprint, July, (2009).

[53] Z. Liu and L. Vandenberghe, *Interior-point method for nuclear norm approximation with application to system identification*, SIAM Journal on Matrix Analysis and Applications, 31 (2009), pp. 1235–1256.

[54] S. Ma, D. Goldfarb, and L. Chen, *Fixed point and Bregman iterative methods for matrix rank minimization*, Mathematical Programming, 128 (2011), pp. 321–353.

[55] Netflix, Inc., *The netflix prize*, http://www.netflixprize.com/.

[56] T. Ngo and Y. Saad, *Scaled gradients on grassmann manifolds for matrix completion*, in Advances in Neural Information Processing Systems 25, P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., 2012, pp. 1421–1429.

[57] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, *An iterative regularization method for total variation-based image restoration*, Multiscale Model. Simul., 4 (2005), pp. 460–489 (electronic).

[58] S. Osher, Y. Mao, B. Dong, and W. Yin, *Fast linearized Bregman iteration for compressive sensing and sparse denoising*, Commun. Math. Sci., (2010), pp. 93–111.

[59] T. Pong, P. Tseng, S. Ji, and J. Ye, *Trace norm regularization: Reformulations, algorithms, and multi-task learning*, SIAM Journal on Optimization, 20 (2010), pp. 3465–3489.

[60] B. Recht, M. Fazel, and P. Parrilo, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, SIAM Rev., 52 (2010), pp. 471–501.

[61] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Phys. D, 60 (1992), pp. 259–268.

[62] Y. Saad, *Numerical methods for large eigenvalue problems*, Manchester Univ Pr, 1992.

[63] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*, Cambridge university press, 2004.

[64] J. Sturm, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11 (1999), pp. 625–653.

[65] K. Toh, M. Todd, and R. Tutuncu, *SDPT3 – a Matlab software package for semidefinite programming*, Optimization Methods and Software, 11 (1999), pp. 545–581.

[66] K.-C. Toh and S. Yun, *An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems*, Pacific J. Optimization, 6 (2010), pp. 615–640.

[67] C. Tomasi and T. Kanade, *Shape and motion from image streams under orthography: a factorization method*, International Journal of Computer Vision, 9 (1992), pp. 137–154.

[68] P. Tseng, *On accelerated proximal gradient methods for convex-concave optimization*, 2008. submitted to SIAM Journal on Optimization.

[69] M. Vichi and G. Saporta, *Clustering and disjoint principal component analysis*, Computational Statistics & Data Analysis, 53 (2009), pp. 3194–3208.

[70] Y. Wei, J. Cai, and M. K. Ng, *Computing Moore-Penrose inverses of Toeplitz matrices by Newton's iteration*, Math. Comput. Modelling, 40 (2004), pp. 181–191.

[71] Z. Wen, W. Yin, and Y. Zhang, *Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm*, Rice University CAAM Technical Report TR10-07. Submitted, (2010).

[72] J. Yang and Y. Zhang, *Alternating direction algorithms for ℓ_1-problems in compressive sensing*, SIAM Journal on Scientific Computing, 33 (2011), pp. 250–278.

[73] W. Yin, S. Osher, D. Goldfarb, and J. Darbon, *Bregman iterative algorithms for ℓ_1-minimization with applications to compressed sensing*, SIAM J. Imaging Sci., 1 (2008), pp. 143–168.

[74] Y. Ying and P. Li, *Distance metric learning with eigenvalue optimization*, The Journal of Machine Learning Research, 13 (2012), pp. 1–26.

[75] X. Yuan and J. Yang, *Sparse and low-rank matrix decomposition via alternating direction methods*, Pacific J. Optimiz., (to appear).

[76] H. Zhang, J.-F. Cai, L. Cheng, and J. Zhu, *Strongly convex programming for exact matrix completion and robust principal component analysis*, Inverse Probl. Imaging, 6 (2012), pp. 357–372.

[77] X. Zhang, M. Burger, X. Bresson, and S. Osher, *Bregmanized nonlocal regularization for deconvolution and sparse reconstruction*, SIAM Journal on Imaging Sciences, 3 (2010), pp. 253–276.

[78] H. Zou, T. Hastie, and R. Tibshirani, *Sparse principal component analysis*, J. Comput. Graph. Statist., 15 (2006), pp. 265–286.