Fast Singular Value Thresholding without Singular Value Decomposition

Jian-Feng Cai * Stanley Osher *

May 2010

Abstract

We are interested in solving the following minimization problem

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

where $\boldsymbol{Y}\in\mathbb{R}^{m\times n}$ is a given matrix, and $\|\cdot\|_F$ is the Frobenius norm and $\|\cdot\|_*$ the nuclear norm. This problem serves as a basic subroutine in many popular numerical schemes for nuclear norm minimization problems, which arise from low rank matrix recovery such as matrix completion. As $\mathcal{D}_\tau(\boldsymbol{Y})$ has an explicit expression which shrinks the singular values of $\boldsymbol{Y}$ and keeps the singular vectors, $\mathcal{D}_\tau$ is referred to singular value thresholding (SVT) operator in the literature. Conventional approaches for $\mathcal{D}_\tau(\boldsymbol{Y})$ first find the singular value decomposition (SVD) of $\boldsymbol{Y}$ and then shrink the singular values. However, such approaches are time consuming under some circumstances, especially when the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is not low compared to the matrix dimension or is completely unpredictable. In this paper, we propose a fast algorithm for directly computing $\mathcal{D}_\tau(\boldsymbol{Y})$ without using SVDs. Numerical experiments show that the proposed algorithm is much more efficient than the approach using the full SVD.

1 Introduction

We are interested in solving the following minimization problem

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

where $\boldsymbol{Y}\in\mathbb{R}^{m\times n}$ is a given matrix and $\tau\in\mathbb{R}^+$ is a fixed parameter. Here and throughout the paper, we use $\|\cdot\|_F$ to denote the matrix Frobenius norm or the square root of the summation of squares of all entries, and $\|\cdot\|_*$ stands for the nuclear norm or the summation of all singular values. Therefore, in (1), we want to find a matrix $\mathcal{D}_\tau(\boldsymbol{Y})$ which is in the vicinity of $\boldsymbol{Y}$ and has an as small as possible nuclear norm. Since $\mathcal{D}_\tau(\boldsymbol{Y})$ is equivalent to shrink the singular values of $\boldsymbol{Y}$ by a soft-thresholding [14] and keep the singular vectors, $\mathcal{D}_\tau(\boldsymbol{Y})$ is called the singular value thresholding (SVT) following from [2]. The SVT serves as a basic subroutine in many popular numerical schemes for nuclear norm minimization problems, which arise from low rank matrix recovery such as matrix completion.

Our motivation to study (1) is mainly from recent burst of research on matrix completion [2, 7, 9, 10], and more generally low-rank matrix recovery [6, 8, 25, 27, 32], via convex optimization. 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

*Department of Mathematics, University of California, Los Angeles, CA 90095. Email: cai,sjo@math.ucla.edu. Research supported by ONR N00014-07-1-0810, ONR N00014-08-1-1119 and an ARO MURI through Rice University.

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 has 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 [9, 10] 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 large enough. The minimization problem (2) is convex and can be recast as a semidefinite programming [15]. Therefore, (2) can be solved by conventional semidifinite programming solvers such as SDPT3 [36] and SeDeMi [35]. 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 [26] for some recent progress on interior-point methods concerning some special nuclear norm-minimization problems.

In order to complete large low rank matrices by solving (2), we have to turn to first-order methods. Here we give several examples of such first-order methods which are quite popular and recently developed.

- The first example is the SVT algorithm in [2]. 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. This algorithm is reformulated as Uzawa's algorithm [1] or linearized Bregman iteration [3, 4, 31, 38]. 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 main step that makes the matrix of low rank is the SVT operator $\mathcal{D}_\tau$. Therefore, the iteration (3) is called SVT algorithm in [2]. 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 [27], which combines the fixed point continuation [18] (also known as proximal forward-backward splitting [12]) with Bregman iteration [30]. 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 [24]. 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. Then the corresponding (partial) augmented Lagrangian (ALM) function is

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

Then, an inexact gradient ascent algorithm is applied to the ALM, and it leads to the following algorithm

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

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 [5, 16] in the imaging community, and it is extended to the decomposition of a matrix into a low-rank matrix plus a sparse matrix in [6, 24, 39].

In all the algorithms mentioned above, the SVT operator $\mathcal{D}_\tau$ serves as a basic tool to produce low-rank matrices. It is required to be computed in each iteration. This fact holds true for not only the above three listed algorithms, but also many other recent popular algorithms for finding low rank matrix via nuclear norm minimization such as [25,37,40], just name a few more. The role of SVT in these algorithms is the same as the vector soft-thresholding [14] in many popular successful ℓ_1-norm minimization algorithms for finding sparse vectors in compressed sensing; see [3,4,13,18,38]. However, different from the vector soft-thresholding, SVT is much harder to compute, as the entries of the unknown matrix are strongly coupled together in (1). The computational efficiency for SVT is crucial to the efficiency of those algorithms for matrix completion and low-rank matrix recovery.

It is well known that $\mathcal{D}_\tau(\boldsymbol{Y})$, the solution of (1), has an explicit expression as follows; see, e.g., [2, 6, 24, 27, 39]. Let the singular value decomposition (SVD) [17] of $\boldsymbol{Y}$ be

$$\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T,$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices and $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_s)$ is a diagonal matrix with diagonals being the singular values of $\boldsymbol{Y}$. Then,

$$\mathcal{D}_\tau(\boldsymbol{Y}) = \boldsymbol{U} \begin{bmatrix} (\sigma_1 - \tau)_+ & & \\ & \ddots & \\ & & (\sigma_s - \tau)_+ \end{bmatrix} \boldsymbol{V}^T, \quad \text{where} \quad (\sigma_i - \tau)_+ = \begin{cases} \sigma_i - \tau, & \text{if } \sigma_i - \tau > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

In other words, the singular vectors of $\boldsymbol{Y}$ are kept, and only the singular values are shrunk by a soft-thresholding. That explains the name of SVT.

Based on (5), there is a natural way to find $\mathcal{D}_\tau(\boldsymbol{Y})$: We first compute SVD of $\boldsymbol{Y}$ and then obtain $\mathcal{D}_\tau(\boldsymbol{Y})$ according to (5). This approach is very popular in the literature. In fact, to the best of our knowledge, all the algorithms mentioned above compute SVT based on SVD. As a result, the efficiency of those algorithms depend highly on the performance of the computation of SVD. Generally, the computation of the full SVD is time consuming. As the size of the matrix $\boldsymbol{Y}$ increases, the computation of the full SVD becomes very slow and prohibitive. Therefore, to achieve good performance of those SVT-based algorithms, special techniques have to be used.

Notice that only those singular values exceeding τ and their corresponding singular vectors contribute to $\mathcal{D}_\tau(\boldsymbol{Y})$. Therefore, a commonly used strategy is to compute only partial SVD instead of the full one, i.e., we compute only those singular values exceeding τ and their associated singular vectors. If the number of singular values exceeding τ is small compared to the matrix dimension $\min\{m, n\}$, then partial SVD can be computed efficiently by some Krylov subspace projection methods [34]. This strategy is adopted in, e.g., [2, 24, 25, 37], where most of them used PROPACK [23] based on the Lanczos procedure with partial re-orthogonalization. As we have to specify the number of singular values/vectors to be computed before calling this SVD package, an important fact that enables us to compute only partial SVD is that the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is somewhat predictable. For example, in [2], we observed that a large parameter τ will yield a sequence of matrices with monotonically increasing rank, and all the matrices in the sequence have a rank lower than r, the rank of the true low rank matrix $\boldsymbol{M}$. Therefore, one can estimate the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ based

on the rank of the matrix in the previous step, and then use this rank estimation as the number of singular values/vectors to be computed in the partial SVD. This finally leads to an efficient algorithm to find SVT $\mathcal{D}_\tau(\boldsymbol{Y})$. Similar strategies are employed in [24, 25, 37].

Under some circumstances, the strategy of computing only partial SVD might not be helpful to accelerate the computation of SVT. One of such circumstances is that the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is not low compared to the matrix dimension $\min\{m, n\}$. This happens when there is no rank control in the algorithm or the rank of the desired matrix $\boldsymbol{M}$ is not very low. As we have mentioned, the efficiency of partial SVD computation depends highly on the number of singular values/vectors to be computed. The computational time of partial SVDs increases dramatically with respect to the number of singular values/vectors to be computed. In fact, it was observed in [24] that when we want to compute more than $0.2\min\{m, n\}$ principal singular vectors/values, using PROPACK is often slower than computing the full SVD. In this case, SVT via partial SVD has no advantage over SVT via the full SVD. Another circumstance under which partial SVD is not helpful is that the rank of $\mathcal{D}_\tau(\boldsymbol{Y})$ is completely unpredictable. This might happen, e.g., when we change the algorithm parameters during the iterations. In the worst case, we have to try many times on the number of singular values/vectors to be computed, following the strategies in, e.g., [2,24]. This finally slows down the algorithm.

In summary, the performance of computing $\mathcal{D}_\tau(\boldsymbol{Y})$ via the full SVD needs to be improved. In the past few decades, a wide range of iterative methods for computing matrix functions of the general form $f(\boldsymbol{Y})$ have been developed, see [21] for a survey. It is valuable to investigate whether some of these iterative methods, or other to be developed, would provide powerful ways for computing $\mathcal{D}_\tau(\boldsymbol{Y})$. This is the main theme of this paper. Observe that we are not interested in the singular values and singular vectors but in $\mathcal{D}_\tau(\boldsymbol{Y})$ as a single matrix. It is wasteful if we compute all the singular values and singular vectors explicitly. Why not skip the step of the full SVD and compute $\mathcal{D}_\tau(\boldsymbol{Y})$ directly? In this paper, we will propose such an algorithm to compute SVT without using SVD. 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 [17] $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ by the method in [19–21]. 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 iteration (c.f. (23) and (24)).

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 algorithm. 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 saves more than 50% CPU time from the algorithm via the full SVD.

The rest of the paper is organized as follows. In Section 2, we give the proposed algorithm. We first illustrate our algorithm for scalars, and the matrix version can be viewed as a natural extension of the scalar case. Numerical experiments are shown in Section 3. Finally, we conclude our paper in Section 4 and give some discussions on possible extensions of our algorithm to compete with SVT via partial SVD.

2 Algorithms

In this section, we propose our algorithm to find $\mathcal{D}_\tau(\boldsymbol{Y})$ that is defined in (1). We first transfer (1) to its dual problem in Section 2.1. Then, we propose our algorithm. We first describe our algorithm for scalars in order to see the idea clearly, and the matrix version can be seen as a natural extension of the scalar case.

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.,

$$\mathcal{P}_\tau(\boldsymbol{Y}) = \arg\min_{\|\boldsymbol{X}\|_2 \le \tau} \|\boldsymbol{X} - \boldsymbol{Y}\|_F, \tag{6}$$

where $\|\cdot\|_2$ is the 2-norm or the maximum singular value of a matrix. Since the 2-norm and the Frobenius norm involved in (6) are all invariant under any unitary transformation, there is an explicit expression of $\mathcal{P}_\tau(\boldsymbol{Y})$ as follows

$$\mathcal{P}_\tau(\boldsymbol{Y}) = \boldsymbol{U} \begin{bmatrix} \min(\sigma_1, \tau) & & \\ & \ddots & \\ & & \min(\sigma_s, \tau) \end{bmatrix} \boldsymbol{V}^T. \tag{7}$$

In views of (5) and (7), we have the following relation between $\mathcal{D}_\tau(\boldsymbol{Y})$ and $\mathcal{P}_\tau(\boldsymbol{Y})$

$$\boldsymbol{Y} = \mathcal{D}_\tau(\boldsymbol{Y}) + \mathcal{P}_\tau(\boldsymbol{Y}). \tag{8}$$

Therefore, if we can find $\mathcal{P}_\tau(\boldsymbol{Y})$, then $\mathcal{D}_\tau(\boldsymbol{Y})$ can be obtained by (8). In other words, the problem of finding $\mathcal{D}_\tau(\boldsymbol{Y})$ is transferred to the problem of finding the projection $\mathcal{P}_\tau(\boldsymbol{Y})$.

Before presenting our algorithm for $\mathcal{P}_\tau(\boldsymbol{Y})$, we remark that the relation (8) holds true not by chance. There is some theory behind that equation. More precisely, (6) is the dual problem of (1), and (8) 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 infinity norm of the vector of singular values. Similar to the vector 1-norm and infinity 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

$$\|\boldsymbol{X}\|_* = \max_{\|\boldsymbol{Z}\|_2 \le 1} \langle \boldsymbol{X}, \boldsymbol{Z} \rangle, \tag{9}$$

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 the definition (1), $\mathcal{D}_\tau(\boldsymbol{Y})$ is a solution of $\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_F^2 + \tau\|\boldsymbol{X}\|_*$. Substituting (9) into this equation, we have that $\mathcal{D}_\tau(\boldsymbol{Y})$ is a solution of the primal problem

$$\min_{\boldsymbol{X} \in \mathbb{R}^{m \times n}} \left(\max_{\|\boldsymbol{Z}\|_2 \le 1} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_F^2 + \tau \langle \boldsymbol{X}, \boldsymbol{Z} \rangle \right). \tag{10}$$

The dual problem is obtained by interchanging the min-max

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

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 (10) or the inner minimization problem in (11). This is exactly (8).

In summary, instead of solving the primal problem (1) directly, we first solve its dual problem (6) and then use the primal-dual relation (8) to get the SVT $\mathcal{D}_\tau(\boldsymbol{Y})$. This strategy was also used in [11] for the Rudin-Osher-Fatemi model [33] in total variation based image denoising. There is a more general theory for the decomposition (8). Indeed, $\mathcal{D}_\tau(\boldsymbol{Y})$ is also known as the Moreau-Yosida proximal operator [22, 28, 29] of the nuclear norm, and (8) is the Moreau's decomposition of $\boldsymbol{Y}$ with respect to the nuclear norm and its conjugate. We refer the interested readers to those references for more details.

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 where $n = m = 1$, i.e., matrices are scalars. In this case, the problem of finding $\mathcal{P}_\tau(\boldsymbol{Y})$ becomes: given a number real y, we want to find

$$\mathcal{P}_\tau(y) = \operatorname{sign}(y) \cdot \min\{|y|, \tau\}. \tag{12}$$

Since our final goal is for matrices where only addition, multiplication, and inversion are available, only those operations are allowed in our algorithm for scalars. We treat the two factors $\operatorname{sign}(y)$ and $\min\{|y|, \tau\}$ in (12) separately. Namely, our algorithm is divided into two steps as follows.

1. Find $w := \operatorname{sign}(y)$ 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 done by Newton's method, and we assume that $y \neq 0$.

For the first step, notice that w takes the value either -1 or 1 which are each solutions of the equation $w^2 = 1$. Applying Newton's method (c.f. [19–21]) yields

$$w_{k+1} = \frac{1}{2}(w_k + w_k^{-1}). \tag{13}$$

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 (13) with $w_0 = y$ always converges to the correct solution and the convergence rate is quadratic.

Lemma 1 *Assume that $y \neq 0$. Let $w = \operatorname{sign}(y)$ and $w_0 = y$. Then, w_k generated by (13) 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 $w_1 = \frac{1}{2}(w_0 + w_0^{-1}) \geq \frac{1}{2}(2 \cdot w_0 \cdot w_0^{-1}) = 1$. 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 (13),

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

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, in order to find $p = \min\{z, \tau\}$, we first 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:

$$p_{k+1} = \frac{p_k^2 - \tau z}{2p_k - z - \tau}. \tag{14}$$

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 into the next lemma.

Lemma 2 *Let $z = |y|$ and $p = \min\{z, \tau\}$. Set $p_0 = 0$. Then, p_k generated by (14) is well-defined and satisfies*

- *When $z \neq \tau$:*

$$|p_{k+1} - p| \leq \min\left\{\frac{1}{|\tau - z|}|p_k - p|^2, \frac{1}{2}|p_k - p|\right\}, \tag{15}$$

- *When $z = \tau$:*

$$|p_{k+1} - p| \leq \frac{1}{2}|p_k - p|. \tag{16}$$

Proof. We first prove by induction that $0 \leq p_k < p$ for all k, and therefore, $2p_k - \tau - z < 0$ and p_{k+1} is well defined. For this purpose, it is obvious that $0 \leq p_0 < p$. 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 (14) are negative. So $p_{k+1} > 0$. The upper bound of p_{k+1} is derived as follows. If $z > \tau$, then $p = \tau$ and, therefore,

$$p_{k+1} - p = \frac{p_k^2 - \tau z}{2p_k - z - \tau} - \tau = \frac{(p_k - \tau)^2}{2p_k - z - \tau} = \frac{(p_k - p)^2}{2p_k - z - \tau} < 0. \tag{17}$$

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

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

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

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

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

and

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

□

2.3 Algorithm for matrices

Now we derive our algorithm for finding $\mathcal{P}_\tau(\boldsymbol{Y})$. It can be seen as a natural extension of the algorithm for scalars in the last subsection. This technique, extending Newton's iteration for scalars to matrices, is widely used in computing functions of matrices; see an review in [21].

Similar to the scalars case, our algorithm for matrices 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 [17, 19]. It factors a given matrix into the product of a unitary matrix, which is the "signum" of the matrix, and a symmetric nonnegative definite matrix, which is the "absolute value" of the matrix. More precisely, for the matrix $\boldsymbol{Y}$, we factor it into

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}, \quad \text{where} \quad \boldsymbol{W} \text{ is unitary, } \boldsymbol{Z} \text{ is symmetric nonnegative definite.} \tag{19}$$

With the polar decomposition, now we transfer the problem of finding $\mathcal{P}_\tau(\boldsymbol{Y})$ defined in (6) to a problem of finding the projection of $\boldsymbol{Z}$ onto the 2-norm ball. Since all the norms involved in (6) are unitary invariant, (6) is equivalent to

$$\mathcal{P}_\tau(\boldsymbol{Y}) = \boldsymbol{W} \cdot \arg\min_{\|\boldsymbol{X}\|_2 \leq \tau} \|\boldsymbol{Z} - \boldsymbol{X}\|_F^2 = \boldsymbol{W}\mathcal{P}_\tau(\boldsymbol{Z}), \qquad \text{where} \qquad \boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}. \tag{20}$$

Therefore, finding $\mathcal{P}_\tau(\boldsymbol{Y})$ is identical to finding $\boldsymbol{W}\mathcal{P}_\tau(\boldsymbol{Z})$. With $\mathcal{P}_\tau(\boldsymbol{Y})$, we use the relation (8) 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 successive two subsections.

Algorithm 1: Algorithm for SVT without SVD.
Input: matrix $\boldsymbol{Y}$
Output: SVT $\mathcal{D}_\tau(\boldsymbol{Y})$
(1) Compute the polar decomposition $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ defined in (19).
(2) Compute the projection $\mathcal{D}_\tau(\boldsymbol{Z}) = \arg\min_{\|\boldsymbol{X}\|_2 \le \tau} \|\boldsymbol{Z} - \boldsymbol{X}\|_F$.
(3) Set $\mathcal{P}_\tau(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{W}\mathcal{P}_\tau(\boldsymbol{Z})$.

2.3.1 Compute the Polar Decomposition

In this subsection, we give the algorithm to compute the polar decomposition (19). The algorithm is from [19, 20]. 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 $\boldsymbol{Y}$. Therefore, to get the polar decomposition of $\boldsymbol{Y}$, one can of course use SVD. However, we do not intend to use SVD to compute it since it is generally time consuming.

We temporarily assume that the matrix $\boldsymbol{Y}$ is nonsingular and square. We use an iteration which is a natural extension of (13) to compute the polar decomposition of $\boldsymbol{Y}$. The iteration is

$$\boldsymbol{W}_{k+1} = \frac{1}{2}\left(\boldsymbol{W}_k + \boldsymbol{W}_k^{-T}\right), \quad k = 0, 1, \ldots, \qquad \boldsymbol{W}_0 = \boldsymbol{Y}, \tag{21}$$

where $\boldsymbol{W}_k^{-T}$ stands for the inverse and transpose of $\boldsymbol{W}_k$. This algorithm is essentially the algorithm proposed in [19, 20]. 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* (19)*. Then $\boldsymbol{W}_k$ generated by* (21) *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, (21) changes only the singular values which are governed by (13). The lemma follows immediately from Lemma 1. □

When the matrix $\boldsymbol{Y}$ is singular or rectangular, the iteration (21) is not available since $\boldsymbol{W}_0 = \boldsymbol{Y}$ is not invertible. Following [19], 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

$$\boldsymbol{Y} = \boldsymbol{O}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{Q}^T, \tag{22}$$

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 [20] 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 (21) 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.

Anyway, we get back to assume that $\boldsymbol{Y}$ is a nonsingular square matrix. In order to further accelerate the convergence of (14), the matrix is scaled in each iteration, as done in [19, 20]. The final algorithm to compute the polar decomposition is described in Algorithm 2.

Algorithm 2: Algorithm for the polar decomposition $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{Z}$ [19, 20].
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 (22) 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 Compute 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 (14) to the matrix case. The iteration is

$$\boldsymbol{P}_{k+1} = (2\boldsymbol{P}_k - \boldsymbol{Z} - \tau \boldsymbol{I})^{-1}(\boldsymbol{P}_k^2 - \tau \boldsymbol{Z}), \quad k = 0, 1, 2, \ldots, \qquad \boldsymbol{P}_0 = \boldsymbol{0}. \tag{23}$$

Similar to the scalar case, the iteration (23) 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 into 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* (23) *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 $\boldsymbol{Z} = \boldsymbol{V'}\boldsymbol{\Sigma'}(\boldsymbol{V'})^T$ be the eigen-decomposition of $\boldsymbol{Z}$. Then, $\boldsymbol{\Sigma'}$ is a diagonal matrix with nonnegative diagonals. So, $\boldsymbol{Y} = (\boldsymbol{W}\boldsymbol{V'})\boldsymbol{\Sigma'}(\boldsymbol{V'})^T$ is the 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 (23), 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 (23), 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 (14). 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 (23), 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 (23) 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 (23) into an equivalent formulation

$$\boldsymbol{P}_{k+1} = \frac{1}{2}\boldsymbol{P}_k + \frac{1}{4}\boldsymbol{Z} + \frac{3\tau}{4}\boldsymbol{I} - (2\boldsymbol{P}_k - \boldsymbol{Z} - \tau \boldsymbol{I})^{-1}(\tau \boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3\tau^2}{4}\boldsymbol{I}). \tag{24}$$

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 greatly compared to (23).

Secondly, notice that all matrices involved in the iteration (24) are symmetric. We can take advantage of this to further reduce the computational cost per step. More specifically, since the matrices $(2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}$ and $\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3\tau^2}{4}\boldsymbol{I}$ are all symmetric and have the same eigenvectors as shown in the proof of Theorem 2, their product $(2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}(\tau\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3\tau^2}{4}\boldsymbol{I})$ 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 [34] to accelerate the convergence of (23) (or equivalently (24)) and enhance its numerical stability. As shown in Theorem 2, the convergence speed of (23) 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 (23), 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 (23), 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 be 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, we use a technique similar to 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

$$\boldsymbol{Z} = \boldsymbol{V}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T + \boldsymbol{V}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^T. \tag{25}$$

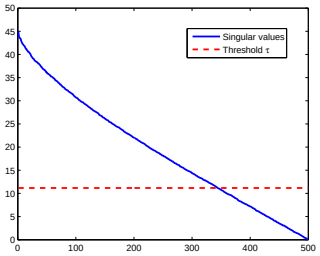
With this decomposition, we have

$$\mathcal{P}_\tau(\boldsymbol{Z}) = \mathcal{P}_\tau(\boldsymbol{V}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^T) + \mathcal{P}_\tau(\boldsymbol{V}_2\boldsymbol{\Sigma}_2\boldsymbol{V}_2^T). \tag{26}$$

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 (25), where $\boldsymbol{\Sigma}_2$ and $\boldsymbol{V}_2$ are the eigenvalues far away from the threshold τ and their associated eigenvectors. According to (26), 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 (24). 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.

Figure 1: Singular values distribution of a 500×500 Gaussian random matrix



Algorithm 3: Algorithm for computing $\mathcal{P}_\tau(\boldsymbol{Z})$
Input: a symmetric nonnegative definite matrix $\boldsymbol{Z}$, a real 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 $\boldsymbol{P}_{k+1} = \frac{1}{2}\boldsymbol{P}_k + \frac{1}{4}\boldsymbol{Z} + \frac{3\tau}{4}\boldsymbol{I} - (2\boldsymbol{P}_k - \boldsymbol{Z} - \tau\boldsymbol{I})^{-1}(\tau\boldsymbol{P}_k - \frac{1}{4}\boldsymbol{Z}^2 - \frac{3\tau^2}{4}\boldsymbol{I})$
(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 the SVT $\mathcal{D}_\tau(\boldsymbol{Y})$. The algorithm is implemented in Matlab using mex programming. The computer is with an Intel Pentium 4 CPU at 3.00GHz and 1.49GB of memory, and the Matlab version is 7.6.0(R2008a). As mentioned in the introduction section, we do not intend to compete our algorithm with SVT via partial SVD, but with SVT via the full SVD which is implemented by calling the Matlab build-in function "svd". Our numerical examples show that our proposed algorithm saves more than 50% of computational time from SVT via the full SVD.

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 singular values distribution, 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 treated as easy example problems for our algorithm. Even so, 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, the relative difference between the final result and $\mathcal{P}_\tau(\boldsymbol{Y})$ via the full SVD is of precision of order 10^{-10}; see this in Table (1). 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 saves more than 50% computational time from the method via the full SVD. For example, when $n = 2000$, the computational time for our algorithm is 194 seconds, and that for SVT via the full SVD is 81.8 seconds.

Next, we test our algorithm for rectangular and singular matrices respectively. The results are shown

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

n	Our algorithm (Alg. 1)				Full SVD	Relative difference
	iters in Alg. 2	# eig's removed	iters in Alg. 3	total time (s)	total time (s)	
500	7	9.5	9	1.6	2.9	7.4×10^{-11}
1000	7	18.8	9	11.1	22.8	1.0×10^{-10}
1500	7	27.2	9	35.6	77.6	8.9×10^{-11}
2000	7	37.2	9	81.8	194	7.1×10^{-11}
2500	7	46.1	9	153	361	8.4×10^{-11}
3000	7	55.4	9	254	657	8.2×10^{-11}

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

size	Our algorithm (Alg. 1)				Full SVD	Relative difference
	iters in Alg. 2	# eig's removed	iters in Alg. 3	total time (s)	total time (s)	
1000×500	5	12.4	9	1.8	4.2	9.5×10^{-11}
2000×1000	5	25	9	12.8	33.5	1.1×10^{-10}
3000×1500	5	37.9	9	40.3	112	9.0×10^{-11}
1000×1000	7	15.9	9	9.8	18.9	8.3×10^{-11}
2000×2000	7	32	9	69.5	161	9.1×10^{-11}
3000×3000	7	44.7	9	221	545	7.9×10^{-11}

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 singular values distribution of Gaussian random matrices, these test problems are not treated as easy problems for our algorithm. From Table 2, we see again that our algorithm takes a small number of iterations to converge and saves more than 50% computational time from the method via the full SVD.

4 Conclusion and Discussion

In this paper, we proposed an algorithm to compute the singular value thresholding (SVT) for a given matrix. The algorithm is in two steps, namely, the polar decomposition step and the projection step, and both steps are done by Newton's method. Numerical experiments show that our algorithm is much faster than the method via the full singular value decomposition (SVD). Our algorithm saves more than 50% computational time from the method via the full SVD.

For future research, we may develop similar algorithms to compete with the method via partial 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 $\boldsymbol{S}^T\boldsymbol{Y}\boldsymbol{T} = \boldsymbol{B}$, 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.

Another possible future research topic is how to integrate the algorithm proposed in this paper with those SVT-based nuclear norm minimization algorithms. One may develop automatic schemes to choose what algorithm to be used for SVT according to the available rank information of the iteration matrices. We leave all these as future research topics.

References

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

[2] J.-F. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. Optimiz.*, 20(4):1956–1982, 2010.

[3] J.-F. Cai, S. Osher, and Z. Shen. Convergence of the linearized Bregman iteration for ℓ_1-norm minimization. *Math. Comp.*, 78(268):2127–2136, 2009.

[4] J.-F. Cai, S. Osher, and Z. Shen. Linearized Bregman iterations for compressed sensing. *Math. Comp.*, 78(267):1515–1536, 2009.

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

[6] E. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Arxiv preprint arXiv:0912.3599*, 2009.

[7] E. Candès and Y. Plan. Matrix completion with noise. *Proceedings of the IEEE*, 2009.

[8] E. Candès and Y. Plan. Tight oracle bounds for low-rank matrix recovery from a minimal number of random measurements. *Arxiv preprint arXiv:1001.0339*, 2010.

[9] E. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

[10] E. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *arXiv*, 903, 2009.

[11] A. Chambolle. An algorithm for total variation minimization and applications. *J. Math. Imaging Vision*, 20(1-2):89–97, 2004. Special issue on mathematics and image analysis.

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

[13] 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(11):1413–1457, 2004.

[14] D. L. Donoho. De-noising by soft-thresholding. *IEEE Trans. Inform. Theory*, 41(3):613–627, 1995.

[15] 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*, volume 3, 2003.

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

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

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

[19] N. Higham. Computing the polar decomposition – with applications. *SIAM J. Sci. Stat. Comput.*, 7(4):1160–1174, 1986.

[20] N. Higham and R. Schreiber. Fast polar decomposition of an arbitrary matrix. *SIAM J. Sci. Stat. Comput.*, 11(4):648–655, 1990.

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

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

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

[24] 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.

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

[26] 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(3):1235–1256, 2009.

[27] S. Ma, D. Goldfarb, and L. Chen. Fixed point and Bregman iterative methods for matrix rank minimization. *Mathematical Programming*, to appear.

[28] J.-J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *C. R. Acad. Sci. Paris*, 255:2897–2899, 1962.

[29] J.-J. Moreau. Proximitéet dualité dans un espace hilbertien. *Bull. Soc. Math. France*, 93:273–299, 1965.

[30] 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(2):460–489 (electronic), 2005.

[31] S. Osher, Y. Mao, B. Dong, and W. Yin. Fast linearized Bregman iteration for compressive sensing and sparse denoising. *Commun. Math. Sci.*, 8(1):93–111, 2010.

[32] B. Recht, M. Fazel, and P. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, to appear.

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

[34] Y. Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.

[35] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1):625–653, 1999.

[36] K. Toh, M. Todd, and R. Tutuncu. SDPT3 – a Matlab software package for semidefinite programming. *Optimization Methods and Software*, 11(12):545–581, 1999.

[37] K.-C. Toh and S. Yun. An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems. *Pacific J. Optimization*, to appear.

[38] 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(1):143–168, 2008.

[39] X. Yuan and J. Yang. Sparse and low-rank matrix decomposition via alternating direction methods. *preprint*, 2009.

[40] X. Zhang, M. Burger, and S. Osher. A unified primal-dual algorithm framework based on Bregman iteration. *J. Sci. Comput.*, to appear.