Speeding Up Singular Value Thresholding For Matrix Completion

Rishabh Ranawat

New York University

Abstract

Low rank matrix completion methods are extremely relevant and effective in scenarios where large matrices (datasets/real world data represented as a matrix) exist but are generally sparse due to missing entires. There has been significant progress made in the domain of matrix completion that has enabled the development of recommender systems on a large scale. One of the common approaches to matrix completion is to reduce it a nuclear norm minimization problem. There have been a number of effective algorithms proposed to tackle such problems. In this paper, we discuss a Singular Value Thresholding (SVT) algorithm for matrix completion proposed by Cai et. al. [1]. The SVT is a simple first-order and an easy to implement algorithm that is efficient at addressing problems that have low rank solutions. The algorithm is iterative and produces a sequence of matrices $\{X^k, Y^k\}$ and at each step, mainly performs a shrinkage based on the singular values. Further, we review the theory behind the algorithm and present some numerical results that back the theorems developed in the original paper. Finally, we try to do some exploration by observing specific variations of the algorithm. In particular, SVT computes the singular values of an iterate matrix per iteration. The authors use the SVD implementation from PROPACK (a FOTRAN library) []. We explore how using other K-SVD algorithms can affect the efficiency/time complexity of the algorithm. We evaluate the performance of the algorithm and its different SVD variants on 3 different datasets - (E1) randomly generated matrices (E2) matrix of Geodesic distances between cities (E3) student learning datasets.

Keywords: Optimization, Singular Value Decomposition, Matrix Completion, Nuclear Norm Minimization

1. INTRODUCTION

The matrix completion problem specifically for low rank matrices is visible in a number of areas such as recommender systems, computer vision, personalized learning, and other areas of engineering. The reason that there do exist such low rank representations of matrices associated with such problems is due to the inherent nature of the data collected. For instance, in the case of the movie recommendations. We know that a user's preferences are only dependent on a few factors and that intuitively hints at the fact that such a matrix would be low rank. An example is the geodesic distance matrix. Such a matrix is known to be low rank and an accurate low-rank representation can be computed efficiently. An area that I explore in this paper, is personalized learning. Specifically, we know from literature that there exists an innate pattern to student learning. So, a matrix consists of students vs. scores on a sequence of related questions could potentially be a low rank matrix.

In this paper, I review the Singular Value Thresholding Algorithm for Matrix Completion proposed by Cai et. al [1]. The SVT algorithm is a simple first order method that is straightforward to implement and primarily uses the idea of soft thresholding to solve the matrix completion problem. The algorithm produces a sequence of matrices $\{X^k, Y^k\}$ and the singular values of Y^k are shrunk using a shrinkage operator. The problem that this iterative algorithm tries to solve is that of minimizing the nuclear norm. This stems from the fact that the shrinkage operator (that we will introduce) is a proximity operator of the nuclear norm of a matrix. We present the algorithm, go through the theory and present some numerical analysis to make sure that the theory holds.

As we will observe the complexity of the SVT algorithm is dependent on the complexity of computing singular values of a matrix. In the paper, the authors use PROPACK (a FORTRAN library) to calculate k-SVDs. I could not get PROPACK to run on my machine (the last release was in 2007) and so I cannot comment on the efficiency of that implementation. However, we explore if other numerical ways of computing the SVD of a matrix would have an effect on two key aspects - time and accuracy. We experiment with MATLAB's svds function, block lanczos algorithm, fast randomized k-SVD. We provide brief descriptions of these algorithms (as the details are not really the main point of this paper) and present the results with some qualitative and quantitative analysis.

The paper is structured as follows: **section 2** A review of the Singular Value Thresholding Algorithm (SVT). This includes going over the theory, providing a basic implementation of the algorithm and presenting some numerical results. Next, **section 3** A summary of the three k-SVD methods - (i) Block Lanczos (ii) Randomized k-SVD (iii) Faster Randomized k-SVD. Finally, **section 4** - Evaluation on the three problems - 4.1 randomly generated matrices 4.2 matrix of Geodesic distances between cities 4.3 student learning datasets.

2. SINGULAR VALUE THRESHOLDING FOR MATRIX COMPLETION

In the following sections, we describe Singular Value Thresholding algorithm in relation to matrix completion in detail.

2.1. ***Nuclear Norm Minimization Problem***

The SVT algorithm is for approximately solving the nuclear norm minimization problem. These problems take the following form:

$$\begin{array}{ll} \text{minimize} & ||X||_* \\ \text{subject to} & \mathcal{A}(X) = b \end{array} \tag{1}$$

where, $\mathcal{A}$ is a linear operator acting on the space of $m \times n$ matrices and $b \in \mathbb{R}^p$. This algorithm is works well for problems that are at an extremely large scales and the solution has low rank. The SVT algorithm is formulated and presented in the case of matrix completion.

Let $\mathcal{P}_\Omega$ be the orthogonal projector onto the span of matrices vanishing outside of Ω so that the $(i,j)th$ component of $\mathcal{P}_\Omega(X)$ is equal to X_{ij} if $(i,j) \in \Omega$ and 0 otherwise. So, now we can state the matrix completion problem as the following:

$$\begin{array}{ll} \text{minimize} & ||X||_* \\ \text{subject to} & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{array} \tag{2}$$

where we are trying to find the optimal $X \in \mathbb{R}^{m\times n}$.

2.2. ***SVT Algorithm***

Now, that we have expressed the matrix completion problem as a minimization problem, we present the SVT algorithm. Before we can completely state algorithm we first need to discuss the singular value shrinkage operator.

Singular value shrinkage operator: We know that the singular value decomposition of a matrix $X \in R^{m\times n}$ [1] of rank r is given by the following:

$$X = U\Sigma V^T$$

The algorithm uses a soft-thresholding operator D_τ, where

$$D_\tau(X) = UD_\tau(\Sigma)V^T$$

and

$$D_\tau(\Sigma) = \Sigma = diag(\{\sigma_i - \tau\})_+$$

Here, what the operator essentially does is that it shrinks the singular values that are lower than τ to 0. This is similar to the soft-thresholding operator. The major implication that this is has is on the rank of the matrix X. For instance, for an extremely large value of τ, the matrix $D_\tau(X)$ would have a much lower rank than the matrix X.

D_τ as the proximity operator associated with the nuclear norm: First, let's define a proximity operator. For any function f that is closed, proper and convex, its proximity operator is defined as the following [2]:

$$P_f^\eta(Y) = \underset{X\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{2\eta}||X - Y||^2 + f(X) \tag{3}$$

where, $\mathcal{H}$ is any Hilbert space with the inner product $\langle ., . \rangle$ and the induced norm $||.||$.

In our case, in order for the singular value shrinkage operator to be the proximity operator the following condition must be satisfied:

$$D_\tau(Y) = \underset{X\in\mathbb{R}^{m\times n}}{\operatorname{argmin}} \frac{1}{2}||X - Y||_F^2 + \tau||X||_*$$

[1] U and V are respectively $m \times r$ and $n \times r$ with orthonormal columns.

[2] Following CMU CVX notes - `https://www.cs.cmu.edu/~suvrit/teach/yaoliang_proximity.pdf`

In order to prove the above statement, we first need to show the following lemma.

Theorem 2.1. *$\forall \tau \geq 0$, $Y \in \mathbb{R}^{m\times n} D_\tau(Y)$ obeys:*

$$D_\tau(Y) = \underset{X}{\operatorname{argmin}} \frac{1}{2}||X - Y||_F^2 + \tau||X||_*$$

Proof. First, we write the function on the RHS as the following:

$$h_0(X) = \tau||X||_* + \frac{1}{2}||X - Y||_F^2$$

Now, we can clearly see that this function is strictly convex and so we can be sure that it has a unique minimizer. In order to prove this theorem we need to show that $D_\tau(Y)$ is equal to this minimizer.

Let's say that $\tilde{X}$ minimizes $h_0(X)$. We know from the properties of a strictly convex function that $\tilde{X}$ minimizes $h_0(X)$ if and only if, $0 \in \partial h_0(\tilde{X})$ [3]

This suggests the following:

$$0 \in \tilde{X} - Y + \tau\partial||\tilde{X}||_* \tag{4}$$

Further, it has been shown in **Lemma Appendix A.1** that

$$\partial||X||_* = \{UV^T + W : W \in \mathbb{R}^{m\times n}, U^TW = 0, WV = 0, ||W||_2 \leq 1\} \tag{5}$$

Now, let $\tilde{X} := D_\tau(Y)$. We can decompose $Y = U_0\Sigma_0V_0^* + U_1\Sigma_1V_1^T$, where U_0, V_0 and U_1, V_1 are the singular vectors associated with singular values greater than τ and smaller than τ respectively. Substituting this notation in the definition:

$$\tilde{X} = U_0(\Sigma_0 - \tau I)V_0^T$$

[3]We say that Z is a subgradient of a convex function $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ at X_0, if

$$f(X) \geq f(X_0) + \langle Z, X - X_0\rangle$$

So, we can reach the following statement

$$\begin{aligned} Y - \tilde{X} &= U_0\Sigma_0 V_0^T + U_1\Sigma_1 V_1^T - U_0\Sigma_0 V_0^T - \tau U_0 V_0^T \\ &= U_1\Sigma_1 V_1^T - \tau U_0 V_0^T \\ &= \tau(U_0 V_0^T + W) \end{aligned}$$

where, $W = \tau^{-1}U_1\Sigma_1 V_1^T$. However, we know from **Lemma Appendix A.1** that $U_0^T W = 0$. Further, we know that Σ_1 is a diagonal matrix with the singular values $< \tau$. This means that $||W||_2 \leq 1$ because we have the inverse τ term there. This suggests that $Y - \tilde{X} \in \partial ||X||_*$. Well, this shows that 4 holds. Hence, we can be sure that the shrinkage operator is a proximity operator of the nuclear norm. □

Okay, so what can we do with this result? We'll get to that question after stating the SVT algorithm.

Algorithm: Now, we can state the SVT algorithm. First, we select a value of $\tau > 0$ and a sequence of $\{\delta_k\}$ of positive step sizes. We start with an initial value of the Y iterate and perform the following steps iteratively for k_{max} iterations:

$$\begin{aligned} &X^k = \mathcal{D}_\tau(Y^{k-1}) \\ &Y^k = Y^{k-1} + \delta_k P_\Omega(M - X^k) \end{aligned} \qquad (6)$$

The stopping criterion is the precision forced on the relative residual error. It can be stated as the following:

$$||P_\Omega(X^k - M)||_F / ||P_\omega M||_F \leq \epsilon$$

The L.H.S. is the relative residual error.

2.3. ***Convergence Analysis for Matrix Completion***

Now, we will show that by following 6, the sequence $\{X^k\}$ from 6 converges to a unique solution to the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & \tau ||X||_* + \frac{1}{2}||X||_F^2 \\ \text{subject to} \quad & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{aligned} \qquad (7)$$

Further, we can observe how this problem is closely related to 2. So, if we can show that the SVT algorithm converges to a unique solution to this problem, then in some sense we have shown that the SVT approximately solves the matrix completion problem. This is also because here f_τ is closely related to the proximity operator of the nuclear norm (which we showed was the shrinkage operator).

First, we need to establish the strong convexity of $f_\tau = \tau||X||_* + \frac{1}{2}||X||_F^2$. Consider the following:

Lemma 2.2. ***Let*** $Z \in \partial f_\tau(X)$ ***and*** $Z' \in \partial f_\tau(X')$***. Then,***

$$\langle Z - Z', X - X' \rangle \geq ||X - X'||_F^2 \tag{8}$$

Proof. We have the result that $Z = \tau Z_0 + Z$ iff $Z = \tau Z_0 + X$, wehere $Z_0 \in \partial||X||_*$.

$$\langle Z - Z', X - X' \rangle = \langle \tau Z_0 + X - \tau Z_0' + X', X - X' \rangle$$

$$= \tau Z_0 X - \tau Z_0' X - \tau Z_0 X' + \tau Z_0' X' + XX - 2XX' + X'X'$$

This can be written as the following using the definition of Frobenius norm and matrix inner product properties.

$$= \tau\langle Z_0 - Z_0', X - X' \rangle + ||X - X'||_F^2$$

Now, to prove the statement of the lemma if we can show that $\langle Z_0 - Z_0', X - X' \rangle \geq 0$ holds then that is sufficient. From **lemma Appendix A.1**, we know that $||Z_0|| \leq 1$ and $\langle Z_0, X \rangle = ||X||_*$. We can use these to derive the following inequality:

$$|\langle Z_0, X' \rangle| \leq ||Z_0||_2||X'||_* \leq ||X'||_*$$

Similarly, we end up getting:

$$|\langle Z_0', X \rangle| \leq ||X||_*$$

Finally, we can draw the following conclusion:

$$\langle Z_0 - Z_0', X - X' \rangle = \langle Z_0, X \rangle + \langle Z_0', X' \rangle - \langle Z_0, X' \rangle - \langle Z_0', X \rangle$$

$$= ||X||_* + ||X'||_* - \langle Z_0, X' \rangle - \langle Z_0', X \rangle \geq 0$$

Hence, we have proved that $\langle Z - Z', X - X' \rangle \geq ||X - X'||_F^2$.

$\square$

We need some additional background. We refer to [2] to make the link between the above stated lemma and the final convergence statement.

Theorem 2.3. *Let f be a differentiable function defined on a nonempty closed convex set $Q \in \mathbb{R}^n$, and let ∇f be Lipschitz continuous on Q with Lipschitz constant L, that is,*

$$||\nabla f(y) - \nabla f(x)||_2 \leq L||y - x||_2$$

for $x, y \in Q$. Given that we are trying minimize f, assume that there does exist a solution. In this case, the sequence $\{x^k\}$ generated by SVT (there is a more general way the paper states but SVT is part of that generalization) with stepsize δ_k satisfying

$$0 < \epsilon < \delta_k \leq 2/(L + 2\gamma), \epsilon, \gamma > 0$$

then if f is pseudo convex on Q, then x^k minimizes f as required. [2]

Thus, the following convergence theorem is reached using the **Theorem 2.3**.

Theorem 2.4. *The sequence $\{X^k\}$ obtained via 6 converges to the unique solution of 7 provided that $0 < inf \delta_k \leq sup \delta_k < 2$.*

Essentially, what the theorem is states is that given that we have proved strong convexity of the function $f_\tau(X)$, we know that the aforementioned minimization problem will converge to the unique solution of the problem if we follow the SVT algorithm iteration.

2.4. ***Key Features of SVT***

There are two main features of using SVT:

- **Low Rank Property** The iterates of X i.e., $\{X^k\}$ are generally low-rank. This result has not been proven analytically, but can be easily observed empirically. The primary reason for this behavior is the singular value shrinkage operator. By the nature of the algorithm, larger the τ is better suited for the algorithm. However, larger τ implies that a number of smaller singular values are going to get dropped off due to the shrinkage operator.

- **Sparsity** Another important characteristic is the fact that Y^k is sparse or rather it's sparsity is directly dependent on Ω (or the number of known entries). This suggests that in implementation we do not have to store a dense representation of the matrix.

2.5. ***Implementation of SVT***

The following is the most basic direct implementation of the SVT algorithm. Following is the pseudo code for the SVT algorithm:

Algorithm 1 SVT Algorithm

```
1: procedure SVT(Matrix, $\Omega$, $\tau$, $\delta$, $\epsilon$, maxiters)
2:     Yk = zeros(size(Matrix))
3:     for 1 $\rightarrow$ maxiters
4:         U, $\Sigma$, V = svd(Yk);
5:         $\Sigma = D_\tau(\Sigma)$
6:         Xk = $U\Sigma V^T$
7:         if Relative Residual Error < $\epsilon$: return Xk
8:         $Yk_{ij} = \begin{cases} 0 & \textbf{if}(i,j) \notin \Omega \\ Yk_{ij} + \delta(Matrix_{ij} - Xk_{ij}) & \textbf{if}(i,j) \in \Omega \end{cases}$
9:     end
10:    return Xk
```

The MATLAB implementation of the above algorithm can be found in **Appendix B**. The parameters above are set specific to the situation. We will discuss these in detail in the following section when we use the algorithm on different examples.

2.6. ***Numerical Experiments To Show Convergence***

Theorem 2.4 proves that the SVT algorithm is gauranteed to converge for the nuclear norm minimization problem (actually, the proximity operator minimization) given that the step size obeys the stated bound. In this section, we numerically test this theorem. We run the SVT algorithm on a randomly generated 1000×1000 matrix with rank 10 for a maximum of 100 iterations. The stopping criterion is when the relative residual error is less than $\epsilon = 1e - 4$. We test the theorem for a different set of fixed $\delta = 1 : 6$. The size of Ω was approximately 30% of the total number of elements. The MATLAB listings to run these experiments is provided in **Appendix C**. The following results were obtained:

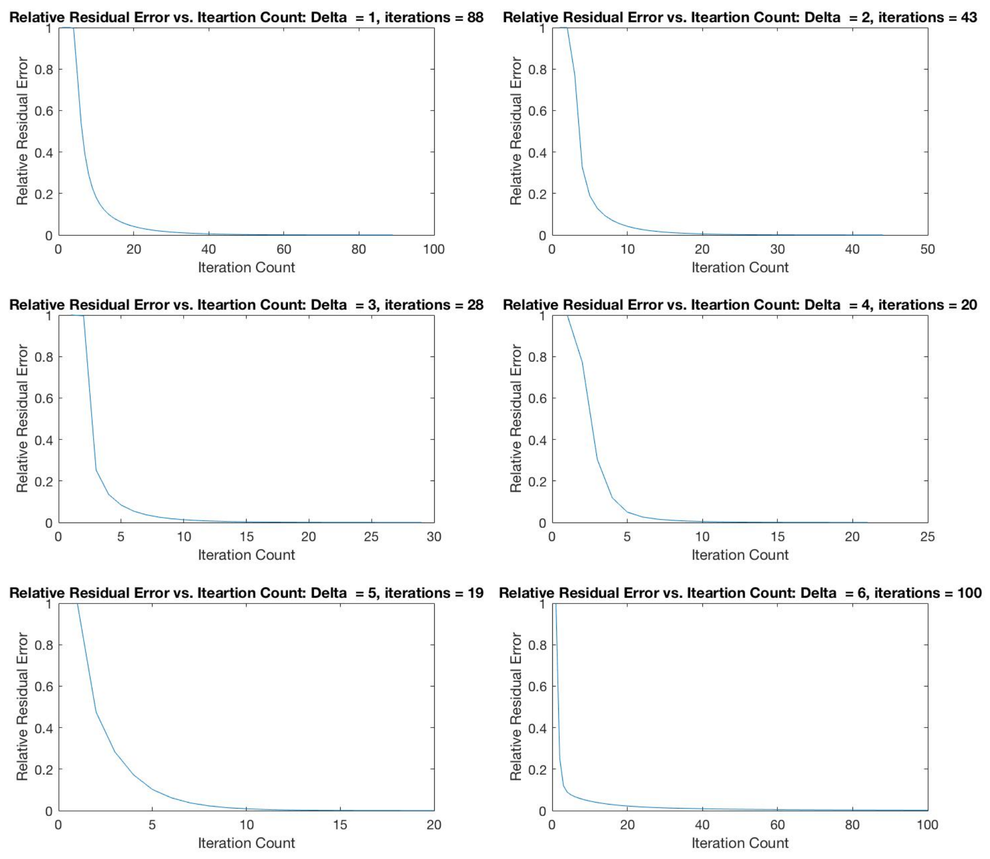

Figure 1: Testing the Convergence Theorem with varying δ for a random 1000 by 1000 matrix with rank 10 with a maximum number of iterations = 100

To summarize the results (in case the figure is too small): for $\delta \leq 5$, the iteration converged in less than 100 iterations. For $\delta = 6$, it didn't converge. This shows that the algorithm is extremely efficient and converges rather quickly. The time taken was reasonable (given in this version of the code we generate all SVDs at every step). One thing to note is that the bound that **Theorem 2.4** provides seems to be a sufficient condition and not necessary

or maybe it depends on how precise you expect your results to be. In this case, $\epsilon = 1e - 4$ and so maybe the iterations where we used a $\delta > 2$, also converged.
To summarize, we can be sure that the SVT algorithm works well as far as the minimization problem 2 is concerned. The theorem provides an analytical bound and the numerical results that were obtained seem to concur with the theory.

Now we explore a different aspect of the algorithm and that is the computation of SVDs in the iterations.

3. SUMMARY OF k-SVD ALGORITHMS

As we have noted, in **Algorithm 1** version in every iteration, we are computing all possible Singular Values of the Y^k iterate. This is inefficient and we can definitely improve on it. This is the primary motivation of this particular section.

3.1. ***Implementation Modification to SVT***

The idea here is that we only need to compute the singular values that are greater than the threshold τ cause anyway the other singular values are shrunk to 0. However, there still exists a problem - how do we know which singular values are greater than τ and then only compute those? In order to answer this question we take an incremental approach (the authors also propose such an approach [1]). The following is the pseudo code for such an implementation.

Algorithm 2 SVT Algorithm with kSVD

```
1: procedure SVT(Matrix, $\Omega$, $\tau$, $\delta$, $\epsilon$, k0, l, maxiters)
2:     rk = 0
3:     Yk = $k0 \times delta(P_\Omega(Matrix))$;
4:     for 1 $\rightarrow$ maxiters
5:         sk = rk + 1
6:         U, $\Sigma$, V = kSVD(Yk, k)
7:         while(cond)
8:             U, $\Sigma$, V = ksvd(Yk, k)
9:             $sings = diag(\Sigma)$;
10:            sk = sk + l;
11:            cond = sings(sk-l) > $\tau$
12:        end
13:        rk = $max\{j : \sigma_j > \tau\}$
14:        $Xk = \sum_{j=1}^{rk}(\sigma_j - \tau)U_jV_j$
15:        if Relative Residual Error < $\epsilon$: return Xk
16:        $Yk_{ij} = \begin{cases} 0 & \textbf{if}(i,j) \notin \Omega \\ Yk_{ij} + \delta(Matrix_{ij} - Xk_{ij}) & \textbf{if}(i,j) \in \Omega \end{cases}$
17:    end
18:    return Xk
```

As you may have noticed, there are two additional parameters. l is the incremental step for calculating the number of singular values and $k0$ is used for initialization of the Y^k iterate (details can be found in the MATLAB listing). The MATLAB listing for the above pseudo code is provided **Appendix D**. We try to answer the question - *What effect does using different k-SVD algorithms in the SVT iteration have on the matrix completion problem?*. In order to answer this question we review existing implementations of some k-SVD algorithms and implement them in the SVT context. In the following sections (3.2 3.3, 3.4), we briefly describe 3 approaches to computing $k-SVD$. We just give a brief overview of the algorithm and their run time complexity. We provide the appropriate citations to the papers that propose these algorithms.

3.2. ***MATLAB svds***

MATLAB offers an implementation of the K-SVD algorithm. FMATLAB does under the hood to calculate $k - SVDs$. It computes a given number of singular vectors via ARPACK and it is actually just a wrapper for an eigs call on the "squarized" matrix [4]. This would be a good baseline to start off with.

```
S = svds(A,K,'largest') computes K singular values based ...
    on SIGMA:
'largest' - compute K largest singular values. This is the ...
    default.
```

3.3. ***SVT with Block Lanczos SVD (SVT-BL)***

This is considered as the standard solution to the k-SVD problem and is the Block Lanczos Iteration algorithm [3].

Algorithm 3 Block Lanczos Algorithm

```
1: procedure SVD-BL(A, k)
2:     $s = k + O(1)$ (oversampling parameter)
3:     $q = O(log\frac{n}{\epsilon})$ maxiterations
4:     $S = Sketch(n, s)$
5:     $C = AS$
6:     $K = [C, (AA^T)C, \dots, (AA^T)^{q-1}C]$
7:     $[U, S, V] = svd(Q'A)$
8:     $U = QU$
```

Notation: Let $A \in \mathbb{R}^{m\times n}$ be the given matrix, $S \in \mathbb{R}^{n\times s}$ be a sketching matrix, e.g. random projection or column selection matrix, and $C = AS \in \mathbb{R}^{m\times s}$ be a sketch of A. The size of C is much smaller than A, but C preserves some important properties of A.

The above algorithm runs is $O(mnk)$ time, however, it takes up a lot of memory. Thus, for large scale purposes this might not be the best option.

3.4. ***SVT with Randomized k-SVD (SVT-R)***

Another class of algorithms are randomized algorithms for k-SVD approximation. A simple version is the following: [3]

[4]This was someone on stackexchange. I could not find any official documentation with regards to this

Algorithm 4 Randomized k-SVD

```
1: procedure SVD-RAND(A, k)
2:     $s = O(\frac{k}{\epsilon})$
3:     $S = Sketch(n, s)$
4:     $C = AS$
5:     $[Q, R] = qr(C)$
6:     $[U, S, V] = svds(Q'A, k)$
7:     $U = QU$
```

The motivation behind this algorithm is that if $C = AS \in \mathbb{R}^{m \times s}$ is a good sketch of A, the column space of C should roughly contain the columns of A_k this is the low rank approximation property. There is a theoretical bound provided for this here []. In this case, the time complexity of the algorithm is $O(nnz(A)k/\epsilon)$.

4. EVALUATION

Now, we have the main SVT algorithm sketched out and a few variants of it we evaluate these on 3 sets of problems. The first objective of this evaluation is to observe whether or not there algorithm performs efficiently and secondly, does using different k-SVD algorithms provide any benefit.

4.1. ***SVT for Random Matrix (E1)***

In this section, we run the SVT algorithm on a random 50 by 50 matrix with rank 3. We set the parameters as the following: τ is 100 times the highest singular value of the matrix and $\delta_k = 2$ i.e., fixed step size. Further, the max number of iterations were about 200 and the tolerance level was $1e - 4$.The size of Ω was approximately 30% of the total number of elements.
The following results were obtained:

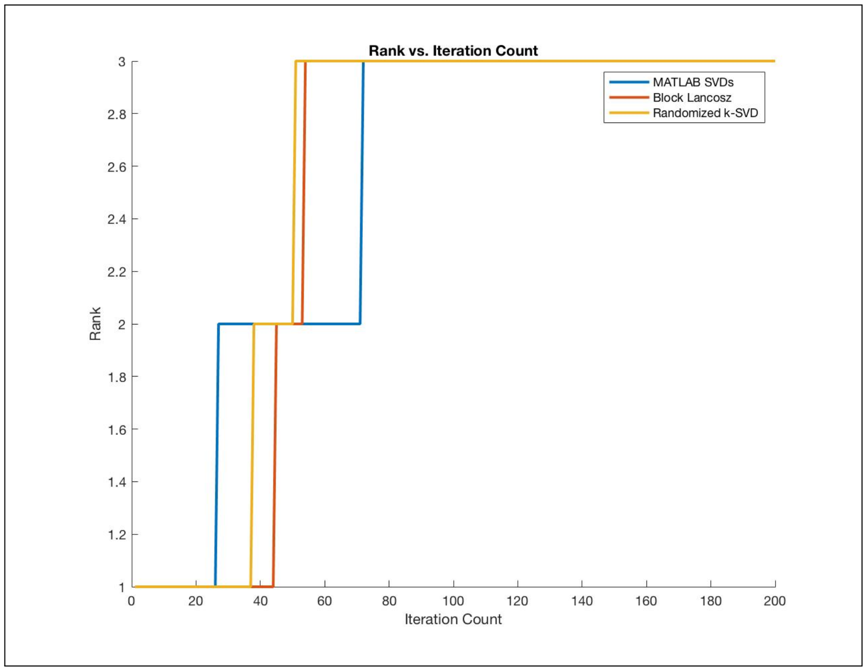

Figure 2: Rank vs. Iterations for a random 50 by 50 matrix with rank 3

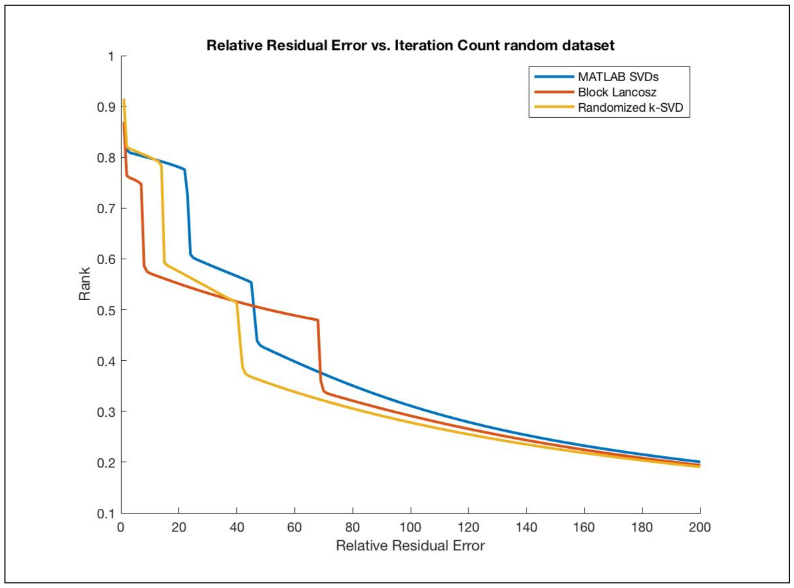

Figure 3: Relative Residual Error vs. Iteration Count for a random 50 by 50 matrix with rank 3

Observations The following were observed:

- The plots seem to concur with theory in that as the number of iterations increase, the rank also increases. Further, we can observe how the relative residual error drops fast as the number of iterations increase. This is an indication of the fact that a low rank matrix representation exists and can be approximated with a relatively low error.
- On comparing results with the results that the authors get, we can be sure that our implementation works correctly.
- As far as time is concerned, the following average (over 3 trials) results were obtained: 1.50s(MATLAB svds), 0.4830s (SVT-BL) 4.8646s (SVT-R). This suggests that SVT-BL performs the best as far as time is concerned.
- The algorithm seems to converge although we cut it out at 200 maximum iterations. However, one can observe that as the number of iterations increase the difference between the three SVD methods decreases.

4.2. ***SVT for Cities Dataset (E2)***

The authors use this algorithm to approximate a matrix of distances. Here, we try to reproduce those results by implementing it ourselves. The dataset has been provided by at *reference*. This dataset is essentially a 312 by 312 matrix where, the ij^{th} entry is the distance between the i^{th} and the j^{th} city. There have been numerical tests that clearly suggest that such geodesic matrices can be approximated as low rank matrices. It has been observed that a 3 low rank approximation of such a matrix gives a pretty good result i.e., $\frac{||M_3||_F}{||M||_F} = 0.9933$. We use the same parameters as the authors $\tau = 1e + 7$ and $\delta = 2$. The size of Ω was approximately 30% of the total number of elements.The following results were obtained:

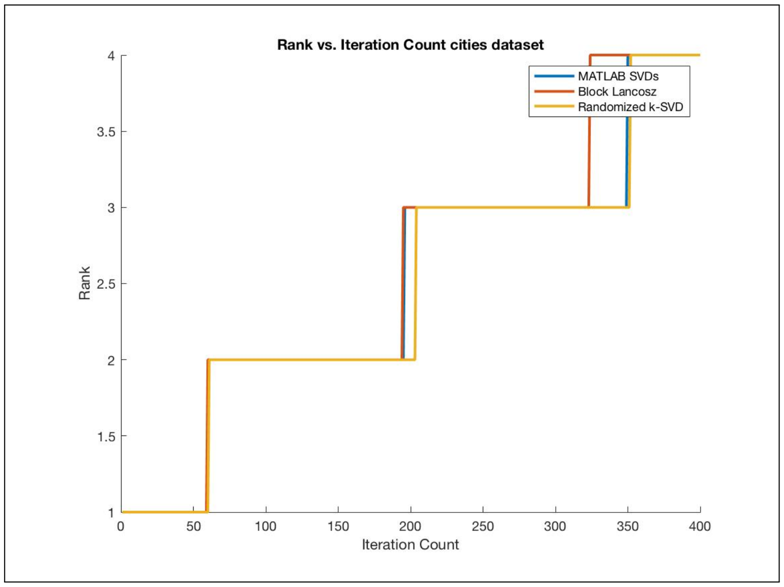

Figure 4: Rank vs. Iterations for the cities dataset

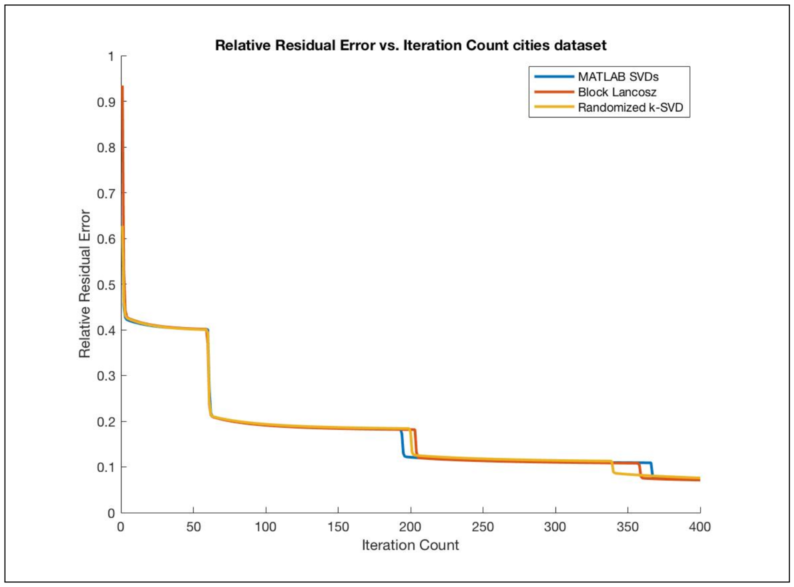

Figure 5: Relative Residual Error vs. Iteration Count for the cities dataset

Observations: Following are some of the observations:

- These results match our intuition that a reasonably accurate low rank approximation of the a distance matrix does exist. As we can see all three algorithms seem to converge. The final relative residual error was approximate 7.84e-02 at the 400th iteration.

- As far as time is concerned, the following average (over 3 trials) results were obtained: 6.43s (MATLAB svds), 5.34s (SVT-BL) 104.18s (SVT-R).

4.3. ***SVT for Education (E3)***

In this section, I describe the applications of the SVT algorithm in the educational domain. The primary idea as stated earlier is that there exists an underlying structure to the way a student learns and given the microscopic level at which today online platforms assess student learning, there is a massive amount of granular data available. In this section, I explore the application of matrix completion to educational data using the SVT algorithm.

The ASSISTments dataset consists of data collected on a personalized learning system [4]. This dataset consists of a student-response records and various attributes as far as questions, types of questions, outcomes, number of attempts and so on are concerned. In this part, we use the SVT algorithm to find a low rank matrix representation of a matrix that consists of the number of attempts a student takes to get a question correct. The matrix looks like the following:

$$NStudents \left\{ \overbrace{\begin{pmatrix} 1 & 2 & 0 & . & . & . & 4 & 1 \\ 0 & 0 & 1 & . & . & . & 1 & 0 \\ . \\ . \\ . \\ 0 & 4 & 2 & . & . & 3 & 1 & 1 \end{pmatrix}}^{QQuestions} \right.$$

where, the ij^{th} entry is the number of attempts a student took to get a question right. The dataset originally consists of 4200 students and 29000 questions. However, my machine cannot hold that big a matrix in memory. So, I randomly sample a matrix with 100 students and 100 questions.

Here, I am essentially trying to observe if there exists a low rank representation and if there does to what extent can we approximate it (i.e., reduce the relative residual error). The parameters used in this case were $\tau = 100 \times max(SVD(matrix))$ and $\delta = 2$. The size of Ω was approximately 30% of the total number of elements.

The following results were obtained:

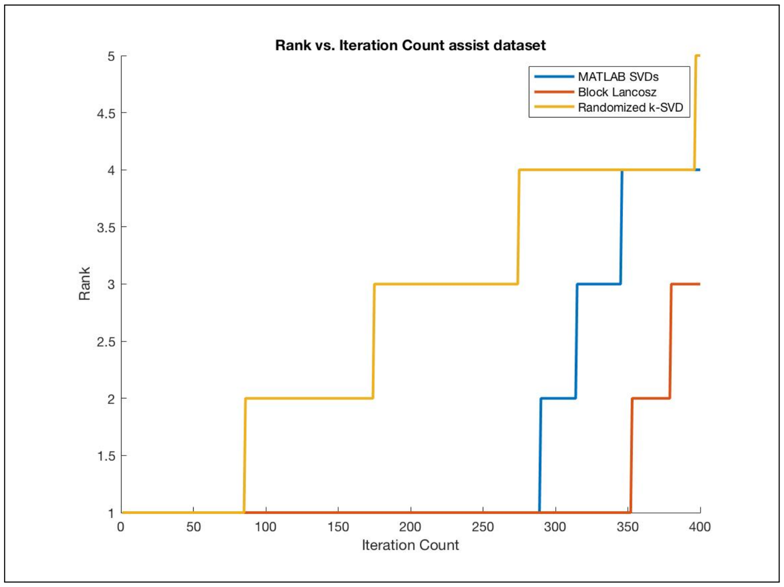

Figure 6: Rank vs. Iterations for the ASSIST dataset

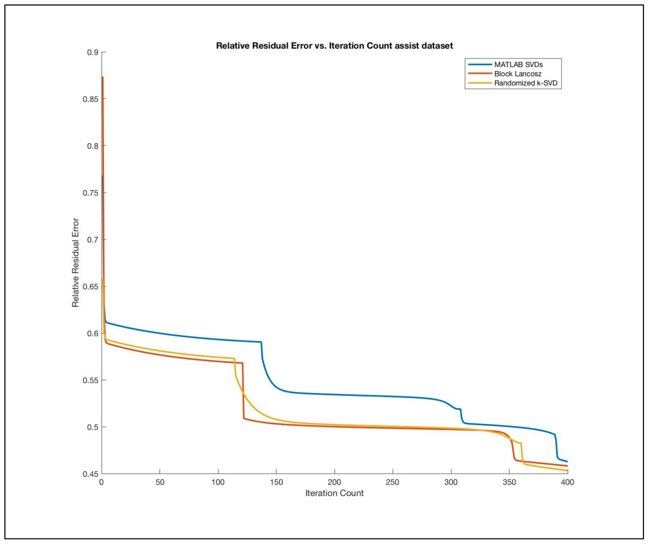

Figure 7: Relative Residual Error vs. Iteration Count for the ASSIST dataset

Observations: Following are some of the observations on the experiments for this dataset:

- The relative residual error is relatively high compared to our previous examples after 400 iterations. One of the reasons is probably that the rank of the matrix needs to be higher than 4 for a better approximation. This could be explored further if there was more compute power and one could let the experiment run to convergence.

- There is a significant discrepancy in the rank of the matrix and the number of iterations between different methods of computing SVD. This might be an indication of compounding error that gets magnified as the number of approximations increases. It is difficult to speculate as to which SVD performs the worst. However, assuming that MATLAB's version is accurate enough we might speculate that there could be a much better algorithm than the randomized version that we utilized.

- As far as time is concerned, the following average (over 3 trials) results were obtained: 4.84s(MATLAB svds), 2.62s (SVT-BL), 10.86s (SVT-R).

5. CONCLUSION

To conclude, in this paper we reviewed the Singular Value Thresholding Algorithm for Matrix Completion in detail. We presented the algorithm, convergence analysis, MATLAB implementation and numerical results to show its convergence. Further, we tried to experiment with different $k-SVD$ algorithms to see if they had any significant effect on the performance of SVT. In order to do so, we looked at a few different algorithms such as the Block-Lanczos iteration algorithm, the randomized $k-SVD$ algorithm and of course, MATLAB's inbuilt *svds* algorithm. Overall, the SVT algorithm is extremely efficient at approximating low rank matrices and can be used in a variety of applications. We also observed that the cities-distance matrix is a low rank matrix and one could obtain an approximate low rank representation of the education dataset if there was additional compute power available.

Future Work: In this paper, I only worked through the equality constraint in the optimization formulation. However, this can be easily extended to inequality constraints. It would be interesting to try examples in that domain as well. I would like to explore if we could formulate the optimization problem by taking certain scenarios into context. For instance, for the education dataset maybe a slight variation to the nuclear norm minimization method would provide better results. Also, I would like to get into the details of more k-svd algorithms and do a more in depth analysis of these. There has been literature that tries to make SVT faster by using rank revealing SVD methods [5]

6. REFERENCES

[1] J. Cai, E. J. Candès, Z. Shen, A singular value thresholding algorithm for matrix completion, SIAM Journal on Optimization 20 (2010) 1956–1982.

[2] Y. C. Cheng, On the gradient-projection method for solving the nonsymmetric linear complementarity problem, Journal of Optimization Theory and Applications 43 (1984) 527–541.

[3] S. Wang, A practical guide to randomized matrix computations with MATLAB implementations, CoRR abs/1505.07570 (2015).

[4] D. Selent, T. Patikorn, N. Heffernan, Assistments dataset from multiple randomized controlled experiments, in: Proceedings of the Third (2016)

ACM Conference on Learning @ Scale, L@S '16, ACM, New York, NY, USA, 2016, pp. 181–184.

[5] H. Ji, W. Yu, Y. Li, A rank revealing randomized singular value decomposition (R3SVD) algorithm for low-rank matrix approximations, CoRR abs/1605.08134 (2016).

Appendix A. Additional Lemmas

Lemma Appendix A.1. [5] *For some matrix $A \in \mathbb{R}^{m\times n}$ the*

$$\partial||X||_* = \{UV^T + W : U^TW = 0, WV = 0, ||W||_2 \leq 1, W \in \mathbb{R}^{m\times n}\}$$

where $A = U\Sigma V^T$.

Proof. An important feature is that at each A, the subdifferential of the nuclear norm admits a subspace $\mathcal{T}$ upon which it can be decomposed.

$$\mathcal{T} = \{UY^T + XV^T : X \in \mathbb{R}^{m\times r}, Y \in \mathbb{R}^{n\times r}\} \cap \{\text{matrices with orthonormal rows}\}$$

Now, let $\Pi_{\mathcal{T}}$ and $\Pi_{\mathcal{T}^\perp}$ denote projections upon $\mathcal{T}$ and its orthogonal complement. Now we know that two matrices P and Q are orthogonal if and only if $\forall \mu : ||A + \mu B|| \geq ||A||$. So, we have the following:

$$||UY^T + XV^T + \mu W||^2 \geq ||UY^T + XV^T||^2$$

Thus, we can be sure that $\mathcal{T}$ and W are orthogonal.
Okay, so now we can write the characterization as:

$$\partial||A||_* = \{Z : \Pi_{\mathcal{T}}(Z) = UV^T, ||\Pi_{\mathcal{T}^\perp} \leq 1\}$$

Further, using the dual norm of the nuclear norm:

$$\partial||A||_* = \{Z : \langle Z, A\rangle = ||A||_*, ||Z||_*^* \leq 1\}$$

Further, we know that the trace norm's dual is the spectral norm. We get:

$$\partial||A||_* = \{Z : \langle Z, A\rangle = ||A||_*, ||Z||_2 \leq 1\}$$

And now, using some linear algebra techniques it can be shown that: $\langle Z, A\rangle = ||A||_*$ and $||Z||_2 \leq 1$.
Hence, shown.

□

[5] Credit to CMU CVX Notes

Appendix B. MATLAB Code for Direct SVT Implementation

```matlab
% Singular Value Thresholding Algorithm for Marix Completion
% The following is a direct implementation without ...
    worrying about
% efficiency.
function [Xopt, ranks, relativeResidualError, ...
    relativeError, nucs, iters] = ...
     SVTAlgorithm(matrix, mask, tau, Δ, epsilon, maxiters)

     % records for plotting
     ranks = [];
     relativeResidualError = [];
     relativeError = [];
     nucs = [];

     % Y^{k} matrix initialization
     Y = zeros(size(matrix));

     for i = 1:maxiters
         % compute the SVD
         [U, S, V] = svd(Y);

         % shrinkage Operator. Essentially, the singular values
         % that are less than tau are shrunk to 0.
         S = max(S−tau, 0);

         % the X^{k} iterate only comprises is a product of
         % the left and right singular vectors that ...
             correspond to above
         % threshold singular values.
         X = U*S*V';

         % orthogonal projection on to omega
         Y = Y + Δ*mask.*(matrix−X);

         % relative residual error for stopping criterion
         rse = norm(mask .* (X − matrix), 'fro') / ...
             norm(mask .* matrix, 'fro');

         % recording for plots
         relativeResidualError = [relativeResidualError rse];

         % stopping criterion and debugging messages
```

```
        if(rse < epsilon)
            Xopt = X;
            fprintf('error %d, iteration %d rank %d \n', ...
                rse, i, rank(Xopt))
            return
        else
            fprintf('error %d, iteration %d rank \n', rse, i)
        end
        iters = i;
    end
    Xopt = X;
end
```

Appendix C. MATLAB Code for Running Convergence Experiments

```
function [] = convergenceExperiements(M , N, r, maxiters)
    randn('state',1000);

    matrix = randn(M,r)*randn(r,N);

    tau = 1e+3;
    epsilon = 1e-4;

    omega_size = round(0.3*M*N);
    mask = zeros(size(matrix));
    pair_ixs = randperm(M * N, omega_size);
    row_ixs = mod(pair_ixs, M) + 1;
    col_ixs = floor((pair_ixs - 1) / M) + 1;
    mask(sub2ind([M, N], row_ixs, col_ixs)) = 1;

    figure(1);
%      title(sprintf('Rank vs. Iteration Count for varying △...
    '));
    for △ = 1:6
        [Xopt, ranks, relativeResidualError, ...
            relativeError, nucs, iters] = ...
            SVTAlgorithm(matrix, mask, tau, △, epsilon, ...
            maxiters);
        subplot(3, 2, △);
        plot(1:size(relativeResidualError, 2), ...
            relativeResidualError);
        title(sprintf('Relative Residual Error vs. ...
            Iteartion Count: Delta  = %d, iterations = ...
            %d', Δ, iters));
        xlabel('Iteration Count');
        ylabel('Relative Residual Error');
    end



end
```

Appendix D. MATLAB Code for Running k-SVD Experiments

```
% The followng is the implementation of the kSVTAlgorithm.
function [Xopt, ranks, relativeResidualError, ...
    relativeError] = ...
    kSVTAlgorithm(matrix, mask, tau, Δ, epsilon, maxiters, ...
        svdtype)

    % As suggested by the author
    k0 = ceil(tau/(Δ*norm(mask.*matrix, 2)));

    % skip size for singular values
    l = 5;
    rk = 0;

    % plotting data
    ranks = [];
    relativeResidualError = [];
    relativeError = [];

    % Y initialization
    Y = k0*Δ*(mask.*matrix);

    for i = 1:maxiters
        sk = rk + 1;

        % calculate only sk singular values
        cond = sk < min(size(matrix));
        while(cond)
            switch svdtype
                case 1
```

```
                    % MATLAB svds
                    [U, S, V] = svds(Y, sk, 'largest');
                case 2
                    % Block Lanczos Method
                    [U, S, V] = BLSVD(Y, sk, ...
                       round(log(size(matrix, 2)/epsilon)));
                case 3
                    % Randomized SVD
                    [U, S, V] = rSVD(Y, sk, ...
                       round(sk/epsilon));
                otherwise
                    [U, S, V] = svds(Y, sk, 'largest');
            end
            sings = diag(S);
            sk = sk + l;

            cond = (sings(sk-l) > tau && sk < ...
               min(size(matrix)));

        end

        % update rk
        for j = 1:size(sings, 1)
            if(sings(j) > tau)
                rk = j;
            else
                break
            end
        end

        % update X iterate
        X = zeros(size(matrix));
        for j = 1:rk
            X = X + (sings(j)-tau)*U(:, j)*V(:, j)';
        end

        % rse calculation
        rse = norm(mask .* (X - matrix), 'fro') / ...
           norm(mask .* matrix, 'fro');

        ranks = [ranks rank(X)];
        relativeResidualError = [relativeResidualError rse];
        relativeError = [relativeError norm(X-matrix, ...
           'fro')/norm(matrix, 'fro')];

        if(rse < epsilon)
            Xopt = X;
            fprintf('error %d, iteration %d rank %d \n', ...
                rse, i, rank(Xopt))
            return
        else
            fprintf('error %d, iteration %d rank %d \n', ...
                rse, i, ranks(i))
        end

        % orthogonal projection for Y
        Y = mask.*(Y + Δ*(matrix-X));
    end
    Xopt = X;
end
```