Limited Memory Block Krylov Subspace Optimization for Computing Dominant Singular Value Decompositions

Xin Liu* Zaiwen Wen† Yin Zhang‡

March 22, 2012

Abstract

In many data-intensive applications, the use of principal component analysis (PCA) and other related techniques is ubiquitous for dimension reduction, data mining or other transformational purposes. Such transformations often require efficiently, reliably and accurately computing dominant singular value decompositions (SVDs) of large unstructured matrices. In this paper, we propose and study a subspace optimization technique to significantly accelerate the classic simultaneous iteration method. We analyze the convergence of the proposed algorithm, and numerically compare it with several state-of-the-art SVD solvers under the MATLAB environment. Extensive computational results show that on a wide range of large unstructured matrices, the proposed algorithm can often provide improved efficiency or robustness over existing algorithms.

Keywords. subspace optimization, dominant singular value decomposition, Krylov subspace, eigenvalue decomposition

1 Introduction

Singular value decomposition (SVD) is a fundamental and enormously useful tool in matrix computations, such as determining the pseudo-inverse, the range or null space, or the rank of a matrix, solving regular or total least squares data fitting problems, or computing low-rank approximations to a matrix, just to mention a few. The need for computing SVDs also frequently arises from diverse fields in statistics, signal processing, data mining or compression, and from various dimension-reduction models of large-scale dynamic systems. Usually, instead of acquiring all the singular values and vectors of a matrix, it suffices to compute a set of dominant (i.e., the largest) singular values and their corresponding singular vectors in order to obtain the most valuable and relevant information about the underlying dataset or system. The purpose of this paper

*Academy of Mathematics and Systems Science, Chinese Academy of Sciences, CHINA (liuxin@lsec.cc.ac.cn). Research supported in part by NSFC grant 11101409 and 10831006.

†Department of Mathematics and Institute of Natural Sciences, Shanghai Jiaotong University, CHINA (zw2109@sjtu.edu.cn). Research supported in part by NSFC grant 11101274.

‡Department of Computational and Applied Mathematics, Rice University, UNITED STATES (yzhang@rice.edu). Research supported in part by NSF Grant DMS-0811188, ONR Grant N00014-08-1-1101, and NSF Grant DMS-1115950.

is to present an algorithm for efficiently, reliably and accurately computing dominant SVDs of *large and unstructured* matrices.

To fix notation, let us consider a real matrix $A \in \mathbb{R}^{m\times n}$ ($m \leq n$ without loss of generality) and a given positive integer $k \ll \min(m,n)$. The task is to find two orthogonal matrices $U_k = [u_1, \ldots, u_k] \in \mathbb{R}^{m\times k}$ and $V_k = [v_1, \ldots, v_k] \in \mathbb{R}^{n\times k}$, and a diagonal matrix $\Sigma_k \in \mathbb{R}^{k\times k}$ whose diagonal entries are the k largest singular values of A, say, $\sigma_1 \geq \cdots \geq \sigma_k \geq 0$, such that

$$A_k \triangleq \sum_{i=1}^{k} \sigma_i u_i v_i^{\mathrm{T}} = \underset{\mathbf{rank}(W)\leq k}{\arg\min} \|A - W\|_{\mathrm{F}}^2, \tag{1}$$

where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm for matrices. For convenience, we will refer to such an approximation $A \approx A_k = U_k \Sigma_k V_k^{\mathrm{T}}$ as the k-th dominant (or truncated) singular value decomposition of A, or simply the dominant SVD without explicitly mentioning the number k, which is typically much smaller than m.

During decades of research, numerous iterative algorithms have been developed for computing dominant SVDs of A, mostly based on computing eigen-pairs (i.e., eigenvalues and eigenvectors) of the symmetric matrix $B = AA^{\mathrm{T}}$ (or $A^{\mathrm{T}}A$), or alternatively those of

$$B = \begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix},$$

with or without additional shifts. The classic simple subspace, or simultaneous, iteration method, extends the idea of the power method which computes the largest eigenvalue and its eigenvector (see [16, 17, 24, 26] for example), performing repeated matrix multiplication followed by orthogonalization. More elaborative algorithms including Arnoldi methods [14, 13], Lanczos methods [21, 12], Jacobi-Davidson methods [23, 2], to cite a few for each type, are specifically designed for large-scale but sparse matrices or other types of structured matrices.

Traditionally, research on large-scale eigenvalue (or singular value) problems has primarily focused on computing a subset of eigenvalues and eigenvectors (or singular values and vectors) of large sparse matrices. Special algorithms for computing dominant SVDs of large unstructured, hence dense, matrices seem few and far between. This state of affair is evident when examining survey articles and reference books on eigenvalue and/or singular value computations (e.g., [19, 7, 25, 22, 10]). On the other hand, large-scale but unstructured matrices do arise from applications, such as image processing and computational materials science (e.g., [27, 20]), which involve large amounts of data even though their dimensions are relatively small compared to those of large sparse matrices. This is especially true in recent years when the demand for computing dominant SVDs of large unstructured matrices has become increasingly pervasive in data-intensive applications. Clearly, for any large enough dense matrix A, the approach of computing the full SVD of A then truncating to the k-th dominant SVD is not a practical option when k is considerably smaller than the sizes of A.

In some applications, it suffices to have rough approximations to data matrices using not-so-accurate dominant SVDs. One approach for computing approximate SVDs is to use the Monto Carlo method [6]

which approximates the dominant singular values and corresponding singular vectors by extracting a submatrix from the target matrix. Recently, a randomized method for approximating dominant SVDs is analyzed in [9] where one applies the simultaneous iteration once or a few times, starting from a randomly generated initial guess. In these two works, theoretical results are developed to estimate approximation quality.

In the present paper, we aim to develop a general-purpose algorithm for computing dominant SVDs of large unstructured matrices, to a high precision if so desired. Our approach is using a *block Krylov subspace optimization* technique, to be explained below, to accelerate the basic simultaneous iteration method for computing the k largest eigenvalues of AA^{T} (or $A^{\mathrm{T}}A$) and their corresponding eigenvectors that we will also refer to as the *leading eigenvectors*.

It is well known that the k leading eigenvectors of AA^{T} maximize the following Rayleigh-Ritz function under orthogonality constraint:

$$\max_{X\in\mathbb{R}^{m\times k}} \ \|A^{\mathrm{T}}X\|_{\mathrm{F}}^2, \text{ s.t. } \ X^{\mathrm{T}}X = I. \tag{2}$$

In our proposed approach, we utilize (2) and a subspace optimization technique to accelerate the classic simple subspace iterations. Specifically, in addition to the usual subspace iteration in which the current iterate of X is multiplied by AA^{T}, we also solve a restricted version of (2) in a subspace spanned by a few previous iterates of X. For example, at iteration i, we may add an extra subspace constraint to (2) so that

$$X \in \mathbf{span}\left\{X^{(i-1)}, AA^{\mathrm{T}}X^{(i-1)}\right\}$$

where the above inclusion means that all the k columns of X are linear combinations of the $2k$ columns on the right-hand side. In our approach, the number of previous iterates, or *the length of the memory*, can be adjusted from iteration to iteration. Recall that a Krylov subspace generated by a matrix B and a vector x is defined as the span of $\{x, Bx, B^2x, \cdots, B^{d-1}x\}$ for some positive integer d. Therefore, we will call our approach a *limited memory block Krylov subspace optimization* technique. A key observation is that, like the classic simple subspace iteration method, the proposed algorithm only requires one matrix-block multiplication of the form $AA^{\mathrm{T}}X$ per iteration; that is, the subspace optimization step requires no extra matrix-block multiplication of this kind.

1.1 Contributions

The classic simple subspace (or simultaneous) iteration (SSI) method has two main advantages: (i) the use of simultaneous matrix-block multiplications instead of individual matrix-vector multiplications enables fast memory access and parallelizable computation on modern computer architectures, (ii) when computing a sequence of closely related eigen-pairs, starting from a block matrix facilitates the reuse of available eigenspace information in the form of warm-start strategies. However, a severe shortcoming of the SSI method is that its convergence speed depends greatly on eigenvalue distributions that becomes intolerably slow with unfavorable eigenvalue distributions. This shortcoming basically prevents the SSI method from being used as a general-purpose solver. A few SSI acceleration techniques exist, such as the Chebyshev acceleration (e.g., [18, 25]), but their effectiveness and applicability remain limited. Presently, general-purpose

solvers for large-scale eigenvalue/singular value problems are largely designed for sparse or otherwise structured matrices.

In this work, we propose and study an SSI-based but much accelerated algorithm for computing dominant SVDs of large unstructured matrices. At each iteration, the algorithm maximizes the Rayleigh-Ritz function in a subspace spanned by a few previous iterate blocks with an adjustable dimension. In doing so, it does not require any additional matrix-block multiplication. The algorithm is of a general-purpose type capable of obtaining highly accurate solutions, as opposed to only rough approximations. Extensive numerical experiments have been conducted to verify the effectiveness of the algorithm with different singular value distributions. Computational evidence shows that, under the Matlab environment, a simple implementation of the proposed algorithm, called LMSVD, already outperforms some state-of-the-art SVD solvers on a wide range of test matrices within the target class.

The idea of subspace optimization has often been employed in optimization such as in nonlinear programming (see, for example, [8, 4, 30]). A more closely related work is "Locally Optimal Block Preconditioned Conjugate Gradient Method" (LOBPCG) proposed by Knyazev [11] for solving generalized eigenvalue problems. The subspace in each iteration of LOBPCG consists of the current iterate $X^{(i)}$, the previous iterate $X^{(i-1)}$ and a residual vectors (possibly preconditioned). Our choice of subspace is more general and its dimension is adjustable. Moreover, the subproblems we choose to solve are small eigenvalue problems, while LOBPCG solves small generalized eigenvalue subproblems. Since both approaches use subspace optimization, in this paper, we will present numerical comparison results with LOBPCG. Another work by Yang, Meza and Wang [28] applies a special-purpose, three-block subspace optimization technique to a nonlinear eigenvalue problems in electronic structure computation. Computational results are reported in both [11] and [28], but few theoretical results are given.

This paper also includes a convergence analysis for our proposed acceleration method for SSI. Due to the introduction of intermediate variables and additional structural changes, the convergence analysis for the original SSI method no longer applies to our approach. Using a different analysis, we are able to obtain global convergence results under reasonable assumptions.

1.2 Organization

The rest of this paper is organized as follows. We describe the motivation of our approach, the algorithmic framework and the detailed algorithm in section 2; our convergence analysis is presented in section 3; and several implementation issues are discussed in section 4. Extensive numerical results are presented in section 5 to evaluate the robustness and efficiency of the proposed algorithm. Finally, we conclude the paper in section 6.

1.3 Notations

The trace of a matrix $M \in \mathbb{R}^{n\times n}$ is denoted by $\mathbf{trace}(M)$, and the main diagonal of M is denoted by $\mathbf{diag}(M)$. Given a symmetric matrix $M \in \mathbb{R}^{n\times n}$, $\mathbf{\Lambda}(M)$ is a diagonal matrix whose diagonal elements are the eigenvalues M in descending order. Superscripts for matrices are iteration counters if they are in

parentheses; otherwise they are exponents. Further notations will be explained as they appear during the progress of the paper.

2 Limited Memory Block Krylov Subspace Optimization

Let us reiterate that our goal is to compute the k-th dominant SVD of a matrix $A \in \mathbb{R}^{m\times n}$ as is defined in problem (1), and our method is based on accelerating the simple subspace (or simultaneous) iteration (SSI) method via solving problem (2) in a chosen subspace at each iteration.

2.1 Motivation and Framework

Starting from an initial point $X^{(0)} \in \mathbb{R}^{m\times k}$, SSI computes the next iterate $X^{(i+1)}$ by the formula

$$X^{(i+1)} \in \mathbf{orth}\left(AA^{\mathrm{T}}X^{(i)}\right), \tag{3}$$

where $\mathbf{orth}(M)$ denotes the set of orthonormal bases for the range space of M. As such, the iterates of SSI, with a possible exception for the initial guess, satisfy the orthogonality condition $X^{\mathrm{T}}X = I$. When $k = 1$, the SSI method reduces to the well-known power method for computing the single largest eigenvalue and its eigenvector. In the SSI method, the orthonormalization step is indispensable (for example, see [19] for more details).

For an unstructured matrix A, the computational costs of the matrix-block multiplication (i.e., $AA^{\mathrm{T}}X$) and orthonormalization in (3) are $O(nmk)$ and $O(mk^2)$, respectively. In most applications, the approximating rank k is far less than the dimension m. Hence, the matrix-block multiplications of the type $AA^{\mathrm{T}}X$ constitute the dominate computational cost of SSI. Obviously, an acceleration will be achieved if one can reduce the number of iterations without having to incur extra matrix-block multiplications or other significant overhead. To achieve the goal of reducing the number of iterations, we propose to modify the basic SSI framework as follows. We replace the last iterate $X^{(i)}$ in the right-hand side of (3) by an "improved" intermediate iterate $\hat{X}^{(i)}$ so that

$$X^{(i+1)} \in \mathbf{orth}\left(AA^{\mathrm{T}}\hat{X}^{(i)}\right), \tag{4}$$

where, for a chosen subspace $\mathcal{S}^{(i)}$ with a block Krylov subspace structure,

$$\hat{X}^{(i)} := \underset{X\in\mathbb{R}^{m\times k}}{\arg\max} \ \|A^{\mathrm{T}}X\|_{\mathrm{F}}^2, \ \text{ s.t. } X^{\mathrm{T}}X = I, \ X \in \mathcal{S}^{(i)}. \tag{5}$$

Again, $X \in \mathcal{S}^{(i)}$ means all columns of X are from the subspace $\mathcal{S}^{(i)}$. We will soon specify the definition of $\mathcal{S}^{(i)}$ and show that the cost of solving (5) can be kept relatively low when the dimension of the subspace $\mathcal{S}^{(i)}$ is relatively small.

We now give the framework of our proposed algorithm that will be called LMSVD-F for ease of reference. Further details about this framework will be specified in the next subsection.

Algorithm 1: Framework LMSVD-F

Input A and k, initialize $X^{(0)} \in \mathbb{R}^{m\times k}$ and set $i = 0$.
while *"not converged"* **do**
  Specify the block Krylov subspace $\mathcal{S}^{(i)}$;
  `/* Block Subspace Optimization */`
  Solve $\hat{X}^{(i)} := \arg\max\{\|A^{\mathrm{T}}X\|_{\mathrm{F}}^2 : X^{\mathrm{T}}X = I,\ X \in \mathcal{S}^{(i)}\}$;
  `/* Simultaneous Iteration */`
  Compute $X^{(i+1)} \in \mathbf{orth}(AA^{\mathrm{T}}\hat{X}^{(i)})$;
  Increment i and continue.

2.2 Algorithm Details

We now describe the selection of the subspace $\mathcal{S}^{(i)}$ which is constructed from a *limited memory* of the last a few iterates. Its choice is of course not unique. We first consider the subspace spanned by the current i-th iterate and the previous p iterates; i.e.,

$$\mathcal{S}^{(i)} := \mathbf{span}\left\{X^{(i)}, X^{(i-1)}, ..., X^{(i-p)}\right\}, \tag{6}$$

where the *memory length* $p \geq 0$ will be specified in Section 4.1.

We collect the current and the other p saved iterate blocks in (6) into a matrix

$$\mathbf{X} = \mathbf{X}_p^{(i)} := \left[X^{(i)}, X^{(i-1)}, ..., X^{(i-p)}\right] \in \mathbb{R}^{m\times q} \tag{7}$$

where $q = (p+1)k$ is the total number of columns in $\mathbf{X}_p^{(i)}$. For notational simplicity, from here on we often choose to drop the superscript and subscript from quantities like $\mathbf{X}_p^{(i)}$ whenever no confusion would arise. Also note that the collection matrix $\mathbf{X}$ is boldfaced to make it distinct from its blocks. Similarly, we define

$$\mathbf{Y} = \mathbf{Y}_p^{(i)} := A^{\mathrm{T}}\mathbf{X}_p^{(i)} := \left[A^{\mathrm{T}}X^{(i)}, A^{\mathrm{T}}X^{(i-1)}, ..., A^{\mathrm{T}}X^{(i-p)}\right] \in \mathbb{R}^{m\times q}, \tag{8}$$

which is also saved in memory. We emphasize that SSI already computes these blocks in $\mathbf{Y}$ and we only need to save them once computed.

It is clearly that $X \in \mathcal{S}^{(i)}$ if and only if $X = \mathbf{X}V$ for some $V \in \mathbb{R}^{q\times k}$, and the subspace optimization problem (5) is equivalent to a generalized eigenvalue decomposition problem:

$$\max_{V\in\mathbb{R}^{q\times k}} \|(A^{\mathrm{T}}\mathbf{X})V\|_{\mathrm{F}}^2, \ \text{ s.t. } \ V^{\mathrm{T}}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)V = I. \tag{9}$$

However, numerical difficulty may arise in solving (9) as the matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ can become numerically rank deficient. A more stable approach, which we will implement, is to find an orthonormal basis for $\mathcal{S}^{(i)}$, say,

$$\mathbf{Q} = \mathbf{Q}_p^{(i)} \in \mathbf{orth}\left(\mathbf{X}_p^{(i)}\right),$$

and to express a matrix $X \in \mathcal{S}^{(i)}$ as $X = \mathbf{Q}V$ for some $V \in \mathbb{R}^{q\times k}$. Here we assume that $\mathbf{X}$ has a full rank and will later relax this assumption. We now convert the generalized eigenvalue problem (9) into an equivalent eigenvalue problem

$$\max_{V\in\mathbb{R}^{q\times k}} \|\mathbf{R}V\|_{\mathrm{F}}^2, \ \text{ s.t. } \ V^{\mathrm{T}}V = I, \tag{10}$$

where

$$\mathbf{R} = \mathbf{R}_p^{(i)} := A^{\mathrm{T}}\mathbf{Q}_p^{(i)}. \tag{11}$$

Next we describe how to calculate the matrix product $\mathbf{R}$ in (11) from historical information without any additional computation involving the matrix A. Since $\mathbf{Q} \in \mathbf{orth}(\mathbf{X})$ and we assume that $\mathbf{X}$ has a full rank, there exists a nonsingular matrix $C \in \mathbb{R}^{q\times q}$ such that $\mathbf{X} = \mathbf{Q}C$. Hence, $\mathbf{Q} = \mathbf{X}C^{-1}$, and $\mathbf{R}$ in (11) can be computed as

$$\mathbf{R} = A^{\mathrm{T}}\mathbf{Q} = (A^{\mathrm{T}}\mathbf{X})C^{-1} = \mathbf{Y}C^{-1}, \tag{12}$$

where $\mathbf{Y} = A^{\mathrm{T}}\mathbf{X}$ is accessible from our limited memory. Once $\mathbf{R}$ is available, we can solve (10) by computing the k leading eigenvectors of the $q \times q$ matrix $\mathbf{R}^{\mathrm{T}}\mathbf{R}$. Let a solution to (10) be $\hat{V}$. The matrix product in Line 5 of the framework LMSVD-F can then be assembled as

$$AA^{\mathrm{T}}\hat{X}^{(i)} = A\mathbf{R}\hat{V} = A\mathbf{Y}C^{-1}\hat{V}. \tag{13}$$

The remaining issue is how to efficiently and stably compute $\mathbf{Q}$ and $\mathbf{R}$ even when the matrix $\mathbf{X}$ is numerically rank deficient. We use the following procedure in our implementation.

Noting that each block in $\mathbf{X}$ is individually orthonormal, we choose to keep the latest block $X^{(i)}$ intact, and project the rest of the blocks onto the null space of $(X^{(i)})^{\mathrm{T}}$, obtaining

$$\mathbf{P}_X = \mathbf{P}_X^{(i)} := \left(I - X^{(i)}(X^{(i)})^{\mathrm{T}}\right)\left[X^{(i-1)} \ \cdots \ X^{(i-p)}\right]. \tag{14}$$

Next we perform an orthonormalization of $\mathbf{P}_X$ via the eigenvalue decomposition of its Gram matrix

$$\mathbf{P}_X^{\mathrm{T}}\mathbf{P}_X = U_X\Lambda_X U_X^{\mathrm{T}}, \tag{15}$$

where U_X is orthogonal and Λ_X is diagonal. It can be easily verified that the matrix

$$\mathbf{Q} = \mathbf{Q}_p^{(i)} := \left[X^{(i)}, \ \mathbf{P}_X U_X \Lambda_X^{-\frac{1}{2}}\right] \in \mathbf{orth}\left(\mathbf{X}_p^{(i)}\right), \tag{16}$$

provided that Λ_X is invertible. The above procedure can be stabilized by monitoring the numerical rank of $\mathbf{P}_X$, or specifically the eigenvalues on the diagonal of the matrix Λ_X in (15). We choose to implement the following two-step *stabilization scheme*:

Step 1 Delete the columns of $\mathbf{P}_X$ whose Euclidean norms are below a threshold $\epsilon_1 > 0$.

Step 2 Delete the eigenvalues in Λ_X, and corresponding columns in U_X, that are less than $\epsilon_2 > 0$.

With a slight abuse of notation, we will continue to use $\mathbf{P}_X$, U_X and Λ_X to denote their stabilized versions, respectively, after possible deletions. Therefore, a stable construction of $\mathbf{Q}$ is still given by formula (16). After this stable orthonormalization, the corresponding $\mathbf{R}$ matrix can be generated as

$$\mathbf{R} = \mathbf{R}_p^{(i)} := \left[Y^{(i)},\ \mathbf{P}_Y U_X \Lambda_X^{-\frac{1}{2}} \right], \tag{17}$$

where before the stabilization procedure we had

$$\mathbf{P}_Y = \mathbf{P}_Y^{(i)} := \left[Y^{(i-1)}\ \cdots\ Y^{(i-p)} \right] - Y^{(i)} (X^{(i)})^{\mathrm{T}} \left[X^{(i-1)}\ \cdots\ X^{(i-p)} \right], \tag{18}$$

but some of the columns of $\mathbf{P}_Y$ may have been deleted corresponding to those deleted columns of $\mathbf{P}_X$ during the stabilization steps. After the removal of numerical rank deficiency, the $\mathbf{R}$ matrix in (17) is well defined as is the $\mathbf{Q}$ matrix in (16).

In summary, the algorithm performs eigenvalue decompositions on two small symmetric positive definite matrices: $\mathbf{P}_X^{\mathrm{T}}\mathbf{P}_X$ in (15) and $\mathbf{R}^{\mathrm{T}}\mathbf{R}$ in (10). The sizes of the two matrices are pk and $(p+1)k$, respectively, and frequently smaller due to deletions. Our computational experience indicates that in general p should be set to 2 or 3, or at most 4 but not greater. Consequently, when k is sufficiently smaller than m, it holds that $(p+1)k \ll m \leq n$. In such a case, any adequate eigenvalue solvers for dense matrices can be utilized to solve the two small eigenvalue problems without significantly affecting the overall performance of our algorithm.

2.3 LMSVD Algorithm Statement

Based on the description above, we state our full Algorithm with the exception about our strategy for selecting the memory length (or block size) p that is to be specified later. For ease of reference, the algorithm will be referred to as LMSVD.

Algorithm 2: Limited Memory Subspace Optimization for SVD (LMSVD)

Input A and k. Initialize $\mathbf{X} \equiv X^{(0)} \in \mathbb{R}^{m\times k}$, $\mathbf{Y} \equiv Y^{(0)} = A^{\mathrm{T}}X^{(0)}$, and $p = i = 0$;
while *"not converged"* **do**
    `/* Block Subspace Optimization */`
    Compute $\mathbf{P}_X$ by (14) and perform stabilization **Step 1**;
    Compute $\mathbf{P}_Y$ by (18) with the same column deletions as for $\mathbf{P}_X$;
    Compute the eigenvalue decomposition of $\mathbf{P}_X^{\mathrm{T}}\mathbf{P}_X$ in (15);
    Perform stabilization **Step 2** to possibly shrink Σ_X and U_X;
    Compute $\mathbf{R}$ by (17) and eigenvalue decomposition of $\mathbf{R}^{\mathrm{T}}\mathbf{R}$;
    Let $\hat{V}$ solve (10), consisting of the k leading eigenvectors of $\mathbf{R}^{\mathrm{T}}\mathbf{R}$;
    Compute $\hat{X}^{(i)} = \mathbf{Q}\hat{V}$ and $\hat{Y}^{(i)} = \mathbf{R}\hat{V}$ (which equals $A^{\mathrm{T}}\hat{X}^{(i)}$);
    `/* Simultaneous Iteration */`
    Compute $X^{(i+1)} \in \mathbf{orth}(A\hat{Y}^{(i)})$ and $Y^{(i+1)} = A^{\mathrm{T}}X^{(i+1)}$;
    Increment i, update p, $\mathbf{X}$ and $\mathbf{Y}$, and continue.

In Line 10, the orthonormalization is done by QR factorization in our implementation. It is worth noting that the solution $\hat{X}^{(i)}$ to problem (10) has orthonormal columns.

It is also reasonable to use the intermediate iterates $\hat{X}^{(j)}$, $j = i-1, \cdots, i-p$, to construct a limited memory block Krylov subspace

$$\hat{\mathcal{S}}^{(i)} := \mathbf{span}\left\{X^{(i)}, \hat{X}^{(i-1)}, ..., \hat{X}^{(i-p)}\right\}, \tag{19}$$

to be used in place of $\mathcal{S}^{(i)}$ in the LMSVD framework. In our experiments, this alternative subspace construction works equally well, very often slightly better. Hence, we choose to use the above subspace $\hat{\mathcal{S}}^{(i)}$ as the default construction in our LMSVD implementation.

2.4 Complexity per Iteration

On unstructured matrices, the per-iteration operation count of the classic SSI method is $O(nmk)$ for the matrix-block multiplications, and $O(mk^2)$ for the orthonormalization (say, by QR factorization). On the other hand, the additional cost per-iteration of LMSVD is chiefly $O(mk^2(1+p)^2)$ for finding an orthonormal basis for the subspace $\mathcal{S}^{(i)}$, and $O(k^3(1+p)^3)$ for solving the two small eigenvalue problems. Comparing the total additional cost of LMSVD to the total cost of SSI, i.e.,

$$O\left((m+k+kp)k^2(1+p)^2\right) \quad \text{vs} \quad O\left(mk(n+k)\right),$$

we can see that when $p \ll k \ll m \leq n$, the above reduces to $O(k(1+p)^2)$ vs $O(n+k)$. In this case, the former is clearly dominated by the latter, and a considerable acceleration to the SSI method becomes achievable if our subspace optimization strategy can significantly reduce the number of iterations required.

The argument above, however, hinges on the fact that the matrices under consideration are unstructured and dense. For highly structured matrices such as sparse matrices, the cost of matrix-vector multiplications involving A may become much lower than $O(nm)$. In this case, it is likely that the overhead introduced by our subspace optimization becomes too high to offer any benefits.

3 Convergence Analysis

In this section, we study convergence properties of the proposed algorithm LMSVD. In the first subsection, we describe the first-order necessary optimality conditions of (2) and restate the existing convergence results for SSI. Then we give a convergence result for a class of SSI-based algorithms, including both SSI and LMSVD. The convergence is generally to a critical point of (2); but under mild assumptions, it also implies convergence to the global optimal solution of (2).

3.1 Preliminaries

For notational simplicity, let

$$B = AA^{\mathrm{T}},$$

and $\lambda_1 \geq ... \geq \lambda_m \geq 0$ be the eigenvalues of B. Since the matrix $X^{\mathrm{T}}X$ is symmetric, the Lagrangian multiplier Λ corresponding to $X^{\mathrm{T}}X = I$ is a symmetric matrix as well. The Lagrangian function of problem (2) is

$$\ell(X, \Lambda) = \frac{1}{2}\left(\mathbf{trace}(X^T BX) - \mathbf{trace}(\Lambda(X^{\mathrm{T}}X - I))\right).$$

Due the orthogonality constraint $X^T X = I$, the linear independence constraint qualification always holds true for (2). Therefore, the first-order necessary conditions for optimality of (2) are that

$$\partial_X \ell(X, \Lambda) = BX - X\Lambda = 0 \ \text{ and } \ X^{\mathrm{T}}X = I. \tag{20}$$

Multiplying both sides of the first equation in (20) by X^{T} and using $X^{\mathrm{T}}X = I$, we obtain the expression

$$\Lambda = X^{\mathrm{T}}BX. \tag{21}$$

Combining (20) and (21), plus the orthogonality constraint, we can write the first-order necessary optimality conditions of (2) into

$$(I - XX^{\mathrm{T}})BX = 0 \ \text{ and } \ X^{\mathrm{T}}X = I. \tag{22}$$

It is clear that (i) if X is a critical point, so is XQ for any orthogonal matrix Q; and (ii) the columns of any critical point X span a nontrivial invariant subspace of B (but need not to be eigenvectors of B).

To facilitate further discussions, we denote the distance between a matrix $X \in \mathbb{R}^{m\times k}$ and a set $\mathcal{L} \subset \{X \mid X \in \mathbb{R}^{m\times k}\}$ as $\mathbf{dist}(X, \mathcal{L})$, i.e.,

$$\mathbf{dist}(X, \mathcal{L}) := \min_{Y\in\mathcal{L}} \|X - Y\|_{\mathrm{F}}. \tag{23}$$

We use $\mathcal{L}^*$ and $\mathcal{L}^c$ to denote, respectively, the sets of optimal solutions and critical points of (2); that is,

$$\mathcal{L}^* := \{X \in \mathbb{R}^{m\times k} \mid X \text{ is an optimal solution of (2)}\}, \tag{24}$$

$$\mathcal{L}^c := \{X \in \mathbb{R}^{m\times k} \mid X \text{ satisfies the conditions (22)}\}. \tag{25}$$

3.2 Convergence of SSI

Let $v_1, v_2, ..., v_m$ be the unit eigenvectors of B associated with $\lambda_1 \geq ... \geq \lambda_m \geq 0$, respectively, and in particular $V_k := [v_1, v_2, ..., v_k] \in \mathbb{R}^{m\times k}$ consisting of the k leading eigenvectors. We first state two standard assumptions for SSI.

Assumption 3.1. *(a) The initial iterate $X^{(0)}$ satisfies the condition that $V_k^T X^{(0)}$ is nonsingular. (b) There is a gap between the k-th and the $(k+1)$-th eigenvalues of B, namely, $\lambda_k > \lambda_{k+1} \geq 0$.*

SSI has the following well-known convergence property.

Proposition 3.1. *Let $\{X^{(i)}\}$ be generated by SSI. Under Assumption 3.1, the distance between i-th iterate $X^{(i)}$ and the optimal solution set $\mathcal{L}^*$ of* (2) *converges to zero, i.e.,*

$$\lim_{i\to+\infty} \mathbf{dist}(X^{(i)}, \mathcal{L}^*) = 0,$$

Moreover, an upper bound for the asymptotic convergence rate is given by

$$\lim_{i\to+\infty} \frac{\mathbf{dist}(X^{(i+1)}, \mathcal{L}^*)}{\mathbf{dist}(X^{(i)}, \mathcal{L}^*)} \leq \frac{\lambda_{k+1}}{\lambda_k} < 1.$$

We should point out that convergence properties of SSI are normally stated in terms of Ritz-value and Ritz-vector sequences (see [19] and [25], for example) that requires further calculations in addition to the main SSI operations given in (4). However, the statement in Proposition 3.1, which better suits our purpose in this paper, is by no mean weaker than other more "standard" forms of convergence results for SSI.

3.3 Convergence of an accelerated class

The introduction of intermediate iterates $\hat{X}^{(i)}$ also introduces complications in the analysis of the accelerated subspace iterations. Unable to directly extend the convergence proof of SSI to LMSVD, we take a different approach and establish a more general result that requires no assumption whatsoever. Naturally, the conclusion is not as strong as the existing SSI result in Proposition 3.1. The result is Theorem 3.1 and is applicable to a whole class of SSI-based algorithms that we call the SSI+ class defined below, which includes both SSI and LMSVD as special cases. Recall that $\Phi(X) = \|A^{\mathrm{T}}X\|_{\mathrm{F}}^2$.

Algorithm 3: SSI+ Class

Input matrix $A \in \mathbb{R}^{m\times n}$ ($m \leq n$) and positive integer $k \leq m$.
Initialize $X^{(0)} \in \mathbb{R}^{m\times k}$ so that $(\hat{X}^{(0)})^{\mathrm{T}}\hat{X}^{(0)} = I$ and $A^{\mathrm{T}}X^{(0)} \neq 0$.
for $i = 0, 1, 2, \cdots$ **do**
    Find $\hat{X}^{(i)} \in \mathbb{R}^{m\times k}$ such that $(\hat{X}^{(i)})^{\mathrm{T}}\hat{X}^{(i)} = I$ and $\Phi(\hat{X}^{(i)}) \geq \Phi(X^{(i)})$.
    Let $X^{(i+1)} \in \mathbf{orth}(AA^{\mathrm{T}}\hat{X}^{(i)})$, and update $k = \mathbf{rank}(X^{(i+1)})$ if necessary.

Clearly, if we set $\hat{X}^{(i)} = X^{(i)}$ at every iteration, we recover SSI.

Theorem 3.1. *Let the sequence $\{\hat{X}^{(i)}\}$ be generated by a member of the SSI+ class. Then any cluster point of the sequence is a critical point of* (2) *that spans a non-trivial invariant subspace of AA^T; i.e.,*

$$\lim_{i\to+\infty} \mathbf{dist}(X^{(i)}, \mathcal{L}^c) = 0,$$

We note that the sequence is bounded by construction, hence a cluster point exists. In the rest of this subsection, we will develop a proof for Theorem 3.1 based on two technical lemmas, which show monotonicity of $\Phi(X)$ during iterations and establish a connection to the critical point condition (22).

Lemma 3.1. *Let $B \in \mathbb{R}^{m\times m}$ be symmetric positive semidefinite with rank $r > 0$. For any nonzero vector $x \in \mathbb{R}^m$, there holds*

$$(x^TB^3x)(x^Tx) - (x^TB^2x)(x^TBx) \geq \lambda_r(B)\left((x^TB^2x)(x^Tx) - (x^TBx)^2\right), \tag{26}$$

where $\lambda_r(B) > 0$ is the smallest positive eigenvalue of B. Moreover, whenever $x^Tx = 1$ and $Bx \neq 0$, inequality (26) *implies*

$$\frac{x^TB^3x}{x^TB^2x} \geq x^TBx + \frac{\lambda_r(B)}{\|B\|_2^2}\left(x^TB^2x - (x^TBx)^2\right). \tag{27}$$

Proof. Without loss of generality, we assume that B is diagonal; otherwise, it suffices to replace x by Q^Tx and B^ℓ by $Q^TB^\ell Q$ for $\ell = 1, 2, 3$, where the columns of Q consist of the unit eigenvectors of B. We now prove (26) for a diagonal matrix B. By expanding the terms in the left-hand side of (26), we obtain

$$\begin{aligned}
& (x^{\mathrm{T}}B^3x)(x^{\mathrm{T}}x) - (x^{\mathrm{T}}B^2x)(x^{\mathrm{T}}Bx) \\
= & \sum_{i=1}^m B_{ii}^3x_i^2\sum_{j=1}^m x_j^2 - \sum_{i=1}^m B_{ii}^2x_i^2\sum_{j=1}^m B_{jj}x_j^2 \\
= & \sum_{1\leq i,j\leq m} x_i^2x_j^2(B_{ii}^3 - B_{ii}^2B_{jj}) \\
= & \sum_{1\leq i<j\leq m} x_i^2x_j^2(B_{ii}^3 - B_{ii}^2B_{jj}) + \sum_{1\leq j<i\leq m} x_i^2x_j^2(B_{ii}^3 - B_{ii}^2B_{jj}) \\
= & \sum_{1\leq i<j\leq m} x_i^2x_j^2(B_{ii}^3 + B_{jj}^3 - B_{ii}^2B_{jj} - B_{ii}B_{jj}^2) \\
= & \sum_{1\leq i<j\leq m} x_i^2x_j^2(B_{ii} + B_{jj})(B_{ii} - B_{jj})^2.
\end{aligned}$$

In the above derivation, we should note the changes in the range of summation.

Similarly, the second term in the right-hand side of (26) can be expanded into

$$(x^{\mathrm{T}}B^2x)(x^{\mathrm{T}}x) - (x^{\mathrm{T}}Bx)^2 = \sum_{1\leq i<j\leq m} x_i^2x_j^2(B_{ii} - B_{jj})^2.$$

If $B_{ii} + B_{jj} = 0$, then $B_{ii} - B_{jj} = 0$ due to the nonnegativity of $\mathbf{diag}(B)$; Otherwise,

$$B_{ii} + B_{jj} \geq \max(B_{ii}, B_{jj}) \geq \lambda_r(B).$$

Combining all above together, we deduce that

$$(x^{\mathrm{T}}B^3x)(x^{\mathrm{T}}x) - (x^{\mathrm{T}}B^2x)(x^{\mathrm{T}}Bx) \quge \lambda_r(B)\sum_{1\leq i<j\leq m} x_i^2x_j^2(B_{ii} - B_{jj})^2$$

which proves (26). Finally, when $x^{\mathrm{T}}x = 1$ and $Bx \neq 0$, we divide the left-hand side of (26) by $x^{\mathrm{T}}B^2x$ and

the right-hand side by $\|B\|_2^2 \geq x^{\mathrm{T}} B^2 x$, leading to (27). □

Lemma 3.2. *Let $B \in \mathbb{R}^{m\times m}$ be symmetric positive semidefinite with rank $r > 0$, and let $X \in \mathbb{R}^{m\times k}$ satisfy $X^T X = I$ and $0 < \ell = \mathbf{rank}(BX) \leq \min(r, k)$. Then, for any $Y \in \mathbf{orth}(BX)$ there holds that*

$$\mathbf{trace}(Y^T BY) - \mathbf{trace}(X^T BX) \geq \frac{\lambda_r(B)}{\|B\|_2^2} \left\|(I - XX^T)BX\right\|_F^2. \tag{28}$$

Proof. We first note that for any given $Y_1,\ Y_2 \in \mathbf{orth}(BX)$ there exists an orthogonal matrix $U \in \mathbb{R}^{\ell\times\ell}$ so that $Y_1 = Y_2 U$, implying $\mathbf{trace}(Y_1^{\mathrm{T}} BY_1) = \mathbf{trace}(Y_2^{\mathrm{T}} BY_2)$. Hence, it suffices to prove (28) for a particular $Y \in \mathbf{orth}(BX)$.

Let $BX = UDV^{\mathrm{T}}$ be the economy-form singular value decomposition of BX; namely, $U \in \mathbb{R}^{m\times\ell}$ and $V \in \mathbb{R}^{k\times\ell}$ have orthonormal columns and $D \in \mathbb{R}^{\ell\times\ell}$ is diagonal and positive definite. We select $Y = U \in \mathbf{orth}(BX)$, or equivalently,

$$Y = BXVD^{-1} = BZD^{-1}$$

where $Z = XV$ satisfies $Z^T Z = I$. It is easy to see that

$$\mathbf{trace}(Y^{\mathrm{T}} BY) = \mathbf{trace}((BZD^{-1})^{\mathrm{T}} B(BZD^{-1})) = \mathbf{trace}(D^{-1} Z^{\mathrm{T}} B^3 ZD^{-1}). \tag{29}$$

Hence, for $i = 1, 2, ..., \ell$,

$$\begin{aligned} (Y^{\mathrm{T}} BY)_{ii} &= z_i^{\mathrm{T}} B^3 z_i / D_{ii}^2, \\ 1 = (Y^{\mathrm{T}} Y)_{ii} &= z_i^{\mathrm{T}} B^2 z_i / D_{ii}^2, \end{aligned} \tag{30}$$

where z_i is the i-th column of Z. It follows from (30) that

$$\mathbf{trace}(Y^{\mathrm{T}} BY) = \sum_{i=1}^{\ell} \frac{z_i^{\mathrm{T}} B^3 z_i}{z_i^{\mathrm{T}} B^2 z_i}, \tag{31}$$

In view of (31) and (27), we deduce

$$\begin{aligned} \mathbf{trace}(Y^{\mathrm{T}} BY) &\geq \sum_{i=1}^{\ell} \left(z_i^{\mathrm{T}} Bz_i + \frac{\lambda_r(B)}{\|B\|_2^2} \left(z_i^{\mathrm{T}} B^2 z_i - (z_i^{\mathrm{T}} Bz_i)^2 \right) \right) \\ &= \mathbf{trace}(Z^{\mathrm{T}} BZ) + \frac{\lambda_r(B)}{\|B\|_2^2} \sum_{i=1}^{\ell} \left(\|Bz_i\|_2^2 - (z_i^{\mathrm{T}} Bz_i)^2 \right) \\ &\geq \mathbf{trace}(Z^{\mathrm{T}} BZ) + \frac{\lambda_r(B)}{\|B\|_2^2} \left(\|BZ\|_{\mathrm{F}}^2 - \|Z^{\mathrm{T}} BZ\|_{\mathrm{F}}^2 \right). \end{aligned}$$

Recall that $Z = XV$, $BX = UDV^{\mathrm{T}}$ and $V^{\mathrm{T}} V = I$. We derive that

$$\mathbf{trace}(Z^{\mathrm{T}} BZ) = \mathbf{trace}(V^{\mathrm{T}} (VDU^{\mathrm{T}}) XV) = \mathbf{trace}(VDU^{\mathrm{T}} X) = \mathbf{trace}(X^{\mathrm{T}} BX).$$

Similarly, we can prove that $\|BZ\|_{\mathrm{F}}^2 = \|BX\|_{\mathrm{F}}^2$ and $\|Z^{\mathrm{T}}BZ\|_{\mathrm{F}}^2 = \|X^{\mathrm{T}}BX\|_{\mathrm{F}}^2$. Therefore,

$$\begin{aligned}
\mathbf{trace}(Y^{\mathrm{T}}BY) &\geq \mathbf{trace}(Z^{\mathrm{T}}BZ) + \frac{\lambda_r(B)}{\|B\|_2^2}\left(\|BZ\|_{\mathrm{F}}^2 - \|Z^{\mathrm{T}}BZ\|_{\mathrm{F}}^2\right) \\
&= \mathbf{trace}(X^{\mathrm{T}}BX) + \frac{\lambda_r(B)}{\|B\|_2^2}\left(\|BX\|_{\mathrm{F}}^2 - \|X^{\mathrm{T}}BX\|_{\mathrm{F}}^2\right) \\
&= \mathbf{trace}(X^{\mathrm{T}}BX) + \frac{\lambda_r(B)}{\|B\|_2^2}\left\|BX - X(X^{\mathrm{T}}BX)\right\|_{\mathrm{F}}^2 \\
&= \mathbf{trace}(X^{\mathrm{T}}BX) + \frac{\lambda_r(B)}{\|B\|_2^2}\left\|(I - XX^{\mathrm{T}})BX\right\|_{\mathrm{F}}^2,
\end{aligned}$$

which completes the proof. □

Now we are ready to present a proof for Theorem 3.1.

Proof. By initialization, $\Phi(X^{(0)}) = \|A^{\mathrm{T}}X^{(0)}\|_{\mathrm{F}}^2 > 0$. Now we show that $\{\Phi(X^{(i)})\}$ is nondecreasing, implying that the sequence is bounded away from the zero matrix. After each iteration, the change in $\Phi(X) = \mathbf{trace}(X^{\mathrm{T}}AA^{\mathrm{T}}X)$ is

$$\Phi(X^{(i+1)}) - \Phi(X^{(i)}) \geq \Phi(X^{(i+1)}) - \Phi(\hat{X}^{(i)}), \tag{32}$$

since $\Phi(\hat{X}^{(i)}) \geq \Phi(X^{(i)})$ by construction. By Lemma 3.2, for $B = AA^{\mathrm{T}}$,

$$\Phi(X^{(i+1)}) - \Phi(\hat{X}^{(i)}) \geq \frac{\lambda_r(B)}{\|B\|_2^2}\left\|\left(I - \hat{X}^{(i)}(\hat{X}^{(i)})^{\mathrm{T}}\right)B\hat{X}^{(i)}\right\|_{\mathrm{F}}^2, \tag{33}$$

where $\lambda_r(B) > 0$ is the smallest positive eigenvalue of B. It follows from (32) and (33) that the sequence $\{\Phi(X^{(i)})\}$ is monotonically non-decreasing. It is clearly positive bounded above and convergent, hence by virtue of (33) forcing

$$\lim_{i\to+\infty}\left\|\left(I - \hat{X}^{(i)}(\hat{X}^{(i)})^{\mathrm{T}}\right)B\hat{X}^{(i)}\right\|_{\mathrm{F}}^2 = 0. \tag{34}$$

Consequently, every limit point is a critical point satisfying (22). The conclusion of the theorem follows readily by standard arguments. □

3.4 Remarks on the convergence result

Theorem 3.1 is applicable to a broad class of SSI acceleration schemes. For example, any method producing an approximate, feasible solution to the subspace optimization problem (5) is admissible as long as the function $\Phi(X)$ value is not decrease. The theorem is established under the absolutely minimal assumption; i.e., the iterations have a nontrivial start. Even in the case of SSI, the result is not implied by existing results since a gap in singular values is not necessary.

As perhaps is expected, the conclusion of Theorem 3.1 is also weak in the sense of only guaranteeing convergence to a critical point, but not to a global maximizer. However, in the context of solving (5),

guaranteed convergence to a critical point is not as weak as it could be for a more general non-concave maximization problem since (5) does not have any local non-global maximizer, as has been long known (and can be shown easily from second-order optimality conditions). As such, it seems highly unlikely for a subspace maximization scheme like LMSVD to be attracted to any saddle point. Indeed, we will later present numerical evidence to demonstrate that LMSVD almost always converges to optima even it is started extremely close to a non-optimal saddle point. In our extensive numerical experiments, LMSVD behaves as robustly as SSI, if not more so, in terms of converging to a global maximizer.

Finally, we mention that if we add the assumption that the sequence $\{V_k^{\mathrm{T}} X^{(i)}\}$, i.e., coefficient matrices associated with the dominant eigenvector of AA^{T}, remains uniformly nonsingular, then under Assumption 3.1 we could derive convergence results for LMSVD similar to those in Proposition 3.1 for SSI.

4 Practical Issues

Recall that the subspace $\mathcal{S}^{(i)}$ is constructed from the current iterate and p previous ones. In this section, we describe a strategy for selecting the memory length p from iteration to iteration. We also discuss the choice for the number of guard vectors that are commonly used in eigenvalue solvers, and specify the stopping rule used in LMSVD.

4.1 Memory Length

The memory length p, used for constructing the subspace $\mathcal{S}^{(i)}$ in (6), is a crucial parameter to the performance of our algorithm. The simplest way is to assign a constant integer value $p_{\max}$ to p at every iteration once the iteration counter i reaches $p_{\max}$; that is, at iteration i,

$$p = \min(i, p_{\max}). \tag{35}$$

In general a larger $p_{\max}$ leads to a smaller number of iterations, but increasing $p_{\max}$ also increases the computational costs per iteration. Our computational experiments indicate that usually a good balance is attained for $p_{\max} \in \{2, 3, 4\}$.

We have also found that an adaptive strategy on selecting p is useful to improving the performance of LMSVD. As the iterate sequence converges, the neighboring iterates tend to become more and more linearly dependent. Therefore, once judged appropriate it is beneficial to shrink the memory by deleting a block from the memory, reducing the size of later subspace optimization problems. Specifically, after $p_{\max}$ iterations, we activate the following adaptive memory size strategy:

$$p = \left\lceil \frac{N_c(\mathbf{R})}{k} \right\rceil - 1, \tag{36}$$

where $\lceil t \rceil$ is the smallest integer greater than or equal to t, and $N_c(\mathbf{R})$ is the number of columns in $\mathbf{R}$ (see Line 7 of Algorithm LMSVD) which can be smaller than $(p+1)k$ due to possible deletions done in the two stabilization steps. Combining (35) and (36), we reach our formula for selecting the memory length p at the

i-th iteration:

$$p = \min\left\{i,\ \left\lceil \frac{N_c(\mathbf{R})}{k} \right\rceil - 1,\ p_{\max}\right\} \tag{37}$$

which is nonnegative. Generally, p initially increases to reach $p_{\max}$, then becomes non-increasing with a probability to decrease to a smaller value, even possibly to zero. Of course, when the memory length p becomes zero, our method reduces to the classic SSI.

4.2 Guard Vectors

It is well-known that when the SSI method is applied to a symmetric positive semidefinite matrix $B \in \mathbb{R}^{m\times m}$ with $m \times k$ block iterates, the convergence to the eigenspace of the first $r \leq k$ leading eigenvectors has an asymptotic rate

$$\lambda_{k+1}/\lambda_r \leq \lambda_{k+1}/\lambda_k$$

where $\lambda_j,\ j = 1, 2, \cdots, m$, are the eigenvalues of B in a descending order. If we really only need to compute the first $r < k$ leading eigen-pairs, the additional $k - r$ vectors are called "guard vectors" and play the role of accelerating convergence. In general, the more guard vectors are used, the less iterations are needed for convergence, but at a higher cost per iteration on memory and computing time. Our algorithm, built on the basis of the SSI framework, also benefits from the use of guard vectors.

Now assume that the r-th dominant SVD of $A \in \mathbb{R}^{m\times n}$ ($m \leq n$) is to be computed. We implement the following standard choice for $k > r$:

$$k = \min\{2r, r + 10, m\}. \tag{38}$$

4.3 Stopping Rule

It is most natural to monitor convergence by observing the magnitude of the residual matrix

$$\mathrm{Res} := (I - XX^{\mathrm{T}})AA^{\mathrm{T}}X = AY - XY^{\mathrm{T}}Y \tag{39}$$

where $Y = A^{\mathrm{T}}X$ which is computed at each iteration, preferably in a relative sense; but it requires some additional matrix multiplications. To reduce unnecessary overheads, in LMSVD we use a two-level strategy with a pair of stopping criteria: a "low-cost" one and a "high-cost" one. We examine the low-cost criterion at each iteration, and activate the high-cost one only when the low-cost one is satisfied.

The low-cost criterion is based on examining the r leading Ritz values of AA^{T} in the subspace $\mathcal{S}^{(i)}$ that are available after the subspace optimization problem (10) is solved at each iteration. Specifically, the criterion is, for some tolerance $\epsilon_\ell > 0$,

$$\left\|\mathbf{\Lambda}_r^{(i)} - \mathbf{\Lambda}_r^{(i-1)}\right\|_{\mathrm{F}} < \epsilon_\ell \left\|\mathbf{\Lambda}_r^{(i)}\right\|_{\mathrm{F}} \tag{40}$$

where $\mathbf{\Lambda}_r^{(i)}$ consists of the r leading eigenvalues of $(\mathbf{R}_p^{(i)})^{\mathrm{T}}\mathbf{R}_p^{(i)}$ with $\mathbf{R}_p^{(i)}$ given by (17). This criterion not

only can be observed at almost no additional cost, but also is a reasonably accurate indicator of convergence.

Once the current iteration satisfies the low-cost criterion (40), we start to check the residual in (39) in a relative sense. To avoid extra computation, we evaluate the residual at the intermediate variable $\hat{X}^{(i-1)}$. Specifically, the high-cost criterion is, for some tolerance $\epsilon_h > 0$,

$$\left\| \mathrm{Res}^{(i-1)} e_j \right\|_2 < \epsilon_h \, \lambda_{\max}^{(i)}, \quad j = 1, ..., r, \tag{41}$$

where $\lambda_{\max}^{(i)}$ is the largest eigenvalue of $(\mathbf{R}_p^{(i)})^{\mathrm{T}} \mathbf{R}_p^{(i)}$, e_j is the j-th column of the identity matrix, and

$$\mathrm{Res}^{(i-1)} = A\hat{Y}^{(i-1)} - \hat{X}^{(i-1)} (\hat{Y}^{(i-1)})^{\mathrm{T}} \hat{Y}^{(i-1)}. \tag{42}$$

We note that only the first r columns of $\mathrm{Res}^{(i-1)}$ need to be evaluated, and the SSI method already does the calculation of $A\hat{Y} = AA^{\mathrm{T}}\hat{X}$.

To relate the tolerance ϵ_ℓ for the low-cost criterion to ϵ_h, we choose the formula $\epsilon_\ell = \sqrt{\epsilon_h \varepsilon}$ after some numerical experiments, where ε is the machine epsilon (in double precision, $\varepsilon = 2.2204 \times 10^{-16}$). Empirically, this proves to be a well balanced choice for efficiency and reliability in our two-level strategy. The number of iterations in which the high-cost criterion is checked, on the average, is about 10% of the total number of iterations. We note that the tolerance for the high-cost criterion, ϵ_h, becomes the only free parameter in our stopping rule. For simplicity, in the sequel we will just refer to it as ϵ, which will be varied in some numerical experiments.

5 Numerical Experiments

In this section, we demonstrate the effectiveness of the LMSVD, as a general solver for computing dominant SVDs of unstructured matrices, by numerical experiments on a wide variety of randomly generated matrices as well as a few examples from applications. Our code is implemented in MATLAB. All the experiments were preformed on a HP laptop with Intel® dual core i5-460M CPU at 2.53GHz ($\times 2$) and 8GB of memory running Ubuntu 10.10 and MATLAB 2010b.

We uses r to denote the targeted number of dominant singular values and vectors. Considering the guard vector discussed in section 4.2, a k-dominant SVD problem is actually solved, in which k is determined by (38) where the default value of the parameter $\xi = 10$. The default tolerance value is $\epsilon = 10^{-8}$ in the stopping criterion (41). The limited memory subspace defined in (19) is always used, unless specified otherwise. The maximal blocks of previous iterates in limited memory subspaces is set to $p_{\max} = 3$ in (37). In addition, based on some empirical experiments, we set the tolerance values, ϵ_1 and ϵ_2, used in our two-step stabilization scheme (see the discussion after (15)) as follows: $\epsilon_1 = 5 \times 10^{-8}$ and $\epsilon_2 = \min(\epsilon_h, \sqrt{\varepsilon})$, where ϵ_h is the tolerance in (41) and ε is the machine precision (in double precision, $\varepsilon = 2.2204 \times 10^{-16}$).

5.1 Random problem generation

In this section, unless otherwise specified, our random matrix $A \in \mathbb{R}^{m\times n}$, assuming $m \leq n$ without loss of generality, is constructed by one of the two models below that we call Model 1 (left) and Model 2 (right):

$$A := UDV^{\mathrm{T}} \quad \text{or} \quad A := DR. \tag{43}$$

In Model 1, the matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times m}$ are first generated with i.i.d. standard Gaussian entries and then orthogonalized using (economy-size) QR decompositions, while $D \in \mathbb{R}^{m\times m}$ is a nonnegative diagonal matrix. This construction builds the singular value decomposition of A, but the involved orthogonalization can become costly as matrix sizes increase. In Model 2, $D \in \mathbb{R}^{m\times m}$ is still diagonal, and $R \in \mathbb{R}^{n\times m}$ is a random matrix whose entries are i.i.d. standard Gaussian. Clearly, the construction cost of Model 2 is much lower.

In both models, the diagonal entries of D are

$$D_{ii} := \max\{\beta^{1-i}, \epsilon^2\}, \quad \forall i = 1, 2, ..., m, \tag{44}$$

where ϵ is the tolerance used in the stopping criterion (41). The parameter $\beta \geq 1$ determines the decay rate of singular values, precisely in Model 1 and approximately in Model 2. Generally speaking, the closer β is to 1, the smaller decay there is in singular values of a generated matrix, and the more difficult the test instance becomes.

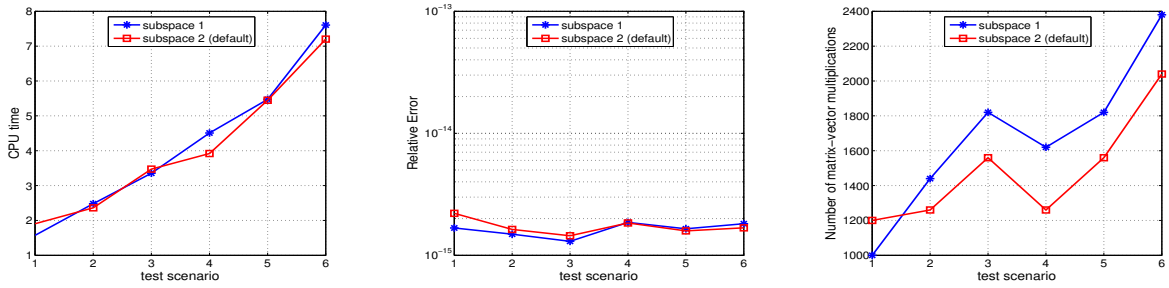
5.2 Sensitivity with respect to different parameters

In this subsection, we justify via numerical evidence the default choices we made in the LMSVD implementation; i.e., the choice of a subspace construction between (6) and (19), the choice of the number of guard vectors, and the choice of $p_{\max}$ – the maximum memory length. For this purpose, we perform three sets of tests using six randomly generated matrices, all from Model 1 where the decay parameter for singular values was set to $\beta = 1.01$. The sizes of the test matrices, (m, n), and the number of computed dominant singular values, r, are set to be $(m, n) = (2000, 4000)$ with $r = 40, 80, 120$, and $(m, n) = (4000, 4000)$ with $r = 80, 120, 160$, totaling 6 different test scenarios.

We first compare the performance of LMSVD using subspaces in (6) and (19) that we call subspaces 1 and 2, respectively, for convenience. The comparison is based on three quantities: the CPU time in seconds, the relative error between the computed and the exact r dominant singular values, say Σ and Σ^* respectively, $||\Sigma^* - \Sigma||/||\Sigma^*||$, and the total number of matrix-vector multiplications. The results in terms of the above three metrics are depicted in plots (a)-(c) of Figure 1, respectively. From these plots, we can see that LMSVD using subspace 1 defined in (19) required fewer number of matrix-vector multiplications than that using subspace 2 defined in(6), and achieved a similar solution quality within a similar amount of CPU time. Hence, we choose to use (19) in all subsequent experiments.

Our next test is to study the behavior of LMSVD with respect to the number of guard vectors. Specifically, we run LMSVD with ξ in (38) going over the set $\{0, 5, 10, 20, 40, r\}$. The corresponding performance

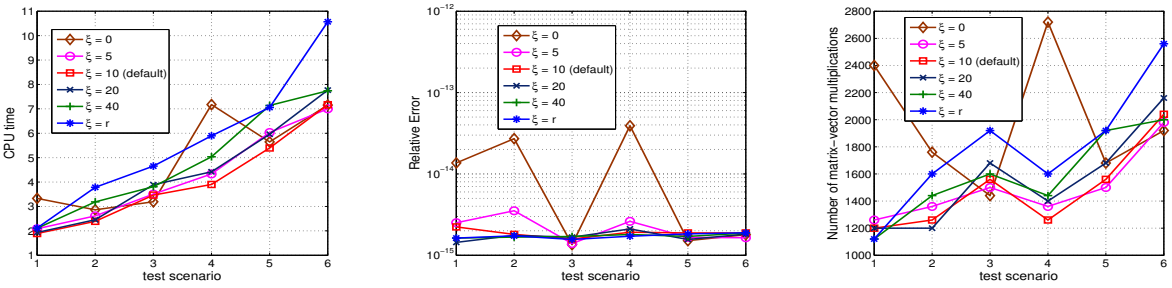
Figure 1: Performance of LMSVD using two different subspaces



(a) CPU time (b) Relative error (c) Matrix-vector multiplications

of LMSVD is illustrated in the plots (a)-(c) of Figure 2. From these results, it appears that $\xi = 10$ is a well balanced choice, and is set as subsequently the default value.

Figure 2: Performance of LMSVD with respect to the number of guard vectors.

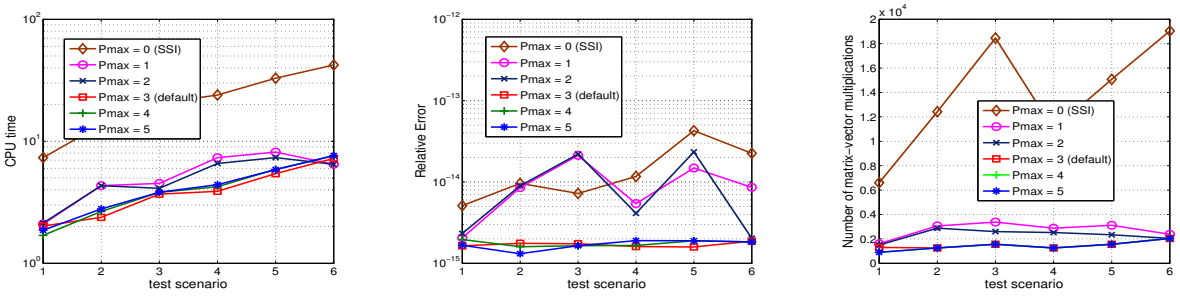


(a) CPU time (b) Relative error (c) Matrix-vector multiplications

We now consider the choice of $p_{\max}$, which determines the longest memory (i.e., number of previous iterates) used in LMSVD. We test 6 different values: $p_{\max} = 0$ to 5. Note that $p_{\max} = 0$ gives the classic SSI method. The performance of LMSVD on this test is presented in plots (a)-(c) of Figure 3, from which we can see a significant performance gap between $p_{\max} = 0$ and $p_{\max} > 0$. A slight overall improvement is still detectable as $p_{\max}$ increases from 1 to 3. Afterward, the benefit of increasing $p_{\max}$ seems to diminish as the cost of subproblem solving outpaces the saving from a reduced iteration number. Consequently, we choose to use $p_{\max} = 3$ as the default value in our tests (although we notice that when the ratio k/r is close to one, a slightly bigger $p_{\max}$ may yield better performance).

It is well known that besides the global maximum and minimum, all other critical points of (2) are saddle points. We present numerical evidence to show that it is highly unlikely for the algorithm to be attracted to a saddle point even under extreme conditions. In this set of experiments, the sizes of the test matrices are $(m, n, r) = (2000, 4000, 40)$. The initial guess $X^{(0)}$ is generated near a saddle point X^S of (2) whose

Figure 3: Performance of LMSVD with respect to $p_{\max}$.



(a) CPU time (b) Relative error (c) Matrix-vector multiplications

columns are singular vectors randomly selected from the matrix U in (43). Specifically,

$$X^{(0)} = X^S + \theta\, \mathbf{randn}(m, k)$$

where $\theta > 0$ controls the distance between $X^{(0)}$ and X^S, and $\mathbf{randn}(m, k)$ is the Maltab command for generating random Gaussian matrices of size m by k. Under the default setting, we compare the behavior of LMSVD with that of the SSI method (i.e., LMSVD with $p_{\max} = 0$). A summary of numerical results is given in Table 5.2.

We observe that when there is a notable decay in singular values, both algorithms could move away from a saddle point and eventually converge, even in the extreme case of $X^{(0)} = X^S$ (presumably due to rounding errors). It becomes more difficult to escape from a saddle point when there is little decay in singular values. This should be expected since in this case the objective value at a saddle point can be extremely close to that at the maximum. It is interesting to note that whenever LMSVD failed to escape, so did the SSI method. Empirically, LMSVD appears to have a similar convergence behavior as SSI, if not a better one. We note that SSI has been proven in theory to possess a guaranteed convergence to the global maximum.

5.3 Performance comparison on random matrices

We now compare the performance of LMSVD with several state-of-the-art SVD solvers including the Matlab built-in function EIGS which interfaces with the Fortran package ARPACK [14], a Matlab version of LOBPCG [11] [1] and a Matlab version of the solver LANSVD in the package PROPACK [12] [2]. In addition, we also compare with SSI without acceleration (i.e., $p_{\max} = 0$ in LMSVD) and with Chebyshev acceleration, called SSI+C, which is SSI augmented by extra steps introduced in section 1 of chapter 6 in [25]. Instead of the dedicated Matlab built-in interface SVDS for singular value decomposition, we choose to directly use the function EIGS since our numerical experiments indicate that the performance of the former is often significantly inferior to that of the latter in many instances. We also mention that the Matlab version of LANSVD performs re-orthogonalization calculations by calling a Fortran subroutine via Matlab's

[1] Downloadable from `http://www.mathworks.com/matlabcentral/fileexchange/48`.
[2] Downloadable from `http://soi.stanford.edu/~rmunk/PROPACK`.

Table 1: An illustration of global convergence to the solution set

Test Settings		LMSVD		SSI	
β	θ	#iter.	error	#iter.	error
1.1	1.e-08	3	5.3548e-15	6	8.8056e-15
1.1	1.e-09	3	2.8147e-15	6	1.4222e-14
1.1	1.e-10	3	5.1257e-15	7	4.4492e-15
1.1	1.e-11	3	2.6859e-15	6	6.2115e-15
1.1	0	3	2.5680e-15	7	3.5907e-14
1.01	1.e-08	8	4.9308e-14	56	1.0840e-12
1.01	1.e-09	7	8.1987e-14	52	9.4457e-13
1.01	1.e-10	7	7.1116e-14	53	9.3774e-13
1.01	1.e-11	8	2.4877e-14	54	1.0526e-12
1.01	0	9	5.5088e-14	62	9.8569e-13
1.001	1.e-08	42	2.1386e-12	150	1.0049e-05
1.001	1.e-09	44	1.4742e-12	150	1.7350e-05
1.001	1.e-10	2	3.8007e+00	2	3.8007e+00
1.001	0	2	3.8007e+00	2	3.8007e+00

MEX external interface.

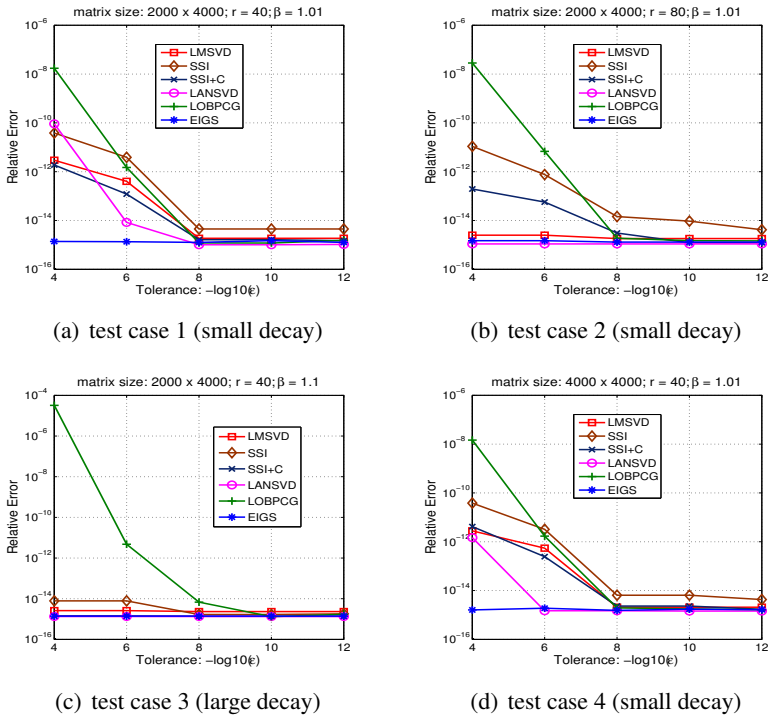
Our experiments are performed on a large set of random matrices generated by the two models described in Section 5.1. The compared quantities are CPU time, the number of matrix-vector multiplications and the solution quality. For LMSVD, each evaluation of the product $AA^{\mathrm{T}}X$ is counted as $2k$ matrix-vector multiplications where k is the number of columns in X defined by (38). Solution quality is measured by the relative error (in 2-norm) of the r computed dominant singular values against their most accurate available values. For matrices generated by Model 1, the exact singular values are known to begin with. For matrices generated by Model 2 where exact solutions are not known, relative errors are calculated with respect to the solutions of EIGS for all algorithms other than EIGS.

In all the experiments throughout this section, EIGS and LOBPCG are invoked with their default parameters. For LANSVD, however, we have observed that its performance in large-decay cases can be improved if its parameter "lanmax" is set to $2r$, while the default value is more appropriate in small-decay cases. Therefore, in order to obtain better results from LANSVD, we use the default value for the parameter "lanmax" if $\beta < 1.1$ and the value $2r$ otherwise.

The first experiment is to evaluate the algorithms with different stopping tolerance values, with the goal of finding a proper tolerance value to be used in later experiments. Generally speaking, when considering a group of algorithms, the relationship between tolerance and solution quality can be complicated due to the existence of multiple factors affecting individual algorithms. For this test, we construct four examples, using Model 1 so that the exact singular values are known, with size parameters (m, n, r, β) set to, respectively, $(2000, 4000, 40, 1.01)$, $(2000, 4000, 80, 1.01)$, $(2000, 4000, 40, 1.1)$ and $(4000, 4000, 40, 1.01)$. We will call them test cases 1 to 4. The results on relative error are presented in Figure 4, where in each plot the horizontal axis represents tolerance value varying from 10^{-4} to 10^{-12}.

We observe from Figure 4 that the accuracy of EIGS is insensitive to tolerance value in the testes cases, always achieving the machine precision even with the largest tolerance value 10^{-4}. Although other three algorithms demonstrate differently sensitivity, they all achieve a precision of order 10^{-15} with tolerance

Figure 4: Relationship between tolerance and relative error.



(a) test case 1 (small decay)

(b) test case 2 (small decay)

(c) test case 3 (large decay)

(d) test case 4 (small decay)

value 10^{-10}. Therefore, we will use 10^{-10} as the default tolerance value for LMSVD, SSI+C, LANSVD and LOBPCG in the rest of the paper.

In addition, since SSI+C clearly outperforms SSI in all tested instances, not only in terms of accuracy as shown in Figure 4, but also in terms of CPU time and the number of matrix-vector multiplications that we are showing here, we will exclude SSI from further comparisons presented hereinafter.

We next evaluate the remaining five solvers in the following three types of randomly generated scenarios constructed by Model 2:

Type I. For each $m \in \{1000, 2000, \ldots, 6000\}$, set $m = n$, the decay rate parameter, see (44), $\beta = 1.01$ and the number of computed singular values $r = 0.03m$. The experiment is repeated for $\beta = 1.1$.

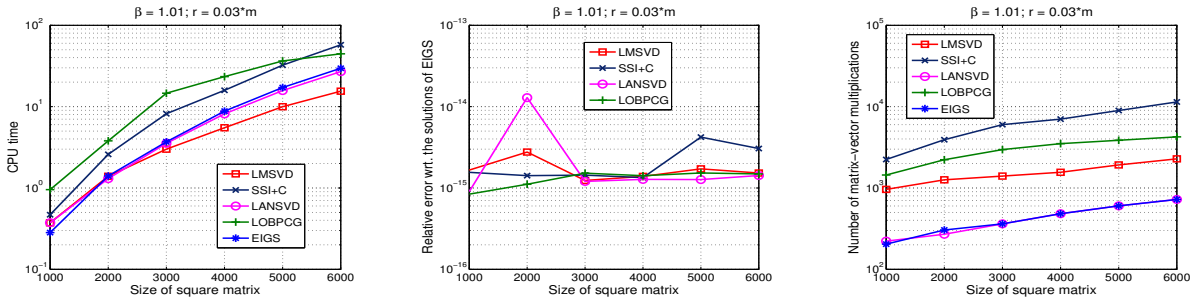
Type II. Set $(m, n) = (2000, 4000)$, $\beta = 1.01$, and vary r from $0.01m$ to $0.06m$. The experiment is repeated for $\beta = 1.1$.

Type III. Set $(m, n) = (2000, 4000)$, $r = 60$, and let $\beta = 1.01 + 0.03i$ for $i = 0, 1, \cdots, 5$.

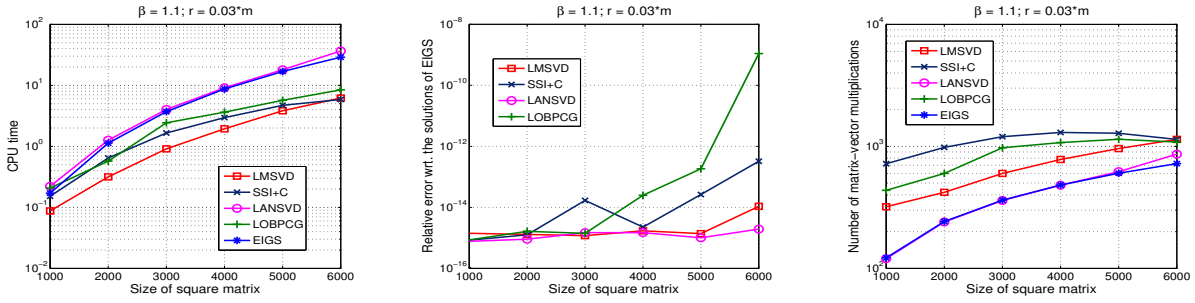
Comparison results from the above three groups of tests are given in Figures 5, 6 and 7, respectively. The following observations should be clear.

- All solvers achieved a good accuracy of 10^{-12} except for one case where LOBPCG failed to do so.
- The two Lanczos-based solvers EIGS and LANSVD generally requires less matrix-vector multiplications than the three subspace-iteration based solvers LMSVD, SSI+C and LOBPCG, though this

Figure 5: Comparison with varying matrix dimension (Type I)



(a) CPU time: $\beta = 1.01$ (b) relative error: $\beta = 1.01$ (c) Matrix-vector multiplications



(d) CPU time: $\beta = 1.1$ (e) relative error: $\beta = 1.1$ (f) Matrix-vector multiplications

advantage does not necessarily mean a higher efficiency in CPU time.

- Relatively speaking, small-decay cases are more favorable to the Lanczos-based solvers, while large-decay cases more favorable to the subspace-iteration based solvers.

In terms of the performance of LMSVD, we can safely make the following observations.

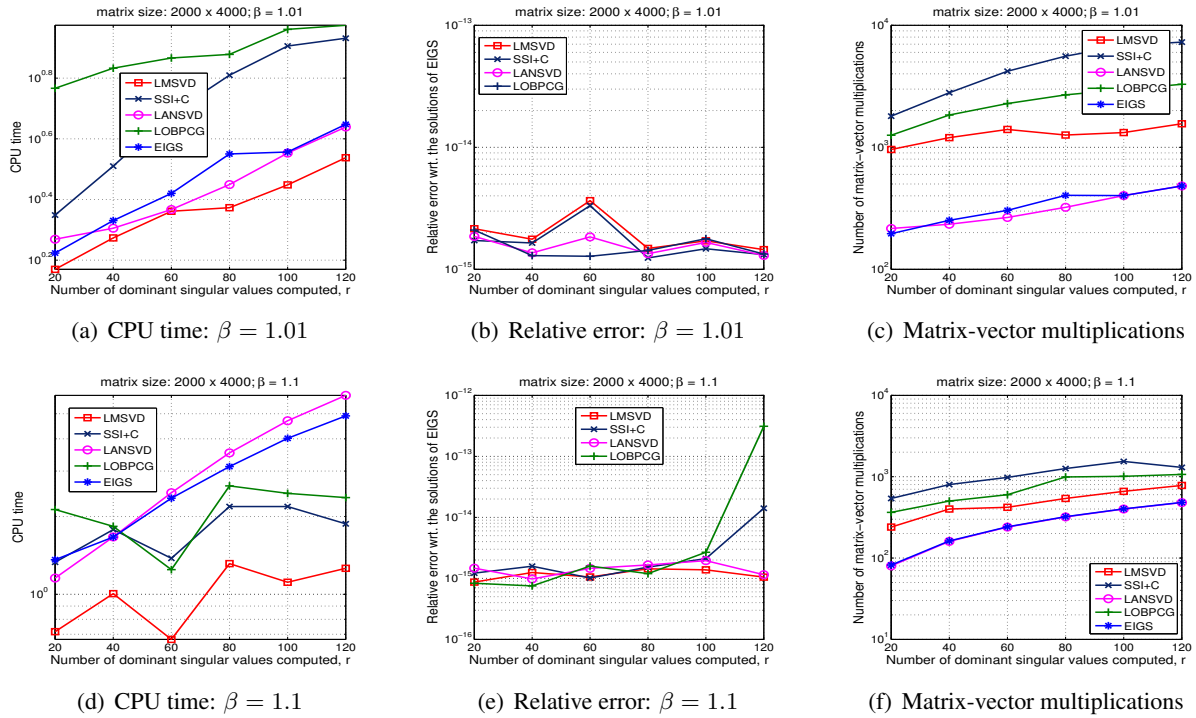
- On large-decay matrices, LMSVD takes as few iterations as SSI+C (though iteration number is not presented here), while on small-decay matrices when SSI+C and LOBPCG become really slow, LMSVD remains competitive with EIGS and LANSVD.

- In all the test instances of small or large decay, the time efficiency of LMSVD is always close to the best, if not the best by itself, among the five tested solvers.

As is noted, when one matrix-block multiplication $AA^{\mathrm{T}}X$ is counted as k matrix-vector multiplications, the two Lanczos-iteration based solvers, EIGS and LANSVD, require considerably fewer matrix-vector multiplications. Even so, LMSVD can still be be faster in terms of CPU time. Presumably, this has to do with memory access patterns in a memory hierarchy of modern computer architecture which is in favor of doing one matrix-block multiplication at once than doing k individual matrix-vector multiplications sequentially. In addition, block matrix multiplications facilitate the use of high efficiency level 3 BLAS subroutines in the linear algebra package LAPACK [1] used by Matlab.

An known advantage of subspace-iteration based algorithms over Lanczos based ones is that the former can easily benefit from a good warm starts, whenever available. To demonstrate this, we construct a sequence

Figure 6: Comparison with varying number of computed singular values (Type II)



(a) CPU time: $\beta = 1.01$ (b) Relative error: $\beta = 1.01$ (c) Matrix-vector multiplications

(d) CPU time: $\beta = 1.1$ (e) Relative error: $\beta = 1.1$ (f) Matrix-vector multiplications

of 2000 by 4000 matrices, starting from a random matrix $A^{(1)}$ generated with $\beta = 1.01$ and each subsequent matrix taking the form of $A^{(j+1)} = A^{(j)} + \frac{1}{5^{j+1}} \frac{W^{(j)}}{\|W^{(j)}\|_F}$, for $j = 1, \cdots, 14$, where $W^{(j)}$ is a random matrix with i.i.d. standard Gaussian elements. For the three subspace-iteration based algorithms, at each step, we set the initial guess as the output of the previous step. The results are plotted in Figure 8. As we can see, as the sequence "converges", the cost of solving each problem generally decreases for the subspace-iteration based algorithms, while it remains flat for EIGS and LANSVD.

Finally, we present an overall evaluation of the five solvers using performance profiles introduced by Moré and Dolan [5]. These profiles provide a way to graphically compare a performance quantity, say $t_{p,s}$ representing the number of iterations or CPU time required by solver s to solve problem p for a group of solvers on a set of test problems.

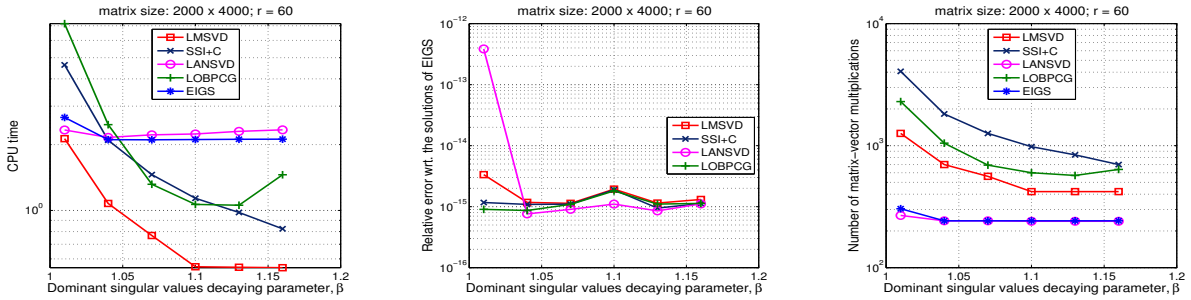
Let us define $r_{p,s}$ to be the ratio between the quantity $t_{p,s}$, obtained on problem p by solver s, over the lowest such quantity obtained by any of the solvers on problem p, i.e., $r_{p,s} := t_{p,s}/\min_s\{t_{p,s}\}$. Whenever solver s fails to solve problem p, the ratio $r_{p,s}$ is set to infinity or some sufficiently large number. Then, for $\tau \geq 0$, the ratio

$$\pi_s(\tau) := \frac{\text{number of problems where } r_{p,s} \leq \tau}{\text{total number of problems}}$$

is the fraction of the test problems that were solved by solver s within a factor $\tau \geq 1$ of the performance obtained by the best solver. The performance plots present $\pi_s(\tau)$ for each solver s as a function of τ. The curves are always monotonically nondecreasing, and the closer a curve to the unity, the better.

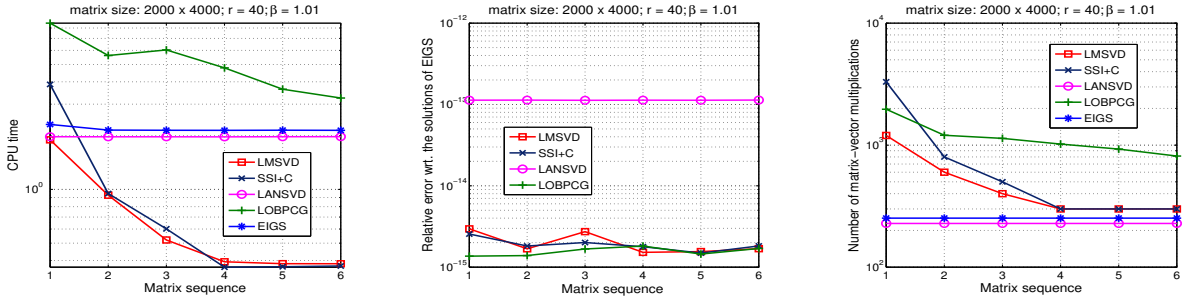
In this experiment, 540 test matrices are generated using Model 2. The sizes of the test matrices are

Figure 7: Comparison with varying decay parameter β (Type III)



(a) CPU time in seconds (b) Relative error (c) Matrix-vector multiplications

Figure 8: warm start.



(a) CPU time in seconds (b) Relative error (c) Matrix-vector multiplications

(m, n) where m ranges from 2000 to 6000 increment 1000, and n ranges from m to 6000 increment 1000 as well. Hence, there are 15 types of matrix shapes. For each of these matrices, we let r range from $0.01m$ to $0.06m$ increment $0.01m$, and β range from 1.01 to 1.15 increment 0.03. In total, this procedure generates $15 * 6 * 6 = 540$ matrices.

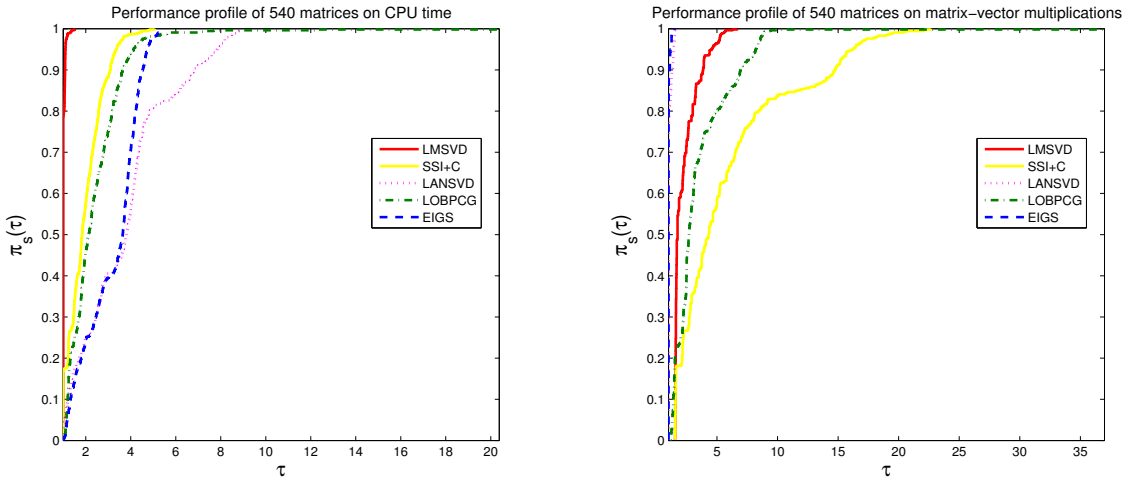
Performance profiles on CPU time and the number of matrix-vector multiplications are given in Figure 9. Plot (a) on the left show that, on the 540 tested problems, (i) LMSVD is the best performer in terms of CPU time, and (ii) it always solved problems in no more than twice of the fastest time among the five solvers. On the other hand, plot (b) shows that both EIGS and LANSVD require fewer matrix-vector multiplications than LMSVD does, while LMSVD uses fewer matrix-vector multiplications than the other two subspace-iteration based algorithms SSI+C and LOBPCG.

In addition, the average relative error with respect to the solutions obtained by EIGS over the 540 test problems are reported in Table 2.

5.4 Performance comparison on an application problem

In this subsection, we test the performance of LMSVD on problems from robust principal component pursuit, which is a recent technique with potential applications in image and video analysis and other applica-

Figure 9: Performance profile



(a) CPU time

(b) the number of matrix-vector multiplications

Table 2: Average relative error with respect to EIGS

LMSVD	SSI+C	LANSVD	LOBPCG
6.5675e-015	9.3449e-015	9.7473e-015	6.5973e-014

tions where data is either incomplete or corrupted. Specifically, the problem is to separate a low-rank matrix L_0 and a sparse matrix S_0 from their given sum $D = L_0 + S_0 \in \mathbb{R}^{m\times n}$. It has been shown in [3] that, under some suitable conditions, its solution can be found by solving the convex optimization problem:

$$\min_{L,S\in\mathbb{R}^{m\times n}} \quad \|L\|_* + \mu\|S\|_1 \quad \text{s.t.} \quad L + S = D, \tag{45}$$

where $\mu > 0$ is a proper weighting factor, $\|L\|_*$ is the nuclear norm of L (the sum of its singular values) and $\|S\|_1$ is the sum of the absolute values of all entries of S (not the matrix 1-norm).

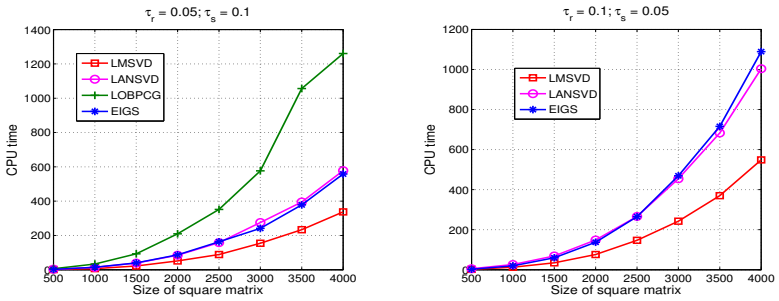
There are a number of methods proposed for solving (45), such as the exact and inexact augmented Lagrange multiplier (ALM and IALM) algorithms proposed in [15] and [29], respectively. However, no matter what method is used to solve (45), the main cost is the computation of the dominant SVD of certain matrices related to the nuclear norm term. Here we study how the performance of the IALM solver changes as its default SVD solver LANSVD is replaced by another SVD solver, in particular by LMSVD.

We first test the algorithms on matrices of the form $D = L_0 + S_0$, where L_0 is rank-r and S_0 is sparse. Specifically, $L_0 = XY^{\mathrm{T}}$, where X and Y are $n \times r$ i.i.d. standard Gaussian random matrices; $S_0 = \tau\tilde{S}$, where $\tilde{S}$ is a sparse matrix whose nonzero positions are uniformly sampled and nonzero element values are independently chosen from the standard Gaussian distribution with variance $1/n$, and τ is a scalar which makes S_0 roughly the same magnitudes as L_0. The sizes of the test matrices are $m = n =$

$500, 1000, 1500, ..., 4000$. We test two groups of test problems. In each of them, the rank ratio parameter $\tau_r \triangleq r/n$ and the density parameter $\tau_s \triangleq \mathtt{nnz}(S_0)/n^2$ are set to $(0.05, 0.10)$ and $(0.10, 0.05)$, respectively. In all cases, the balancing parameter μ is fixed as $1/n$.

We run IALM with four SVD solvers, LMSVD, LANSVD, EIGS and LOBPCG on the first group of test problems, and with the first three solvers on the second group, excluding LOBPCG due to a runtime error message it gives. Since all solvers achieved a very similar accuracy, we only summarize the CPU time results in Figure 10. We observe that using LMSVD in IALM reduces the CPU time by 30-50% comparing to using LANSVD. The performance of IALM using EIGS is similar to that of using LANSVD, while the results of using LOBPCG are not competitive.

Figure 10: CPU time results on randomly generated matrix separation problems.



(a) CPU time (group 1) (b) CPU time (group 2)

We next consider the case of recovering an image (as an approximate low-rank matrix) from its sum with a sparse random matrix. We test two images: a rank-2 image "checkerboard" of size 512×512 and an approximately low-rank image "brickwall" of size 772×1165, shown in Figures 11 (a) and (b), respectively. The images are corrupted by adding a sparse matrix whose nonzero entries take random values uniformly distributed in $[0, 1]$ and the locations of the nonzero entries are uniformly random. The density level $\mathtt{nnz}(S^*)/m^2$ was set to 5%. On each these problems, IALM calls a SVD solver about 18 times. Table 3 reports the average number of singular values, denoted by "av.dsv", computed at each iteration, the relative error, denoted by "rel.err", of the recovered image, and the CPU time used. The recovered images by IALM with LMSVD are depicted in Figures 11 (a) and (b). Since the recovered images with other SVD solvers are almost visually identical, they are not included here for the sake of space.

Problem Name	IALM+LMSVD			IALM+LANSVD			IALM+LOBPCG			IALM+EIGS		
	av.dsv	rel.err	CPU	av.dsv	rel.err	CPU	av.dsv	rel.err	CPU	av.dsv	rel.err	CPU
checkerboard	59.0	0.23	2.95	59.0	0.23	7.91	59.0	0.23	7.09	59.0	0.23	3.42
brickwall	76.8	0.29	10.78	76.8	0.29	23.55	76.8	0.29	22.43	76.8	0.29	17.05
video-hall	56.4	-	23.74	56.4	-	39.72	56.4	-	115.68	56.4	-	53.64

Table 3: image restoration and video separation

Finally, with different SVD solvers we apply IALM to a video separation problem in [3], which aims to separate a video into a static background and moving objects. Every frame of a video is reshaped into a

long column vector and then collected into a matrix. As such, the number of rows in the matrix equals to the number of pixels and the number of columns equals to the number of frames in a video. We take only part of the clip with 500 frames to reduce the storage and computation required, resulting in a 25344×500 matrix separation problem. In this test, IALM needs to call a SVD solver 7 times. The average number of dominant SVD computed at each iteration and the CPU time are again included in Table 3 as the last row. We observe that IALM with LMSVD requires the least amount of CPU time than with other solvers. A few recovered frames by IALM with LMSVD are given in Figure 11.

6 Conclusions

We propose and study a limited memory subspace optimization technique to accelerate the classical simultaneous subspace iteration (SSI) method for computing truncated singular value decompositions (SVDs) of large and unstructured matrices. The main idea is to introduce an intermediate iterate which maximizes the Rayleigh-Ritz function in a subspace spanned by a few previous iterates without introducing any extra matrix-block multiplication. A stable algorithm is constructed to produce high precision solutions if so specified, and a general convergence result is established based on minimal assumptions.

Comprehensive numerical experiments are conducted to evaluate the performance of our new truncated SVD solver LMSVD in comparison to a few state-of-the-art solvers. Numerical results indicate that LMSVD does accelerate SSI (or SSI plus Chebyshev acceleration) significantly to a point where its performance becomes competitive, ofter superior, to the best publicly available solvers on a wide range of unstructured matrices under the MATLAB environment. We hope that LMSVD will become a new addition to the computational toolbox useful in data-intensive applications.

Finally, we opine that on unstructured matrices an SSI-based solver like LMSVD, rich in level 3 BLAS operations, offers a greater opportunity to be efficiently parallelized on massively parallel computers than that offered by solvers based on Lanczos iterations.

References

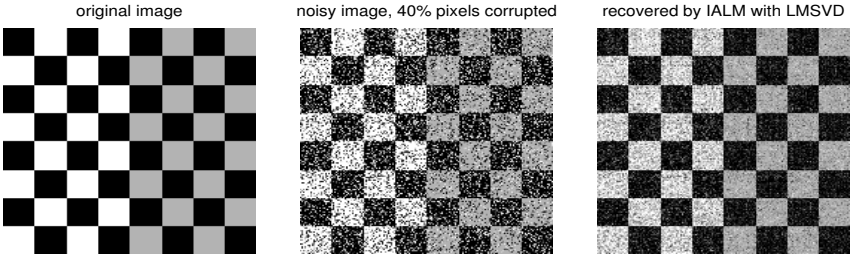
[1] E. ANDERSON, Z. BAI, J. DONGARRA, A. GREENBAUM, A. MCKENNEY, J. DU CROZ, S. HAMMERLING, J. DEMMEL, C. BISCHOF, AND D. SORENSEN, *Lapack: a portable linear algebra library for high-performance computers*, in Proceedings of the 1990 ACM/IEEE conference on Supercomputing, Supercomputing '90, IEEE Computer Society Press, 1990, pp. 2–11.

[2] M. BOLLHÖFER AND Y. NOTAY, *JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices*, Comput. Phys. Comm., 177 (2007), pp. 951–964.

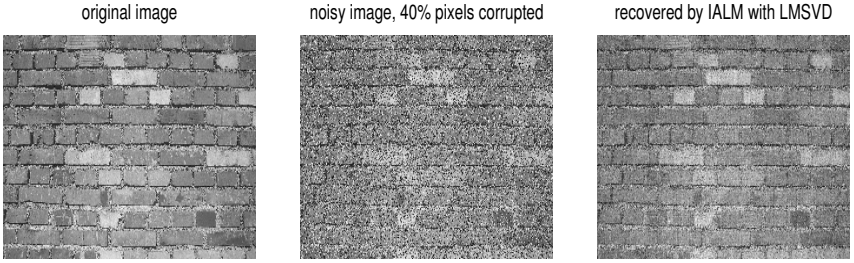
[3] E. J. CANDÈS, X. LI, Y. MA, AND J. WRIGHT, *Robust principal component analysis?*, J. ACM, 58 (2011), pp. 1– 37.

[4] Y. DAI, *Fast algorithms for projection on an ellipsoid*, SIAM J. Optim., 16 (2006), pp. 986–1006.

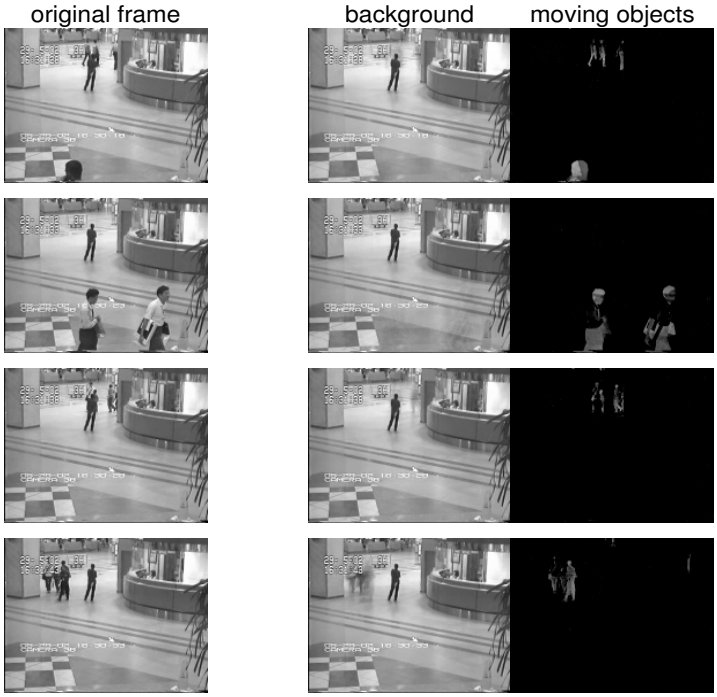
Figure 11: Image restoration and video separation problems.



(a) Recovery results for "checkboard"



(b) Recovery results for "brick wall"



(c) Video separation by IALM with LMSVD

[5] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.

[6] P. DRINEAS, R. KANNAN, AND M. W. MAHONEY, *Fast Monte Carlo algorithms for matrices. II. Computing a low-rank approximation to a matrix*, SIAM J. Comput., 36 (2006), pp. 158–183.

[7] G. H. GOLUB AND C. F. VAN LOAN, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[8] N. I. M. GOULD, D. ORBAN, AND P. L. TOINT, *Numerical methods for large-scale nonlinear optimization*, Acta Numerica, (2005), pp. 299–361.

[9] N. HALKO, P. G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[10] V. HERNÁNDEZ, J. E. ROMÁN, A. TOMÁS, AND V. VIDAL, *A survey of software for sparse eigenvalue problems*, tech. rep., Scalable Library for Eigenvalue Problem Computations, 2009.

[11] A. V. KNYAZEV, *Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[12] R. M. LARSEN, *Lanczos bidiagonalization with partial reorthogonalization*, Aarhus University, Technical report, DAIMI PB-357, September 1998.

[13] R. B. LEHOUCQ, *Implicitly restarted Arnoldi methods and subspace iteration*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 551–562.

[14] R. B. LEHOUCQ, D. C. SORENSEN, AND C. YANG, *ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, vol. 6 of Software, Environments, and Tools, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998.

[15] 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.

[16] H. RUTISHAUSER, *Computational aspects of F. L. Bauer's simultaneous iteration method*, Numer. Math., 13 (1969), pp. 4–13.

[17] H. RUTISHAUSER, *Simultaneous iteration method for symmetric matrices*, Numer. Math., 16 (1970), pp. 205–223.

[18] Y. SAAD, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Mathematics of Computation, 42 (1984), pp. 567–588.

[19] Y. SAAD, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, 1992.

[20] Y. SAAD, J. R. CHELIKOWSKY, AND S. M. SHONTZ, *Numerical methods for electronic structure calculations of materials*, SIAM Rev., 52 (2010), pp. 3–54.

[21] D. C. SORENSEN, *Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations*, in Parallel numerical algorithms (Hampton, VA, 1994), vol. 4 of ICASE/LaRC Interdiscip. Ser. Sci. Eng., Kluwer Acad. Publ., pp. 119–165.

[22] ——, *Numerical methods for large eigenvalue problems*, Acta Numer., 11 (2002), pp. 519–584.

[23] A. STATHOPOULOS AND C. F. FISCHER, *A davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix*, Computer Physics Communications, 79 (1994), pp. 268–290.

[24] G. W. STEWART, *Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices*, Numer. Math., 25 (1975/76), pp. 123–136.

[25] ——, *Matrix algorithms Vol. II: Eigensystems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.

[26] W. J. STEWART AND A. JENNINGS, *A simultaneous iteration algorithm for real matrices*, ACM Trans. Math. Software, 7 (1981), pp. 184–198.

[27] M. TURK AND A. PENTLAND, *Eigenfaces for recognition*, J. Cognitive Neuroscience, 3 (1991), pp. 71–86.

[28] C. YANG, J. C. MEZA, AND L.-W. WANG, *A trust region direct constrained minimization algorithm for the Kohn-Sham equation*, SIAM J. Sci. Comput., 29 (2007), pp. 1854–1875.

[29] X. YUAN AND J. YANG, *Sparse and low-rank matrix decomposition via alternating direction methods*, Pacific Journal of Optimization, (2012).

[30] Y. YUAN, *Subspace techniques for nonlinear optimization*, in Some topics in industrial and applied mathematics, vol. 8 of Ser. Contemp. Appl. Math. CAM, Higher Ed. Press, 2007, pp. 206–218.