PRIMME_SVDS: A PRECONDITIONED SVD SOLVER FOR COMPUTING ACCURATELY SINGULAR TRIPLETS OF LARGE MATRICES BASED ON THE PRIMME EIGENSOLVER

LINGFEI WU* AND ANDREAS STATHOPOULOS*

Abstract. The computation of a few singular triplets of large, sparse matrices is a challenging task, especially when the smallest magnitude singular values are needed in high accuracy. Most recent efforts try to address this problem through variations of the Lanczos bidiagonalization method, but algorithmic research is ongoing and without production level software. We develop a high quality SVD software on top of the state-of-the-art eigensolver PRIMME that can take advantage of preconditioning, and of PRIMME's nearly-optimal methods and full functionality to compute both largest and smallest singular triplets. Accuracy and efficiency is achieved through a hybrid, two-stage meta-method, primme_svds. In the first stage, primme_svds solves the normal equations problem up to the best achievable accuracy. If further accuracy is required, the method switches automatically to an eigenvalue problem with the augmented matrix. Thus it combines the advantages of the two stages, faster convergence and accuracy, respectively. For the augmented matrix, solving the interior eigenvalue is facilitated by a proper use of the good initial guesses from the first stage and an efficient implementation of the refined projection method. We also discuss how to precondition primme_svds and to cope with some issues that arise. The method can be used with or without preconditioning, on large problems, and can be called with its full functionality from MATLAB through our MEX interface. Numerical experiments illustrate the efficiency and robustness of the method.

1. Introduction. The Singular Value Decomposition (SVD) is a ubiquitous computational kernel in science and engineering. Many applications require a few of the largest singular values of a large sparse matrix A and the associated left and right singular vectors (singular triplets). These applications are from diverse areas, such as social network analysis, image processing, textual database searching, and control theory. A smaller, but increasingly important, set of applications requires a few smallest singular triplets. Examples include least square problems, determination of matrix rank, low rank approximation, and computation of pseudospectrum [1, 2, 3, 4]. Recently we have used such techniques to reduce the variance in Monte Carlo estimations of the the trace of the inverse of a large sparse matrix.

It is well known that the computation of the smallest singular triplets presents challenges both to the speed of convergence and the accuracy of iterative methods. In this paper, we mainly focus on the problem of finding the smallest singular triplets. Assume $A \in \Re^{m\times n}$ is a large sparse matrix with full column rank and $m \geq n$. The (economy size) singular value decomposition of A can be written as:

$$A = U\Sigma V^T \tag{1.1}$$

where $U = [u_1, \ldots, u_n] \in \Re^{m\times n}$ is an orthonormal set of the left singular vectors and $V = [v_1, \ldots, v_n] \in \Re^{n\times n}$ is the unitary matrix of the right singular vectors. $\Sigma = diag(\sigma_1, \ldots, \sigma_n) \in \Re^{n\times n}$ contains the singular values of A, $\sigma_1 \leq \ldots \leq \sigma_n$. We will be looking for the smallest $k \ll n$ singular triplets $\{\sigma_i, u_i, v_i\}, i = 1, \ldots, k$.

There are two approaches to compute the singular triplets $\{\sigma_i, u_i, v_i\}$ by using a Hermitian eigensolver. Using MATLAB notation, the first approach seeks eigenpairs of the augmented matrix $B = [0\ A^T; A\ 0] \in \Re^{(m+n)\times(m+n)}$, which has eigenvalues $\pm\sigma_i$ with corresponding eigenvectors $([v_i; u_i], [-v_i; u_i])$, as well as $m - n$ zero eigenvalues [6, 7, 8]. The main advantage of this approach is that iterative methods can potentially

*Department of Computer Science, College of William and Mary, Williamsburg, Virginia 23187-8795, U.S.A.(lfwu@cs.wm.edu, andreas@cs.wm.edu)

compute the smallest singular values accurately, i.e., with residual norm close to $O(\|A\|\epsilon_{mach})$. However, convergence of eigenvalue iterative methods is slow since it is a highly interior eigenvalue problem, and even the use of iterative refinement or inverse iteration involves a maximally indefinite matrix [5]. For restarted iterative methods convergence is even slower, irregular, and often the required eigenvalues are missed since the Rayleigh-Ritz projection method does not effectively extract the appropriate information for interior eigenvectors [29, 30, 33].

The second approach computes eigenpairs of the normal equations matrix $C = A^T A \in \Re^{n\times n}$ which has eigenvalues σ_i^2 and associated eigenvectors v_i. If $\sigma_i \neq 0$, the corresponding left singular vectors are obtained as $u_i = \frac{1}{\sigma_i} A^T v_i$. C is implicitly accessed through successive matrix-vector multiplications. The squaring of the singular values works in favor of this approach with Krylov methods, especially with largest singular values since their relative separations increase. Although the separations of the smallest singular values become smaller, we show in this paper that this approach still is faster than Krylov methods on B because it avoids indefiniteness. On the other hand, squaring the matrix limits the accuracy at which smallest singular triplets can be obtained. Therefore, this approach is typically followed by a second stage of iterative refinement for each needed singular triplet which resolves the required accuracy [11, 12, 13]. However, this one-by-one iterative refinement does not exploit information from other singular vectors and thus is it not as efficient as an eigensolver applied on B with the estimates of the first stage.

The Lanczos bidiagonalization (LBD) method [1, 6] is accepted as an accurate and more efficient method for seeking singular triplets (especially smallest), and numerous variants have been proposed [22, 17, 19, 14, 15, 16, 18, 25]. LBD builds the same subspace as Lanczos on matrix C, but since it works on A directly, it avoids the numerical problems of squaring. However, the Ritz vectors often exhibit slow, irregular convergence when the smallest singular values are clustered. To address this problem, harmonic projection [19, 14], refined projection [17], and their combinations [18] have been applied to LBD. Despite remarkable algorithmic progress, current LBD methods are still in development, with only few existing MATLAB implementations that serve mainly as a testbed for mathematical research. Moreover, we show that a two stage approach based on a well designed eigenvalue code (such as PRIMME) can be more robust and efficient for a few singular triplets. Most importantly, our approach can use preconditioning, something that is not directly possible with LBD but becomes crucial because of the difficulty of the problem even for medium matrix sizes.

The Jacobi-Davidson type SVD method, JDSVD [9, 10], is based on an inner-outer iteration and can also use preconditioning. It obtains the left and right singular vectors directly from a projection of B on two subspaces and, although it avoids the numerical limitations of matrix C, it needs a harmonic [29, 32, 31] or a refined projection method [33, 34] to avoid the irregular convergence of the Rayleigh-Ritz method. JDSVD often has difficulty computing the smallest singular values of a rectangular matrix A, especially without preconditioning, due to the presence of zero eigenvalues of B. Also, JDSVD is only available in a MATLAB research implementation.

Recently, based on [26], an inverse free preconditioned Krylov subspace method, SVDIFP, has appeared for the singular value problem [27]. The implementation includes the robust incomplete factorization (RIF) [28] for the normal equations matrix, but other preconditioners can also be used. To circumvent the intrinsic difficulties of filtering out the zero eigenvalues of the augmented matrix B, the method works with the normal equations matrix C, but computes directly the smallest singular values

of A and not the eigenvalues of C. Thus good numerical accuracy can be achieved but, as we show later, at the expense of efficiency. Moreover, the design of SVDIFP is based on restarting with a single vector, which is not effective when seeking more than one singular values. Finally, the SVDIFP code is only available in MATLAB.

Given the above research activity in SVD algorithms, it is surprising that there is a lack of good quality software for computing the partial SVD, especially with preconditioning. Without preconditioning, SVDPACK [20] and PROPACK [21, 22] implement variants of (block) Lanczos methods. In addition, PROPACK implements an implicitly restarted LBD. However, SVDPACK can only compute largest singular triplets while PROPACK has to leverage shift-and-invert techniques to search for smallest. SLEPc offers some limited functionality for computing the partial SVD problem of a large, sparse rectangular matrix using various eigensolvers working on B or C [23]. It also implements a parallel LBD method but focuses mainly on largest singular values [24]. With the growing size and difficulty of real-world problems, there is a clear need for a high quality SVD solver software that allows for additional flexibility, implements state-of-the-art methods, and allows for preconditioning.

In this paper we address this need by developing a high quality SVD software based on the state-of-the-art package PRIMME (PReconditioned Iterative Multi-Method Eigensolver) [37]. The novelty is not on the interface but on a hybrid, two-stage method that achieves both efficiency and accuracy for both largest and smallest singular values under limited memory. In the first stage, the proposed method primme_svds solves an extreme eigenvalue problem on C up to the user required accuracy or up to the accuracy achievable by the normal equations. If further accuracy is required, primme_svds switches to a second stage where it utilizes the eigenvectors and eigenvalues from C as initial guesses to a Jacobi-Davidson method on B, which has been enhanced by a refined projection method. The appropriate choices for tolerances, transitions, selection of target shifts, and initial guesses are handled automatically by the method. We also discuss how to precondition primme_svds and to cope with possible issues that can arise. Our extensive numerical experiments show that primme_svds can be considerably more efficient than all other methods when computing a few of the smallest singular triplets, even without a preconditioner. With a good preconditioner, the primme_svds method can be much more efficient and more robust than the JDSVD and SVDIFP methods.

In Section 2 we motivate the two stage SVD method based on the convergence of other Krylov methods to the smallest magnitude eigenvalue of B and C. In Section 3, we develop the components of the two stage SVD method. In Section 4, we describe how to precondition primme_svds, and how to dynamically inspect the quality of preconditioning at the two different stages. In Section 5, we present extensive experiments that corroborate our conclusions.

We denote by $\|.\|$ the 2-norm of a vector or a matrix, by A^T the transpose of A, by I the identity matrix, $\kappa(A) = \frac{\sigma_n}{\sigma_1}$, and by $K_m(A, v) = span\{v, Av, \ldots, A^{m-1}v\}$ the m-dimensional Krylov subspace generated by A and the initial vector v.

2. Motivation for PRIMME and a two stage strategy. We first introduce some basic iterative methods for SVD, and discuss some of the features in PRIMME that facilitate the development of a flexible SVD solver. Then, we study both the asymptotic convergence of unpreconditioned Krylov methods applied on C and B, and the quality of the subspaces built by different methods. Even without taking into account the way we extract information from the subspaces, we arrive at the conclusion that a hybrid strategy should be preferred.

2.1. The LBD, JDSVD, and SVDIFP methods. The LBD method, [6, 7], starts with unit vectors p_1 and q_1 and after k steps produces the following decomposition as a partial Lanczos bidiagonalization of A:

$$\begin{aligned} AP_k &= Q_k B_k, \\ A^T Q_k &= P_k B_k^T + r_k e_k^T, \end{aligned} \tag{2.1}$$

where the r_k is the residual vector at k-th step, e_k is the $k-$th orthocanonical vector,

$$B_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & \beta_{k-1} \\ & & & \alpha_k \end{pmatrix} = Q_k^T A P_k,$$

and Q_k and P_k are orthonormal bases of the Krylov subspaces $K_k(AA^T, q_1)$, and $V_k = K_k(A^T A, p_1)$ respectively. With properly chosen starting vectors, LBD produces mathematically the same space as the symmetric Lanczos method on B or C [18, 19].

To approximate the singular triplets of A, LBD solves the small singular value problem on B_k, and uses the corresponding Ritz approximations from Q_k and P_k as left and right singular vectors. To address the rapid loss of orthogonality of P_m and Q_m in finite precision, full [7], partial [22], or one-sided reorthogonalization [14, 18] strategies have been applied to variants of LBD. Because, all such solutions become expensive when k is large, restarted LBD versions have been studied [19, 14, 17, 22]. The goal is twofold: restart with sufficient subspace information to maintain a good convergence, and identify the appropriate Ritz information to restart with. The former problem is tackled with implicit or thick restarting [38]. The latter problem is tackled with combinations of harmonic and refined projection methods. For example, IRLBA [15] uses a thick restarted block LBD with harmonic projection, while IRRHLB [18] first computes harmonic Ritz vectors, and then uses their Rayleigh quotients in a refined projection to extract refined Ritz vectors from P_k and Q_k.

The JDSVD method [9] extends the Jacobi-Davidson method and its correction equation for singular value problems by exploiting the special structure of the augmented matrix B. Similarly to LBD, JDSVD computes singular values, not eigenvalues, of the projection matrix, and the left and right singular vectors from separate spaces. Because good quality approximations are important not only for restarting but also in the correction equation, various projection methods can benefit JDSVD. We introduce only the standard choice where the test and search space are the same.

Let U and V be the bases of the left and right search spaces with dimension k. Computing a singular triplet of $H = U^T AV$ yields (θ, Uc, Vd) as the Ritz approximation of a corresponding singular triplet of A. Then JDSVD obtains new corrections s and t for U and V by solving (approximately) the following correction equation:

$$\begin{pmatrix} P_u & 0 \\ 0 & P_v \end{pmatrix} \begin{pmatrix} -\theta I_m & A \\ A^T & -\theta I_n \end{pmatrix} \begin{pmatrix} P_u & 0 \\ 0 & P_v \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} Av - \theta u \\ A^T u - \theta v \end{pmatrix} \tag{2.2}$$

where $P_v = I_n - vv^T, P_u = I_m - uu^T$. The left and right corrections s, t are then orthogonalized against U and V respectively. Clearly, different Galerkin choices can be used to compute harmonic or refined singular triplets. JDSVD uses thick restarting [38, 39] with possibly extra Ritz vectors retained from the previous iteration, similarly

to the locally optimal Conjugate Gradient recurrence [35]. Most importantly, the JDSVD method can take advantage of preconditioning when solving (2.2).

The SVDIFP method [27] extends the EIGIFP method [26] by computing the smallest singular values as eigenvalues of the generalized eigenvalue problem $(C,\ I)$. Given the current approximation $(x_i, \rho_i = x_i^T C x_i / x_i^T x_i)$, it constructs a new approximation x_{i+1} by a projection of $(C,\ I)$ onto the Krylov subspace,

$$K_m(C - \rho_i I, x_i) = span\{x_i, (C - \rho_i I)x_i, \ldots, (C - \rho_i I)^{m-1} x_i\}. \tag{2.3}$$

The space can be preconditioned by any available preconditioner $LDL^T \approx C - \rho_i I$. To avoid the numerical problems of projecting on C, SVDIFP computes the approximate singular values directly from a two sided projection of C, similarly to the LBD.

2.2. The PRIMME eigenvalue package. Our goal is to enable practitioners to solve a variety of large, sparse singular value problems with unprecedented efficiency, robustness, and accuracy. Our methods should be able to use preconditioning because very slow convergence is a limiting factor for seeking smallest singular triplets. Moreover, large problem size suggests the use of advanced restarting techniques so that memory savings do not impede convergence. These desired functionalities are found in the PRIMME eigensolver [37], which also includes a host of additional features that can further help develop and fine-tune an efficient SVD solver.

PRIMME implements a wide variety of preconditioned eigenvalue algorithms, including the nearly optimal methods GD+k and JDQMR. Near optimality is used in the sense of achieving similar convergence to a similar method with unlimited memory (e.g., unrestarted Lanczos or unrestarted Generalized Davidson). PRIMME also allows for a dynamic choice of the best method based on runtime measurements. It also implements many techniques for improving efficiency and robustness, including block-methods and locking. Unlike Lanczos or LBD, PRIMME can use as many initial vector guesses as there are available. Also given a set of user provided shifts, PRIMME can find interior eigenvalues closest to in absolute value or on the left or right side of each of these shifts. These two features are important for our two stage SVD method because there are very good eigenvalue and eigenvector approximations from the first stage. PRIMME is a parallel, high performance implementation which has proved faster and more robust than almost any other eigensolver when seeking a small number of extreme eigenvalues of large sparse Hermitian matrices. It is natural, therefore, to build our SVD solver on top of it.

First, we need to understand whether PRIMME should be used to solve the SVD as an eigenvalue problem on C or on B, and whether its convergence will be competitive to other SVD methods. The following sections address these issues.

2.3. Asymptotic convergence of Krylov methods on C and B. When seeking largest singular values, it is accepted that Krylov methods on C are faster than on B [9, 19, 27, 13]. The argument is straightforward.

THEOREM 2.1. *Let $\gamma_B = \frac{\sigma_n - \sigma_{n-1}}{\sigma_{n-1} + \sigma_n}$ and $\gamma_C = \frac{\sigma_n^2 - \sigma_{n-1}^2}{\sigma_{n-1}^2 - \sigma_1^2}$ be the gap ratios of the largest eigenvalue of matrices B and C, respectively. Then, for the largest eigenvalue, the asymptotic convergence of Lanczos on C is 2 times faster than Lanczos on B.*

Proof. The asymptotic convergence rate is the square root of the gap ratio. Then:

$$\gamma_C = \frac{(\sigma_n - \sigma_{n-1})(\sigma_n + \sigma_{n-1})^2}{(\sigma_n + \sigma_{n-1})(\sigma_{n-1}^2 - \sigma_1^2)} = \gamma_B \frac{(\sigma_n + \sigma_{n-1})^2}{(\sigma_{n-1}^2 - \sigma_1^2)} > \gamma_B \frac{4\sigma_{n-1}^2}{(\sigma_{n-1}^2 - \sigma_1^2)} = \frac{4\gamma_B}{1 - (\frac{\sigma_1}{\sigma_{n-1}})^2}.$$

Therefore, for $\sigma_1 \approx 0$, the asymptotic convergence rate $\sqrt{\gamma_C} > 2\sqrt{\gamma_B}$. In the less interesting case $\sigma_1 \to \sigma_{n-1}$, Lanczos on C is arbitrarily faster than on B. □

For smallest singular values the literature is less clear, although methods that work on C have been avoided for numerical reasons. In previous experiments we have observed much faster convergence with approaches on C than on B [41]. To obtain some intuition, we perform a basic asymptotic convergence analysis of Krylov methods working on C or on B trying to compute the smallest magnitude eigenvalue.

LEMMA 2.2. *Let the union of two intervals: $K = [-a, -b] \cup [c, d]$, $-b < 0 < c$, and $p_k(x)$ the optimal degree-k polynomial that is as small as possible on K and $p_k(0) = 1$. Let $\epsilon_k = \max_{x \in K} |p_k(x)|$, and $\rho = \lim_{k \to \infty} \epsilon_k^{1/k}$. Then asymptotically:*

$$\rho \simeq 1 - \sqrt{\frac{bc}{da}}.$$

Proof. This is an application of Theorem 5 in [40]. □

This ρ translates to an upper bound for the asymptotic convergence rate of any Krylov solver applied to an indefinite matrix whose spectrum lies in the interval K. Thus it can also be used for the convergence rate to the smallest positive eigenvalue of the augmented matrix B. Assume σ_1 is a simple eigenvalue of B and thus σ_1^2 is a simple eigenvalue of C. Define its gap ratio in C as, $\gamma = \frac{\sigma_2^2 - \sigma_1^2}{\sigma_n^2 - \sigma_2^2}$, and assume $\gamma \ll 1$.

THEOREM 2.3. *Consider the spectrum of the matrix $B - \sigma_1 I$, which lies (except for the zero eigenvalue) in the two intervals: $K = [-\sigma_n - \sigma_1, -2\sigma_1] \cup [\sigma_2 - \sigma_1, \sigma_n - \sigma_1]$. The asymptotic convergence rate for any Krylov solver that finds σ_1 is bounded by:*

$$\rho = 1 - \sqrt{\gamma \; \frac{2\sigma_1}{\sigma_2 + \sigma_1} \; \frac{\sigma_n^2 - \sigma_2^2}{\sigma_n^2 - \sigma_1^2}}.$$

Proof. Clearly, the optimal polynomial $p_k(x)$ of Lemma 2.2 is the best polynomial for finding σ_1. Applying the Lemma for the specific bounds for this interval we get:

$$\frac{bc}{ad} = \frac{2\sigma_1(\sigma_2 - \sigma_1)}{(\sigma_n + \sigma_1)(\sigma_n - \sigma_1)} = \frac{2\sigma_1}{\sigma_2 + \sigma_1} \; \frac{\sigma_2^2 - \sigma_1^2}{\sigma_n^2 - \sigma_1^2} = \frac{2\sigma_1}{\sigma_2 + \sigma_1} \; \frac{\sigma_2^2 - \sigma_1^2}{\sigma_n^2 - \sigma_2^2} \; \frac{\sigma_n^2 - \sigma_2^2}{\sigma_n^2 - \sigma_1^2}.$$

□

LEMMA 2.4. *The bound of the asymptotic convergence rate to σ_1^2 of Lanczos on C is approximately: $q = 1 - 2\sqrt{\gamma}$.*

Proof. The bound on the rate of convergence of Lanczos for σ_1^2 is approximated as $e^{-2\sqrt{\gamma}}$ [5, p. 280]. Taking the first order approximation from Taylor series around 0, we obtain $e^{-2\sqrt{\gamma}} = 1 - 2\sqrt{\gamma} + O(\gamma)$. □

THEOREM 2.5. *A Krylov method on C that computes σ_1^2 has always faster asymptotic convergence rate than a Krylov method on B that finds σ_1, by a factor of*

$$\tau = \frac{1 - \sqrt{\gamma}\sqrt{\frac{2\sigma_1}{\sigma_2 + \sigma_1} \; \frac{\sigma_n^2 - \sigma_2^2}{\sigma_n^2 - \sigma_1^2}}}{1 - 2\sqrt{\gamma}}. \tag{2.4}$$

Proof. For the method on C to be faster it must hold $\tau > 1$ or $\frac{2\sigma_1}{\sigma_2 + \sigma_1} \; \frac{\sigma_n^2 - \sigma_2^2}{\sigma_n^2 - \sigma_1^2} < 4$. Basic manipulations lead to the condition $(4 - 2\frac{\sigma_n^2 - \sigma_2^2}{\sigma_n^2 - \sigma_1^2})\sigma_1 >= -4\sigma_2$. Since $\frac{\sigma_n^2 - \sigma_2^2}{\sigma_n^2 - \sigma_1^2} < 1$ and all $\sigma_i > 0$, the above condition always holds. □

First, we observe that if σ_1 is very close to 0, the normal equations approach becomes arbitrarily faster than the augmented one, as long as σ_2 remains bounded away from 0. Second, it is not hard to see that $\tau = 1 + O(\sqrt{\sigma_2 - \sigma_1})$, which means that the two approaches become similar with highly clustered eigenvalues. In that case, however, using a block method would increase the gap ratios and the gains from the approach on C would be larger again.

Most importantly, the above asymptotic convergence rates reflect optimal methods applied to C and B, and an extraction of the best information from the subspaces. While for extremal eigenvalues near-optimal methods, such as those in PRIMME, coupled with the Rayleigh Ritz procedure can deliver this convergence, for interior problems practical Krylov methods have a hard time achieving this convergence and extracting the best eigenvectors from the subspace. Therefore, we expect in practice the normal equations to be significantly faster than any approach based on B.

2.4. Comparison of subspaces from LBD, JDSVD and an eigenmethod. We extend the discussion on Lanczos to include the two native SVD methods. The relative differences between their convergence can be inferred by studying the subspace they build. A higher dimensional Krylov subspace implies faster convergence, if we assume eigenvector approximations can be extracted effectively from the subspace. We compare LBD, JDSVD, and Lanczos (or equivalently unpreconditioned GD) working on C and on B.

Suppose u_1, v_1 are left and right initial guesses. After k iterations ($2k$ matvecs), Lanczos working on the normal equations matrix C builds:

$$V_k = K_k(A^TA, v_1). \tag{2.5}$$

The LBD method builds both left and right Krylov spaces [14]:

$$U_k = K_k(AA^T, Av_1), \qquad V_k = K_k(A^TA, v_1). \tag{2.6}$$

The JDSVD method also builds two subspaces, each being a direct sum of two Krylov spaces of half the dimension [9]:

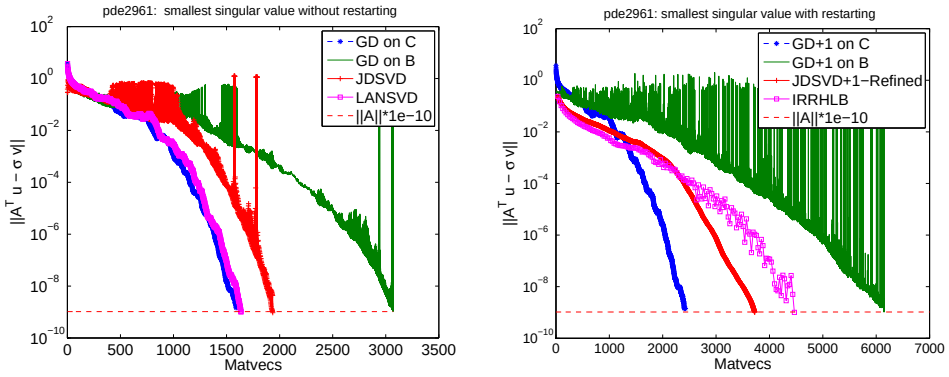
$$U_k = K_{\frac{k}{2}}(AA^T, u_1) \oplus K_{\frac{k}{2}}(AA^T, Av_1), \quad V_k = K_{\frac{k}{2}}(A^TA, v_1) \oplus K_{\frac{k}{2}}(A^TA, A^Tu_1) \tag{2.7}$$

Lanczos working on B builds $K_k(B, [v_1; u_1])$ which does not correspond exactly to the spaces above in general. In the special case of $u_1 = 0$, the subspace is given below:

$$\begin{pmatrix} U_k \\ V_k \end{pmatrix} = \begin{pmatrix} 0 \\ K_{\frac{k}{2}}(A^TA, v_1) \end{pmatrix} \oplus \begin{pmatrix} K_{\frac{k}{2}}(AA^T, Av_1) \\ 0 \end{pmatrix}. \tag{2.8}$$

Clearly, Lanczos (or GD) working on C and LBD build the same Krylov subspace for right singular vectors. The LBD method also builds the Krylov subspace for left singular vectors, and while that helps generate the bidiagonal projection, it does not improve convergence of LBD over Lanczos on C. On the other hand, Lanczos (or GD) on B and JDSVD may generate a k vector subspace, but this comes from a direct sum of Krylov spaces of $k/2$ dimension. Thus, they are expected to take twice the number of iterations of LBD in the worst case. The JDSVD subspace can also be richer than that of Lanczos on B because JDSVD handles the left and right search spaces independently for arbitrary initial guesses.

Figure 2.1(a) demonstrates the relative convergence behavior of these unrestarted methods seeking the smallest singular value of a sample matrix. Only the outer



(a) unrestarted (b) restarted

FIG. 2.1. *Comparing convergence speed of eigenmethods on C, B, Lanczos bidiagonalization, and JDSVD in both unrestarted and restarted case for matrix pde2961. LANSVD implements bidiagonalization without restarting [22] while IRRHLB is currently the most advanced bidiagonalization method with implicit restarting [18].*

iteration of JDSVD is used (inner iterations = 1). The results agree with the above analysis. The convergence speed of LBD is the same as GD on C. JDSVD is slower than LBD or GD but faster than GD on B which is about twice as slow as LBD.

In practice, however, memory and computational requirements necessitate the restarting of these methods. Because LBD, JDSVD, and GD on B extract interior spectral information from the subspaces, critical directions may be dropped during restarting, causing significant convergence slow downs and irregular behavior. The use of harmonic or refined Ritz projections during restart help ameliorate this problem up to a point. However, the problem is still an interior one. In contrast, GD+1 (one of the nearly optimal methods in PRIMME) on C should see a far smaller effect on its convergence. This is because first, it solves an extreme eigenvalue problem, where the Rayleigh-Ritz projection results in optimal eigenvalue convergence, and second, it combines thick restarting with the locally optimal conjugate gradient directions to keep appropriate information during restart [38, 35].

Figure 2.1(b) reflects the above. Once restarting is used, GD+k is faster than any other method. The only disadvantage is the limited accuracy because of the squared conditioning of C. Therefore, a natural idea is to apply another phase to refine the accuracy until user requirements are satisfied. Instead of iterative refinement, we claim that a second stage eigensolver on B is more efficient.

3. Developing the two stage strategy. We develop primme_svds, a two-stage SVD meta-method that uses the suite of methods in PRIMME to first get a fast solution of the eigenvalue problem on C to the best accuracy possible, and then resolve the remaining accuracy with a PRIMME eigensolver on B. We discuss and automate issues of accuracy, convergence tolerance, initial guesses, interior eigenvalues of B, and how to apply preconditioning.

3.1. The first stage of primme_svds. Although an eigensolver on C can be much faster than other methods, the residual norms of the eigenvalues involve $\|C\| = \|A\|^2$. Thus achieving the required numerical accuracy may not be possible.

Let (σ, u, v) be a targeted singular triplet of A and $(\tilde{\sigma}^2, \tilde{v})$ the approximating Ritz

pair from an eigenmethod working on C. Using the approximation $\tilde{u} = A\tilde{v}/\tilde{\sigma}$, we can write the following four residuals:

(3.1)
$$r_v = A\tilde{v} - \tilde{\sigma}\tilde{u}, \quad r_u = A^T\tilde{u} - \tilde{\sigma}\tilde{v}, \quad r_C = C\tilde{v} - \tilde{\sigma}^2\tilde{v}, \quad r_B = B\begin{bmatrix} \tilde{v} \\ \tilde{u} \end{bmatrix} - \tilde{\sigma}\begin{bmatrix} \tilde{v} \\ \tilde{u} \end{bmatrix}.$$

Typically a singular triplet is considered converged when $\|r_v\|$ and $\|r_u\|$ are less than a given tolerance. Since our eigenvalue methods work on C and B we need to relate the above quantities. First, it is easy to see that $r_C = A^T(A\tilde{v}) - \tilde{\sigma}^2\tilde{v} = \tilde{\sigma}A^T\tilde{u} - \tilde{\sigma}^2\tilde{v} = \tilde{\sigma}r_u$. To relate to the norm of the residual of the second stage note that $\|r_B\|^2 = (\|r_v\|^2 + \|r_u\|^2)/(\|\tilde{v}\|^2 + \|\tilde{u}\|^2)$. If the Ritz vector is normalized, $\|\tilde{v}\| = 1$, we also obtain $\|\tilde{u}\| = \|A\tilde{v}/\tilde{\sigma}\| = 1$ and $r_v = 0$. Bringing it all together (see also [41, 27]),

$$\|r_u\| = \frac{\|r_C\|}{\tilde{\sigma}} = \|r_B\|\sqrt{2}. \tag{3.2}$$

Given a user requirement $\|r_u\| < \delta\ \|A\|$, the normal equations and the augmented methods should be stopped when $r_C < \delta\ \tilde{\sigma}\ \|A\|$ and $r_B < \delta\ \|A\|/\sqrt{2}$ respectively. PRIMME's stopping criterion is $\|r_C\| < \delta_C\|C\|$, so we must provide $\delta_C = \delta\ \tilde{\sigma}/\|A\|$. In floating point arithmetic this may not be achievable since $\|r_C\|$ can only be guaranteed to achieve $O(\|C\|\epsilon_{mach})$ [5]. Thus, the criterion for the normal equations becomes,

$$\delta_C = \max\left(\delta\ \tilde{\sigma}/\|A\|, \epsilon_{mach}\right). \tag{3.3}$$

As $\tilde{\sigma}$ is not known a priori, we modify slightly the PRIMME stopping criterion.

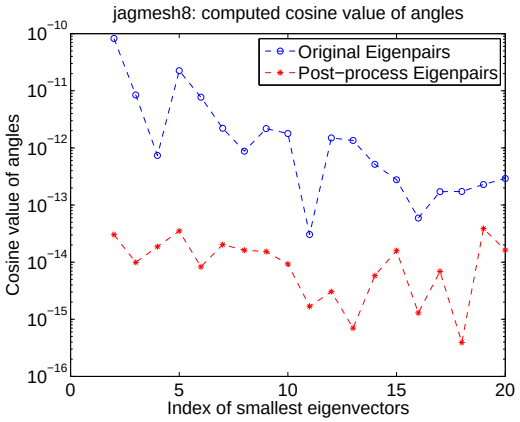
First, note that for the largest σ_n, $\delta_C = \delta$ and thus full residual accuracy is achievable with the normal equations. Since $\sigma \approx \tilde{\sigma}$, based on the Bauer-Fike bound, $|\sigma^2 - \tilde{\sigma}^2| \approx |\sigma - \tilde{\sigma}|(2\tilde{\sigma}) \leq \|r_C\| < \delta_C\|A\|^2 = \tilde{\sigma}\delta\|A\|$ and thus $|\sigma - \tilde{\sigma}| \leq \delta\|A\|/2$ so the singular values are as accurate as can be expected.

This does not hold for smaller, and in particular the smallest few, eigenvalues. Thus, if the user requires $\delta < \|A\|\epsilon_{mach}/\tilde{\sigma}$, primme_svds first makes full use of the first stage and then switches to the second stage working on B to resolve the remaining accuracy of $O(\tilde{\sigma}/\|A\|) < \kappa(A)^{-1}$. For not too ill conditioned matrices, most of the time is then spent on the more efficient first stage.

A second, more subtle issue involves the accuracy of the Ritz vectors from C which are used as initial guesses to B. We have observed that even though their residual norms are below the desired tolerance, the convergence of the interior eigenvalues in B is sometimes (but not often) irregular, with long plateaus, and might not be able to reach machine precision. This occurs when the eigenvalues are highly clustered. On the other hand, it does not occur when only one eigenvalue is sought, which implies that it has to do with the sensitivity of interior eigenvalues to the nearby eigenvectors that we pass as initial guesses [30]. Therefore, before we start stage two, we perform a complete Rayleigh Ritz procedure with the converged eigenvectors of C. Providing the new Ritz vectors as initial guesses completely cures this problem.

To understand the problem as well as the solution, consider the decomposition of the smallest Ritz vector $\tilde{u}_1$ on the exact eigenvectors of C, $\tilde{u}_1 = c_1u_1 + \sum_{i=2}^n c_iu_i$. On exit from the first stage, its residual satisfies $\|r_{\tilde{u}_1}\| < \|C\|\delta_C$, and from Bauer-Fike it also holds, $\sqrt{1-c_1^2} < \|C\|\delta_C$ and $c_i \leq \|C\|\delta_C$. Therefore, if we omit second and higher order terms, the Rayleigh quotient and the residual of $\tilde{u}_1$ can be written as:

$$\mu = \frac{\tilde{u}_1^T A\tilde{u}_1}{\tilde{u}_1^T\tilde{u}_1} = \lambda_1 + \sum_{i=2}^n (\frac{c_i}{c_1})^2\lambda_i, \qquad r_{\tilde{u}_1} = A\tilde{u}_1 - \mu\tilde{u}_1 \approx \sum_{i=2}^n c_i(\lambda_i - \lambda_1)u_i. \tag{3.4}$$



Smallest eigenvalue accuracy	
Original	3.1e-13
Post-processing	1.9e-16
Smallest residual norm	
Original	4.1e-13
Post-processing	1.3e-13

FIG. 3.1. *The cosine of angles ($c_i = \tilde{u}_1^T u_i, i = 2, \ldots, 19$) between the smallest exact eigenvectors and the smallest Ritz vector before and after pre-processing. The table compares the accuracy of the Rayleigh quotient and the residual norm for $\tilde{u}_1$ before and post-processing for matrix jagmesh8.*

As a result of the convergence behavior of iterative methods, the c_i tend to be larger for nearby eigenpairs, and fall drastically as i increases. Then, from (3.4), the accuracy of μ is dominated by nearby $(c_i/c_1)^2$. The post-processing Rayleigh-Ritz uses the k nearby converged Ritz vectors, recomputes the projection with less floating point errors, and rearranges the directions to produce smaller $c_i, i = 2, \ldots, k$, and thus better Ritz values. Of course, the additional Rayleigh-Ritz cannot improve the residual norms without the incorporation of new information in the basis. Even in floating point arithmetic, the improvements are minimal. This is evident also in (3.4) where $r_{\tilde{u}_1}$ depends on the c_i linearly, so the effect of improving the nearby c_i is small. Figure 3.1 shows these effects on a matrix that presented the original problem.

In the second stage, the improvements on c_i translate to a starting search space of better quality, and thus better Ritz (or harmonic Ritz) pairs for the interior eigenvalue problem. The original PRIMME implementation includes a verification phase which checks if any converged eigenpairs have become slightly unconverged. If so, then it performs another Rayleigh-Ritz procedure on the restarted basis and repeats the algorithm. Therefore, it is only a minor modification to, based on a user input, force another Rayleigh-Ritz procedure before PRIMME exits. Algorithm 1 shows how the specific functionality needed for the first stage is given to PRIMME.

Algorithm 1 Dynamic threshold adjusting and additional Rayleigh-Ritz in the first stage of primme_svds

1: Every iteration, PRIMME checks convergence based on an external function that obtains the current largest and targeted Ritz value $\tilde{\sigma}_n$ and $\tilde{\sigma}_i$
2: **if** (target == largest or (target == smallest and $\tilde{\sigma}_i < 1.0$)) **then**
3: adjust PRIMME's $\delta_C = \max(\delta_{user}\ \tilde{\sigma}_i/\tilde{\sigma}_n, \epsilon_{mach})$
4: **else**
5: adjust PRIMME's $\delta_C = \max(\delta_{user}/\tilde{\sigma}_n, \epsilon_{mach})$
6: **end if**
7: If requested, perform Rayleigh-Ritz on the returned vector basis

3.2. The second stage of primme_svds. We argue that solving an eigenvalue problem on matrix B with an approximate eigenspace as initial guess is a better approach than iterative refinement [12, 13] for the following reasons. First, with iterative refinement, eigenvectors are improved one by one without any synergy from the nearby subspace information. In contrast, a good quality eigensolver, such as those in PRIMME, provides global convergence to all desired pairs. Second, an inner-outer eigensolver such as JDQMR stops the inner linear solver dynamically and near-optimally to avoid exiting too early (which increases the number of outer iterations) or iterating too long (which increases the number of inner iterations). We are not familiar with similar implementations for iterative refinement. Third, iterative refinement for clustered interior eigenvalues may not be able to converge to the desired high accuracy due to the lack of proper deflation strategies [43], both at the linear solver and at the outer iteration. Naturally, a well designed eigensolver that employs locking and blocking techniques is more robust to address these problems. Finally, we point out that the correction equation of the Jacobi-Davidson method applied on B^T,

$$(I - ww^T)(B^T - \mu I)(I - ww^T)\tilde{t} = \tilde{\sigma}w - B^T w, \tag{3.5}$$

where $w^T = \begin{bmatrix} u^T v^T \end{bmatrix}$ is equivalent to the iterative refinement proposed in [12] ([9]). Therefore, JDQMR enjoys the benefits of both eigensolvers and iterative refinement.

PRIMME inserts any available initial guesses to its search space and if the guesses are less than the minimum restart size, fills the rest of the positions with a Lanczos space from a random vector to guard against extremely bad guesses. Because this is not true for the second stage, we have modified the Lanczos space to start from the first blocked targeted eigenvalue. Because of irregular convergence of interior eigenvalue problems, sometimes eigenvector approximations that were introduced initially in the search space but have not been targeted yet may degrade in quality or even be displaced. For this reason, when an eigenvector converges and is locked out of the search space, we re-introduce the initial guess of the next vector to be targeted. This resulted in significant improvement in robustness and often in convergence speed.

PRIMME provides remarkable flexibility for seeking interior eigenvalues. It accepts multiple shifts and provides three different ways to select an interior Ritz value: closest in absolute value to each shift (`primme_closest_abs`), or left or on the right of each shift. Because of the very good accuracy of eigenvalue approximations from C, the `primme_closest_abs` option is more effective at selecting the proper Ritz value during the outer iterations. More importantly, with such accurate shifts, the correction equation in JDQMR often returns the exact correction to the eigenvector after the solution of only one linear system. In the case of ill-conditioned matrices, where the shifts from the first stage are less accurate, GD+k may be preferable for the second stage because it provides better global convergence to nearby eigenvalues. However, it is computationally more expensive per iteration than JDQMR [37].

For interior eigenvalues, the Rayleigh-Ritz procedure does not have the same optimality as for extreme eigenvalues, causing the convergence to be irregular. In the GD+k method with sufficiently large search space, such problems are transient and do not affect the convergence and overall speed of the method. This is why PRIMME only implemented the Rayleigh-Ritz method originally. In the second stage of primme_svds, the availability of accurate initial guesses and shifts calls for the JDQMR method. For this method, spurious Ritz values can cause Ritz vectors to fail to converge [34]. The effect can be detrimental also during restart where major eigenvector components may be discarded and need to be recovered [29, 30, 31]. The problem is accentuated in the maximally indefinite case of SVD problems.

We addressed this problem by extending PRIMME's functionality to include the refined projection that minimizes the residual $\|BVy - \tilde{\sigma}Vy\|$ over the search space V and for a given $\tilde{\sigma}$ [33, 34]. Because the shifts $\tilde{\sigma}$ are very accurate, a harmonic Ritz procedure is not necessary, and the refined one is expected to give the best approximation for the targeted eigenpair. Our refined projection method is similar to the one in [10] and [29] that produces refined Ritz vectors for all the required eigenvectors (not just the closest to $\tilde{\sigma}$). Since $\tilde{\sigma}$ remains constant, there is no need to perform a QR factorization of $BV - \tilde{\sigma}V$ at every step. Instead, as part of Gram-Schmidt, we add one more column to the orthonormal matrix Q and to the matrix R. A full QR factorization is only needed at restart. Then, following [34], we compute the refined Ritz vectors by solving the small SVD problem with R, and replace the targeted Ritz value with the Rayleigh quotient of the first refined Ritz vector.

Algorithm 2 PRIMME enhancements and special algorithmic choices needed for the second stage of primme_svds

Initial shifts $\tilde{\sigma}_i$, initial vectors $[\tilde{v}_i\ ; A\tilde{v}_i/\tilde{\sigma}_i], i = 1, ..., k$ $qr\_full = 1$
Build an orthonormal basis V of $[K_{m-k}(B, \tilde{v}_1), \tilde{v}_i]$. Set t as the Lanczos residual
while all k eigenvalues have not converged **do**
  Orthonormalize t against $(v_i)_{i=1}^{m-1}$. Update $v_m = t, w_m = Bv_m, H_{:,m} = V^T w_m$
  if $qr\_full == 1$ **then**
    $W - \tilde{\sigma}_1 V = QR,\ qr\_full = 0$
  else
    During orthogonalization perform one-column QR updating
  end if
  Compute eigendecomposition $H = S\Theta S^T$ with θ_j ordered by closeness to $\tilde{\sigma}_1$
  Compute SVD decomposition of $R = U\Sigma S^T$, Rayleigh quotient $\theta_1 = s_1^T H s_1$
  If $(\sigma_i, [v_i; u_i])$ converged, lock, and re-introduce $[\tilde{v}_{i+1}\ ; A\tilde{v}_{i+1}/\tilde{\sigma}_{i+1}]$ into V
  If restarting, set $qr\_full = 1$
  Obtain the next vector $t = Prec(r)$ (typically with JDQMR)
end while

Solving the small SVD problem for only one shift per iteration reduces the cost of the refined procedure considerably, making it similar to the cost of computing the Ritz vectors. However, the quality of other refined Ritz vectors reduces with the distance of their Rayleigh quotient from $\tilde{\sigma}$, so they may not be as effective in a block algorithm. Nevertheless, these approximations have the desirable property of monotonic convergence as claimed in [10, 29] and also observed in our experiments. This added robustness for JDQMR more than justifies the small additional cost. Algorithm 2 shows all these second stage changes in the context of PRIMME.

3.3. Outline of the implementation. We first developed PRIMME MEX, a MATLAB interface for PRIMME. This exposes the full functionality of PRIMME to a broader class of users, who can now take advantage of MATLAB's built-in matrix times block-of-vectors operators and preconditioners. Its user interface is similar to MATLAB `eigs` allowing it to be called not only by non-expert users but also by experts that can adjust over 30 parameters. The meta-method primme_svds was then implemented as a MATLAB function on top of PRIMME MEX. This allowed flexibility for algorithmically tuning the two stages. Many of the enhancements, such as the refined projection method or a user provided stopping criterion, were implemented directly in PRIMME and will be part of its next release, which will also include a

native C implementation of primme_svds.

Algorithm 3 primme_svds: a hybrid two stage method for SVD

1: Set up the problem and call PRIMME through the PRIMME_MEX function
   Use enhancements of Algorithm 1.
2: **if** (target == smallest) **then**
3: Form shifts and initial guesses for B from the approximations of C
4: Call PRIMME for interior eigenvalues of B through PRIMME_MEX function.
   Use enhancements of Algorithm 2.
5: **end if**
6: Return converged desired singular triplets to user

Algorithm 3 summarizes the primme_svds without considering preconditioning. At a minimum, users must provide the input matrix A, or function pointers that perform matrix-vector operations with A and A^T, or directly with B and/or C. Then, primme_svds sets up the matrix-vector functions for PRIMME. By default, the first stage uses the PRIMME DYNAMIC method to decide between GD+k and JDQMR, and the second stage uses JDQMR. But users can set different eigenmethods for each stage just as they can tune any of PRIMME's parameters. We form the initial guesses for the second stage as $[\tilde{v}_i\ ; \tilde{u}_i]$, where $(\tilde{\sigma}_i^2, \tilde{v}_i)$ are the approximate eigenpairs from C, and $\tilde{u}_i = \frac{1}{\tilde{\sigma}_i} A^T \tilde{v}_i$. When the singular value is extremely small or even zero, the first stage provides low or no accuracy to $\tilde{u}_i$. In this extreme case, it is better to choose $\tilde{u}_i$ as a random vector and orthogonalize it to other left singular vectors computed after the first stage. For tiny and highly clustered singular values, we suggest that the first stage uses locking and block methods that are both available in PRIMME.

4. Preconditioning in primme_svds. The shift-and-invert technique is sometimes thought of as a form of preconditioning. If a direct factorization of the matrix A, C or B is possible, this is often the method of choice for highly indefinite or highly clustered extreme eigenproblems. For smallest singular values, the MATLAB `svds` relies solely on shift and invert ARPACK [46] using the LU factorization of B. PROPACK follows a similar strategy on C, but uses QR factorization. However, as pointed out in [27], for rectangular matrices `svds` often converges to the zero eigenvalues of B rather than the smallest singular value. Our method can also be used in shift-and-invert mode, assuming the user provides the inverted operator as a matrix-vector. For large matrices, however, preconditioners become a necessary alternative.

JDSVD accepts a preconditioner for a square matrix A or, if A is rectangular, leverages a preconditioner for $B - \tau I$ [9]. SVDIFP is combined with the robust incomplete factorization (RIF) method [28] to provide a preconditioner directly for $C - \tau I$ [27]. The advantage is that it works seamlessly for both square and rectangular matrices, but RIF may not be the best choice of preconditioner.

primme_svds accepts any user-provided preconditioning operator. In the most general form, any preconditioner directly for C or B can be used. When $M \approx A$ is available (e.g., the incomplete LU factorization of a square matrix), primme_svds forms $M^{-1}M^{-T}$ and $[0\ M^{-1}; M^{-T}\ 0]$ as the preconditioning operators for PRIMME for the different stages. Moreover, if a preconditioner such as RIF is given $M \approx C^{-1}$, we can build preconditioners for the second stage as $[0\ AM; MA^T\ 0]$. It is not clear in general how to form a preconditioner for C from a preconditioner of B.

4.1. A dynamic two stage method with preconditioning. The analysis in Sections 2.3 and 2.4 holds for Krylov methods but it is less meaningful with preconditioners. Clearly, if two different preconditioners are provided for C and B their relative strengths are not known by primme_svds. But we have also noticed cases where the first stage benefits less than the second stage when a less powerful preconditioner M for A is used to form preconditioners for both C and B. If M is ill-conditioned but its near-kernel space does not correspond well to that of A, it may work for B, but taking M^TM produces an unstable preconditioner for C [42]. On the other hand, with a sufficiently good preconditioner, both methods enjoy similar benefits on convergence. If the relative strengths of the provided preconditioner are known, users can choose the two-stage approach or only one of the stages (e.g., the second one). For the general case, we present a method that, based on runtime measurements, switches dynamically between the normal equations and the augmented approach to identify the most effective one for the given preconditioning. This is shown in Algorithm 4.

To estimate the convergence of the two approaches, we run a set of initial tests alternating between running on C and on B. Because JDQMR relies on good initial guesses which are not available initially, the dynamic algorithm uses only the GD+k method. Once the algorithm decides on the approach, other PRIMME methods can be used as specified by the user. Without loss of generality, we only consider GD+k for our dynamic primme_svds experiments. The approximations obtained from one run are passed as initial guesses to the next run.

We estimate the convergence rate by measuring the average reduction per iteration of the residual norm. To be able to capture the convergence at different phases of the iterative method, we must switch between the two approaches several times. However, switching too frequently incurs a lot of overhead (rebuilding the initial basis, performing extra Rayleigh-Ritz procedures, and possibly convergence loss from restarting the search space). Switching too infrequently may be wasteful when the preconditioner for C does not work well. Thus, we control the maximum number of iterations for the next GD+k run, $maxIter$. This number is always larger than $initIter$ which is a reasonably small number, i.e., 50. If the same approach is chosen in two successive runs, $maxIter$ doubles. If the approach should be switched, $maxIter$ is reduced more aggressively for the next run (Step 12) to avoid wasting too much time on the wrong approach. On the other hand, if one eigenvalue converges in the initial tests or at least two eigenvalues converge later, we stop the dynamic switching and choose the currently faster approach. If the faster approach is on the normal equations, a two-stage method might be necessary to get to full accuracy. Although two or three switches typically suffice to distinguish between approaches, we also limit the number of switches.

5. Numerical experiments. Our first two experiments use diagonal matrices to demonstrate the principle of the two stage method and that the method can compute artificially clustered tiny singular values to full accuracy. Then, we conduct an extensive set of experiments for finding the smallest singular values of several matrices. Large singular values are also computed under the shift-invert setting. The matrix set is chosen to overlap with those in other papers in the literature. We compare against several state-of-the-art SVD methods: JDSVD [9, 10], SVDIFP [27], IRRHLB [18], IRLBA [14], and MATLAB's `svds`. First, we compute a few of the smallest singular triplets on both square and rectangular matrices without a preconditioner. Then we show a performance comparison between the two stage primme_svds method and its dynamic version with different quality of preconditioners. Subsequently, we demon-

Algorithm 4 Dynamic switching between stages for preconditioned primme_svds

Set $initIter$, $maxSwitch$
$numSwitch = numConverged = j = 0, maxIter = initIter, Undecided = true$
Run $initIter$ iterations of GD+k on C and on B and collect initial average convergence rate of both approaches.
while ($numSwitch < maxSwitch$ and $Undecided$) **do**
  Choose estimated faster approach (C or B) for next call
  if ($numSwitch == 0$ **and** $numConverged > 0$) **or** $numConverged > 1$ **then**
    undecided = false (Choose faster approach and no more switching)
  else
    if Same approach is chosen again **then**
      $j = j + 1;\quad maxIter = initIter * 2^j$
    else if Different approach is chosen **then**
      $j = \text{floor}(j/2);\quad maxIter = initIter * 2^j$
    end if
  end if
  $numSwitch = numSwitch + 1$
  Call PRIMME with $maxIter$ and current chosen approach
end while
if All desired singular triplets are found on B **then**
  Return final singular triplets to users
else if All desired singular triplets are found on C **then**
  Return resulting singular triplets to augmented approach
else if Faster approach is on C **then**
  Proceed with the two-stage approach
else if Faster approach is on B **then**
  Continue only with the augmented approach
end if

strate that primme_svds provides faster convergence with a good preconditioner compared with JDSVD and SVDIFP. We also show that primme_svds achieves better performance than `svds` and SVDIFP using shift-and-invert. Finally, we present some numerical results on a real-world problem.

All computations are carried out on a DELL dual socket with Intel Xeon processors at 2.93GHz for a total of 16 cores and 50 GB of memory running the SUSE Linux operating system. We use MATLAB 2013a with machine precision $\epsilon = 2.2 \times 10^{-16}$ and PRIMME is linked to the BLAS and LAPACK libraries available in MATLAB. Our stopping criterion is for the left and right residuals to satisfy,

$$\sqrt{\|r_u\|^2 + \|r_v\|^2} < \|A\|\delta_{user}. \tag{5.1}$$

For JDSVD we use the refined projection method as it performed best in our experiments, which is also consistent with [9]. We choose the default for all parameters except for setting 'krylov = 0' to avoid occasional convergence problems for smallest singular values. SVDIFP is a recent method and its MATLAB implementation is still under development. Its maximum number of inner iterations can be chosen as fixed or adaptive. We run with both choices and report the best result. Also, we use singular triplet residuals as the stopping criterion for SVDIFP instead of the default residual of the normal equations. For IRLBA and IRRHLB, we choose all default parameters as suggested in the code.

All methods start with the same initial guess, `ones(min(m,n),1)`, except for matrix lshp3025 for which a random guess is necessary. We set the maximum number of restarts to 5000 for IRRHLB and IRLBA and to 10000 for JDSVD and primme_svds. Since SVDIFP can only set a maximum number of iterations for each targeted singular triplet, we report that SVDIFP cannot converge to all desired singular values if its overall number of matrix-vector operations is larger than $(maxBasisSize - k) * 5000$. For primme_svds, we set maxBasisSize=35, minRestartSize=21 and experiment with two δ tolerances, 1e-8 and 1e-14. For δ =1e-8, primme_svds does not need to enter the second stage for any of our tests. Since the numbers of matrix-vector products with A and A^T are the same, the tables report as "MV" the number of products with A only. "Sec" is the run time in seconds, and "–" means the method cannot converge to all desired singular values or that the code breaks down. For the first stage of primme_svds we use the GD_Olsen_PlusK method. For the second stage, we run experiments with both GD_PlusK and JDQMR. Since our implementation is mainly in C, we focus on comparing the number of matrix-vector operations as the primary measurement of the performance. However, we also report execution times which is relevant since matrix-vector, preconditioner, and all BLAS/LAPACK operations are performed by the MATLAB libraries.

5.1. Primme_svds for clustered tiny singular values. We illustrate first how our two-stage method works in a seamless manner. We consider a diagonal matrix A = diag([1:10,1000:100:1e6]) and the preconditioner M = A + rand(1,10000)*1e4. In Figure 5.1, the green and black lines show the convergence behaviors of PRIMME on B and C respectively. Indeed, the convergence on B is very slow due to a highly indefinite problem while the accuracy on C is limited and stagnates when reaching its limit. The two stage primme_svds combines the benefits of the two methods, and determines the smallest singular efficiently and accurately as the magenta line shows.

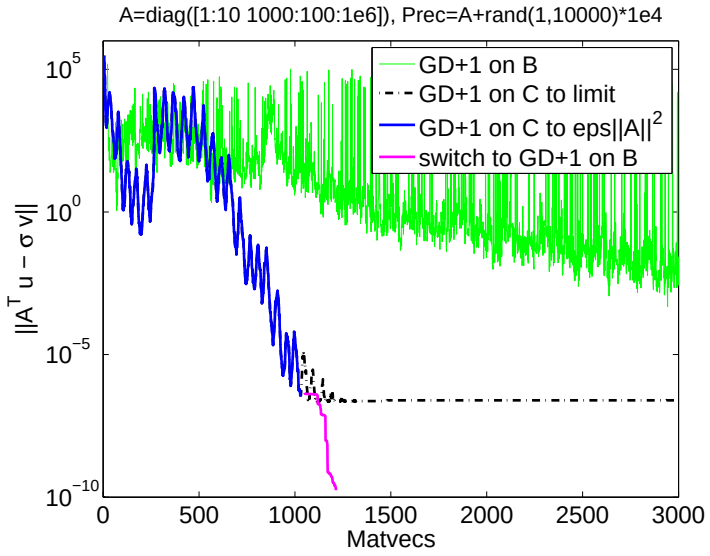


FIG. 5.1. *An example to show how two stage of primme_svds works seamlessly for seeking smallest singular values accurately*

Next we show how primme_svds can determine several clustered tiny singular

values. Consider the matrix A = diag([1e-14 1e-12 1e-8:1e-8:4e-8, 1e-3:1e-3:1]) with identity matrix as a preconditioner. Although locking may be able to determine clustered or multiple singular values, we increase robustness by using a block size of two in PRIMME, for the first stage only. We set the user tolerance $\delta_{user} = 1e-15$ to examine the ultimate accuracy of primme_svds. As shown in Table 5.1, primme_svds is capable of computing all the desired clustered, tiny singular values accurately.

TABLE 5.1
Computation of 10 clustered smallest singular values by primme_svds with block size 2

Singular values	true value	primme_svds	residuals
σ_1	1e-14	9.952750930151887E-15	4.9E-16
σ_2	1e-12	1.000014102726476E-12	8.9E-16
σ_3	1e-8	1.000000003582006E-08	6.5E-16
σ_4	2e-8	2.000000002073312E-08	9.2E-16
σ_5	3e-8	3.000000000893043E-08	9.0E-16
σ_6	4e-8	4.000000001929591E-08	9.2E-16
σ_7	1e-3	1.000000000000025E-03	9.7E-16
σ_8	2e-3	1.999999999999956E-03	8.1E-16
σ_9	3e-3	2.999999999999997E-03	9.1E-16
σ_{10}	4e-3	3.999999999999958E-03	9.8E-16

5.2. Without preconditioning. We compare two variants of primme_svds with four methods, JDSVD, SVDIFP, IRRHLB, and IRLBA on both square and rectangular matrices without preconditioning. Since a good preconditioner is usually not easy to obtain for SVD problems, it is important to examine the effectiveness of a method in this case. We compute the $k = 1, 3, 5, 10$ smallest singular triplets. In order to speed up convergence, $k+3$ eigenvalues are computed when k desired eigenvalues are required in `svds`. A similar strategy is applied to IRRHLB and IRLBA. For primme_svds, we found this is not necessary.

We select six square and six rectangular matrices from other research papers [18, 27] and the University of Florida Sparse Matrix Collections [45]. Table 5.2 lists these matrices along with some of their basic properties. Among them, the matrices pde2961, dw2048, well1850 and lp_ganges have relative larger gap ratios and smaller condition number, and thereby are easy ones. Matrices fidap4, jagmesh8, wang3, deter4, and plddb are hard cases, and matrices lshp3025, ch, and lp_bnl2 are very hard ones. We expect all methods to perform well for solving easy problems. Harder problems tend to magnify the difference between methods.

Tables 5.3, 5.4, 5.5 and 5.6 show that primme_svds variants converge faster and more robustly than all other methods on both square and rectangular matrices. Specifically, table 5.3 shows that for moderate accuracy the normal equations solved with a PRIMME method are significantly faster. For instance, primme_svds is generally at least two or three times faster than other methods when solving hard problems for any number of smallest singular values. When solving easy problems, primme_svds is still several times faster than JDSVD, SVDIFP, and IRLBA. IRRHLB can be comparative with our method only when seeking 10 singular values on easy cases. This is typical behavior of the Lanczos method when looking for a large number eigenvalues [36]. The superiority of primme_svds is achieved as a result of the near-optimal properties of the eigenmethods in PRIMME [35]. We have noticed that even for moderate accuracy, JDSVD, SVDIFP, IRRHLB and IRLBA are all challenged by hard problems, where they are often inefficient or even fail to converge to all desired singular values.

TABLE 5.2

Properties of the test square and rectangular matrices, where $\gamma_m(k) = \min_{i=1}^{k}(gap(\sigma_i))$, *and* $gap(\sigma_i) = min_{j\neq i}|\sigma_i - \sigma_j|$

Matrix	pde2961	dw2048	fidap4	jagmesh8	wang3	lshp3025
order	2961	2048	1601	1141	26064	3025
nnz(A)	14585	10114	31837	7465	77168	120833
$\kappa(A)$	9.5e2	5.3e3	5.2e3	5.9e4	1.1e4	2.2e5
$\|A\|_2$	1.0e1	1.0e0	1.6e0	6.8e0	2.7e-1	7.0e0
$\gamma_m(1)$	8.2e-3	2.6e-3	1.5e-3	1.7e-3	7.4e-5	1.8e-3
$\gamma_m(3)$	2.4e-3	2.9e-4	2.5e-4	1.6e-3	1.9e-5	9.1e-4
$\gamma_m(5)$	2.4e-3	2.9e-4	2.5e-4	4.8e-5	1.9e-5	1.8e-4
$\gamma_m(10)$	7.0e-4	1.6e-4	2.5e-4	4.8e-5	6.6e-6	2.2e-5

Matrix	well1850	lp_ganges	deter4	plddb	ch	lp_bnl2
rows m:	1850	1309	3235	3049	3700	2324
cols n:	712	1706	9133	5069	8291	4486
nnz(A)	8755	6937	19231	10839	24102	14996
$\kappa(A)$	1.1e2	2.1e4	3.7e2	1.2e4	2.8e3	7.8e3
$\|A\|_2$	1.8e0	4.0	1.0e1	1.4e2	7.6e2	2.1e2
$\gamma_m(1)$	3.0e-3	1.1e-1	1.1e-1	4.2e-3	1.6e-3	7.1e-3
$\gamma_m(3)$	3.0e-3	4.5e-2	3.1e-4	5.1e-5	7.7e-4	4.8e-3
$\gamma_m(5)$	3.0e-3	2.4e-3	8.9e-5	5.1e-5	3.6e-4	1.1e-3
$\gamma_m(10)$	2.6e-3	8.0e-5	8.9e-5	2.0e-5	4.0e-5	1.1e-3

Table 5.4 shows smaller differences between methods, reflecting the slower convergence of the augmented method in stage two. We see that the two variants of primme_svds have comparable number of matrix-vector operations, but the JDQMR one typically requires less time if the matrix is sparse enough [35]. For computing 10 eigenpairs, IRRHLB shows a small edge in the number of iterations for two easy cases. However, primme_svds method never misses eigenvalues, is consistently much faster than all other methods, and significantly faster than IRRHLB in the hard cases. Note the slow convergence of JDSVD, since it relies on the augmented matrix to produce all the required accuracy. SVDIFP is also not competitive, partly due to its inefficient restarting strategy. Clearly, demanding higher accuracy for the hard problems does not help the rest of the methods. Note that because of PRIMME's high quality implementation, not only does primme_svds enjoy better robustness but it is also ten times faster than IRRHLB for the cases where IRRHLB takes fewer MVs.

Tables 5.5 and 5.6 show that the advantage of primme_svds is even more significant on rectangular matrices. For example, except for the two easy problems well1850 and lp_ganges, primme_svds is often five or ten times faster than the other methods. The reason is two-fold. First, primme_svds works on C with dimension $\min(m, n)$, which saves memory and computational costs. SVDIFP also shares this advantage. Second, PRIMME's advanced restarting techniques exploits the convergence optimality of exterior eigenvalues of the matrix C. Interestingly, JDQMR converges much faster than GD+k on some hard problems such as plddb, ch and lp_bnl2 in table 5.6. The reason is the availability of excellent shifts from the first stage. We conclude that primme_svds is the fastest method and sometimes the only method that converges for hard problems without preconditioning.

5.3. With preconditioning. What is remarkable from the previous Tables 5.3 to 5.6 is the difficulty of solving for the smallest singular values, even for small matri-

ces. Preconditioning is a prerequisite for addressing larger, practical problems, which limits our choice to primme_svds, JDSVD and SVDIFP.

We first compare our two stage method and our dynamic two-stage method for two different quality preconditioners. We choose $M = LU$, the factorization obtained from MATLAB's ILU function on a square matrix A with parameters `'type=ilutp'`, `'thresh=1.0'`, and varying `'droptol=1e-2'` or `'droptol=1e-3'`. Given these two M, we form the preconditioners for primme_svds as $M^{-1}M^{-T}$ and $[0\ M^{-1}; M^{-T}\ 0]$. Without loss of generality, primme_svds chooses GD+k eigenmethod for the underlying eigensolver PRIMME. We seek ten smallest singular values with tolerance 1e-14.

As shown in Table 5.7, both variants of primme_svds can solve the problems effectively with a good preconditioner (`'droptol=1e-3'`). In this case, the static two stage method is always better than the dynamic one because of the overhead incurred by switching between the two methods. On the other hand, when using the preconditioner with `'droptol=1e-2'`, the two stage primme_svds is slower than the dynamic in some cases, and in the case of lshp3025, much slower. The reason is the inefficiency of the preconditioner in the normal equations. Our dynamic primme_svds can detect the convergence rate difference and choose the faster method to accomplish the remaining computations. Of course, if this issue is known beforehand, users can bypass the dynamic heuristic and call directly the second stage.

Next, we compare the two stage primme_svds with JDSVD and SVDIFP when a good quality of preconditioner is available. Except for preconditioning, all other parameters remain as before. For the first preconditioner we use MATLAB's ILU on a square matrix A. For the second preconditioner we use the RIF MEX function provided in [27] on a rectangular matrix with `'droptol=1e-3'`. The resulting RIF factors $LDL^T \approx A^TA$, where D is diagonal matrix with 0 and 1 elements, are used to construct the pseudoinverses $M^{-1} = L^{-T}L^{-1}A^T$ and $M^{-T} = AL^{-T}L^{-1}$ for preconditioning the second stage of primme_svds and JDSVD. In the SVDIFP code, there is an input parameter 'COLAMD' which computes approximate column minimum degree permutation to obtain sparser LU factors. This technique can be applied to other methods but some modifications are needed. Therefore, we disable this parameter in the SVDIFP code. For the JDSVD code, we try both enabling and disabling the initial Krylov subspace and report the best result.

Tables 5.8 and 5.9 show that a good preconditioner makes the problems tractable, with all three methods solving the problems effectively. Still, in most cases primme_svds provides much faster convergence and execution time on both square and rectangular matrices. We see that when seeking one smallest singular value with high accuracy, JDSVD takes less iterations for one square matrix (wang3), and SVDIFP is competitive in two rectangular cases (plddb and lp_bnl2). This is because these cases require very few iterations, and the first stage of primme_svds forces a Rayleigh-Ritz with 21 extra matrix-vector operations. This robust step is not necessary for this quality of preconditioning. If we are allowed to tune some of its parameters (as we did with JDSVD and SVDIFP) primme_svds does require fewer iterations even in these cases.

5.4. With the shift and invert technique. Shift and invert is also applicable to PRIMME and primme_svds. Moreover, because it turns the eigenvalues closest to the shift to a largest eigenvalue problem, there is no need for the second stage. SVDIFP can utilize shift and invert operator as a preconditioner. Therefore, we report results on primme_svds, SVDIFP, and `svds`. `svds`, uses shift and invert operator on the augmented matrix B. For primme_svds and SVDIFP we use two different factorizations, an LU and a QR factorization of the matrix A. Since `svds` uses a

basis size of 40 for seeking 10 smallest singular values, we give the same basis size to primme_svds and SVDIFP. Also, we disable the 'COLAMD' option in SVDIFP. For SVDIFP the shifts are all zeros, for `svds` we give a shift of 1e-8, and there is no need of shifts for primme_svds due to an extreme eigenvalue problem. We have instrumented the MATLAB native `svds` code to return the number of iterations. To facilitate comparisons, we include the LU and QR factorization times in the running times of all methods, but also report them separately. The tolerance is $\delta =$ 1e-10, and primme_svds uses the DYNAMIC method that switches between GD+k and JDQMR to optimize performance.

Table 5.10 shows that primme_svds is faster than `svds` both in convergence and execution time, partly because it works on C which is smaller in size and allows for faster convergence. primme_svds is much faster than SVDIFP due to its more efficient restarting strategy, and also because an inverted operator may not be as effective when used as a preconditioner. Note that `svds` does not work well on rectangular matrices because B becomes singular and cannot be inverted, and if instead a small shift is used, it finds the zero eigenvalues of B first.

5.5. With real-world problems. We use primme_svds, SVDIFP, and JDSVD to compute the smallest singular triplet of matrices of order larger than 1 million. Some information on these matrices appears in Table 5.11. We apply the two-stage primme_svds(JDQMR) on all test matrices except thermal2, which is solved by dynamic preconditioning primme_svds. The preconditioners are applied similar to our previous experiments with the exception that ILU uses `'thresh=0.1'`, and `'udiag=1'`. The tolerance is $\delta = 1e-12$. The symbol "*" means the method returns results that either did not satisfy the desired accuracy or did not converge to the smallest singular triplet.

Table 5.12 shows the results without or with various preconditioners. Debr is a numerically singular square matrix. primme_svds is capable of resolving this more efficiently than JDSVD, while SVDIFP returns early when it detects that it is not likely to converge to the desired accuracy for left singular vector [27]. All methods easily solve problem cage14 with ILU(0), but primme_svds is much faster. Thermal2 is an ill-conditioned matrix, whose preconditioner turns out to be less effective for C than for B. Therefore, SVDIFP has much slower convergence than JDSVD. Thanks to the dynamic scheme, primme_svds recognizes this deficiency and converges without too many additional iterations, and with the same execution time as JDSVD. However, if we had prior knowledge about the preconditioner's performance, running only at the second stage gives almost exactly the same matrix-vectors as JDSVD and much lower time. Reducing further the overhead of the dynamic heuristic is part of our current research. JDSVD often fails to converge to the smallest singular value for rectangular matrices since it has difficulty to distinguish them from zero eigenvalues of B, as shown in the cases sls and Rucci1. For matrix sls, SVDIFP misconverges to the wrong singular triplet while primme_svds is successful in finding the correct one. SVDIFP and primme_svds have similar performance for solving problem Rucci1. In summary, primme_svds is far more robust and more efficient than either of the other two methods for large problems.

6. Conclusion. Based on the state-of-the-art eigensolver PRIMME, we have developed a full functionality, high quality SVD solver for a few smallest or largest singular values of a large matrix. The key is a two stage meta-method, primme_svds, that in the first stage solves the normal equations as a fast way to get sufficiently accurate approximations, and if further accuracy is needed, solves an interior eigen-

value problem from the augmented matrix. In addition, we have presented several enhancements to the PRIMME eigensolver that allow for an efficient computation of the interior eigenproblem. We have motivated the merit of this approach theoretically, and confirmed its performance through an extensive set of experiments. primme_svds improves on convergence and robustness over other state-of-the-art singular value methods, but most importantly it is based on a highly optimized production software that allows its use, with or without preconditioning, in large real world problems. Currently, primme_svds is available in MATLAB through the MEX interface, and we are planning to release it in the near future. A native C implementation of primme_svds as part of PRIMME is planned next.

REFERENCES

[1] G. H. Golub, and C. F. Van Loan, *Matrix Computations*, 3rd ed., The John Hopkins University Press, Baltimore, London, 1996.

[2] A. Bjorck, *Numerical Method for Least Squares Problems*, SIAM, Philadelphia, PA, 1996.

[3] L. N. Trefethen and David Bau III, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[4] L. N. Trefethen, *Computation of Pseudospectrum*, SIAM, Philadelphia, PA, 2001.

[5] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA, 1998.

[6] G. Golub, and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Indust. Appl. Math. Ser.B Numer. Anal., 2 (1965), pp. 205-224.

[7] G. Golub, F. T. Luk and M. L. Overton, *A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix*, ACM Trans. Math. Software, 7 (1981), pp. 149-169.

[8] J. Cullum, R. A. Willoughby, and M. Lake, *A Lanczos algorithm for computing singular values and vectors of large matrices*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 197-215.

[9] M. E. Hochstenbach, *A Jacobi-Davidson type SVD method*, SIAM J. Sci. Comput., 23 (2001), pp. 606-628.

[10] M. E. Hochstenbach, *Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems*, BIT, 44 (2004), pp. 721-754.

[11] B. Philippe and M. Sadkane, *Computation of the fundamental singular subspace of a large matrix*, Linear Algebra Appl., 257 (1997), pp. 77-104.

[12] J. J. Dongarra, *Improving the accuracy of computed singular values*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 712-719.

[13] Michael W. Berry, Dani Mezher, B. Philippe and Ahmed Sameh, Parallel algorithms for the singular value decomposition (Chapter 4), in Erricos John Kontoghiorghes, editor, *Handbook on parallel computing and statistics*, pages 117-164, Chapman and Hall/CRC, 2005.

[14] J. Baglama and L. Reichel, *Augmented implicitly restarted Lanczos bidiagonalization methods*, SIAM J. Sci. Comput., 27 (2005), pp. 19-42.

[15] J. Baglama and L. Reichel, *Restarted block Lanczos bidiagonalization methods*, Numer. Algorithms, 43 (2006), pp. 251-272.

[16] J. Baglama and L. Reichel, *An implicitly restarted block Lanczos bidiagonalization method using leja shifts*, BIT, 53 (2013), pp. 285-310.

[17] Z. Jia and D. Niu, *An implicitly restarted refined Lanczos bidiagonalization method for computing a partial singular value decomposition*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 246-265.

[18] Z. Jia and D. Niu, *An refined harmonic Lanczos bidiagonalization method and implicitly restarted algorithm for computing the smallest singular triplets of large matrices*, SIAM J. Sci. Comput., 32 (2010), pp. 714-744.

[19] E. Kokiopoulou, C. Bekas, and E. Gallopoulos, *computing the smallest singular triplets with implicitly restarted Lanczos bidiagonalization*, Appl. Numer. Math., 49 (2004), pp. 39-61.

[20] M. Berry, *Large Scale Singular Value Computations*, International Journal of Supercomputer Applications 6:1, (1992), pp. 13-49.

[21] R. M. Larsen, *Lanczos bidiagonalization with partial reorthogonalization*, technical report, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 1998. Available online from http://soi.stanford.edu/ rmunk/PROPACK/

[22] R. M. Larsen, *Combining implicit restarts and partial reorthogonalization in Lanczos bidiagonalization*, available online from http://soi.stanford.edu/ rmunk/PROPACK/

[23] V. Hernandez, J. E. Roman, and V. Vidal. *SLEPc: A scalable and flexible toolkit for the*

solution of eigenvalue problems. ACM Trans. Math. Software, 31(3):351-362, 2005.
[24] V. Hernndez, J. E. Romn and A. Toms. *A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization*. Electron. Trans. Numer. Anal., 31:68-85, 2008.
[25] D. Niu and X. Yuan, *A harmonic Lanczos bidiagonalization method for computing interior singular triplets of large matrices*, Appl. Math. Comput., 218 (2012), pp. 7459-7467.
[26] Gene H. Golub and Qiang Ye, *An Inverse Free Preconditioned Krylov Subspace Method for Symmetric Generalized Eigenvalue Problems*, SIAM J. Sci. Comput., 24 (2002), pp. 312-334.
[27] Qiao Liang and Qiang Ye, *Computing Singular Values of Large Matrices with Inverse Free Preconditioned Krylov Subspace Method*, submitted.
[28] M. Benzi and M. Tuma, *A Robust Preconditioner with Low Memory Requirements for Large Sparse Least Squares Problems*, SIAM J. Sci. Comput., 25 (2003), pp. 499-512.
[29] R.B. Morgan, *Computing interior eigenvalues for large matrices*, Linear Algebra Appl. 154/156 (1991), pp. 289-309.
[30] R.B. Morgan, M. Zeng, *Harmonic projection methods for large non-symmetric eigenvalue problems*, Numer. Linear Algebra Appl.5(1) (1998), pp. 33-55.
[31] G.L.G. Sleijpen, H.A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl, 17(2) (1996), pp. 401-425.
[32] C.C.Paige, B.N.Parlett, H.A. van der Vorst, *Approximate solutions and eigenvalue bounds from Krylov subspaces*, Numer. Linear Algebra Appl. 2 (1995), pp. 115-134.
[33] Z.Jia, *Refined iterative algorithms based on Arnoldi's process for large unsymmetric eigenproblems*, Linear Algebra Appl, 259 (1997), pp. 1-23.
[34] G.W.Stewart, *Matrix Algorithms; Vol. II Eigensystems*, SIAM, Philadelphia, PA, 2001.
[35] A. Stathopoulos, *Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue*, SIAM J. Sci. Comput., 29(2) (2007), pp. 481-514.
[36] A. Stathopoulos and J. R. McCombs, *Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking Many eigenvalue*, SIAM J. Sci. Comput., 29(5) (2007), pp. 2162-2188.
[37] A. Stathopoulos and James R. McCombs, *PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description*, ACM Trans. Math. Software, 37 (2010), pp. 21:1-21.30.
[38] A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic Thick Restarting of the Davidson, and the Implicitly Restarted Arnoldi Methods*, SIAM J. Sci. Comput., 19, 1, (1998) 227-45.
[39] K. Wu and H. Simon, *Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems*, SIAM J. Matrix Analysis and Applications, 22, 2, (2000) 602-616.
[40] J. Shen, G. Strang, and A. J. Wathen, *The Potential Theory of Several Intervals and Its Applications*, Applied Mathematics and Optimization, 44(1) (2001), pp. 67-85.
[41] L. Wu, and A. Stathopoulos, *Enhancing the PRIMME Eigensolver for Computing Accurately Singular Triplets of Large Matrices*, Tech Report: WM-CS-2014-03, March, Department of Computer Science, College of William and Mary, 2014.
[42] Y. Saad, and M. Sosonkina, *Enhanced Preconditioners for Large Sparse Least Squares Problems*, Tech Report: umsi-2001-1, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.
[43] M. Arioli and J. Scott, *Chebyshev acceleration of iterative refinement*, Numerical Algorithm, 1017-1398, 2013, 1-18.
[44] H.A. van der Vorst, *Computational Methods for Large Eigenvalue Problems*, In P.G. Ciarlet and J.L. Lions(eds), Handbook of Numerical Analysis, Volume VIII, North-Holand, 2002, pp. 3-179.
[45] T. A. Davis, and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1-1:25.
[46] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.

TABLE 5.3

Seeking 1, 3, 5, and 10 smallest singular triplets of square matrices with user tolerance 1e-8. The primm_svds method has two variants: primme_svds(GD+k) uses GD+k eigenmethod while primme_svds(JDQMR) uses JDQMR eigenmethod respectively in the second stage

	δ = 1e-8 Matrix:	pde2961		dw2048		fidap4	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(1st stage only)	2165	1.8	1766	1.2	4750	3.2
1	JDSVD	4269	4.8	3840	3.1	4379	3.8
1	SVDIFP	3433	7.9	2520	2.7	11572	13.3
1	IRRHLB	3755	44.6	3228	35.1	8839	97.9
1	IRLBA	12292	8.7	7684	3.3	47332	20.3
3	primme_svds(1st stage only)	2643	2.2	2135	1.5	5661	3.8
3	JDSVD	7195	8.3	5776	4.8	14334	12.7
3	SVDIFP	11466	29.6	12123	14.7	26285	28.7
3	IRRHLB	3718	44.6	3225	36.1	14303	156.2
3	IRLBA	14676	9.9	7656	4.1	69456	38.5
5	primme_svds(1st stage only)	3118	2.6	2431	1.7	6890	5.4
5	JDSVD	9076	12.2	7514	8.1	16270	16.3
5	SVDIFP	17272	46.4	17564	21.8	45560	51.8
5	IRRHLB	4301	53.8	2978	34.6	13184	152.5
5	IRLBA	10759	7.7	6083	3.4	60207	34.3
10	primme_svds(1st stage only)	4894	4.2	3912	2.8	10087	6.8
10	JDSVD	14906	20.8	11683	10.4	20934	19.7
10	SVDIFP	39143	110.6	28996	36.6	97653	115.7
10	IRRHLB	4809	63.5	3445	43.8	12025	147.6
10	IRLBA	8786	6.8	5482	3.3	38946	18.4

	δ = 1e-8 Matrix:	jagmesh8		wang3		lshp3025	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(1st stage only)	5472	2.9	6579	40.1	11312	10.4
1	JDSVD	12343	8.2	11353	83.8	37225	40.9
1	SVDIFP	20223	16.3	14063	228.1	27413	73.0
1	IRRHLB	30105	317.7	19689	632.1	46845	565.8
1	IRLBA	–	–	–	–	–	–
3	primme_svds(1st stage only)	5915	3.2	7589	58.3	12250	10.9
3	JDSVD	13861	8.7	17865	127.3	50282	57.5
3	SVDIFP	146224	123	48696	1090	153524	518.0
3	IRRHLB	25497	271	19001	619.5	42201	518.4
3	IRLBA	139266	48.6	129876	577.2	–	–
5	primme_svds(1st stage only)	6679	3.7	8673	68.7	14126	12.5
5	JDSVD	18173	12.8	22441	157.3	66034	80.9
5	SVDIFP	–	–	68774	1671	146336	569.2
5	IRRHLB	18314	197.8	17963	570	93239	1186.6
5	IRLBA	85043	29.9	82551	359.1	–	–
10	primme_svds(1st stage only)	8861	4.8	19445	154.2	19755	18.7
10	JDSVD	21209	14.7	–	–	86780	115.6
10	SVDIFP	–	–	–	–	–	–
10	IRRHLB	52043	608.1	16975	537	–	–
10	IRLBA	43798	15.1	52490	250.7	–	–

TABLE 5.4

Seeking 1, 3, 5, and 10 smallest singular triplets of square matrices with user tolerance 1e-14. The primm_svds method has two variants: primme_svds(GD+k) uses GD+k eigenmethod while primme_svds(JDQMR) uses JDQMR eigenmethod respectively in the second stage.

	δ = 1e-14 Matrix:	pde2961		dw2048		fidap4	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(GD+k)	2986	2.7	2421	1.9	6018	5.4
1	primme_svds(JDQMR)	2925	3.0	2374	1.6	5942	3.9
1	JDSVD	6106	7.6	5061	4.5	6436	5.9
1	SVDIFP	5992	14.0	5241	5.6	21325	24.8
1	IRRHLB	6328	75.5	4561	50.9	14078	155.4
1	IRLBA	21700	15.3	7684	3.3	96868	51.4
3	primme_svds(GD+k)	4243	6.1	3679	3.7	8846	8.8
3	primme_svds(JDQMR)	4226	3.7	3602	2.5	8775	5.8
3	JDSVD	10517	11.6	8911	7.1	10781	9.0
3	SVDIFP	20613	49.6	23290	27.6	48603	56.3
3	IRRHLB	6241	77.0	4443	50.0	19059	215.4
3	IRLBA	24096	21.4	7684	3.3	111756	60.8
5	primme_svds(GD+k)	5579	7.1	4776	3.7	11976	9.1
5	primme_svds(JDQMR)	5569	5.0	4753	3.3	11910	7.8
5	JDSVD	14554	19.1	12266	11.7	14906	14.6
5	SVDIFP	31506	77.0	33117	39.9	83622	94.7
5	IRRHLB	6218	79.2	4193	49.1	18098	211.2
5	IRLBA	16629	13.7	9763	5.5	84847	47.2
10	primme_svds(GD+k)	9337	12.1	8069	7.9	19805	16.2
10	primme_svds(JDQMR)	9333	8.6	8014	6.0	19433	13.2
10	JDSVD	24498	29.6	20351	20.5	25125	24.5
10	SVDIFP	73847	192.5	53315	63.4	–	–
10	IRRHLB	6371	86.8	4589	58.6	17393	214.2
10	IRLBA	12497	12	7796	4.8	56026	34.7

	δ = 1e-14 Matrix:	jagmesh8		wang3		lshp3025	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(GD+k)	7080	5.0	9160	74.8	15674	19.3
1	primme_svds(JDQMR)	7043	3.6	8957	65.5	15922	13.9
1	JDSVD	13608	9.6	16457	105.4	42835	53.1
1	SVDIFP	–	–	36675	916.6	–	–
1	IRRHLB	43869	466.5	27470	1003	57912	693
1	IRLBA	–	–	–	–	–	–
3	primme_svds(GD+k)	8859	5.7	14184	179.7	18910	20.8
3	primme_svds(JDQMR)	8836	4.4	13360	91.1	18788	15.2
3	JDSVD	17029	11.0	41900	387	48731	59.4
3	SVDIFP	–	–	122469	3080	–	–
3	IRRHLB	31210	330.1	29035	1094	63806	799.5
3	IRLBA	–	–	–	–	–	–
5	primme_svds(GD+k)	11569	7.5	21396	302.7	24743	27.5
5	primme_svds(JDQMR)	11344	5.9	18668	119.6	24398	20.8
5	JDSVD	22573	15.2	57454	441.4	62646	75.9
5	SVDIFP	–	–	–	–	–	–
5	IRRHLB	23498	331.4	22985	730.1	99395	1258.9
5	IRLBA	124411	53.6	–	–	–	–
10	primme_svds(GD+k)	17344	10.8	37898	480.7	39852	42.4
10	primme_svds(JDQMR)	17233	8.8	33995	245.1	39591	32.7
10	JDSVD	29613	21.0	91290	871.9	–	–
10	SVDIFP	–	–	–	–	–	–
10	IRRHLB	55673	827.4	43309	1679.4	–	–
10	IRLBA	59595	27.4	–	–	–	–

TABLE 5.5

Seeking 1, 3, 5, and 10 smallest singular triplets of rectangular matrices with user tolerance 1e-8. The primm_svds method has two variants: primme_svds(GD+k) uses GD+k eigenmethod while primme_svds(JDQMR) uses JDQMR eigenmethod respectively in the second stage.

	δ = 1e-8 Matrix:	well1850		lp_ganges		deter4	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(1st stage only)	519	0.4	233	0.2	235	0.2
1	JDSVD	1563	3.3	771	0.7	760	1.5
1	SVDIFP	1352	1.4	403	0.5	330	1.4
1	IRRHLB	872	8.7	345	3.4	500	6.6
1	IRLBA	1060	0.5	260	0.2	292	0.4
3	primme_svds(1st stage only)	584	0.4	242	0.3	1943	2.0
3	JDSVD	2773	2.4	7052	7.1	–	–
3	SVDIFP	2779	2.7	990	1.1	6794	29.2
3	IRRHLB	847	8.5	325	3.3	11896	170.1
3	IRLBA	816	0.4	246	0.2	17616	17.7
5	primme_svds(1st stage only)	668	0.5	536	0.7	2853	3.1
5	JDSVD	4203	3.5	15533	16.1	–	–
5	SVDIFP	4169	4.1	1979	2.2	21725	99.1
5	IRRHLB	872	9.3	926	10.3	12644	187
5	IRLBA	931	0.5	802	0.5	25515	26.3
10	primme_svds(1st stage only)	956	0.7	1036	0.9	4240	4.3
10	JDSVD	85053	73.3	–	–	–	–
10	SVDIFP	7279	7.2	9213	10.1	28303	132.3
10	IRRHLB	827	10.0	2851	34.7	–	–
10	IRLBA	788	0.4	890	0.5	42038	44.3

	δ = 1e-8 Matrix:	plddb		ch		lp_bnl2	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(1st stage only)	2513	2.4	10408	19.3	16581	12.2
1	JDSVD	11100	16.2	74592	139.1	69576	94.2
1	SVDIFP	6097	18.2	15843	60.9	18653	34.9
1	IRRHLB	23161	290.4	–	–	–	–
1	IRLBA	31684	21.1	21700	22.4	–	–
3	primme_svds(1st stage only)	2457	3.3	21071	31.8	20023	15.0
3	JDSVD	13531	20.0	–	–	101696	135.9
3	SVDIFP	19934	60.7	31431	127.4	55302	119
3	IRRHLB	23032	293.3	–	–	–	–
3	IRLBA	16836	11.1	–	–	–	–
5	primme_svds(1st stage only)	2607	2.7	27342	34.6	25387	19.0
5	JDSVD	17832	27.3	–	–	–	–
5	SVDIFP	22565	68.7	49137	215.3	96769	197.2
5	IRRHLB	20015	261.5	–	–	–	–
5	IRLBA	13755	10.4	–	–	–	–
10	primme_svds(1st stage only)	3355	3.3	36820	43.0	36420	26.9
10	JDSVD	55100	100.4	–	–	–	–
10	SVDIFP	40019	125.8	–	–	–	–
10	IRRHLB	14907	210.2	–	–	–	–
10	IRLBA	8617	7.4	–	–	–	–

TABLE 5.6

Seeking 1, 3, 5, and 10 smallest singular triplets of rectangular matrices with user tolerance 1e-14. The primm_svds method has two variants: primme_svds(GD+k) uses GD+k eigenmethod while primme_svds(JDQMR) uses JDQMR eigenmethod respectively in the second stage.

	δ = 1e-14 Matrix:	well1850		lp_ganges		deter4	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(GD+k)	637	0.4	546	0.5	455	0.6
1	primme_svds(JDQMR)	638	0.4	512	0.3	448	0.4
1	JDSVD	1838	1.9	1002	0.9	9351	18.3
1	SVDIFP	2306	2.3	697	0.8	514	2.4
1	IRRHLB	1368	13.8	500	5.1	1213	17.0
1	IRLBA	1700	0.7	356	0.2	548	0.5
3	primme_svds(GD+k)	894	0.5	506	0.4	3356	3.4
3	primme_svds(JDQMR)	883	0.5	505	0.3	3319	2.9
3	JDSVD	71259	57.3	4357	3.9	–	–
3	SVDIFP	5016	5.0	1540	1.7	13383	58.8
3	IRRHLB	1137	12.3	528	5.5	–	–
3	IRLBA	1206	0.6	396	0.2	33426	27.2
5	primme_svds(GD+k)	1158	0.7	1065	0.8	5097	5.0
5	primme_svds(JDQMR)	1147	0.6	1039	0.7	5064	4.5
5	JDSVD	–	–	15302	13.5	–	–
5	SVDIFP	7250	7.0	3442	3.7	41197	184
5	IRRHLB	1196	12.9	1493	16.4	17342	257.5
5	IRLBA	1378	0.5	1268	0.5	45445	38.9
10	primme_svds(GD+k)	1846	1.0	2044	1.4	9644	9.7
10	primme_svds(JDQMR)	1839	1.0	1976	1.1	9437	8.3
10	JDSVD	–	–	–	–	–	–
10	SVDIFP	13327	12.6	17584	20.0	74899	330.4
10	IRRHLB	1069	12.3	4435	53.5	–	–
10	IRLBA	1014	0.4	1372	0.6	66861	57.5

	δ = 1e-14 Matrix:	plddb		ch		lp_bnl2	
k	Method	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(GD+k)	3594	3.8	24937	42.5	25549	25.0
1	primme_svds(JDQMR)	3355	2.6	16606	15.0	20564	14.5
1	JDSVD	72964	99.9	–	–	89420	113.7
1	SVDIFP	12995	38.7	20993	80	34769	68.2
1	IRRHLB	32957	418.8	–	–	–	–
1	IRLBA	63236	39.3	52484	43.1	–	–
3	primme_svds(GD+k)	8213	12.9	81205	184.5	92755	138.1
3	primme_svds(JDQMR)	4119	3.1	32604	30.0	30688	21.8
3	JDSVD	–	–	–	–	–	–
3	SVDIFP	47169	134.8	58073	240	–	–
3	IRRHLB	27556	356.1	–	–	–	–
3	IRLBA	26016	17.0	–	–	–	–
5	primme_svds(GD+k)	13145	22.5	115175	266	84784	113.9
5	primme_svds(JDQMR)	5216	3.8	44269	41.1	40641	28.8
5	JDSVD	–	–	–	–	–	–
5	SVDIFP	57706	176.5	–	–	–	–
5	IRRHLB	27332	360.2	–	–	–	–
5	IRLBA	23891	16.3	–	–	–	–
10	primme_svds(GD+k)	14591	23.3	–	–	–	–
10	primme_svds(JDQMR)	7633	5.6	74120	67.5	68111	47.8
10	JDSVD	–	–	–	–	–	–
10	SVDIFP	92668	288.2	–	–	–	–
10	IRRHLB	17811	248.8	–	–	–	–
10	IRLBA	11913	8.7	–	–	–	–

TABLE 5.7

Comparing two preconditioning methods for seeking ten smallest singular triplets. Primme_svds(two stage) is the two stage primme_svds we proposed before while primme_svds(dynamic) is the dynamic switching approach between the normal equations approach and the augmented approach.

δ = 1e-14 Matrix:		pde2961		dw2048		fidap4	
droptol	primme_svds	MV	Sec	MV	Sec	MV	Sec
1e-3	two stage	166	0.4	211	0.7	210	1.3
1e-3	dynamic	242	0.4	283	0.7	286	1.4
1e-2	two stage	258	0.5	673	1.6	813	3.2
1e-2	dynamic	307	0.5	668	1.5	1043	3.7

δ = 1e-14 Matrix:		jagmesh8		wang3		lshp3025	
droptol	primme_svds	MV	Sec	MV	Sec	MV	Sec
1e-3	two stage	163	0.5	306	5.5	209	3.0
1e-3	dynamic	223	0.6	396	5.5	273	3.6
1e-2	two stage	990	3.1	736	8.9	7631	132.4
1e-2	dynamic	547	1.6	1038	9.6	696	10.1

TABLE 5.8

Seeking smallest singular triplet with ILU, droptol = 1e-3. The primm_svds method has two variants: primme_svds(GD+k) uses GD+k eigenmethod while primme_svds(JDQMR) uses JDQMR eigenmethod respectively in the second stage. We report the time for generating the preconditioner and running each method separately.

	δ = 1e-8 Matrix:	fidap4		jagmesh8		wang3		lshp3025	
	ILU Time:	0.1		0.1		2.8		0.1	
k	Method	MV	Sec	MV	Sec	MV	Sec	MV	Sec
1	primme_svds(1st stage only)	15	0.1	13	0.1	46	2.5	19	0.3
1	JDSVD	67	0.7	34	0.3	45	2.0	56	1.5
1	SVDIFP	58	0.4	51	0.2	132	5.7	82	1.7
10	primme_svds(1st stage only)	117	0.6	91	0.2	185	2.5	122	1.7
10	JDSVD	342	3.1	287	1.4	320	15.7	364	10.5
10	SVDIFP	691	3.0	561	1.2	1179	29.1	1187	21.9

	δ = 1e-14 Matrix:	fidap4		jagmesh8		wang3		lshp3025	
	ILU Time:	0.1		0.1		2.8		0.1	
1	primme_svds(GD+k)	62	0.6	52	0.1	102	1.8	66	0.5
1	primme_svds(JDQMR)	64	0.3	55	0.1	106	1.1	68	0.5
1	JDSVD	78	1.5	45	0.3	67	3.0	79	1.2
1	SVDIFP	98	0.6	100	0.3	235	8.3	159	2.3
10	primme_svds(GD+k)	210	1.3	163	0.5	306	5.5	209	3.0
10	primme_svds(JDQMR)	251	1.2	215	0.5	402	5.0	265	3.5
10	JDSVD	573	5.5	408	1.9	518	14.6	606	26.9
10	SVDIFP	1152	5.4	981	1.9	1991	51.4	1897	29.1

TABLE 5.9

Seeking smallest singular triplet with RIF, droptol = 1e-3. The primm_svds method has two variants: primme_svds(GD+k), denoted as p(GD+k), uses GD+k eigenmethod while primme_svds(JDQMR), denoted as p(JDQMR), uses JDQMR eigenmethod respectively in the second stage. We report the time for generating the preconditioner and running each method separately.

δ = 1e-8 Matrix:		fidap4		jagmesh8		lshp3025		deter4		plddb		lp_bnl2	
	RIF Time:	1.5		0.5		4.9		11.0		0.4		1.6	
k	Method	MV	Sec	MV	Sec	MV	Sec	MV	Sec	MV	Sec	MV	Sec
1	primme_svds	291	0.4	119	0.1	295	0.6	27	0.2	10	0.1	15	0.1
1	JDSVD	1729	8.9	1311	3.6	2674	21.7	122	3.8	67	0.3	89	0.5
1	SVDIFP	513	1.4	622	0.9	732	5.3	142	2.1	29	0.1	49	0.1
10	primme_svds	1224	1.8	307	0.5	784	2.8	405	2.4	52	0.1	74	0.1
10	JDSVD	8131	39.5	2356	5.7	–	–	–	–	–	–	–	–
10	SVDIFP	4359	10.3	3118	4.2	5308	40.8	2278	45.4	390	1.0	453	1.0

δ = 1e-14 Matrix:		fidap4		jagmesh8		lshp3025		deter4		plddb		lp_bnl2	
	RIF Time:	1.5		0.5		4.9		11.0		0.4		1.6	
k	Method	MV	Sec	MV	Sec	MV	Sec	MV	Sec	MV	Sec	MV	Sec
1	p(GD+k)	521	1.0	207	0.3	466	1.5	82	0.5	45	0.1	66	0.1
1	p(JDQMR)	544	1.0	229	0.2	514	1.5	87	0.4	45	0.1	69	0.1
1	JDSVD	2037	10.5	1410	3.6	3004	24.5	188	5.5	122	0.5	134	0.6
1	SVDIFP	843	2.1	990	1.3	1394	8.5	221	4.3	48	0.1	63	0.1
10	p(GD+k)	2074	3.2	562	0.8	1364	4.8	769	2.5	152	0.5	192	0.5
10	p(JDQMR)	2604	4.9	641	0.9	1616	7.0	877	5.3	247	0.5	242	0.5
10	JDSVD	–	–	12057	28.2	–	–	–	–	–	–	–	–
10	SVDIFP	7470	15.1	5024	6.1	9999	83	3705	64.5	748	1.0	862	1.0

TABLE 5.10

Seeking 10 smallest singular triplets using shift and invert technique. LU(A) and QR(A) are the time for LU factorization and QR factorization on the matrix A respectively. We report the time of each method including their running time and associated factorization time.

δ = 1e-10	fidap4		jagmesh8		lshp3025		deter4		plddb		lp_bnl2	
Method	MV	Sec	MV	Sec	MV	Sec	MV	Sec	MV	Sec	MV	Sec
LU(A) time	–	0.02	–	0.01	–	0.06	–	0.01	–	0.01	–	0.01
primme_svds	31	0.10	26	0.07	35	0.22	167	14.4	47	0.28	35	1.01
SVDIFP	380	0.90	316	0.31	400	1.22	1177	168.4	418	1.92	432	9.3
QR(A) time	–	0.02	–	0.01	–	0.04	–	0.53	–	0.01	–	0.10
primme_svds	31	0.29	26	0.08	29	0.53	166	9.1	27	0.08	36	0.48
SVDIFP	383	2.24	316	0.37	422	3.59	1177	55.4	418	0.55	432	3.13
svds	73	0.33	61	0.23	65	0.36	–	–	–	–	–	–

TABLE 5.11

Basic information of some real-world test matrices

Matrix	debr	cage14	thermal2	sls	Rucci1
rows m:	1048576	1505785	1228045	1748122	1977885
cols n:	1048576	1505785	1228045	62729	109900
nnz(A)	4194298	27130349	8580313	6804304	7791168
σ_1	1.11E-20	9.52E-02	1.61E-06	9.99E-1	1.04E-03
$\kappa(A)$	3.6E+20	1.01E+1	7.48E+6	1.30E+3	6.74E+3
Application	undirected graph	directed graph	thermal	Least Squares	Least Squares
Preconditioner	No	ILU(0)	ILU(1e-3)	RIF(1e-3)	RIF(1e-3)

TABLE 5.12

Seeking the smallest singular triplet for real world problems. We report the time of each method including their running time and associated factorization time (PRtime) separately.

δ = 1e-12		primme_svds			SVDIFP			JDSVD		
Matrix	PRtime	MV	Sec	RES	MV	Sec	RES	MV	Sec	RES
debr	—	539	84.3	3e-12	403*	245.7*	2.0e-01	1971	474.6	2e-12
cage14	2e0	19	11	4e-13	33	28	5.9e-13	111	185	7e-14
thermal2	3.69e+03	419	506	7e-12	–	–	4.4e-09	309	535	4e-12
sls	3.29e3	1779	170	1e-09	408*	328*	1.1e-09	–	–	2
Rucci1	6.92e4	4728	1087	7e-12	4649	6464	5.8e-12	–	–	4.8e-03