A GOLUB-KAHAN DAVIDSON METHOD FOR ACCURATELY COMPUTING A FEW SINGULAR TRIPLETS OF LARGE SPARSE MATRICES [*]

STEVEN GOLDENBERG, ANDREAS STATHOPOULOS, ELOY ROMERO [†]

Abstract. Obtaining high accuracy singular triplets for large sparse matrices is a significant challenge, especially when searching for the smallest triplets. Due to the difficulty and size of these problems, efficient methods must function iteratively, with preconditioners, and under strict memory constraints. In this research, we present a Golub-Kahan Davidson method (GKD), which satisfies these requirements and includes features such as soft-locking with orthogonality guarantees, an inner correction equation similar to Jacobi-Davidson, locally optimal +k restarting, and the ability to find real zero singular values in both square and rectangular matrices. Additionally, our method achieves full accuracy while avoiding the augmented matrix, which often converges slowly for the smallest triplets due to the difficulty of interior eigenvalue problems. We describe our method in detail, including implementation issues that arise. Our experimental results confirm the efficiency and stability of our method over the current implementation of PHSVDS in the PRIMME software package.

Key words. Singular Value Decomposition, Iterative Methods

AMS subject classifications. 65F04,65B04,68W04,15A04

1. Introduction.

Assuming a large sparse matrix, $A \in \Re^{m,n}$ with $m \geq n$, the economy size singular value decomposition (SVD) is given by

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

where $\mathbf{U} \in \Re^{m,n}$ and $\mathbf{V} \in \Re^{n,n}$ are orthonormal bases and $\Sigma = diag(\sigma_1, \ldots, \sigma_n) \in \Re^{n,n}$ with $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_n$ is a diagonal matrix containing the singular values of A. The singular triplets of A are defined as $(\mathbf{u}_i, \sigma_i, \mathbf{v}_i)$, where bold face differentiates from search space vectors in this paper. When using inexact arithmetic, we have the left and right singular value residuals, defined as $r_u = A^T u - \sigma v$ and $r_v = Av - \sigma u$ respectively.

This decomposition has become increasingly important and is frequently used in fields like statistics for principal component analysis [14], computer science for image compression [23] and web search clustering [21], and genomics for expression data processing [2]. More specifically, finding the smallest singular triplets is useful for total least squares problems and the determination of the effective rank of a matrix [9], and for variance reduction of inverse operators [7].

Additionally, finding high accuracy solutions is crucial when running in a single or low precision environment. In single precision, matrix multiplication can only provide 1.2E-7$\|A\|$ of accuracy, and in practice this bound is optimistic for iterative solvers due to accumulated error. Despite this limitation, single-precision calculations have become increasingly important for deep learning applications [11] which are often resistant to errors and therefore require less than full double precision. Reducing the precision of matrix vector multiplications can provide speed ups on CPUs due to increased vectorization, and GPUs can obtain speedups of 2x-4x [32]. In addition, using single precision cuts the storage requirements in half. Specifically, the use of

*This work is supported by NSF under grant No. ACI SI2-SSE 1440700, and by DOE under an ECP grant.

†College of William and Mary, (sgoldenberg@email.wm.edu, andreas@cs.wm.edu, eloy@cs.wm.edu).

single precision calculations is encouraged by Advanced Micro Devices (AMD) for OpenCL applications [1], and half precision, which can only provide 1E-3$\|A\|$ digits of accuracy, has been growing in popularity on NVIDIA's GPUs [18].

When the matrix A is large enough, it can be inefficient to compute the SVD with dense methods. Furthermore, applications often require only a few of the largest or smallest singular values and vectors. These considerations have lead to the use of iterative algorithms like Golub-Kahan-Lanczos (GKL) also known as Lanczos bidiagonalization [8]. However, when the solution requires many iterations, it may be infeasible to store all the GKL vectors necessary for full or partial reorthogonalization. To solve this, restarted versions of GKL that limit the maximum basis size, such as IRLBA [4], have been developed. Additionally, other methods have emerged, such as Jacobi-Davidson (JDSVD) [12], the Preconditioned Hybrid SVD method (PHSVDS) [31], and the Preconditioned Locally Minimal Residual method (PLMR_SVD) [28]. These methods can use the more advanced +k (also known as locally optimal) restarting and can take advantage of preconditioning, which can provide significant speedups for difficult problems.

In general without preconditioning or +k restarting, these methods build Krylov spaces on the normal equations matrix $C = A^TA$ or on the augmented matrix,

$$B = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}. \tag{1.2}$$

We denote a k-dimensional Krylov space on a square matrix A with initial vector v_1 by $K_k(A, v_1) = span\{v_1, Av_1, \ldots, A^{k-1}v_1\}$ and $\|\cdot\|$ denotes the Euclidean norm.

Frequently, methods that build their search space with B, like JDSVD and PLMR_SVD, are able to achieve accuracy of $\|r_B\| < O(\|A\|\epsilon_{mach})$ when searching for the smallest singular triplets, where ϵ_{mach} is the working machine precision and $r_B = [r_u; r_v]$ is the eigenvalue residual on B. However, B has singular values $\pm\sigma_i$ [22], so searching for the smallest singular triplets is a highly interior eigenvalue problem that can converge slowly. Worse, when A is rectangular, the spectrum of B contains $m - n$ zero eigenvalues that are not in the spectrum of A. Therefore, methods on B are unable to distinguish real zero singular values of A within the spectrum when $m \neq n$.

Alternatively, methods that build $K_k(C, v_1)$ explicitly are only able to achieve accuracy $O(\|C\|\epsilon_{mach}) = O(\|A\|^2\epsilon_{mach})$ for the eigenvalue residual on C, r_C. Additionally, r_C is equivalent to a scaling of r_u, as seen in equation 1.3.

$$r_C = A^TAv - \sigma^2 v = \sigma(A^Tu - \sigma v) = \sigma r_u. \tag{1.3}$$

Thus, if $\sigma_1 \neq 0$, the norm of the singular value residual when searching for the smallest singular value cannot be better than $O(\|A\|\kappa(A)\epsilon_{mach})$, where $\kappa(A) = \frac{\sigma_n}{\sigma_1}$ is the condition number of A. Despite the squaring of the spectrum, these methods usually converge faster than methods on B, both in theory and in practice, due to the extremal problem they solve. Furthermore, these methods are often able to find real zero singular values of A, as the corresponding eigenproblem on C does not introduce extraneous zero eigenvalues.

In this work, we introduce a Golub-Kahan Davidson method (GKD), which keeps the convergence of methods on C, but attains the full accuracy of methods on B. Specifically, we define full accuracy to be $\sqrt{\|r_u\|^2 + \|r_v\|^2} < \|A\|\epsilon_{mach}$. First, we discuss related methods such as GKL, JDSVD, PLMR_SVD and PHSVDS, followed by a detailed description of our method including implementation details. Lastly, we provide experimental results that highlight the capabilities of GKD compared to the current implementation of PHSVDS in the PRIMME software package.

1.1. Related Work. GKL [16] builds two vector bases, one for the right space $K_k(A^TA, v_1)$ and one for the left space $K_k(AA^T, Av_1)$. It builds the second basis while computing the first one without additional matrix vector multiplications (matvecs). More importantly, it avoids directly multiplying vectors with A^TA and thus avoids the numerical problems associated with working on C. This is done by keeping two orthogonal spaces, U and V, where the last vector of V, v_k, is used to expand U as $u_k = Av_k$ and the last vector of U, u_k, is used to expand V as $v_{k+1} = A^Tu_k$. These new vectors are orthonormalized to the previous ones in their corresponding bases and the coefficients from this process are used to create the bidiagonal projection matrix U^TAV. GKL solves the smaller singular value problem on this projection matrix to approximate the singular triplets.

While GKL is considered to be one of the most accurate and effective algorithms for finding small singular triplets, the standard version is unrestarted and cannot be preconditioned. Therefore, GKL tends to be computationally slow for poorly separated triplets of large matrices. Many restarted versions have been developed [5, 4, 13] but use primarily implicit or thick restarting [29] and thus are unable to maintain the convergence of the unrestarted method. Locally optimal (also known as +k) restarting uses vectors from successive iterations in a way similar to a nonlinear conjugate gradient and has been shown to converge similarly to an unrestarted method for both eigenvalue [15, 27, 26] and singular value problems [31].

SVDIFP [17] implements an inner-outer method where the inner one builds a preconditioned Krylov space $K_k(M(C - \rho_i I), x_i)$, where M is a preconditioner for C and (x_i, ρ_i) is the approximate right singular vector and value at the i-th step of the outer iteration. SVDIFP is able to avoid numerical problems, at least for the right singular vectors, by using a two sided projection similarly to GKL. SVDIFP's structure, however, does not allow for many of the optimization techniques of Davidson-type methods which can significantly improve convergence [31].

JDSVD [12] works on B by using two independent subspaces rather than one. It is an inner outer method that expands both spaces by solving a Jacobi-Davidson type correction equation on B. Without preconditioning, restarting, or solving the correction equation, the JDSVD outer method builds subspaces that span the following Krylov spaces:

$$U_k = K_{\frac{k}{2}}(AA^T, u_1) \oplus K_{\frac{k}{2}}(AA^T, Av_1), \quad V_k = K_{\frac{k}{2}}(A^TA, v_1) \oplus K_{\frac{k}{2}}(A^TA, A^Tu_1). \tag{1.4}$$

These spaces are similar to the ones used in GKL, but crucially, each space is the sum of two different spaces of half dimension. This allows JDSVD to take advantage of initial guesses for both the left and right singular vectors. However, it also means that the outer solver in JDSVD requires twice as many matvecs to build a space of equal Krylov dimension. Furthermore, if we choose initial vectors that satisfy $v_1 = A^Tu_1$, the outer iteration of JDSVD becomes wasteful as it builds the same space as a GKL with half the dimension (in this case the spaces $K_{\frac{k}{2}}(A^TA, v_1)$ and $K_{\frac{k}{2}}(A^TA, A^Tu_1)$ in (1.4) differ only by one vector). This is also true of eigensolvers on B as seen below,

$$B^2 \begin{bmatrix} v \\ Av \end{bmatrix} = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}^2 \begin{bmatrix} v \\ Av \end{bmatrix} = \begin{bmatrix} A^TAv \\ AA^T(Av) \end{bmatrix}. \tag{1.5}$$

The inner correction equation used in JDSVD often allows for faster convergence than standard eigenvalue methods on B while maintaining the ability to converge to full accuracy. Despite these benefits, it can still suffer from the same issues as other eigenmethods on B.

PHSVDS [31] exploits the different advantages of eigenmethods on B and C by utilizing each in a two-stage method. The first stage can use any state-of-the-art eigensolver on C, which gives it fast convergence until either the user tolerance is met or until switching to a second stage using an eigensolver on B is necessary to reach the remaining user tolerance. Switching to an eigensolver on B after a fully converged first stage can effectively utilize good initial guesses from the first stage on C, and thus PHSVDS can avoid resolving the entire accuracy on an indefinite problem. Its implementation in PRIMME can use any of the two near-optimal eigensolvers, GD+k or JDQMR. This two-stage approach has been shown to be faster than eigensolvers on B alone, and typically has better performance than other SVD methods.

While PHSVDS has shown significant improvements, it is still limited by the speed of eigensolvers on B when the matrix is ill-conditioned. It converges quite well for problems that do not need to switch stages, but eigensolvers on C cannot converge to high accuracy if the smallest singular value is nearly 0. Once it switches to the second stage on B, a significant slowdown occurs associated with interior problems and methods based on the augmented matrix. We see later than GDK converges with the near-optimal speed of GD+k on C down to $O(\|A\|\epsilon_{mach})$.

PLMR_SVD [28] is a recent method based on a stationary iteration that uses two separate four-term recurrences to build the following spaces,

$$\text{span}\{v^{(i)}, r_u^{(i)}, P(A^T r_v^{(i)} - \sigma r_u^{(i)}), v^{(i-1)}\}$$

$$\text{span}\{u^{(i)}, r_v^{(i)}, P(A r_u^{(i)} - \sigma r_v^{(i)}), u^{(i-1)}\},$$

where $v^{(i)}$ and $u^{(i)}$ are the i-th approximations of the right and left singular vectors respectively, and $r_v^{(i)} = P(Av^{(i)} - \sigma u^{(i)})$ and $r_u^{(i)} = P(A^T u^{(i)} - \sigma v^{(i)})$ are their preconditioned right and left residuals respectively. Without a preconditioner, PLMR_SVD is equivalent to GD+1 with a 3-vector basis (or LOBPCG) on B. There may be additional benefits to building the spaces separately, but PLMR_SVD lacks the subspace acceleration present in GD+k and JDSVD, which can provide superlinear convergence.

2. Main Contribution.

In the following section, we describe the proposed method, GKD, in detail, especially focusing on the selection of approximate singular triplets from our subspaces and the implementation of our restarting method. Additionally, we discuss error accumulations that occur due to restarting and the mitigation strategy required to ensure reliable performance for high accuracy calculations. Finally, we extend GKD to an inner-outer method that solves a Jacobi-Davidson correction equation.

2.1. Algorithm.

Our algorithm is designed to mimic the numeric nature of GKL by keeping two orthonormal bases for the right and left space, V and Q respectively, which are built without multiplying directly with A^TA. Instead, we build Q such that $AV = QR$ is the economy QR factorization of AV. Then, we extend V with a left residual based on a Galerkin extraction from R. Without preconditioning or +k restarting, this process is identical to GKL, building the right and left spaces $K_q(A^TA, v_1)$ and $K_q(AA^T, Av_1)$ after q iterations or $2q$ matvecs. Since both the extraction of approximate triplets through the SVD of R and the expansion of the spaces avoid a direct multiplication with C, we avoid the squaring of the norm and condition number that occurs with eigensolvers on C.

Specifically, we extract approximate singular triplets from these spaces using a Rayleigh-Ritz procedure that is adapted for the SVD. Given search spaces $\mathcal{Q} \subset \mathbb{R}^{\mathrm{m}}$ and $\mathcal{V} \subset \mathbb{R}^{\mathrm{n}}$, we can determine approximations (u, σ, v) with the following two Galerkin conditions on the right and left residuals,

$$\begin{aligned} Av - \sigma u &\perp \mathcal{Q}, \\ A^T u - \sigma v &\perp \mathcal{V}. \end{aligned} \tag{2.1}$$

Since $u \in \mathcal{Q}$ and $v \in \mathcal{V}$, we can write $u = Qx$ and $v = Vy$, where Q and V form k-dimensional orthonormal bases of $\mathcal{Q}$ and $\mathcal{V}$ respectively. Additionally, $AV = QR \Rightarrow Q^T AV = R$, which allows us to rewrite the conditions as follows:

$$\begin{aligned} Q^T AVy = \sigma Q^T Qx &\Rightarrow Ry = \sigma x \\ V^T A^T Qx = \sigma V^T Vy &\Rightarrow R^T x = \sigma y. \end{aligned} \tag{2.2}$$

Therefore, solving the singular value decomposition on R with singular triplets (x, σ, y) satisfies both constraints and provides approximations to the singular triplets of A.

To expand the right search space, we take the approximations from the above Rayleigh-Ritz extraction and use them to form the left residual $r_u = A^T u - \sigma v$. Then, we can choose to expand V with this r_u directly, or with the preconditioned residual Pr_u, where P is a suitable preconditioner for $A^T A$ or for $A^T A - \sigma I$, if available.

We expand the left space Q with Av_{i+1} instead of a preconditioned right residual. This differentiates the method from JDSVD with the goal of producing a faster converging outer method. Specifically, from (1.3) the left residual r_u is colinear with the residual r_C of the Generalized Davidson (GD) method [20] on the matrix C, which is also colinear with the new GKL direction for V. In addition, the Rayleigh-Ritz on C used by GD gives the same answer as (2.2),

$$V^T A^T AVy = \sigma y \Rightarrow R^T Ry = \sigma y,$$

so, in exact arithmetic, GKD is equivalent to GD solving the eigenproblem on $A^T A$. Without preconditioning or restarting, it is also equivalent to GKL and thus it is twice as fast as JDSVD if the latter is used only as an outer method. By construction, GKD has similar numerical properties as GKL, whereas the accuracy of GD is limited by working directly on $A^T A$. GKD can also be used with thick and +k restarting, which in exact arithmetic makes it equivalent to GD+k on C, the first stage method of PHSVDS, but without the numerical limitations. Algorithm 2.1 shows the restarted and preconditioned version of GKD when seeking one singular triplet. Although the orthogonalization of step 13 can be avoided without preconditioning [24], it is needed for high accuracy and in our more general method that allows for flexible preconditioning. Furthermore, the algorithm can be extended to find more than one singular triplets by using soft or hard locking. A block version is similarly possible.

2.2. Restarting and Locking. Our restart procedure takes the current best approximations to the s singular triplets closest to the user specified target, $\tilde{\sigma}$, and uses them together with those from the +k restarting to compress V, Q and R down to dimension $s + k$. The steps for building the restarted V follow closely the description in [26] and are shown in lines 1-7 of Algorithm 2.2.

The simplest method to restart Q and R, without recomputing the QR factorization of the restarted AVt, is to set them as $Q\tilde{Q}$ and $\tilde{R}$ respectively, where $Rt = \tilde{Q}\tilde{R}$ is the QR factorization of Rt with $t = [Y_1, v_{new}]$ from line 6 of Algorithm 2.2. This

Algorithm 2.1 GKD Iteration

```
1: Define target $\tilde{\sigma}$, initial vector $v_1$, max basis size $q$, tolerance $\delta$, preconditioner $P$, and $i = 1$
2: Build $V = [v_1]$, $Q = [\frac{Av_1}{\|Av_1\|}]$, and $R = \|Av_1\|$
3: while $\sqrt{\|r_u\|^2 + \|r_v\|^2} > \|A\|\delta$ do
4:     while $i < q$ do
5:         Compute SVD of $R$
6:         Choose the singular triplet $(x, \sigma_r, y)$ of $R$ nearest to the target $\tilde{\sigma}$
7:         Save $v_{old} = y$ for +k restarting
8:         Set $u = Q(:, 1:i)x$, $v = V(:, 1:i)y$
9:         Compute left residual: $r_u = A^T u - \sigma_r v$
10:        $V(:, i+1) = Pr_u$
11:        Orthogonalize $V(:, i+1)$ against $V(:, 1:i)$
12:        $Q(:, i+1) = AV(:, i+1)$
13:        Orthogonalize $Q(:, i+1)$ against $Q$ and update $R(:, i+1)$
14:        $i = i + 1$
15:    end while
16:    call Algorithm 2.2 to restart
17: end while
```

Algorithm 2.2 Restart Procedure

```
1: Define restart size $s$ and target $\tilde{\sigma}$
2: Compute SVD of $R = X\Sigma_r Y^T$
3: Choose $s$ singular triplets of $R$ closest to $\tilde{\sigma}$ (called $(X_1, \Sigma_r^{(1)}, Y_1)$)
4: Save the remaining singular triplets from the SVD of R, $(X_2, \Sigma_r^{(2)}, Y_2)$
5: $v_{new} \leftarrow$ Orthogonalize saved +k vectors $[v_{old}; 0]$ from main iteration against $Y_1$
6: $t = [Y_1, v_{new}]$
7: $V = Vt$
8: if Reset criteria is met then
9:     Reorthogonalize $V$ and build $Q$ and $R$ such that $AV = QR$
10: else
11:    QR factorize $\Sigma_r^{(2)} Y_2^T v_{old} = \tilde{Q}\tilde{R}$
12:    Set $Q = Q[X_1 X_2 \tilde{Q}]$ and $R = \begin{bmatrix} \Sigma_r^{(1)} & 0 \\ 0 & \tilde{R} \end{bmatrix}$.
13: end if
```

can introduce numerical error of magnitude $O(\|R\|\epsilon_{mach})$, which can be as large as $O(\|A\|\epsilon_{mach})$. Although this error is acceptable for a single QR factorization, the error accumulates over many restarts causing the factorization not to correspond to the actual AV and eventually causing loss of convergence. It is possible to intelligently compute Q and R to avoid direct multiplications with R through the already available SVD of R as seen below,

$$
\begin{aligned}
AVt = QRt &= Q \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} \Sigma_r^{(1)} & 0 \\ 0 & \Sigma_r^{(2)} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & Y_2^T v_{old} \end{bmatrix} \\
&= Q \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_r^{(2)} Y_2^T v_{old} \end{bmatrix}.
\end{aligned} \tag{2.3}
$$

From (2.3), the new Q and R can be obtained with minimal effort by performing a QR factorization $\Sigma_r^{(2)} Y_2^T v_{old} = \tilde{Q}\tilde{R}$. The restarted Q and R are given in Line 12 of Algorithm 2.2. This strategy has better numerical behavior because we separate the space of small singular values that are kept in thick restarting (X_1) from the +k restarting space which has correction directions over the entire singular space (including those of large magnitude). By explicitly decoupling $\Sigma_r^{(1)}$ and $\tilde{R}$ in R, any errors in $\tilde{R}$ do not affect the ability of the algorithm to compute the smallest eigenvectors and they only affect the correction directions. Moreover, as the +k algorithm typically uses only $k = 1$ previous vectrors, no errors are expected.

To accurately find many singular triplets, we implement two versions of locking. The first, hard-locking, locks singular vectors out of the search space explicitly once the required user tolerance is reached. At every iteration, we orthogonalize the vector added to V against the locked right singular vectors, as well as the previous vectors in V. In practice, the vectors added to Q do not require orthogonalization against the locked left singular vectors. The second, soft-locking, merely flags converged singular triplets while leaving them in the basis.

It is known that hard locking can cause stagnation in some rare cases or when the number of locked vectors is large. This is caused by the error still present in the locked vectors, which may contain critical directions for other singular triplets [25]. We have not seen any matrices in this paper that exhibit this behavior. However, soft-locking can provide left and right singular vectors that are orthogonal to machine precision, while hard-locking only obtains left singular vectors orthogonal up to $O(\|A\|\delta)$. Therefore, we present only soft-locking results in this paper. We intend to address the issues with hard-locking more thoroughly in the future.

2.3. Resetting. Since $AV = QR$, the right residual $r_v = Av - \sigma u$ should be zero throughout our procedure,

$$r_v = Av - \sigma u = AVy - Q(\sigma x) = AVy - QRy = (AV - QR)y = 0. \tag{2.4}$$

Generally, this means we can avoid the extra matrix-vector multiplication (or storage for AV) necessary to compute r_v. In practice though, $\|r_v\|$ cannot be better than $O(\|A\|\epsilon_{mach})$ due to the multiplication AV when computing the left space. Worse, $\|r_v\|$ grows as $O(\sqrt{\text{numRestarts}}\|A\|\epsilon_{mach})$, which has also been noticed in [30]. Therefore, our method must calculate $\|r_v\|$ explicitly when $\|r_u\| < \|A\|\delta$, where δ is the user selected tolerance. This ensures we meet the convergence criteria of Algorithm 2.1.

The errors we observe in r_v may grow large enough to exceed the user tolerance, which would make convergence impossible. These errors come from two main sources. The first source is from the loss of orthogonality of V, and the second is the loss of accuracy of the QR factorization of AV. We have found experimentally that both of these errors can impede or halt convergence as the SVD of R no longer corresponds to the singular triplets in A. We note that this issue is rare and only occurs when $\delta \approx \epsilon_{mach}\sqrt{\text{numRestarts}}$. To correct these errors, we implement a resetting procedure that reorthogonalizes V, and rebuilds Q and R directly from a newly computed AV.

It is critical to only reset sparingly, as rebuilding Q and R from scratch takes $s + k$ matvecs to obtain AV and a full QR factorization. Additionally, resetting can cause an increase in the residual norm by a factor of $\kappa(A)$, which may require a few iterations to reduce back to its previous level. In order to track the errors mentioned above, we have devised two inexpensive criteria that help to avoid unnecessary resets. From (2.4), we can estimate errors in the QR factorization directly from the norm of

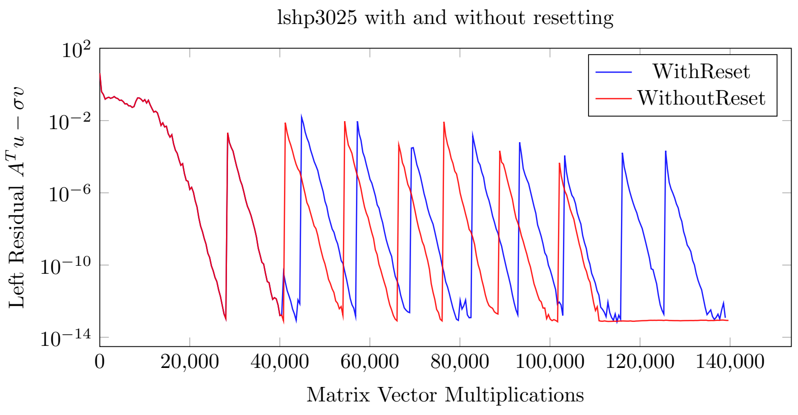


FIG. 1. *Demonstrating the need for resetting on lshp3025 ($\|A\| = 7$) with GKD ($q = 35$, $s = 15$, $\delta =$1E-14, and $k = 1$).*

the right residual. We choose to reset when $\|r_u\| < 1.25\|r_v\|$, as the errors in the QR factorization directly impact the convergence of r_u. Experimentally, we have found a few cases where the small 25% buffer between r_u and r_v is needed to detect potential stagnation.

The error in the orthogonality of V may also cause failures to converge. Therefore, we estimate how large $\|E\| = \|V^TV - I\|$ can be before it begins to affect convergence. Based on the Galerkin conditions, we should have solved the equivalent eigenproblem, $R^TRy = V^TA^TAVy = \sigma^2V^TVy$. In practice, we solve $R^TRy = V^TA^TAVy = \sigma^2y$ regardless of the orthonormality of V. Therefore, we obtain a Ritz vector and Ritz value that will not converge to a 0 residual for the original problem, since $V^TV \neq I$. However, the Ritz pair produced by our inexact Galerkin can be considered as a Ritz pair of an exact Galerkin condition applied to the nearby generalized eigenproblem $A^TAVy = \sigma^2MVy$ where $M = V(V^TV)^{-2}V^T$ as seen below,

$$V^TA^TAVy = \sigma^2V^TMVy = \sigma^2V^TV(V^TV)^{-2}V^TVy = \sigma^2y. \tag{2.5}$$

In order to correctly monitor and maintain convergence, the residual we use for expansion, $r_C = \sigma r_u = A^TAv - \sigma^2v$, should not drift too far from this exact residual, $r_E = A^TAv - \sigma^2V(V^TV)^{-2}V^Tv$, where $v = Vy$. Assuming $\|E\| < 1$, we have

$$\begin{aligned}\|r_E - r_C\| &= \sigma^2\|Vy - V(V^TV)^{-1}y\| \\ &\leq \sigma^2\|V\|\|I - (V^TV)^{-1}\| = \sigma^2\|V\|\|I - (I+E)^{-1}\| \\ &\leq \sigma^2(1 + \|E\|)\|(I+E)^{-1}\|\|E\| \\ &\leq \sigma^2(1 + \|E\|)\left\|I + \sum_{i=1}^{\infty} E^i\right\|\|E\| \\ &= \sigma^2\|E\| + O(\sigma^2\|E\|^2).\end{aligned} \tag{2.6}$$

Since we want $r_u = r_C/\sigma$ to converge to tolerance $\|A\|\delta$, we limit the distance $\|r_E - r_C\| < \|A\|\delta\sigma$. Thus, from (2.6), we perform a reset when $\|E\| \geq \|A\|\delta/\sigma$. In practice we have noticed only a few situations where this criteria caused a reset.

To demonstrate this problem, we ran lshp3025, a problem from the SuiteSparse Matrix Collection [6], which requires thousands of restarts before convergence. Properties of this problem can be found in Table 1. The criteria outlined in the previous paragraphs combine to avoid the stagnation seen in Fig. 1. Due to the very low tolerance of 1E-14 $= 50 * \epsilon_{mach}$, approximately 2,500 restarts or 35,000 matvecs may cause the reset criteria to be met. It is clear our criteria is somewhat conservative, as resets occur approximately every 40,000 matvecs, even when the method is able to converge without it. However, without resetting, the method completely stagnates at around 110,000 matvecs. Moreover, with or without resets, we observe convergence to the first 8 smallest singular values in a similar number of matvecs (110,000), even though adding resets should increase the overall number of matvecs. This indicates the increased stability of the method also can improve performance slightly.

2.4. Inner Solver. Inner-outer solvers like JDSVD and the JDQMR implementation in PRIMME utilize extra matvecs inside of an inner solver as a refinement step to improve the convergence speed of the outer iterations. By solving a related linear system, these methods can provide a significant speedup in time for problems that have a relatively inexpensive matrix-vector multiplication. Furthermore, solving this linear system can reduce the residual of the solution without requiring the expansion of the outer basis. Consequently, the number of orthogonalizations as well as the number restarts are reduced, which avoids their associated error and resets. This is particularly critical for problems that require a significant number of iterations.

GKD can be extended to a Jacobi-Davidson variant, GKJD, that expands the subspace V by the approximate solution of the correction equation

$$(I - vv^T)(A^TA - \sigma^2 I)(I - vv^T)t = -r_u \tag{2.7}$$

instead of applying a preconditioner at line 10 of Algorithm 2.1. Here, and for the remainder of this section, σ without a subscript denotes the shift used for the inner solver, which may be different than the user specified target $\tilde{\sigma}$ or the current approximate singular value. As before, σ_i will denote the ith singular value. The inner equation can also utilize a preconditioner, improving convergence further. In particular, our inner solver is based on the symmetric Quasi-Minimal Residual method (QMRs) used in PRIMME's JDQMR. QMRs can utilize indefinite preconditioners and solve indefinite systems which may occur when σ lies in the interior of the spectrum.

In order to avoid over utilizing the inner method when convergence is poor or the correction equation does not match the desired singular values, or under utilizing the inner method when convergence is good, extra steps must be taken. Due to the smooth convergence of QMRs, we can include dynamic stopping conditions based on estimated eigenvalue residuals to stop the linear solve in a near-optimal way. We have adopted the same QMRs solver and dynamic criteria used in PRIMME's JDQMR [26].

Our inner solver for (2.7) works directly on $A^TA - \sigma^2 I$ so its numerical stability needs to be justified. As with an outer iteration on A^TA, no numerical issues are expected when σ is in the largest part of the spectrum, but when seeking the smallest part, singular values below $O(\|A\|\sqrt{\epsilon_{mach}})$ will become indistinguishable when squared. However, the solution of the inner correction equation still provides useful directions even when a few singular values of A are below $O(\|A\|\sqrt{\epsilon_{mach}})$. The reason is well understood numerically and it is why inverse iteration works well despite a nearly singular linear system [22, sec. 4.3].

Assume there are k singular values below the noise level, i.e., $\sigma_k \leq \|A\|\sqrt{\epsilon_{mach}} < \sigma_{k+1}$, and a shift $\sigma \leq \|A\|\sqrt{\epsilon_{mach}}$. If we ignore the projectors for simplicity, the numerically computed solution of (2.7), $\tilde{t}$, satisfies

$$\tilde{t} = t + \mathbf{V}(\Sigma^2 - \sigma^2)^{-1}\mathbf{V}^T E\tilde{t}, \tag{2.8}$$

where the backward error satisfies $\|E\| \leq ||A^T A||\epsilon_{mach}$. Therefore, the relative forward error is a vector $\frac{\tilde{t}-t}{\|\tilde{t}\|} = \sum_{i=1}^{n} \mathbf{v}_i c_i$ with the coefficients satisfying

$$|c_i| = \frac{|\mathbf{v}_i^T E\tilde{t}|}{|\sigma_i^2 - \sigma^2|\|\tilde{t}\|} \leq \frac{\|A\|^2\epsilon_{mach}}{|\sigma_i^2 - \sigma^2|}. \tag{2.9}$$

For $i > k$, we have $\sigma_i \geq \sigma_{k+1} > \|A\|\sqrt{\epsilon_{mach}}$, and thus $|c_i| = O(\frac{\|A\|^2}{\sigma_i^2}\epsilon_{mach}) < 1$. As the separation increases, $\sigma_{k+1} \gg \|A\|\sqrt{\epsilon_{mach}}$, we have $c_i \ll 1$ and the errors in the $\mathbf{v}_i, i > k$, directions become negligible. For $i \leq k$, we have $|\sigma_i^2 - \sigma^2| < \|A\|^2\epsilon_{mach}$ and thus the corresponding c_i could blow up. In practice, calculations at the noise level of the arithmetic will limit $c_i = O(1)$ but either way these $\mathbf{v}_i, i \leq k$, directions dominate the correction vector.

The behavior is similar when the backward error is at the level of the residual norm at which we solve (2.7), i.e., $\|E\| \leq \|A\|^2\theta$, for some tolerance θ. Typically we ask for a residual norm reduction relative to $\|r_u\|$ but this can be translated to a θ. Then, the $|c_i|$ in (2.9) have the same bounds as above only multiplied by θ/ϵ_{mach}. Since the approximate solution has $\|t\| = O(\theta)$, the effect of the noise error is larger.

We can view the noise of the numerically computed correction $\tilde{t}$ as the application of a low pass filter with the diagonal matrix $diag(c_i)$, where the $i < k$ singular components dominate the result. Clearly, the inner iteration cannot differentiate between these k smallest singular directions which look like a multiplicity. However, the Rayleigh Ritz of the outer method has no problems approximating these singular vectors as long as their k-dimensional space is sufficiently represented in the outer search space.

If the outer method in GKJD has a restart size $s \geq k$ and the gap σ_{k+1}/σ_k is large, then the filter ensures that all $\mathbf{v}_i, i = 1, \ldots, k$, will be approximated well after k outer iterations. As the gap narrows, the filter boosts also directions of larger singular values up to σ_f, where $\frac{\|A\|^2}{\sigma_f^2}\epsilon_{mach}$ starts to become negligible. Therefore, the outer method may take more than k iterations, although convergence depends on the gaps in the "filtered" $\sigma_1, \ldots, \sigma_f$ spectrum, which has much smaller spread than the entire spectrum.

The situation is similar if the restart size $s < k$ and σ_{k+1}/σ_k is large, since the search space cannot capture all small singular vectors, so convergence will occur based on the perceived gaps after the implicit application of the filter. In the extreme case of $s \ll k$ and/or very small spectral gaps, we can expect the method to be slow. However, in such ill-conditioned problems, no better algorithmic options exist without a preconditioner.

Figures 2 and 3 show examples of how GKJD with dynamic stopping conditions for the inner iteration can converge even when several singular values are below $\|A\|\sqrt{\epsilon_{mach}}$. They also show that GKJD is competitive and sometimes faster than GKD in terms of matrix-vector products, in addition to the benefit of a less expensive iteration. The matrices have a specified spectrum Σ and random left and right singular vectors.

In Figure 2 the matrix has 16 singular values below $\|A\|\sqrt{\epsilon_{mach}}$ but we limit GKD and GKJD to a restart size of only 15. Even with this limitation, GKJD is able to converge to the smallest singular triplet with relative accuracy of 1E-14, and it does so three times faster than GKD. Additionally, with only a few extra outer iterations, GKJD can find 14 of the smallest singular values.

The difference seen between GKD and GKJD is due to the large number of

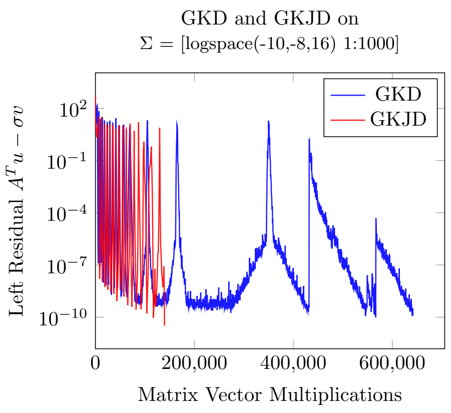


FIG. 2. *Convergence of GKD and GKJD when there are more SVs below $\sqrt{\epsilon_{mach}}$ than the MaxBasisSize (q = 35, s = 15).*

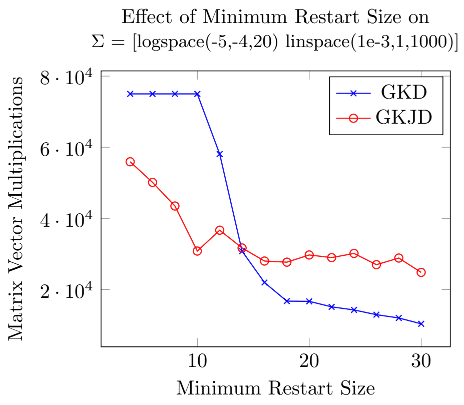


FIG. 3. *Convergence of GKJD on a problem with 20 SVs below $\sqrt{\epsilon_{mach}}$ in single precision with varying minimum restart sizes. (Maximum Matvecs = 75,000, q = 50)*

restarts for GKD and their associated error. As the errors caused by restarts grows above the relative tolerance within approximately 2,000 restarts (40,000 matvecs), GKD may have numerical issues and not converge although this behavior is sensitive to the choice of random orthonormal bases U and V. Since GKJD performs orders of magnitude fewer outer iterations, it is not affected by this source of error heavily and therefore is not sensitive to the random left and right singular spaces. With a marginally less strict tolerance, GKD does not exhibit this behavior.

In Figure 3 we consider an example where the matrix has 20 singular values below the $\|A\|\sqrt{\epsilon_{mach}}$ threshold. We use single precision arithmetic, which allows for relatively larger spectral gaps that make convergence tractable. We search for the smallest singular value with a maximum basis size of 50, the dynamic inner stopping criteria, and a tolerance of 1E-5 for all tests while varying the restart size used by the GKD and GKJD. We see that smaller restart sizes do not impede convergence of GKJD and only slow it down by less than a factor of two. However, the effects of a small restart size are much more severe on GKD, which is unable to converge to the desired tolerance within 75,000 matvecs for restart sizes less than 10. This shows that GKJD is able to rebuild the space lost during restarting much more quickly than GKD, as the inner equation can sufficiently filter out directions corresponding to the unwanted portions of the spectrum.

3. Benefits over PHSVDS.

3.1. Avoiding the Augmented Problem.

As mentioned earlier, methods on B often exhibit problems due to the interior nature of the spectrum that they work on. In order to demonstrate these issues, Figure 4 shows convergence on the problem A = diag([1e-10, 2e-10, 5e-10, 1e-9, 3e-9, 1e-8, 1e-6, 1e-4, 1:1000]). First, this problem is very poorly conditioned ($\kappa(A)$ = 1E13) and since the 6 smallest singular values are below 1E-8, the first stage of PHSVDS is unable to distinguish them from zero. Second, because the spectrum is reflected across 0 for the augmented problem, it is very difficult to converge only to the positive part of the spectrum.

In searching for 3 singular values to a user tolerance of 1E-14, PHSVDS took more than 4 times more matvecs, but more importantly, it missed 5 smaller singular

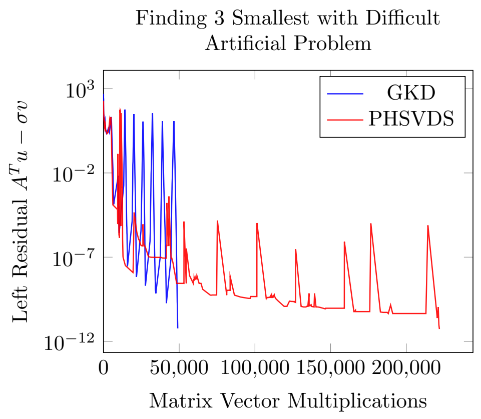


FIG. 4. *Convergence of PHSVDS on a poorly conditioned problem ($\kappa(A)$ = 1E+13)*

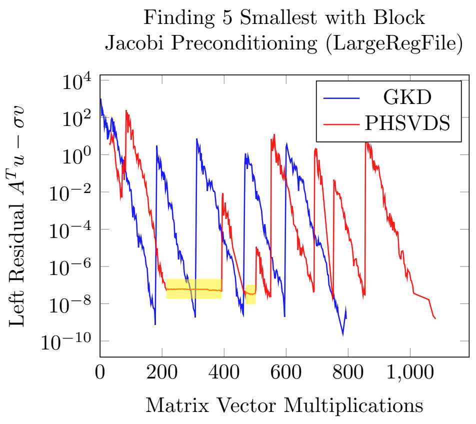


FIG. 5. *Stagnations caused by a failure to fully converge in the first stage of PHSVDS (κ =1.1E+4)*

values as the third converged value was 1e-4. Even worse, the vectors that were returned for left and right spaces were not orthogonal, as $\|Q^TQ - I\| \approx \|V^TV - I\| \approx$ 6E-5. Therefore, the true residuals after orthogonalization did not meet the full user tolerance. Comparatively, GKD converges to all 6 of the smallest singular values and did so with fully orthogonal left and right vectors. As we can see from the figure, the convergence for GKD is fairly smooth, converging to each of the six singular values below 1E-8 before finishing. This is a vast improvement over the second stage of PHSVDS, which exhibits irregular convergence with large spikes in the left residual and long stagnations.

3.2. Switching Problems. One of the biggest practical advantages of GKD over PHSVDS or any two stage algorithm is that it avoids the need to switch. For PHSVDS, choosing the right time to switch is crucial so as to give the best possible initial guesses to the second stage in order to avoid excessive use of the second stage on B. However, if an overly optimistic bound is used, it may cause stagnations in the first stage before switching. In general, it can be difficult to converge down to the theoretical limit for the first stage in practice, and determining the minimum constant above the theoretical limit that works for every problem is most likely impossible. Worse, preconditioning can increase this difficulty as it can cause errors that are difficult to account for within the switching criteria.

Specifically, we found these switching issues to occur when testing PHSVDS on LargeRegFile (another matrix from the SuiteSparse Collection [6]) with Block Jacobi preconditioning and δ =1E-12. It is clear from the highlighted portions of Figure 5 that PHSVDS is unable to meet the convergence criteria for the first stage. In fact, while the case shown in Figure 5 is able to reach the criteria eventually, most cases like this stagnate completely. For example, the same problem (LargeRegFile) when solved with an inner solver (JDQMR) is never able to meet the first stage convergence criteria. Since GKD never requires switching methods, we can avoid these problems entirely and provide more reliable convergence.

3.3. Space and Time Comparisons. For computations on large matrices, it is important to consider the convergence rate, the space requirements, and the total work that the algorithm requires. Therefore, we provide a short comparison of the latter between our method and PHSVDS before presenting numerical results in Section 4.

GKD requires storage for two spaces, V and Q that are $n \times q$ and $m \times q$ respectively where q is the maximum basis size. In the PRIMME implementation of PHSVDS, a similar same space is required to store the resulting left and right singular vector approximations. However, the first stage of PHSVDS requires a working memory set of two spaces of size $n \times q$, for V and $A^T AV$. Therefore, for square matrices, the working space required for the first stage of PHSVDS is equivalent to GKD. For very tall and skinny matrices ($n \ll m$), the first stage of PHSVDS uses a reduced memory footprint for most of the computation, but only if the user can guarantee that switching to the second stage will not be required. Otherwise, the second stage of PHSVDS will require two spaces of dimension $(m + n) \times q$. This corresponds to double the storage requirement of GKD. For very large problems, this might force the user to reduce the max basis size in order to store the bases in memory.

In terms of execution cost, GKD performs two orthogonalizations per iteration, one for V and one for Q, while the first stage of PHSVDS performs only one orthogonalization for V. Therefore, with low required accuracy where the second stage is not involved, PHSVDS is more efficient per step computationally. For robustness, primme_svds implements the second stage of PHSVDS using refined extraction which requires two orthogonalizations on vectors of dimension $m+n$ and thus has double the orthogonalization cost of GKD. Additionally, these vectors of size $m + n$ incur more error in dot product computations, so baseline calculations will not be as accurate. When using low precision calculations (single or half), these errors become even more important to avoid if possible.

4. Numerical Results.

To verify our algorithm's performance, we utilized the same matrices given in the original PHSVDS publication [31] as well as three matrices with dimension larger than one million from [30]. These matrices are publicly available through the SuiteSparse Matrix Collection [6] and represent real world applications. These problems are quite difficult for iterative solvers and are used to stress test the capabilities of GKD and PHSVDS. Since these matrices are sparse, we provide their dimensions and the number of non-zero entries of A, $nnz(A)$, as well as the norm of A, $\|A\|$, the condition number of A, $\kappa(A)$, and the gap ratio for σ_1, $\gamma_1 = (\sigma_2 - \sigma_1)/(\sigma_n - \sigma_2)$.

The matrices listed in Table 1 and Table 2 are listed from least to most difficult (left to right) as generally their condition numbers increase, and the gap ratios for their smallest singular values decrease. It should be noted that none of these matrices are particularly poorly conditioned, and do not require the second stage in PHSVDS to improve the singular vector estimates more than a few orders of magnitude. Therefore, the benefits we would expect to gain on very poorly conditioned problems are significantly larger.

We restrict GKD and PRIMME's PHSVDS Matlab interface, primme_svds, to a maximum basis size of 35 vectors, a minimum restart size of 15 vectors and a user tolerance of $\delta =$ 1E-14 for the smaller matrices and $\delta =$ 1E-12 for the larger ones. We also enforce one retained vector from the previous iteration (for +1 restarting) except for the three large cases, where we enforce +2 restarting. Additionally, we choose to soft lock converged triplets, but due to the interior nature of the augmented method in primme_svds, we are unable to set soft-locking for the second stage while searching for

Matrix	pde2961	dw2048	fidap4	jagmesh8	wang3	lshp3025
dimension	2961	2048	1601	1141	26064	3025
nnz(A)	14585	10114	31837	7465	77168	120833
$\kappa(A)$	9.5E+2	5.3E+3	5.2E+3	5.9E+4	1.1E+4	2.2E+5
$\|A\|$	1.0E+1	1.0E+0	1.6E+0	6.8E+0	2.7E-1	7.0E+0
γ_1	8.2E-3	2.6E-3	1.5E-3	1.7E-3	7.4E-5	1.8E-3

TABLE 1
Basic Properties of Square Matrices

Matrix	well1850	lp_ganges	deter4	plddb	ch	lp_bnl2
rows	1850	1309	3235	3049	3700	2324
columns	712	1706	9133	5069	8291	4486
nnz(A)	8755	6937	19231	10839	24102	14996
$\kappa(A)$	1.1E+2	2.1E+4	3.7E+2	1.2E+4	2.8E+3	7.8E+3
$\|A\|$	1.8E+0	4.0E+0	1.0E+1	1.4E+2	7.6E+2	2.1E+2
γ_1	3.0E-3	1.1E-1	1.1E-1	4.2E-3	1.6E-3	7.1E-3

TABLE 2
Basic Properties of Rectangular Matrices

Matrix	sls	Rucci1	LargeRegFile
rows	1,748,122	1,977,885	2,111,154
columns	62,729	109,900	801,374
nnz(A)	6,804,304	7,791,168	4,944,201
$\kappa(A)$	1.3E+3	6.7E+3	1.1E+4
$\|A\|$	1.3E+3	7.0E+0	3.1E+3
γ_1	8E-7	5E-5	3E-7

TABLE 3
Basic Properties of Large Scale Matrices

the smallest singular triplets. It should be noted that hard-locking generally improves performance for our method when searching for more than one singular value, but does not provide the same orthogonality guarantees and is subject to the numerical issues mentioned earlier.

4.1. Unpreconditioned Results. We compare GD+k (implemented as the default MIN_MATVECS method in primme_svds) against GKD, and the JDQMR method (MIN_TIME in primme_svds) against GKJD. As shown in Figure 6, GKD and GKJD require fewer matrix-vector multiplications than their primme_svds counterparts for all matrices. Also, the matrices that show the largest benefits are lshp3025, wang3, jagmesh8, and lp_ganges. As expected, these correspond to the matrices that required more significant use of the second stage in primme_svds, due to their larger $\kappa(A)$.

For most cases, we see a slight drop off in performance when searching for the 10 smallest singular values, but this is mostly caused by different implementations of soft-locking. Since primme_svds uses two stages, the first stage soft locks each vector at a tolerance above the user specified tolerance. However, since they are soft-locked, the first stage of primme_svds can improve the initial guesses to the second stage in some cases, since it leaves the estimated singular triplets in the basis while converging to other vectors. To check this, we ran GKD using a pseudo two-stage implementation that mimics the primme_svds behavior. This was done by converging to all 10 singular values to a higher tolerance first ($\kappa(A)\|A\|\epsilon_{mach}$), before converging to the full user tolerance. In this case, GKD can further improve performance for soft-locking over primme_svds.

For rectangular matrices, we also tested whether our method could find a true

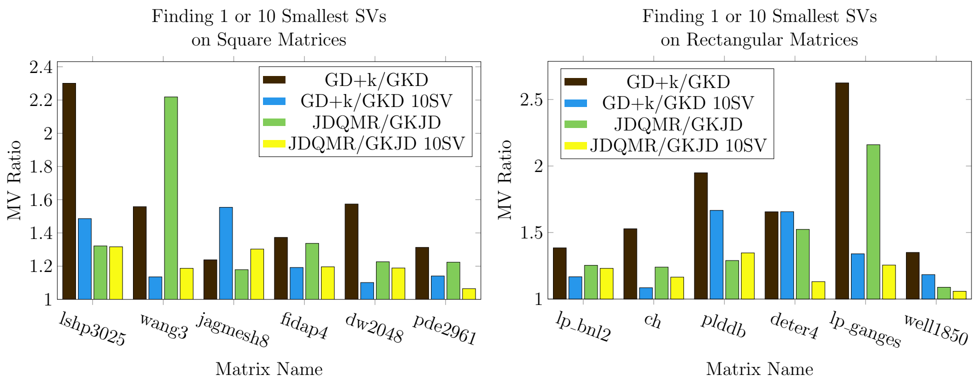
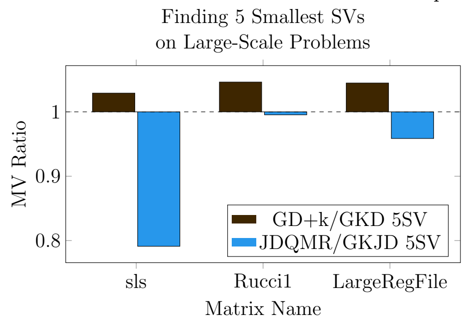


Fig. 6. *Unpreconditioned Results*



	sls	Rucci1	LargeRegFile
GKD	60298	112668	28766
GD+k	62050	117882	30056
GKJD	50859	138750	27652
JDQMR	40236	138118	26508

Fig. 7. *Large-Scale Unpreconditioned Results. Required matvecs for GKD, GD+k, GKJD and JDQMR are shown in the table. Note that for sls, GKJD finds 3 of the singular values with multiplicity 14 while JDQMR finds only 2.*

zero singular value by appending one extra column to the matrix equal to the first column. GKD is able to find the real zero in all cases. primme_svds will not return this numerically zero value, as outlined in its documentation, since its second stage has no way to distinguish real zeros from the null space created by the augmented matrix.

For the large scale matrices, Figure 6 shows a fairly even performance between primme_svds and GKD/GKJD. This is expected as the tolerance is higher (tol = 1E-12) than the small cases, and therefore primme_svds only uses the second stage sparingly. The biggest difference is seen for sls and for the inner-outer methods (JDQMR/GKJD), where the high multiplicity (14) at the second smallest singular value causes issues with convergence. Specifically, JDQMR only converges to two of these numerically equal singular values before finding five converged triplets, while GKJD is able to recognize the higher multiplicity and spends extra iterations finding a third. We also note that the number of matvecs for GKD/GKJD are significantly smaller than the numbers for SLEPc's implementation of LBD reported in [30].

In general, iterative methods may have trouble finding multiplicities or may converge out of order causing the methods to miss directions [19]. This is especially true for Krylov solvers which, in exact arithmetic, are unable to find more than one eigenvector corresponding to a multiplicity. In order to solve this problem, many algorithms including PHSVDS can utilize a block solver where the block size approximates the degree of the multiplicity [5, 3, 10]. Additionally, multiple initial guesses

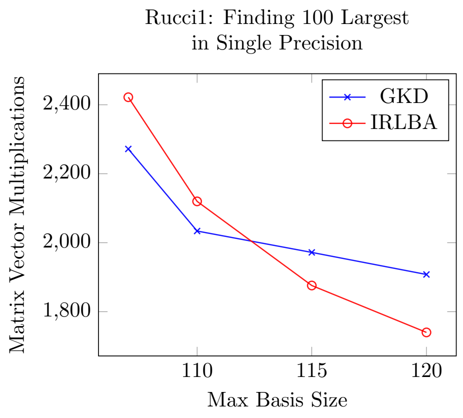


FIG. 8. *Similar performance can be achieved with a relatively small basis size even when searching for 100 values.*

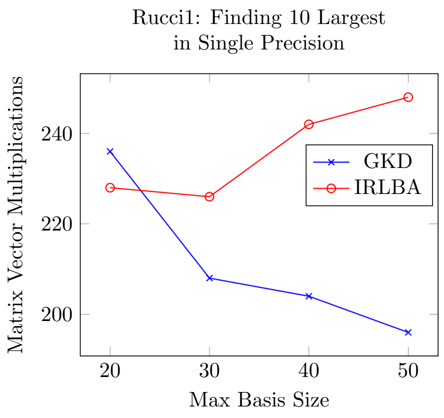


FIG. 9. *IRLBA wastes matrix vector multiplications building a full basis without checking convergence.*

can be used to reduce the likelihood of initial vectors being deficient in the invariant space of the multiplicity. Both of these ideas would be simple extensions that could be added to GKD to improve robustness.

4.2. Single Precision Results. In order to demonstrate the versatility of our method, we ran tests in single precision looking for the largest 10 or 100 singular values of matrices to tolerance δ = 1E-4. Although much less taxing on the solver, these kinds of requirements are common in many SVD applications. We compare our results to IRLBA (which is the default method in MATLAB's `svds` for largest singular values). Since we are looking for low accuracy, we omit results from PRIMME since it would use only the first stage which is equivalent to GKD.

Figures 8 and 9 report results on Rucci1. We also ran these tests on sls and LargeRegFile, but convergence was achieved in too few iterations (requiring only one restart) so all methods were similar. We vary the maximum basis size to understand how GKD compares when the user has more or less space than IRLBA uses as a default. When searching for 100 singular triplets, we choose basis sizes close to 100 to mimic the situation where space is at a premium and only a small number of extra vectors can be stored. For 10 singular triplets, we show how IRLBA compares to GKD when the basis size is much larger than the number for desired triplets.

Figure 8 shows that both IRLBA and GKD provide fairly similar results for 100 singular values. GKD performs better in the most extreme memory limitation as it can selectively target the desired values when building its space. However, when there is more room to build a Krylov space, this targeting is no longer required.

Figure 9 shows increased advantages of GKD when fewer singular values are needed. For 10 singular values, the standard version of IRLBA defaults to a maximum basis size of 30. In some cases, the system may have additional space for a larger basis size which can improve convergence. However, since IRLBA generally only checks convergence after a full basis is built, a larger basis size can limit how often IRLBA performs these checks. This allows GKD to outperform IRLBA, even though they obtain nearly identical performance for smaller basis sizes.

4.3. Preconditioned Results. We provide a preconditioner for the small matrices built using Matlab's ILU with the ilutp factorization, a drop-tolerance of 1E-3,

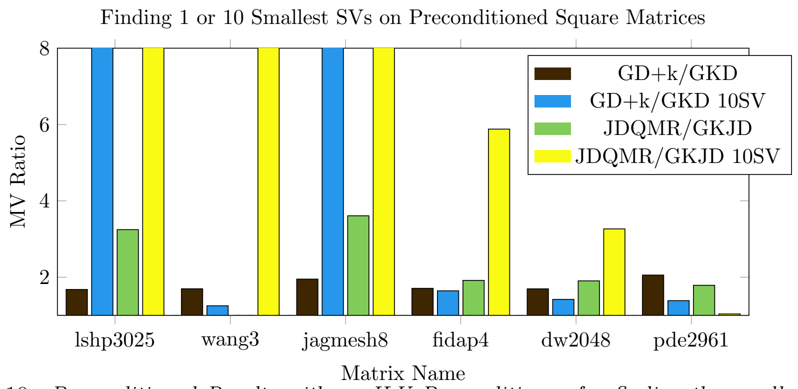
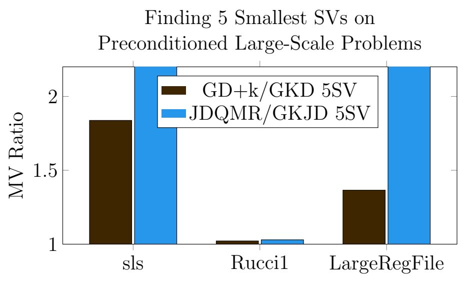


FIG. 10. *Preconditioned Results with an ILU Preconditioner for finding the smallest and 10 smallest singular triplets.*



	sls	Rucci1	LargeRegFile
GKD	6515	16074	810
GD+k	11972	16426	1106
GKJD	8204	18198	1266
JDQMR	DNF	18734	DNF

FIG. 11. *Large-Scale Results with Block Jacobi Preconditioner (block size=600 on A^TA) for the 5 smallest singular triplets. Required matvecs for GKD,GD+k, GKJD and JDQMR are shown in the table.*

and a pivot threshold of 1.0. Our results show the significant benefit of an effective preconditioner, as all of the small problems required less than 150 matvecs when searching for one singular value with GKD. However, these preconditioners sometimes caused significant issues for primme_svds, as it was unable to converge for lshp3025 when searching for the 10 smallest singular values, and exhibited significant difficulty converging to 10 singular values for wang3, jagmesh8 and fidap4. Specifically, when searching for 10 singular values, wang3 requires 12x more matvecs for JDQMR, and jagmesh8 requires 56x and 14x more matvecs for GD+k and JDQMR respectively. These issues are caused by primme_svds' switching issues mentioned earlier.

For the three large matrices, ILU becomes significantly more expensive, so we use a Block-Jacobi preconditioner, inverting exactly diagonal blocks of A^TA each of size 600. This is relatively inexpensive to compute and it is also parallelizable. Again, we see a significant decrease in matvecs as all three problems required less than 15% of the matvecs needed for the unpreconditioned cases. For Rucci1 the convergence differences between our methods and primme_svds are negligible, but for sls and LargeRegFile, GKD and GKJD provide significant improvements in speed and robustness. Again, as seen earlier in Figure 5, primme_svds' switching criteria are too stringent for preconditioned cases, which causes slowdowns for GD+k on LargeRegFile. Worse, primme_svds' JDQMR suffers stagnations that cause failures to converge when preconditioned on sls and LargeRegFile.

The 80% improvement on sls over GD+k comes from primme_svds being unable to separate the directions corresponding to the large degree multiplicity. During additional testing, we found the number of matvecs required to find the 5 smallest singular values with primme_svds is only marginally less than the number required to find 10. Since primme_svds is unable to appropriately separate the directions corresponding to the multiplicity, it converges to all 10 values concurrently. However, GKD is able to distinguish these directions and converge smoothly for each one individually, providing a substantial improvement. Testing GKD to converge to 10 values as well, we still found an improvement over primme_svds, however the gap between the two methods was significantly reduced.

5. Conclusions.

We have presented GKD, a new method for finding the smallest singular triplets of large sparse matrices to full accuracy. Our method works iteratively, under limited memory, with preconditioners, while including features such as soft-locking with orthogonality guarantees, +k restarting, and the ability to find real zero singular values in both square and rectangular matrices. Additionally, GKJD adds a Jacobi-Davidson inner solver for the A^TA correction equation into GKD, which can lower execution time when the matrix-vector multiplication operation is inexpensive and can reduce the errors caused by restarting. Both of these methods have shown to be more reliable and efficient than PHSVDS, and thus over other SVD methods, for nearly all cases.

REFERENCES

[1] I. Advanced Micro Devices, *AMD OpenCL optimisation guide*. https://developer.amd.com/amd-accelerated-parallel-processing-app-sdk/opencl-optimization-guide. Accessed 2018-02-14.

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

[3] J. Baglama, D. Calvetti, and L. Reichel, *IRBL: An implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems*, SIAM Journal on Scientific Computing, 24 (2003), pp. 1650–1677.

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

[5] J. Baglama and L. Reichel, *Restarted block Lanczos bidiagonalization methods*, Numerical Algorithms, 43 (2006), pp. 251–272.

[6] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.

[7] A. S. Gambhir, A. Stathopoulos, and K. Orginos, *Deflation as a Method of Variance Reduction for Estimating the Trace of a Matrix Inverse*, SIAM J. Sci. Comput., 39 (2017), pp. A532–A558, https://doi.org/10.1137/16M1066361.

[8] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2 (1965), pp. 205–224.

[9] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[10] R. G. Grimes, J. G. Lewis, and H. D. Simon, *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*, SIAM Journal on Matrix Analysis and Applications, 15 (1994), pp. 228–272.

[11] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in International Conference on Machine Learning, 2015, pp. 1737–1746.

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

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

[14] I. Jolliffe, *Principal component analysis*, Wiley Online Library, 2002.

[15] A. V. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM journal on scientific computing, 23 (2001), pp. 517–541.

[16] R. M. Larsen, *Lanczos bidiagonalization with partial reorthogonalization*, DAIMI Report Series, 27 (1998).

[17] Q. Liang and Q. Ye, *Computing singular values of large matrices with an inverse-free preconditioned Krylov subspace method*, Electronic Transactions on Numerical Analysis, 42 (2014), pp. 197–221.

[18] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, *NVIDIA tensor core programmability, performance & precision*, CoRR, abs/1803.04014 (2018), http://arxiv.org/abs/1803.04014, https://arxiv.org/abs/1803.04014.

[19] J. R. McCombs and A. Stathopoulos, *Iterative validation of eigensolvers: a scheme for improving the reliability of Hermitian eigenvalue solvers*, SIAM Journal on Scientific Computing, 28 (2006), pp. 2337–2358.

[20] K. Meerbergen and R. Morgan, *Inexact methods*, in Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[21] S. Osiński, J. Stefanowski, and D. Weiss, *Lingo: Search results clustering algorithm based on singular value decomposition*, in Intelligent information processing and web mining, Springer, 2004, pp. 359–368.

[22] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, 1980.

[23] H. Prasantha, H. Shashidhara, and K. B. Murthy, *Image compression using svd*, in Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on, vol. 3, IEEE, 2007, pp. 143–145.

[24] H. D. Simon and H. Zha, *Low-rank matrix approximation using the Lanczos bidiagonalization process with applications*, 21 (2000), pp. 2257–2274.

[25] A. Stathopoulos, *Locking issues for finding a large number of eigenvectors of Hermitian matrices*, tech. report, Tech Report WM-CS-2005-09, Computer Science, The College of William & Mary, 2005.

[26] A. Stathopoulos, *Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. part i: Seeking one eigenvalue*, SIAM J. Sci. Comput., 29 (2007), pp. 481–514.

[27] A. Stathopoulos and Y. Saad, *Restarting techniques for (Jacobi-)Davidson symmetric eigenvalue methods*, Electr. Trans. Numer. Anal., 7 (1998), pp. 163–181.

[28] E. Vecharynski, *Preconditioned Iterative Methods for Linear Systems, Eigenvalue and Singular Value Problems*, PhD thesis, University of Colorado at Denver, Denver, CO, USA, 2011. AAI3456056.

[29] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.

[30] L. Wu, E. Romero, and A. Stathopoulos, *PRIMME_SVDS: A high-performance preconditioned SVD solver for accurate large-scale computations*, arXiv preprint arXiv:1607.01404, (2016).

[31] L. Wu and A. Stathopoulos, *A preconditioned hybrid SVD method for accurately computing singular triplets of large matrices*, SIAM Journal on Scientific Computing, 37 (2015), pp. S365–S388.

[32] P. Zhang and Y. Gao, *Matrix multiplication on high-density multi-GPU architectures: theoretical and experimental investigations*, in International Conference on High Performance Computing, Springer, 2015, pp. 17–30.