Electronic Transactions on Numerical Analysis.
Volume 42, pp. 197-221, 2014.
Copyright © 2014, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

COMPUTING SINGULAR VALUES OF LARGE MATRICES WITH AN INVERSE-FREE PRECONDITIONED KRYLOV SUBSPACE METHOD[*]

QIAO LIANG[†] AND QIANG YE[†]

Dedicated to Lothar Reichel on the occasion of his 60th birthday

Abstract. We present an efficient algorithm for computing a few extreme singular values of a large sparse $m\times n$ matrix C. Our algorithm is based on reformulating the singular value problem as an eigenvalue problem for C^TC. To address the clustering of the singular values, we develop an inverse-free preconditioned Krylov subspace method to accelerate convergence. We consider preconditioning that is based on robust incomplete factorizations, and we discuss various implementation issues. Extensive numerical tests are presented to demonstrate efficiency and robustness of the new algorithm.

Key words. singular values, inverse-free preconditioned Krylov Subspace Method, preconditioning, incomplete QR factorization, robust incomplete factorization

AMS subject classifications. 65F15, 65F08

1. Introduction.

Consider the problem of computing a few of the extreme (i.e., largest or smallest) singular values and the corresponding singular vectors of an $m \times n$ real matrix C. For notational convenience, we assume that $m \geq n$ as otherwise we can consider C^T. In addition, most of the discussions here are valid for the case $m < n$ as well with some notational modifications. Let $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_n$ be the singular values of C. Then nearly all existing numerical methods are based on reformulating the singular value problem as one of the following two symmetric eigenvalue problems:

$$\sigma_1^2 \leq \sigma_2^2 \leq \cdots \leq \sigma_n^2 \quad \text{are the eigenvalues of } C^TC \tag{1.1}$$

or

$$-\sigma_n \leq \cdots \leq -\sigma_2 \leq -\sigma_1 \leq \underbrace{0 = \cdots = 0}_{m-n} \leq \sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_n$$

are the eigenvalues of the *augmented matrix*

$$M := \begin{bmatrix} 0 & C \\ C^T & 0 \end{bmatrix}. \tag{1.2}$$

Namely, a singular value of C can be obtained by computing the corresponding eigenvalue of either $A := C^TC$ or M.

Computing a few selected eigenvalues of a large symmetric matrix is a subject that has been well studied in the last few decades; see [4, 37] for surveys. To compute a few extreme eigenvalues of a large symmetric matrix, the standard method of choice is the Lanczos algorithm [13, p. 304] or the implicitly restarted Lanczos method [39] (ARPACK [30]). Their speed of convergence depends on how well the desired eigenvalues are separated from the rest of the spectrum. When the (relative) separation is small or the desired eigenvalues lie in the interior of the spectrum, a shift-and-invert spectral transformation is usually used to

[*]Received October 17, 2013. Accepted October 28, 2014. Published online on December 10, 2014. Recommended by Yousef Saad. The first author is supported in part by NSF under Grant DMS-1317424. The second author is supported in part by NSF under Grant DMS-1317424 and DMS-1318633.

[†]Department of Mathematics, University of Kentucky, Lexington, KY 40506, USA
({qiao.liang, qye3}@uky.edu).

accelerate the convergence; see [13, 14]. This requires inverting or factorizing a shifted matrix. For sparse matrices, a factorization may create excessive fill-ins of the zero entries, which results in significant memory and operation costs. When the factorization of the shifted matrix is inefficient or infeasible, several methods have been developed that employ either inexact shift-and-invert or some *preconditioning* transformations. The Jacobi-Davidson method [38], the JDCG algorithm [34], the locally preconditioned conjugate gradient method (LOBPCG) [26, 27], and the inverse-free preconditioned Krylov subspace method [20, 32] are some of the methods in this class. There is a large body of literature on various aspects of the large symmetric eigenvalue problem; see [1, 2, 4, 15, 19, 33, 37, 40, 45] and the references therein for further discussions.

To compute a few extreme singular values of C, we can apply the Lanczos algorithm or the implicitly restarted Lanczos algorithm to one of the two formulations (1.1) and (1.2), and this can often be done implicitly. Indeed, several methods have been introduced that exploit the special structure and the associated properties of these eigenvalue problems. The Lanczos bidiagonalization method introduced in [17] is a widely used method for the singular value problems that implicitly applies the Lanczos method to the formulation (1.1). A robust implementation called `lansvd` is provided in PROPACK [29]. The implicit restart strategy has been developed for the Lanczos bidiagonalization algorithm in [3] and [28], which also include the robust MATLAB implementations `irlba` and `irlanb`, respectively. Other aspects of the Lanczos bidiagonalization algorithm are discussed in [10, 24, 42]. These methods based on the Lanczos algorithm for the eigenvalue problem (1.1) work well when the corresponding eigenvalue is reasonably well separated. However, their convergence may be slow if the eigenvalue is clustered, which turns out to be often the case when computing the smallest singular values through (1.1). Specifically, for the formulation (1.1), the spectral separation for σ_1^2 as an eigenvalue of C^TC may be much smaller than the separation of σ_1 from σ_2 since

$$\frac{\sigma_2^2-\sigma_1^2}{\sigma_n^2-\sigma_2^2}=\frac{\sigma_2-\sigma_1}{\sigma_n-\sigma_2}\frac{\sigma_1+\sigma_2}{\sigma_n+\sigma_2}\ll\frac{\sigma_2-\sigma_1}{\sigma_n-\sigma_2}$$

(assuming $\sigma_2 \ll \sigma_n$). On the other hand, for the formulation (1.2), σ_1 is an interior eigenvalue of M, for which a direct application of the Lanczos algorithm does not usually result in convergence.

The shift-and-invert transformation is a standard method to deal with clustering or to compute interior eigenvalues. For example, to compute a few of the smallest singular values, MATLAB's routine `svds` applies ARPACK [30, 39] to the augmented matrix M (1.2) with a shift-and-invert transformation. This works well for square matrices. However, for computing the smallest singular value of a non-square matrix, a subtle difficulty arises in using the shift-and-invert transformation for M because M is singular, and with a shift close to 0, the method often converges to one of the $m-n$ zero eigenvalues of M rather than to σ_1. On the other hand, one can avoid the shift-and-invert procedure by considering the Jacobi-Davidson method for the augmented matrix (1.2), and a method of this type, called JDSVD, has been developed in [21, 22] that efficiently exploits the block structure of (1.2). The JDSVD method replaces the shift-and-invert step by approximately solving so-called correction equations using a preconditioned iterative method. When computing σ_1 as an interior eigenvalue of the augmented matrix (1.2), convergence of JDSVD appears to strongly depend on the quality of the preconditioner for the correction equation. This demands a good preconditioner for M or $M-\mu I$, which is unfortunately difficult to construct when $m \neq n$ owing to the singularity of M.

It appears that the augmented matrix formulation (1.2) has some intrinsic difficulties when it is used to compute a few of the smallest singular values of a non-square matrix because of the existence of the zero eigenvalues of M. For this reason, we propose to reconsider formulation (1.1) in this situation. While (1.1) has the advantage of a smaller dimension in the underlying eigenvalue problem, a clear disadvantage is that there is no efficient method to carry out the shift-and-invert transformation $(C^TC - \mu I)^{-1}$ other than explicitly forming C^TC. Note that C^TC is typically much denser than C and explicitly computing C^TC may result in a loss of accuracy with the condition number being squared. In the case of $\mu = 0$, which can be used to compute a singular value σ_1 that is sufficiently close to 0, the inverse of C^TC can be implicitly obtained by computing the QR factorization of C. This is the approach taken in `lansvd` of PROPACK [29]. However, since a complete QR factorization of a sparse matrix may be expensive owing to possible excessive fill-ins of the zero entries, it is interesting to study other approaches that use incomplete factorizations instead. Other drawbacks of (1.1) include the need to compute left singular vectors when they are required and the potential loss of accuracy caused by computing σ_1^2 when σ_1 is tiny; see Section 3. In particular, the computed left singular vectors may have low accuracy if the singular values are small; see the discussions in Section 3.

In this paper, we propose to address the small separation of σ_1^2 in the formulation (1.1) by considering a preconditioned Krylov subspace method. Specifically, we shall implicitly apply the inverse-free preconditioned Krylov subspace method of [20] (or its block version [36]) to $A = C^TC$. As already discussed, the standard shift-and-invert transformation is not practical for (1.1) as it requires a factorization of $C^TC - \mu I$. The inverse-free preconditioned Krylov subspace method is an effective way to avoid the shift-and-invert transformation for computing a few extreme eigenvalues of the symmetric generalized eigenvalue problem $Ax = \lambda Bx$, where A and B are symmetric with B positive definite. In this method, an approximate eigenvector x_k is iteratively improved through the Rayleigh-Ritz projection onto the Krylov subspace

$$\mathcal{K}_m(H_k, x_k) := \operatorname{span}\left\{x_k, H_kx_k, H_k^2x_k, \ldots, H_k^mx_k\right\},$$

where $H_k := A - \rho_kB$ and ρ_k is the Rayleigh quotient of x_k. The projection is carried out by constructing a basis for the Krylov subspace through an inner iteration, where the matrices A and B are only used to form matrix-vector products. The method is proved to converge at least linearly, and the rate of convergence is determined by the spectral gap of the smallest eigenvalue of H_k (rather than the original eigenvalue problem as in the Lanczos method). An important implication of this property is that a congruence transformation of (A, B) derived from an incomplete LDL^T factorization of a shifted matrix $A - \mu B$ may be applied to reduce the spectral gap of the smallest eigenvalue of H_k and hence, to accelerate the convergence to the extreme eigenvalue. This is referred to as preconditioning. A block version of this algorithm has also been developed in [36] to address multiple or severely clustered eigenvalues.

In applying the inverse-free preconditioned Krylov subspace method [20, 36] to the matrix $A = C^TC$, we directly construct the projection of C rather than the projection of C^TC used for the eigenvalue problem. In this way, we compute approximations of σ_1 directly from the singular values of the projection of C rather than using the theoretically equivalent process of computing approximations of σ_1^2 from the projection of C^TC. By computing σ_1 directly, we avoid the pitfall of a loss of accuracy associated with computing σ_1^2 if σ_1 is tiny. On the other hand, the potential difficulty with the accuracy of the computed left singular vector in this case is intrinsic to the approach of C^TC. An efficient implementation of the inverse-free preconditioned Krylov subspace method depends on the construction of a preconditioner derived from an incomplete LDL^T factorization of $C^TC - \mu I$. Constructing a preconditioner for C^TC has been discussed extensively in the literature in the context of solving least squares problems (see [5, 8, 9, 16, 31, 35, 43]), and one method well suited for our problem is the robust incomplete factorization (RIF) of [8, 9]. For the shifted matrix $C^TC - \mu I$, however, there is no known effective method for computing a factorization without forming C^TC first. It turns out that the robust incomplete factorization (RIF) can be easily adapted to construct an LDL^T factorization of the shifted matrix $C^TC - \mu I$ without forming C^TC. Our numerical results demonstrate that the RIF preconditioner in combination with the inverse-free preconditioned Krylov subspace method leads to a very efficient preconditioned algorithm for the singular value problem. Numerical tests also exhibit that this method is particularly competitive for computing a few of the smallest singular values of non-square matrices.

The paper is organized as follows. Section 2 develops the inverse-free preconditioned Krylov subspace method for the singular value problem. Section 3 presents the robust incomplete factorization (RIF) preconditioner for $C^TC - \mu I$. Section 4 briefly describes a MATLAB implementation called `svdifp` that we have developed. Section 5 presents some numerical examples comparing our method with several existing programs for the singular value problem. We conclude in Section 6 with some remarks. We consider real matrices throughout, but all results can be generalized to complex matrices in a trivial way.

Throughout, $\|\cdot\|$ denotes the 2-norm for both vectors and matrices. $\|\cdot\|_1$ denotes the 1-norm. $\|\cdot\|_{\max}$ denotes the max norm, i.e., the largest entry of the matrix in absolute values. $\langle x,y\rangle := x^Ty$ is the Euclidean inner product, and for a symmetric positive definite matrix A, $\langle x,y\rangle_A := x^TAy$ is the A-inner product.

2. The inverse-free preconditioned Krylov subspace method.

We compute the singular values of C by computing the eigenvalues of $A = C^TC$. To address the problem of slow convergence caused by the reduced spectral gap of σ_1^2 in the Lanczos algorithm, we apply the inverse-free preconditioned Krylov subspace projection method of [20], whose speed of convergence can be accelerated by preconditioning using some incomplete factorizations. We first describe this basic method for the eigenvalue problem in Section 2.1. We then develop a corresponding algorithm for the singular value problem in Section 2.2.

2.1. The generalized eigenvalue problem.

Consider the problem of computing the smallest eigenvalue of the generalized eigenvalue problem for (A, B), i.e., $Ax = \lambda Bx$. Note that we need to discuss the generalized eigenvalue problem here even though the singular value problem will be formulated as a standard eigenvalue problem for C^TC because our preconditioning scheme will actually transform it to an equivalent generalized eigenvalue problem.

In an iterative process, assume that x_k is an approximate eigenvector at step k. We construct a new approximation x_{k+1} by the Rayleigh-Ritz projection of (A, B) onto the Krylov subspace

$$\mathcal{K}_m(A - \rho_k B, x_k) = \text{span}\left\{x_k, (A-\rho_k B)x_k, \ldots, (A-\rho_k B)^m x_k\right\},$$

where

$$\rho_k = \rho(x_k) := \frac{x_k^T A x_k}{x_k^T B x_k}$$

is the Rayleigh quotient and m is a parameter to be chosen. Specifically, let Z_m be the matrix consisting of the basis vectors of $\mathcal{K}_m(A - \rho_k B, x_k)$. We then form the matrices $A_m = Z_m^T(A - \rho_k B)Z_m$ and $B_m = Z_m^T B Z_m$ and find the smallest eigenvalue μ_1 and a corresponding eigenvector h for (A_m, B_m). Then the new approximate eigenvector is

$$x_{k+1} = Z_m h, \tag{2.1}$$

and, correspondingly, the Rayleigh quotient

$$\rho_{k+1} = \rho_k + \mu_1 \tag{2.2}$$

is a new approximate eigenvalue. This is the basic process of the inverse-free Krylov subspace method; see [20] or Algorithm 2.2 below for a formal description. The construction of the basis vectors Z_m for $K_m(A - \rho_k B, x_k)$ is accomplished using either the Lanczos method or the Arnoldi method with the B-inner product; see [20] for a more detailed discussion.

It is shown in [20, Theorem 3.2] that ρ_k always converges to an eigenvalue and x_k converges into the direction of an eigenvector. Furthermore, the following theorem characterizes the speed of convergence.

THEOREM 2.1 ([20, Theorems 3.2 and 3.4]). *Let A and B be symmetric with B positive definite, and let $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of (A, B). Let (ρ_k, x_k) be the approximate eigenpair at step k of the inverse-free Krylov subspace method defined by (2.1) and (2.2), and assume that $\lambda_1 < \rho_0 < \lambda_2$. Then ρ_k converges to λ_1. Furthermore, if $\mu_1 < \mu_2 \leq \cdots \leq \mu_n$ are the eigenvalues of $A - \rho_k B$, then*

$$\rho_{k+1} - \lambda_1 \leq (\rho_k - \lambda_1)\epsilon_m^2 + 2(\rho_k - \lambda_1)^{3/2}\epsilon_m \left(\frac{\|B\|}{\sigma_2}\right)^{\frac{1}{2}} + \mathcal{O}\left((\rho_k - \lambda_1)^2\right), \tag{2.3}$$

where

$$\epsilon_m = \min_{p \in \mathcal{P}_m, p(\mu_1)=1} \max_{i \neq 1} |p(\mu_i)|$$

and $\mathcal{P}_m$ denotes the set of all polynomials of degree not greater than m.

This theorem demonstrates that ρ_k converges at least with the rate of ϵ_m^2, which is determined by the spectral distribution of $A - \rho_k B$. It is also shown in [20, Corollary 3.5] that, asymptotically, the eigenvalues of $A - \rho_k B$ in this bound can be replaced by those of $A - \lambda_1 B$ to simplify it. Namely, letting $0 = \gamma_1 < \gamma_2 \leq \cdots \leq \gamma_n$ be the eigenvalues of $A - \lambda_1 B$, we have

$$\frac{\rho_{k+1} - \lambda_1}{\rho_k - \lambda_1} \leq 4\left(\frac{1-\sqrt{\psi}}{1+\sqrt{\psi}}\right)^{2m} + 4\left(\frac{1-\sqrt{\psi}}{1+\sqrt{\psi}}\right)^{m}\left(\frac{\|B\|}{\sigma_2}\right)^{\frac{1}{2}}(\rho_k - \lambda_1)^{\frac{1}{2}} + \mathcal{O}(\rho_k - \lambda_1), \tag{2.4}$$

where

$$\psi := \frac{\gamma_2 - \gamma_1}{\gamma_n - \gamma_1} = \frac{\gamma_2}{\gamma_n}.$$

By (2.4) (or Theorem 2.1), convergence of the inverse-free Krylov subspace method can be accelerated by increasing the relative gap between γ_1 and γ_2 (or μ_1 and μ_2). This can be achieved by a congruent equivalent transformation, which is called preconditioning. Specifically, we can compute an LDL^T factorization of $A - \lambda_1 B$ that is scaled such that

$$L^{-1}(A - \lambda_1 B)L^{-T} = D = \text{diag}(1, \ldots, 1, 0). \tag{2.5}$$

We then consider the preconditioned problem

$$(\hat{A}, \hat{B}) := (L^{-1}AL^{-T}, L^{-1}BL^{-T}), \tag{2.6}$$

which has exactly the same eigenvalues as the pencil (A, B). However, applying our algorithm to $(\hat{A}, \hat{B})$, the speed of convergence depends on the spectral gap of

$$\hat{A} - \lambda_1 \hat{B} = L^{-1}(A - \lambda_1 B)L^{-T} = D,$$

which has exactly two eigenvalues, namely $\gamma_1 = 0$ and $\gamma_2 = \cdots = \gamma_n = 1$. Then $\psi = 1$, and $\rho_{k+1} - \lambda_1 = \mathcal{O}\left((\rho_k - \lambda_1)^2\right)$, which implies quadratic convergence of $\rho^{(k)}$. In general, one may use a step-dependent preconditioner by computing the factorization $A - \rho_k B = L_k D_k L_k^T$. Then, using (2.3), we also obtain quadratic convergence of $\rho^{(k)}$; see [20] for details.

The preconditioning strategies discussed above are ideal situations where the full LDL^T factorization is computed. A practical way of implementing this is to compute an incomplete factorization of $A - \mu B$ as an approximation, where μ is an approximation of λ_1. With such a matrix L, we may expect the eigenvalues of $\hat{A} - \lambda_1 \hat{B} = L^{-1}(A - \lambda_1 B)L^{-T}$ to be clustered around two points. Then $\psi \approx 1$, which results in accelerated convergence by (2.4); see [36] for an analysis. We can also construct a preconditioner L_k from an incomplete LDL^T factorization of $A - \rho_k B$ at each step to reduce ϵ_m and hence to accelerate convergence. This is a more costly approach, but it can be used to update a preconditioner when it appears ineffective.

As in the preconditioned conjugate gradient method for linear systems, the preconditioning transformation (2.6) can be carried out implicitly. Indeed, all we need is to construct, at the iteration k, a basis for the transformed Krylov subspace

$$L^{-T}\hat{\mathcal{K}}_m(\hat{H}_k, L^T x_k) = \mathcal{K}_m(L^{-T}L^{-1}H_k, x_k),$$

where $H_k = A - \rho_k B$ and $\hat{H}_k = \hat{A} - \rho_k \hat{B} = L^{-1}H_k L^{-T}$. This is achieved by using matrix-vector multiplications with $L^{-T}L^{-1}H_k$, and the only operation involving the preconditioning transformation is $L^{-T}L^{-1}$; see [20] for details. For completeness, we state the following preconditioned algorithm from [20, Algorithm 4] (with a possibly step-dependent preconditioner L as discussed above and a construction of a B-orthonormal basis of the Krylov subspace).

ALGORITHM 2.2. *Inverse free preconditioned Krylov subspace method for* (A, B).

```
1   Input $m$ and an initial approximate eigenvector $x_0$ with $\|x_0\| = 1$;
2   $\rho_0 = \rho(x_0)$;
3   For $k = 0, 1, 2, \ldots$ until convergence
4       construct a preconditioner $L$;
5       construct a $B$-orthonormal basis $\{z_0, z_1, \ldots, z_m\}$ for
            $\mathcal{K}_m(L^{-T}L^{-1}(A - \rho_k B), x_k)$;
6       form $A_m = Z_m^T(A - \rho_k B)Z_m$, where $Z_m = [z_0, z_1, \ldots, z_m]$;
7       find the smallest eigenvalue $\mu_1$ and an eigenvector $h$ of $A_m$;
8       $\rho_{k+1} = \rho_k + \mu_1$ and $x_{k+1} = Z_m h$.
9   End
```

The above algorithm computes the smallest eigenvalue only. To find additional eigenvalues, we need to use a deflation procedure. A natural deflation process discussed in [32] is to shift the eigenvalues that have been computed and then apply the algorithm. Specifically, assume that (λ_i, v_i) (for $1 \le i \le \ell$) are ℓ eigenpairs that have been computed, and let $V_\ell = [v_1, \ldots, v_\ell]$ satisfy $V_\ell^T B V_\ell = I$. Then $AV_\ell = BV_\ell \Lambda_\ell$, where $\Lambda_\ell = \mathrm{diag}\{\lambda_1, \ldots, \lambda_\ell\}$. If $\lambda_{\ell+1} \le \lambda_{\ell+2} \le \ldots \le \lambda_n$ are the remaining eigenvalues of (A, B), then $\lambda_{\ell+1}$ is the smallest eigenvalue of

$$(A_\ell, B) := (A + (BV_\ell)\Sigma(BV_\ell)^T, B), \tag{2.7}$$

where $\Sigma = \mathrm{diag}\{\alpha - \lambda_i\}$ and α is any value chosen to be greater than $\lambda_{\ell+2}$. Therefore $\lambda_{\ell+1}$ can be computed by applying the inverse-free preconditioned Krylov subspace algorithm to (A_ℓ, B). For the singular value problem for C to be discussed in the next section, we apply the inverse-free preconditioned Krylov subspace to $A = C^TC$ implicitly by applying the projection on C. However, the deflation (2.7) changes the structure to $C^TC + (BV_\ell)\Sigma(BV_\ell)^T$, for which an implicit projection is difficult to construct. In this setting, it is more desirable to work with (A, B) directly.

An alternative approach is to project the Krylov subspace to the B-orthogonal complement of $\mathcal{V}_\ell := \operatorname{span}\{v_1, \ldots, v_\ell\}$. Namely, we apply projections directly on (A, B) but replace the Krylov subspace $\mathcal{K}_m(A-\rho_k B, x_k)$ or in the preconditioned algorithm, the subspace $\mathcal{K}_m(L^{-T}L^{-1}(A - \rho_k B), x_k)$, respectively, by the projected subspace

$$\mathcal{K}_m((I - V_\ell V_\ell^T B)(A - \rho_k B), x_k) \text{ or } \mathcal{K}_m((I - V_\ell V_\ell^T B)L^{-T}L^{-1}(A - \rho_k B), x_k). \tag{2.8}$$

This enforces that all approximate eigenvectors obtained are in the B-orthogonal complement of $\mathcal{V}_\ell$ and hence their convergence to an eigenvector corresponding to one out of the eigenvalues $\{\lambda_{\ell+1}, \ldots, \lambda_n\}$ provided the iteration converges. This deflation approach has the advantage of not changing the matrix $A = C^TC$ for the singular value problem. However, its convergence property is not understood as the existing theory (Theorem 2.1) is not readily applicable to the setting of projected Krylov subspaces. However, our numerical experiments show that this deflation strategy works as intended.

2.2. The singular value problem for C. We consider the singular value problem for an $m \times n$ matrix C. We apply Algorithm 2.2 to the eigenvalue problem $A = C^TC$ and $B = I$. However, a direct application involves computing the eigenvalue ρ_k of the projection problem involving A_m, which converges to σ_1^2. One potential difficulty associated with this approach is that ρ_k computed in this way may have a larger error if σ_1 is very small (relative to $\|C\|$). Specifically, if $\tilde{\rho}_k$ is the computed Ritz value, it follows from the standard backward error analysis [18] that $\tilde{\rho}_k$ is the exact eigenvalue of a perturbed matrix $A_m + E_m$ with $\|E_m\| = \mathcal{O}(\mathbf{u})\|A_m\|$, where $\mathbf{u}$ is the machine precision. Then

$$|\tilde{\rho}_k - \rho_k| \le \mathcal{O}(\mathbf{u})\|A_m\| \le \mathcal{O}(\mathbf{u})\|A\| = \mathcal{O}(\mathbf{u})\|C\|^2$$

and

$$|\sqrt{\tilde{\rho}_k} - \sqrt{\rho_k}| \le \mathcal{O}(\mathbf{u})\|C\| \frac{\|C\|}{\sqrt{\tilde{\rho}_k} + \sqrt{\rho_k}} \approx \mathcal{O}(\mathbf{u})\|C\|\kappa(C)/2\,,$$

where $\kappa(C) = \sigma_n/\sigma_1$ is the condition number of C. In particular, the relative error

$$\frac{|\sqrt{\tilde{\rho}_k} - \sqrt{\rho_k}|}{\sqrt{\rho_k}} \le \mathcal{O}(\mathbf{u}) \frac{\|C\|^2}{\sqrt{\rho_k}(\sqrt{\tilde{\rho}_k} + \sqrt{\rho_k})} \approx \mathcal{O}(\mathbf{u})\kappa(C)^2/2$$

is proportional to $\kappa(C)^2$. Thus, very little relative accuracy may be expected if $\kappa(C)$ is of order $1/\sqrt{\mathbf{u}}$. In contrast, a backward stable algorithm should produce an approximation of σ_1 with absolute error of the order of $\mathcal{O}(\mathbf{u})\|C\|$ and the relative error of the order of $\mathcal{O}(\mathbf{u})\kappa(C)$. We note that the above discussion is based on a worst case upper bound. It is likely pessimistic particularly in the bound of $\|A_m\|$, but it does highlight the potential loss of accuracy when one approximates σ_1 through computing σ_1^2; see Example 5.1 in Section 5.

To achieve the desired backward stability, we propose to construct a two-sided projection of C, from which we compute the approximate singular values directly. This is similar to the Lanczos bidiagonalization algorithm, where a bidiagonal projection matrix is constructed whose singular values directly approximate the singular values of C. Algorithmically, we construct an orthonormal basis $\{z_0, z_1, \ldots, z_m\}$ for $\mathcal{K}_m(L^{-T}L^{-1}(A - \rho_k I), x_k)$ and simultaneously an orthonormal basis $\{y_0, y_1, \ldots, y_m\}$ for $\operatorname{span}\{Cz_0, Cz_1, \ldots, Cz_m\}$ as follows.

First, $f_{0,0} = \|Cz_0\|_2$, and $y_0 = Cz_0/f_{0,0}$. Then, for $i = 1, \ldots, m$, we generate z_i and y_i by

$$\begin{aligned} f_{i,i}z_i &= L^{-T}L^{-1}(C^TCz_{i-1} - \rho_k z_{i-1}) - f_{0,i}z_0 - f_{1,i}z_1 - \cdots - f_{i-1,i}z_{i-1}, \\ g_{i,i}y_i &= Cz_i - g_{0,i}y_0 - g_{1,i}y_1 - \cdots - g_{i-1,i}y_{i-1}, \end{aligned} \tag{2.9}$$

where $f_{j,i} = z_j^T L^{-T}L^{-1}(C^TCz_{i-1} - \rho_k z_{i-1})$, $g_{j,i} = y_j^T Cz_i$, and $f_{i,i}$ and $g_{i,i}$ are chosen so that $\|y_i\| = \|z_i\| = 1$. Assuming $\dim(\mathcal{K}_m(L^{-T}L^{-1}(A - \rho_k I), x_k)) = m + 1$, the recurrence for z_i does not breakdown, and the process leads to an orthonormal basis $\{z_0, z_1, \ldots, z_m\}$. It is easy to show that the recurrence for y_i does not breakdown either and $\{y_0, y_1, \ldots, y_m\}$ is orthonormal. Let $Y_m = [y_0, y_1, \ldots, y_m]$. Then $CZ_m = Y_m G_m$, where $G_m = [g_{ij}]_{i,j=0}^m$. It follows that $Z_m^T(C^TC)Z_m = G_m^T G_m$. If $\sigma_k^{(1)}$ is the smallest singular value of G_m, then $(\sigma_k^{(1)})^2$ is the smallest eigenvalue of $A_m = Z_m^T(C^TC)Z_m$, i.e., the so constructed value $(\sigma_k^{(1)})^2$ is equal to ρ_{k+1} in Algorithm 2.2.

By computing $\sigma_k^{(1)}$ directly, we avoid a possible loss of accuracy. Specifically, if $\tilde{\sigma}_k^{(1)}$ is the computed singular value of G_m using the standard SVD algorithm such as `svd`, then it follows from backward stability that $\tilde{\sigma}_k^{(1)}$ is the exact singular value of $G_m + F_m$ for some F_m with $\|F_m\| = \mathcal{O}(\mathbf{u})\|G_m\|$. Then

$$|\tilde{\sigma}_k^{(1)} - \sigma_k^{(1)}| \le \mathcal{O}(\mathbf{u})\|G_m\| \le \mathcal{O}(\mathbf{u})\|C\|,$$

and hence,

$$\frac{|\tilde{\sigma}_k^{(1)} - \sigma_k^{(1)}|}{\sigma_k^{(1)}} \le \mathcal{O}(\mathbf{u})\kappa(C).$$

Thus, as Algorithm 2.2 converges, i.e., $\sqrt{\rho_k} = \sigma_k^{(1)} \to \sigma_1$, the approximate singular value $\tilde{\sigma}_k^{(1)}$ can approximate σ_1 with a relative accuracy of the order of $\mathcal{O}(\mathbf{u})\kappa(C)$.

To compute additional eigenvalues, we use a deflation based on the projected Krylov subspace (2.8) and compute the direct projection of C in the same way. We summarize this process in the following algorithm, Algorithm 2.3, that computes the $(\ell + 1)$st smallest singular value when the first ℓ singular values have already been computed.

ALGORITHM 2.3. *Inverse free preconditioned Krylov subspace method for SVD.*

```
1  Input: $m$; $V_\ell = [v_1, \ldots, v_\ell]$ with $C^TCv_i = \sigma_i^2 v_i$ and $V_\ell^T V_\ell = I$;
          initial right singular vector $x_0$ such that $\|x_0\| = 1$ and $V_\ell^T x_0 = 0$;
2  initialize: $\rho_0 = \|Cx_0\|$; $G_m = [g_{ij}] = 0 \in \mathbb{R}^{(m+1)\times(m+1)}$;
3  For $k = 0, 1, 2, \ldots$ until convergence
4          construct a preconditioner $L$;
5          $z_0 = x_k$; $w = Cz_0$; $m' = m$;
6          $g_{0,0} = \|w\|$ and $y_0 = w/g_{0,0}$;
7          For $i = 1 : m$
8              $z_i = (I - V_\ell V_\ell^T)L^{-T}L^{-1}(C^Tw - \rho_k z_{i-1})$
9              For $j = 0 : i - 1$
10                 $z_i = z_i - (z_j^T z_i)z_j$;
11             End
12             If $\|z_i\| \ne 0$, $z_i = z_i/\|z_i\|$; otherwise, $m' = i$ and break;
13             $w = Cz_i$; $y_i = w$;
14             For $j = 0 : i - 1$
15                 $g_{j,i} = y_j^T y_i$ and $y_i = y_i - g_{j,i}y_j$;
```

```
16            End
17            $g_{i,i} = \|y_i\|$ and $y_i = y_i/g_{i,i}$;
18        End
19        compute the smallest singular value $\sigma_{k+1}^{(1)}$ of $G_m = [g_{ij}]_{i,j=0}^{m'}$,
              and a corresponding unit right singular vector $h$;
20        $\rho_{k+1} = (\sigma_{k+1}^{(1)})^2$, $x_{k+1} = [z_0, z_1, \ldots, z_{m'}]h$;
21  End
```

We make some remarks concerning Algorithm 2.3. The presented algorithm has deflation included, where ℓ singular values and right singular vectors are given as inputs. When none is given, it computes the smallest singular value σ_1 by setting $\ell = 0$ and V_ℓ to the empty matrix. At line 4, a preconditioner needs to be constructed such that $LDL^T \approx C^TC - \mu I$ for μ equal to ρ_k or a fixed initial value. An algorithm based on RIF to compute an incomplete factor L implicitly from C will be discussed in the next section. As stated, different preconditioners may be used for different iteration steps, but for efficiency reasons, we usually use the same preconditioner. Line 9 implements the deflation and preconditioning techniques implicitly. The *for* loop at lines 7–18 constructs an orthonormal basis $\{z_0, z_1, \ldots, z_{m'}\}$ for the Krylov subspace and simultaneously an orthonormal basis $\{y_0, y_1, \ldots, y_{m'}\}$ such that $CZ_{m'} = Y_{m'}G_{m'}$, where $m' = \dim(\mathcal{K}_m((I - V_\ell V_\ell^T)L^{-T}L^{-1}(A - \rho_k B), x_k)) - 1$. Then $G_{m'} = Y_{m'}^T CZ_{m'}$. Its smallest singular value and a corresponding right singular vector h are computed to construct a new approximate right singular vector at lines 19–20.

The process is theoretically equivalent to Algorithm 2.2 as applied to $A = C^TC$ and $B = I$. When no preconditioning is used, i.e., $L = I$, the inverse-free Krylov subspace method is simply the Lanczos method for A with a restart after m iterations. When the preconditioning is used, we effectively transform the standard eigenvalue problem for C^TC to the equivalent generalized eigenvalue problem for $(\hat{A}, \hat{B}) = (L^{-1}C^TCL^{-T}, L^{-1}L^{-T})$, to which the inverse-free Krylov subspace method is applied.

In the `eigifp` implementation [32] of the inverse-free preconditioned Krylov subspace method for the eigenvalue problem, an LOBPCG-type (locally optimal preconditioned conjugate gradient [25, 27]) subspace enhancement was also included to further accelerate convergence. Note that, in the LOBPCG method, the steepest descent method is modified by adding the previous approximate eigenvector x_{k-1} to the space spanned by the current approximation and its residual $\mathrm{span}\{x_k, (A - \rho_k B)x_k\}$ to construct a new approximate eigenvector. It results in a conjugate gradient-like algorithm that has a significant speedup in convergence over the steepest descent method. This idea has also been used in `eigifp` [32] by adding the previous approximate eigenvector x_{k-1} to the Krylov subspace $\mathcal{K}_m(A - \rho_k B, x_k)$, which is also found to accelerate convergence in many problems. As the extra cost of adding this vector is quite moderate, we also use this subspace enhancement in our implementation for the singular value problem. Algorithmically, we just need to add after the *for* loop at lines 7–18 the construction of an additional basis vector $z_{m'+1}$ by orthogonalizing $x_k - x_{k-1}$ against $\{z_0, z_1, \ldots, z_{m'}\}$. Note that we have used $x_k - x_{k-1}$ rather than x_{k-1} for orthogonalization because orthogonalizing x_{k-1} against $z_0 = x_k$ typically leads to a cancellation when $x_k \approx x_{k-1}$. To avoid possible cancellations, we implicitly compute the orthogonalization of $x_k - x_{k-1}$ against $z_0 = x_k$ through

$$d = \tilde{Z}_{m'} \begin{bmatrix} -\frac{\hat{h}^T\hat{h}}{h_1} \\ \hat{h} \end{bmatrix}, \quad \text{where} \quad h = \begin{bmatrix} h_1 \\ \hat{h} \end{bmatrix} \in \begin{bmatrix} \mathbb{R} \\ \mathbb{R}^{m\times 1} \end{bmatrix}$$

is the unit right singular vector of the projection matrix G_m, and $\tilde{Z}_{m'}$ is the matrix of the basis vectors at line 19 of the previous step (step $k-1$) of Algorithm 2.3, i.e., $x_k = \tilde{Z}_{m'}h$

and $x_{k-1} = \tilde{Z}_{m'} e_1$, where $e_1 = [1, 0, \dots, 0]^T$. It is easy to check that

$$d = \tilde{Z}_{m'} h - \frac{1}{h_1} \tilde{Z}_{m'} e_1 = x_k - \frac{1}{h_1} x_{k-1} = \frac{1}{h_1}(x_k - x_{k-1}) - \frac{h_1 - 1}{h_1} x_k$$

and $x_k^T d = 0$. Therefore, a new basis vector that extends the subspace with $x_k - x_{k-1}$ can be obtained by orthogonalizing d against $\{z_1, \dots, z_{m'}\}$ as

$$f_{m'+1,m'+1} z_{m'+1} = d - f_{1,m'+1} z_1 - \cdots - f_{m',m'+1} z_{m'}.$$

Moreover, $Cd = CZ_{m'}[-\frac{\hat{h}^T \hat{h}}{h_1}, \hat{h}]^T$, and then $Cz_{m'+1}$ can be computed without the explicit multiplication by C from

$$Cz_{m'+1} = \frac{Cd - f_{1,m'+1} Cz_1 - \cdots - f_{m',m'+1} Cz_{m'}}{f_{m'+1,m'+1}},$$

from which $y_{m'+1}$ and an additional column of G are computed as in (2.9). However, with possible cancellations in the last formula, $Cz_{m'+1}$ may be computed with large errors and we suggest to compute $Cz_{m'+1}$ explicitly when high accuracy is needed.

The algorithm we have presented computes approximate singular values and simultaneously the corresponding right singular vectors only. In applications where singular triplets are required, we can compute approximate left singular vectors from the right singular vectors obtained. This is a limitation of the $C^T C$ formulation (1.1) and Algorithm 2.3, where we reduce the eigenvalue residual r_p of the approximate singular value and right singular vector pair $(\sigma_k^{(1)}, x_k)$ (with $\|x_k\| = 1$),

$$r_p := \|C^T C x_k - (\sigma_k^{(1)})^2 x_k\|.$$

From this residual, the errors of the approximate singular value $\sigma_k^{(1)}$ and approximate right singular vector x_k can be bounded as (see [13, p. 205])

$$|\sigma_k^{(1)} - \sigma_1| \le \frac{r_p^2}{(\sigma_k^{(1)} + \sigma_1)\text{gap}} \quad \text{and} \quad \sin \angle(x_k, v_1) \le \frac{r_p}{\text{gap}},$$

where we assume that σ_1 is the singular value closest to $\sigma_k^{(1)}$, v_1 is a corresponding right singular vector, and $\text{gap} = \min_{i \ne 1} |\sigma_k^{(1)} - \sigma_i|$. When a corresponding left singular vector is needed, it can be obtained as

$$w_k = C \frac{x_k}{\sigma_k^{(1)}} \tag{2.10}$$

provided that $\sigma_k^{(1)} \ne 0$. Then the accuracy of the approximate singular triplet $(\sigma_k^{(1)}, w_k, x_k)$ can be assessed by

$$r_t := \left\| \begin{bmatrix} Cx_k - \sigma_k^{(1)} w_k \\ C^T w_k - \sigma_k^{(1)} x_k \end{bmatrix} \right\| = \left\| M \begin{bmatrix} w_k \\ x_k \end{bmatrix} - \sigma_k^{(1)} \begin{bmatrix} w_k \\ x_k \end{bmatrix} \right\|.$$

It is easily checked that

$$r_t = \frac{r_p}{\sigma_k^{(1)}}. \tag{2.11}$$

Therefore, for a tiny singular value, a small residual r_p for the pair $(\sigma_k^{(1)}, x_k)$ does not imply a small residual r_t for the singular triplet $(\sigma_k^{(1)}, w_k, x_k)$. Indeed, the constructed left singular vector w_k may not be a good approximation even if x_k is a good approximate right singular vector as indicated by r_p. This appears to be an intrinsic difficulty of the C^TC formulation (1.1). Specifically, in the extreme case of $\sigma_1 = 0$, a corresponding left singular vector is any vector in the orthogonal complement of $\mathcal{R}(C)$ (the range space of C), and it can not be obtained from multiplying a right singular vector by C or any vector in the subspace $C\mathcal{K}_m(C^TC, x_k) = \mathcal{K}_m(CC^T, Cx_k) \subset \mathcal{R}(C)$. In this case, we need to consider CC^T on a new random initial vector to compute a left singular vector.

An alternative formulation for the left singular vector is obtained by calculating the left singular vector g of the projection matrix G_m at line 19 of Algorithm 2.3 and then forming

$$w_k = [y_0, y_1, \dots, y_{m'}]g \,. \tag{2.12}$$

It is easy to check that this is theoretically equivalent to (2.10) if $\sigma_k^{(1)} \neq 0$. However, an advantage of this formulation is that it is still defined even when $\sigma_k^{(1)} = 0$ although the quality of the approximation is not assured. Our numerical experiments indicate that (2.12) is in general similar to (2.10) but may lead to a slightly better left singular vectors in some problems. In our implementation, we use (2.11) to estimate the residual of singular triplets to avoid the cost of computing it, but at termination, we use (2.12) to compute a left singular vector and then recompute its residual.

In the algorithm, we have used r_p for the convergence test unless a left singular vector is required. When this is indeed needed, r_t is used for the convergence test. As discussed above, however, r_t may never converge to 0 if $\sigma_k^{(1)}$ is extremely small. Therefore, in order to properly terminate the iteration in such a situation, we propose to monitor the magnitude of the computed singular value, and when an extremely small singular value (i.e., of the order of machine precision) is detected, the stopping criterion should be switched to using r_p. By properly terminating the iteration according to r_p, we can still obtain a sufficiently good approximate singular value and a right singular vector. After that, we can then separately apply the algorithm to C^T to compute a left singular vector.

In the following algorithm, we present an extension of Algorithm 2.3 by the subspace enhancement steps and termination criteria discussed above. It also includes the additional step needed in computing the left singular vector when it is required.

ALGORITHM 2.4. *Inverse free preconditioned Krylov subspace method for SVD with LOBPCG enhancement.*

```
1–17   See lines 1–17 of Algorithm 2.3.
18     $z_{m'+1} = d_k; w = Cd_k;$
19     For $j = 0 : m'$
20         $z_{m'+1} = z_{m'+1} - (z_j^T z_{m'+1})z_j;$
21         $w = w - (z_j^T z_{m'+1})Cz_j;$
22     End
23     If $\|z_{m'+1}\| \neq 0$
24         $z_{m'+1} = z_{m'+1}/\|z_{m'+1}\|, w = w/\|z_{m'+1}\|; y_{m'+1} = w;$
25         For $j = 0 : m'$
26             $g_{j,m'+1} = y_j^T y_{m'+1}$ and $y_{m'+1} = y_{m'+1} - g_{j,m'+1}y_j;$
27         End
28         $g_{m'+1,m'+1} = \|y_{m'+1}\|$ and $y_{m'+1} = y_{m'+1}/g_{m'+1,m'+1};$
29         $m' = m' + 1;$
30     End
```

```
31        compute the smallest singular value $\sigma^{(1)}_{k+1}$ of $G_m = [g_{ij}]_{i,j=0}^{m'}$,
              and a corresponding unit right singular vector $h$;
32        $d_{k+1} = Z_{m'}(h - e_1/h_1); Cd_{k+1} = CZ_{m'}(h - e_1/h_1)$;
33        $\rho_{k+1} = (\sigma^{(1)}_{k+1})^2$, $x_{k+1} = Z_{m'}h$;
34        $res = \|C^T Cx_{k+1} - \rho_{k+1}x_{k+1}\|$;
35        If *singular triplet* is desired and $\sigma^{(1)}_{k+1} > \mathbf{u}\|C\|^2$
36            $res = res/\sigma^{(1)}_{k+1}$;
37        End
38        test convergence using $res$;
39  End
40  If *singular triplet* is desired
41        compute a left singular vector $g$ of $G_m$ and $w_{k+1} = [y_0, y_1, \ldots, y_{m'}]g$;
42  End
43  Output: $(\sigma^{(1)}_{k+1}, x_{k+1})$ or, if *singular triplet* is required, $(\sigma^{(1)}_{k+1}, w_{k+1}, x_{k+1})$.
```

In Algorithm 2.4, lines 1–17 are identical to Algorithm 2.3. In lines 18–30, the subspace is expanded with $x_k - x_{k-1}$ using the method mentioned earlier. In line 32 the orthogonalization of $x_{k+1} - x_k$ against x_{k+1} is computed to be used in the next iteration. The algorithm, by default, computes the singular value and the right singular vectors. If singular triplets are desired, in lines 35–37 an appropriate residual is computed to be used for testing convergence. This is only for the purpose of terminating the iteration. At convergence, however, we compute the left singular vector w_{k+1} and the residual of the singular triplets explicitly.

Finally, we mention that the algorithm can be adapted trivially to compute the largest singular value. Namely, to compute the largest singular values of C, we just need to modify line 19 in Algorithm 2.3 and line 31 in Algorithm 2.4 to calculate the largest singular value of G_m and a corresponding right singular vector, and the rest of the algorithm remains the same. It is easy to see that the convergence theory of [20] extends to this case. We also note that the above algorithm is based on a vector iteration for computing a single singular value. A block matrix iteration version of the inverse-free preconditioned Krylov subspace method has been developed in [36] to compute multiple eigenvalues or extremely clustered eigenvalues. It can be adapted as in Algorithm 2.3 to the task of computing multiple or extremely clustered singular values. Here, we omit a formal statement of the algorithm; see [36].

3. Preconditioning by robust incomplete factorizations (RIF).

In this section, we discuss how to construct a preconditioner L, i.e., an approximate LDL^T factorization $C^T C - \mu I = LDL^T$, where $\sqrt{\mu}$ is an approximation of the singular value to be computed and D is a diagonal matrix of 0 or ± 1. This generally requires forming the matrix $C^T C - \mu I$, which may be much denser than C and hence leads to a denser L. In addition, forming the matrix is associated with a potential loss of information in very ill-conditioned cases although this appears not to pose a problem when only an approximate factorization is sought [23].

For computing the smallest singular value, $\mu = 0$ is a natural first choice for the shift. In this case, we need an incomplete factorization of a symmetric positive semidefinite matrix, for which numerous techniques have been developed; see [6] for a survey. Indeed, if $\mu = 0$, the problem is the same as constructing a preconditioner for the linear least squares problem. One method that has been well studied is the incomplete QR factorization; see [5, 16, 31, 35, 43]. The incomplete QR factorization methods, such as the incomplete modified Gram-Schmidt method or the incomplete Givens rotation method, can be used here to construct a preconditioner for computing the smallest singular values that are close to 0. While these methods are effective and often result in a much faster convergence, they tend to have high intermediate storage requirements in our experiences; see [9] as well. Moreover, they can not deal with the case $\mu \neq 0$. On the other hand, Benzi and Tuma propose a method for constructing a preconditioner for C^TC in [9] called robust incomplete factorization (RIF). This method can be easily adapted to the computation of an incomplete LDL^T factorization for $C^TC - \mu I$ and is found to have more moderate fill-ins. We discuss now the RIF preconditioner for the SVD algorithm.

Let $A \in \mathbb{R}^{n\times n}$ be a sparse symmetric positive definite matrix. The idea of RIF is to obtain the factorization $A = L^TDL$ by applying an A-orthogonalization process to the unit basis vectors $e_1, e_2, \ldots, e_n$ (i.e., $I = [e_1, e_2, \ldots, e_n]$). It will become a Gram-Schmidt process for the unit basis vectors with respect to the inner product $\langle x, y\rangle_A := x^TAy$, i.e., for $i = 1, 2, \ldots, n$,

$$z_i = e_i - \sum_{j=1}^{i-1} \frac{\langle e_i, z_j\rangle_A}{\langle z_j, z_j\rangle_A} z_j\,. \tag{3.1}$$

This is the classical Gram-Schmidt (CGS) process. The corresponding modified Gram-Schmidt (MGS) process can be implemented by updating the basis vector z_i initialized as $z_i = e_i$ ($1 \le i \le n$) by the following nested loop: for $j = 1, 2, \ldots, n$, orthogonalize each z_i (for $i = j+1, \ldots, n$) against z_j by

$$z_i = z_i - \frac{\langle z_i, z_j\rangle_A}{\langle z_j, z_j\rangle_A} z_j. \tag{3.2}$$

This updating process allows discarding z_j to free the memory once it is orthogonalized against all z_i (for $i = j+1, \ldots, n$). Let

$$l_{ij} = \frac{\langle z_i, z_j\rangle_A}{\langle z_j, z_j\rangle_A} \quad \text{if } i \ge j,$$

and set $l_{ij} = 0$ if $i < j$. Then $L = [l_{ij}]$ is a unit lower triangular matrix, and this process results in an A-orthogonal matrix $Z = [z_1, z_2, \ldots, z_n]$ such that $I = ZL^T$. Then $Z^TAZ = D$ implies $A = LDL^T$, where $D = \text{diag}[d_1, d_2, \ldots, d_n]$ and $d_j = \langle z_j, z_j\rangle_A$.

Clearly, by (3.1), $z_i \in \text{span}\{e_1, e_2, \ldots, e_i\}$, and Z is upper triangular. Since CGS (3.1) and MGS (3.2) are theoretically equivalent, (3.2) can be formulated as

$$z_i = z_i - l_{ij}z_j, \quad \text{with } l_{ij} = \frac{\langle e_i, z_j\rangle_A}{\langle z_j, z_j\rangle_A}\,,$$

which is computationally more efficient (see [7]) for a problem like $A = C^TC$. In addition, as A is sparse, $\langle e_i, z_j\rangle_A = e_i^TAz_j$ may be structurally zero for many i, j resulting in a sparse matrix L. The A-orthogonalization process can efficiently exploit the property $l_{ij} = 0$ by skipping the corresponding orthogonalization step. Furthermore, one may also drop the entry l_{ij} and skip the orthogonalization if l_{ij} is sufficiently small. This would result in an incomplete factorization called robust incomplete factorization (RIF).

RIF has also been used in [9] to efficiently construct preconditioners for C^TC for a full rank matrix $C \in \mathbb{R}^{m\times n}$ arising from the normal equation for the least squares problem. An advantage of RIF for C^TC is that the C^TC-orthogonalization process can be carried out using C only as

$$z_i = z_i - l_{ij}z_j, \quad \text{with } l_{ij} = \frac{\langle Cz_i, Cz_j\rangle}{\langle Cz_j, Cz_j\rangle}, \tag{3.3}$$

for $j = 1, 2, \dots, n$ and $i = j + 1, \dots, n$, where $\langle \cdot, \cdot \rangle$ is the Euclidean inner product. In this setting, the following CGS formulation of l_{ij}

$$z_i = z_i - l_{ij} z_j, \quad \text{with } l_{ij} = \frac{\langle Ce_i, Cz_j \rangle}{\langle Cz_j, Cz_j \rangle}$$

is preferred over the MGS formulation because of the need to compute Cz_i in MGS (3.3) each time z_i is updated, whereas only Ce_i (the i-th column of C) is needed in CGS. Since we are only interested in an incomplete factorization by applying a dropping threshold for z_i and l_{ij}, the difference in stability between CGS and MGS is not significant. Also, the computation of l_{ij} requires forming Cz_j once for each z_j, which involves sparse-sparse matrix-vector multiplications and can be efficiently computed as a linear combination of a few columns of C; see [9]. We also observe that the inner products in l_{ij} involve two sparse vectors as well.

If we multiply both sides of (3.3) by C, it is possible to get around the computation of $w_i := Cz_i$ as a matrix-vector multiplication in MGS (3.3) by computing it through the updating formula

$$w_i = w_i - l_{ij} w_j, \quad \text{with } l_{ij} = \frac{\langle w_i, w_j \rangle}{\langle w_j, w_j \rangle}, \tag{3.4}$$

which maintains the MGS form. However, since the matrix L is all we need, it is not necessary in this formula to compute z_i anymore. Indeed, since w_i is initialized as Ce_i, (3.4) is just the modified Gram-Schmidt process in the Euclidean inner product applied to the columns of C, and it becomes the MGS method for the QR factorization of C. However, with w_i initialized as Ce_i and z_i initialized as e_i, the generated sequence w_i is expected to be much denser than the corresponding z_i, which appears to be the case in our experiments. This may be the main motivation of using the A-orthogonalization in RIF.

We observe that the same process can be extended to our problem of constructing an LDL^T factorization for $A := C^TC - \mu I$ with a shift $\mu \approx \sigma_1^2$. The corresponding orthogonalization process is

$$z_i = z_i - l_{ij} z_j, \quad \text{with } l_{ij} = \frac{\langle Ce_i, Cz_j \rangle - \mu \langle e_i, z_j \rangle}{\langle Cz_j, Cz_j \rangle - \mu \langle z_j, z_j \rangle},$$

for $j = 1, 2, \dots, n$ and $i = j+1, \dots, n$. Now, if $\mu < \sigma_1^2$, then $C^TC - \mu I$ is positive definite, and with the divisor in l_{ij} being nonzero, the process is well defined.

If $\mu = \sigma_1^2$, then $C^TC - \mu I$ is positive semidefinite and the process may encounter a zero division if $\langle Cz_j, Cz_j \rangle - \mu \langle z_j, z_j \rangle = 0$ for some j. However, in this case, $(C^TC - \mu I)z_j = 0$, and then $\langle Cz_i, Cz_j \rangle - \mu \langle z_i, z_j \rangle = 0$ for any i. Then we do not need to carry out the orthogonalization against z_j. Continuing the process, we still obtain $z_1, z_2, \dots, z_n$ such that $\langle Cz_j, Cz_i \rangle - \mu \langle z_j, z_i \rangle = 0$ but $Z^TAZ = D$ will have zeros in the diagonal. However, this does not cause any problem as we still have $C^TC - \mu I = LDL^T$, and by using a scaled L, we have D with 0 and 1 as diagonal elements. This is precisely the factorization needed; see (2.5).

If $\mu > \sigma_1^2$, then $C^TC - \mu I$ is indefinite and the process may breakdown with the occurrence of $\langle Cz_j, Cz_j \rangle - \mu \langle z_j, z_j \rangle = 0$ but $(C^TC - \mu I)z_j \neq 0$ for some j. In practice, the exact breakdown is unlikely, but we may encounter a near breakdown $\langle Cz_j, Cz_j \rangle - \mu \langle z_j, z_j \rangle \approx 0$, which may cause an instability in the process. However, since we are only interested in an incomplete factorization which incur a perturbation through dropping small elements, we propose to modify the pivot by simply setting $\langle Cz_j, Cz_j \rangle - \mu \langle z_j, z_j \rangle$ to some nonzero scalar such as the dropping threshold and skip the orthogonalization against z_j. This perturbation is consistent with the dropping strategy in the incomplete factorization and would amount to a perturbation to z_j of the order of magnitude of the dropping threshold. In any case, it only affects the quality of the preconditioner and hence efficiency of the overall algorithm, but it does not reduce the accuracy of the singular value computed by our method. In our experiences, the algorithm handles modest indefiniteness very well, but the quality of the preconditioner deteriorates as the matrix indefiniteness increases.

The incomplete LDL^T factorization provided by RIF needs to be scaled so that D has diagonals equal to $0, \pm 1$ for its use as a preconditioner for the singular value problem. This can be achieved by multiplying L by $D^{1/2}$ on the right. The following is the RIF algorithm as adapted from [9] with the columns of L scaled.

ALGORITHM 3.1. *Robust Incomplete Factorization of* $C^TC - \mu I$.

```
1   Input: $\eta_1$ (drop threshold for $L$) and $\eta_2$ (drop threshold for $Z$);
2   initialization: $Z = [z_1, z_2, \ldots, z_n] = I$; $L = [l_{ij}] = I \in \mathbb{R}^{n\times n}$;
3   For $j = 1$ to $n$
4       $d_j = \langle Cz_j, Cz_j\rangle - \mu\langle z_j, z_j\rangle$;
5       $l_{jj} = \sqrt{|d_j|}$;
6       If $l_{jj} > \max\{\eta_1\|Ce_j\|_1, \mathbf{u}\}$
7           For $i = j+1$ to $n$
8               $p_{ij} = \langle Cz_j, Ce_i\rangle - \mu\langle z_j, e_i\rangle$;
9               If $|p_{ij}|/l_{jj} \geq \max\{\eta_1\|Ce_j\|_1, \mathbf{u}\}$
10                  $z_i = z_i - \frac{p_{ij}}{d_j} z_j$ and $l_{ij} = \mathrm{sgn}(p_{jj}) \cdot p_{ij}/l_{jj}$;
11                  If $|z_i(\ell)| < \eta_2\|z_i\|_1$ for any $\ell$, set $z_i(\ell) = 0$;
12              End
13          End
14      Else
15          $l_{jj} = \max\{\eta_1\|Ce_j\|, \mathbf{u}\}$
16      End
17  End
```

We present some remarks concerning Algorithm 3.1. At line 6, we test the divisor l_{jj} for near-breakdown. If a near-breakdown occurs, we set l_{jj} to the breakdown threshold $\max\{\eta_1\|Ce_j\|_1, \mathbf{u}\}$ at line 15 and skip the orthogonalization process. Here, we note that the threshold is chosen to be relative to the norm of Ce_j as Cz_j is constructed from it through orthogonalization and $\mathbf{u}$ is added to the definition of the threshold to deal with the possible situation of $Ce_j = 0$. We skip the orthogonalization of z_i if l_{ij} is below the given threshold $\max\{\eta_1\|Ce_j\|_1, \mathbf{u}\}$. In that case, l_{ij} is set to 0. To further improve the efficiency of the algorithm, we also apply a dropping rule to z_i at line 11 by setting all entries of z_i that are below the threshold $\eta_2\|z_i\|_1$ to 0. This will maintain Z as sparse as possible and improve the efficiency of the algorithm. In our experiments, the quality of the constructed preconditioner appears to depend more on the magnitude of η_2 than that of η_1. So η_2 is chosen to be much smaller than η_1. In our implementation, we set $\eta_1 = 10^{-3}$ and $\eta_2 = 10^{-8}$ as the default values. Finally, on output, the algorithm produces an approximate factorization $C^TC - \mu I \approx LDL^T$ with D having only $0, \pm 1$ as diagonal elements.

4. Robust implementation. One advantage of the inverse-free preconditioned Krylov subspace method is its simplicity of the implementation with the number of inner iterations being the only parameter to select. We have implemented Algorithm 2.3 in combination with the RIF preconditioner (Algorithm 3.1) in a black-box MATLAB implementation for the singular value problem. The program called `svdifp` is used in our numerical tests.

ETNA
Kent State University
http://etna.math.kent.edu

Our program svdifp is based on the MATLAB program eigifp [32], which implements the inverse-free preconditioned Krylov subspace method with several algorithmic enhancements for the generalized eigenvalue problem. We have incorporated many features of eigifp into our implementation, but the core iteration involves the construction of the projection of C as outlined in Algorithm 2.3. Noting that for Algorithm 2.3, the only required user input is m (the inner iteration) and a preconditioner, we have adopted the same strategy used in eigifp in determining m; see [32]. Namely, m can be either specified by the user or, by default, adaptively determined by the program according to its effect on the rate of convergence. Note that experiments have shown that an optimal value of m is larger if the problem is more difficult, while it is smaller if the problem is easier (e.g., with a good preconditioner). On the other hand, to determine a preconditioner, we first need an approximate singular value as a shift for the RIF preconditioner. Here different strategies will be used depending on whether computing the largest or the smallest singular values is the goal.

For computing the smallest singular value, we assume 0 is a good initial approximate singular value, and, using 0 as the shift, we compute a preconditioner by Algorithm 3.1 and carry out a preconditioned iteration.

For computing the largest singular value, the standard Lanczos bidiagonalization algorithm [17] should work well because the spectral separation is typically doubled through the C^TC formulation (1.1), i.e.

$$\frac{\sigma_n^2-\sigma_{n-1}^2}{\sigma_{n-1}^2-\sigma_1^2}=\frac{\sigma_n-\sigma_{n-1}}{\sigma_{n-1}-\sigma_1}\frac{\sigma_n+\sigma_{n-1}}{\sigma_{n-1}+\sigma_1}\approx 2\frac{\sigma_n-\sigma_{n-1}}{\sigma_{n-1}-\sigma_1}.$$

However, for problems with clustered largest singular values, the preconditioning approach can still be very beneficial. One difficulty then is that there is no good approximate singular value readily available initially, and no preconditioner can be derived. Following the strategy in eigifp [32], we start the iteration with no preconditioning, and when a sufficiently good approximate singular value σ has been found as determined by the residual, we compute a preconditioner for $C^TC-\mu I$ by Algorithm 3.1 with the shift $\mu=\sigma^2+r_p$ and then continue the iteration with preconditioning, where r_p is the residual and hence μ is an upper bound for the true singular value. This typically leads to accelerated convergence.

In both cases, the program monitors the approximate singular value obtained and the convergence rate and may update the preconditioner using an updated approximate singular value as the shift if a significant deviation of the singular value from the shift is detected. The same strategy is followed when computing several singular values with deflation. The program can be run with no required user input. However, it also allows various optional parameters, which the user may supply to improve performance. These include the inner iteration m, the RIF thresholds, an initial approximate singular value (which can be used to compute a preconditioner), or a preconditioner itself, among others.

5. Numerical examples.

In this section, we present some numerical examples to demonstrate the capability and efficiency of the preconditioned inverse-free Krylov subspace method for the singular value problem. We compare our MATLAB implementation svdifp with several existing programs (i.e., irlba of Baglama and Reichel [3], jdsvd of Hochstenbach [21, 22], lansvd of Larson [29], and svds of MATLAB, which is based on ARPACK [30] of Lehoucq, Sorenson, and Yang). The program irlba [3] implements an augmented implicitly restarted Lanczos bidiagonalization algorithm. The program jdsvd [21, 22] implements a Jacobi-Davidson method on the augmented matrix formulation. (Note that a program based on the Jacobi-Davidson method for C^TC has also been developed recently [23].) The code lansvd [29] implements the Lanczos bidiagonalization algorithm for R^{-1} from the QR factorization of $C = QR$ for computing the smallest singular value. The MATLAB program `svds` implements ARPACK [30] and uses the inverse of M (or $M - \mu I$) in the formulation (1.2) for computing the smallest singular value. We note that `svdifp` and `jdsvd` compute one singular value at a time, while `irlba`, `lansvd`, and `svds` can compute several singular values simultaneously. On the other hand, `svdifp` and `jdsvd` can use preconditioners to accelerate convergence, while `irlba`, `lansvd`, and `svds` have to use the shift-and-invert approach.

In the first three examples, we test the programs on computing the smallest singular value, while in the fourth example we demonstrate the capability of `svdifp` in computing several of the largest singular values using deflation. All the executions were carried out using MATLAB version 8.0.0.783 from MathWorks on a PC with an Intel quad-core i7-2670QM with 2.20GHz and 12 GB of RAM running Ubuntu Linux 12.04. The machine epsilon is $\mathbf{u} \approx 2.2 \cdot 10^{-16}$. The performance parameters we consider for comparisons are the residual of the approximate singular triplet obtained, the number of matrix-vector multiplications where applicable, and the CPU time. The CPU time is gathered with on-screen outputs suppressed. For the methods that require some factorization of the matrix, we also consider the number of non-zeros in the factors, which indicates the memory requirements and their potential limitations.

We first present an example that tests the capability of `svdifp` to compute tiny singular values accurately. We also show that applying `eigifp` directly to the eigenvalue problem for C^TC may result in a loss of accuracy for the computed singular value. Here, in using `eigifp`, the matrix-vector multiplication C^TCx is obtained by computing Cx first and then multiplying by C^T. Even though C^TC is not explicitly formed, the singular value is obtained from the projection of C^TC, potentially resulting in a loss of accuracy; see the discussion in Section 2.

EXAMPLE 5.1. We consider the following matrix

$$C = U\Sigma V^T, \qquad \text{with } \Sigma = \begin{bmatrix} D \\ 0 \end{bmatrix} \quad \in \mathbb{R}^{m\times n},$$

where $D = \operatorname{diag}(1, 1/2^4, \ldots, 1/n^4)$ and U and V are random orthogonal matrices generated by `U=orth(rand(m,m))` and `V=orth(rand(n,n))` in MATLAB. We test and compare the accuracy of the smallest singular value computed by `svdifp` and `eigifp` with $n = 100$ and $m = 100$ or $m = 200$. In either case, the exact smallest singular value of C is $\sigma_1 = 10^{-8}$, and the second smallest singular value is approximately $1.041 \cdot 10^{-8}$. The convergence is tested using the criterion $\|C^TCv_1 - \sigma_1^2 v_1\| < \eta\|C\|^2$, and to achieve the best accuracy possible, we use a very small threshold $\eta = 10^{-19}$ and run the iteration until the residual stagnates. Both methods are run without preconditioning and with the number of inner iteration set to 20.

Table 5.1 lists the best smallest singular values and their residuals obtained. For `svdifp`, with $\kappa(C) = 10^8$, the residual deceases to about 10^{-18} and the computed value of σ_1 has a relative error of the order of $10^{-10} \approx \mathbf{u}\kappa(C)$. This is the best accuracy one may expect from a backward stable method. On the other hand for `eigifp`, the residual decreases and then stagnates at around 10^{-16}. The relative error of the computed singular values oscillates around 10^{-4}, and no better approximation can be obtained. The singular value computed by applying `eigifp` directly lost about 5 digits of accuracy in this case.

It is interesting to observe that with a good preconditioning, `eigifp` appears to be able to compute σ_1 accurately. Note that C is a dense matrix and the default preconditioner

TABLE 5.1
Example 5.1. σ_1: computed smallest singular value by `svdifp` *and* `eigifp`*, Res:* $\|C^T C v_1 - \sigma_1^2 v_1\|$.

	$m = 100$		$m = 200$	
	σ_1	Res	σ_1	Res
`svdifp`	1.0000000008e-08	1e-20	1.00000000001e-08	2e-20
`eigifp`	1.0001e-08	8e-17	1.00008e-8	8e-17

TABLE 5.2
Test matrices used for Examples 5.2 and 5.3.

No.	Matrix	Size	Non-zeros	σ_1	$\kappa(C)$	source
			Square Matrix			
1	dw2048	2048×2048	10114	4.68e-4	2.03e3	Matrix Market
2	fidap004	1601×1601	31837	6.57e-4	2.39e3	Matrix Market
3	hor131	434×434	41832	1.53e-5	4.31e4	Matrix Market
4	jagmesh1	936×936	6264	5.63e-3	1.23e3	Matrix Market
5	lshp	3025×3025	20833	1.03e-4	6.78e4	Matrix Market
6	pde2961	2961×2961	14585	1.62e-2	6.42e2	Matrix Market
7	pores3	532×532	3474	2.67e-1	5.61e5	Matrix Market
8	sherman	1000×1000	3750	3.23e-4	1.56e4	Matrix Market
			Rectangular Matrix			
9	well1033	1033×320	4372	1.09e-2	1.66e2	Matrix Market
10	well1850	1850×712	8755	1.61e-2	1.11e2	Matrix Market
11	lpi_cplex1	5224×3005	10947	6.39e-2	3.13e3	UFLSMC
12	qiulp	1900×1192	4492	7.57e-1	4.08e1	UFLSMC
13	ge	10099×16369	44825	1.08e-3	1.28e7	UFLSMC
14	p010	10099×19090	118000	1.50e-1	1.18e2	UFLSMC
15	lp_ganges	1309×1706	6937	1.87e-4	2.13e4	UFLSMC
16	cep1	1521×4769	8233	1.00e0	1.49e1	UFLSMC
17	gen2	1121×3264	81855	1.41e0	3.35e1	UFLSMC
18	Maragal_5	3320×4654	93091	7.11e-46	2.30e46	UFLSMC
19	lp_ship12s	1151×2869	8284	0	-	UFLSMC

constructed by `eigifp` is the (complete) LDL^T factorization. However, if we use a preconditioner that is constructed from $A^T A$ by artificially dropping the entries of A that are smaller than 10^{-3}, then a similar loss of accuracy occurs.

Next, we test and compare `svdifp` with several existing programs for the SVD on computing the smallest singular value for a set of test problems. The test matrices consist of both square and non-square matrices taken from the Matrix Market [11] and the University of Florida Sparse Matrix Collection [12]. They are listed in Table 5.2 together with some basic information on the matrices (the smallest singular values are computed by MATLAB's `svd(full(A))`).

Since these programs may have very different approaches and have different assumptions on computing resources, we carry out the tests in two different settings. We first consider those programs in Example 5.2 that do not use any exact factorization for the inverse, i.e., `svdifp`, `jdsvd`, and `irlba`. Since `svdifp` and `jdsvd` can be implemented with or without preconditioning, we test them first with preconditioning and then without preconditioning together with `irlba`. In the second test (Example 5.3), we consider `svds` and `lansvd`, where the LU factorization of M and the QR factorization of C are respectively

TABLE 5.3
Example 5.2 with preconditioning. CPU: CPU time, MV: # of matrix-vector multiplications, nnz: number of non-zeros of the preconditioner, Res: $\|[Cv_1 - \sigma_1 u_1; C^T u_1 - \sigma_1 v_1]\|/\|C\|_1$.

	`svdifp`				`jdsvd`			
No.	CPU	MV	nnz	Res	CPU	MV	nnz	Res
				Square Matrix				
1	0.6	179	25564	9e-7	0.4	136	49019	2e-11
2	1.5	223	91593	1e-7	0.9	102	179673	2e-8
3	0.6	3545	15719	5e-7	0.1	148	11740	3e-10
4	0.4	289	33065	6e-7	0.7	146	67112	6e-10
5	7.3	1103	170276	8e-7	1.7	100	425650	6e-10
6	1.9	113	69291	3e-8	0.3	126	89000	2e-9
7	0.04	25	4870	3e-13	0.09	96	46461	3e-7
8	0.2	355	13695	3e-7	0.1	84	11630	2e-7
				Rectangular Matrix				
9	0.03	91	2235	2e-10	2.8	750	59291	1e-7
10	0.08	69	6325	7e-8	9.6	426	312083	1e-7
11	0.4	69	8995	2e-7	9.0	320	49318	2e-7
12	0.2	91	13620	1e-8	1.2	350	94671	3e-7
13	10.4	91	110017	5e-7	1689.	20052	141008	1e-4
14	13.1	157	138793	2e-7	474.	438	11276604	1e-7
15	0.3	91	18573	9e-9	10.6	358	421304	2e-13
16	2.0	113	106822	3e-8	1.1	266	41793	6e-7
17	4.3	267	297609	9e-7	36023.	36846	8055182	1e-3
18	28.0	24	997991	3e-2[a]	9002.	3744	8666363	7e-7
19	0.08	24	6868	7e-2[b]	0.5	136	65642	4e-8

[a]For this matrix, σ_1 =7.11e-46 according to MATLAB's `svd`. Although `Res` =3e-2, the residual defined by $\|C^T Cv_1 - \sigma_1^2 v_1^2\|$ is 3e-24, while the computed singular value is 2e-25. The singular values returned by `jdsvd` for this matrix is 3e-5. Also note that 113 singular values of this matrix are smaller than the machine precision and the second smallest is 1.7e-31.

[b]For this matrix, $\sigma_1 = 0$ according to MATLAB's `svd`. Although `Res` = 6e-2, the residual defined by $\|C^T Cv_1 - \sigma_1^2 v_1^2\|$ is 2e-25, while the computed singular value is 4e-27. The singular values returned by `jdsvd` for this matrix is 6e-7. Also note that 35 singular values are smaller than the machine precision. The second smallest singular value is 0 as well and the third one is 1.3e-18.

computed for the shift-and-invert. To facilitate a comparison, we consider `svdifp` using the R factor from the QR factorization of C as a preconditioner. Namely, if a complete factorization is possible, `svdifp` may also take advantage of it by using a more effective preconditioner although this is not the best way to use the program.

EXAMPLE 5.2. We consider the performance of `svdifp`, `jdsvd`, and `irlba` in computing the smallest singular value of the matrices in Table 5.2. For matrices with $m < n$, we consider their transposed instead. We set the initial vector for all three methods to be the same random vector generated by `randn(n,1)`. We also select the parameters in the three codes so that each method carries out about the same number of matrix-vector multiplications in each inner iteration. Specifically, for `svdifp`, we set the number of inner iterations m to 10. In `jdsvd`, the maximum number of steps of the inner linear solver is set to 10, which is also its default value. We use the default settings of `jdsvd` for all other parameters. In particular, the refined extraction of the Ritz vector is used throughout, and the dimension of the search subspace varies between 10 and 20. In `irlba`, we set $k = 1$ (the number of desired singular values) and $adjust = 8$ (the number of initial vectors added to the k restart vectors to form an initial subspace). They are chosen so that the dimension of the initial subspace is consistent with the default choices: $k = 6,\ adjust = 3$. All other parameters in `irlba` are set to their default values. Then `irlba` applies 10 bidiagonalization steps after each restart. Based on these settings, all three methods carry out approximately 22 matrix-vector multiplications (by C or C^T) in each outer iteration. We set the maximum number of outer iterations to 10000 for all, and, unless stated otherwise, the stopping criterion is

$$\texttt{Res} := \|[Cv_1 - \sigma_1 u_1; C^T u_1 - \sigma_1 v_1]\| / \|C\|_1 < 10^{-6}, \tag{5.1}$$

where (σ_1, u_1, v_1) is the approximate singular triplet obtained at step k.

We first compare `svdifp` and `jdsvd`, both of which allow using preconditioning to accelerate convergence. In `svdifp`, the default RIF preconditioner is used, i.e., an incomplete factorization of C^TC is constructed by Algorithm 3.1 with the default choices of thresholds $\eta_1 = 10^{-3}$ and $\eta_2 = 10^{-8}$. In `jdsvd`, a preconditioner is needed for solving a correction equation in the inner iteration, and we use the routine `create_prec_jdsvd.m` that accompanies `jdsvd` to construct a preconditioner for M. Specifically, for square matrices, we compute the ILU factorization of C, from which a preconditioner for M is constructed. For non-square matrices, we compute the ILU factorization of M, but because of the singularity of M, breakdown often occurs, in which case the ILU factorization of a shifted matrix $M - \mu I$ is used where $\mu = 2^p \cdot 10^{-2}\|M\|_{\max}$ and p is the first non-negative integer that stops the breakdown. The dropping threshold for all ILU factorizations is 10^{-3}. In addition, `jdsvd` uses BiCGSTAB [41] as the inner linear solver when a preconditioner is present.

In Table 5.3 the results of this test are presented. In the table, `nnz` is the number of non-zeros in the preconditioner (L for `svdifp` and both L and U for `jdsvd`). In the `MV` column, we list the number of matrix-vector multiplications by either C or C^T. `Res` is the relative residual of the approximate singular triplet (5.1).

We observe that `svdifp` achieves satisfactory convergence within 10000 iterations in all problems. For matrices 18 and 19, the singular values are extremely small and therefore the residual of the singular triplet computed by (2.11) is not expected to converge. For these two problems, the termination criterion is switched to using the eigenvalue residual $\|C^TCv_1 - \sigma_1^2 v_1\|$ instead when a singular value of the order of the machine precision is detected (see the discussion on the left singular vectors in Section 2), and then, even though `Res` is fairly large, the computed singular values, which are given in the footnotes, are actually very good approximations already. Therefore, with the limitation of not returning any good left singular vector in such cases, `svdifp` still produces good approximate singular values and right singular vectors. `jdsvd` also achieve satisfactory convergence within 10000 iterations in all but problems 13 and 17. For those two problems, the preconditioned linear solvers in the inner iterations of `jdsvd` converge early in less than the maximum 10 iterations allowed, which is why the total matrix-vector multiplications are less than the maximum possible. Matrix 17 is also a difficult problem with 138 singular values clustered between 1.41421 and 1.41425. In terms of performance measured by `MV` and `CPU`, `jdsvd` outperforms the other methods slightly for square problems, while `svdifp` does that for non-square problems. In terms of `nnz`, RIF in `svdifp` has substantially less memory requirement.

We next compare `svdifp` and `jdsvd` without preconditioning. They are also compared with `irlba`. When no preconditioner is present, `jdsvd` uses MINRES as the inner linear solver. For `irlba`, we only report its results with a one-sided full reorthogonalization, which is the default setting. We list the results of this test in Table 5.4. For problems 18 and 19 with extremely small singular values, the convergence test is switched to use the eigenvalue residual $\|C^TCv_1 - \sigma_1^2 v_1\|$, but at termination, the residual of the singular triplet with the left

TABLE 5.4
Example 5.2 without preconditioning. CPU: CPU time, MV: # of matrix-vector multiplications, Res: $\|[Cv_1 - \sigma_1 u_1; C^T u_1 - \sigma_1 v_1]\|/\|C\|_1$.

	`svdifp`			`jdsvd`			`irlba`		
No.	CPU	MV	Res	CPU	MV	Res	CPU	MV	Res
				Square Matrix					
1	2.2	8033	1e-06	2.1	7542	9e-7	2.8	13856	9e-7
2	4.9	18901	1e-06	6.0	21830	1e-6	21.9	104496	8e-7
3	20.1	220002	5e-04	34.8	220038	1e-5	25.9	220018	2e-2
4	9.5	81227	1e-06	4.8	26308	9e-7	13.4	90350	1e-6
5	27.3	69457	1e-06	20.7	62476	1e-6	59.8	220018	3e-2
6	3.5	9023	1e-06	3.0	9280	1e-6	6.1	23668	9e-7
7	21.1	220002	2e-03	35.7	220038	2e-5	27.2	220018	2e-2
8	15.1	127185	9e-07	23.7	127134	1e-6	32.6	220018	3e-2
				Rectangular Matrix					
9	1.2	7153	1e-06	0.4	2284	6e-7	0.2	1206	7e-8
10	0.5	2467	8e-07	0.6	2262	1e-6	0.3	1888	8e-8
11	0.7	1697	1e-06	0.4	1074	6e-7	0.2	634	2e-7
12	0.3	1257	1e-06	0.7	2900	1e-6	0.2	1228	1e-7
13	500.	220002	1e-03	189.	220038	3e-5	167.	220018	2e-2
14	18.4	6669	1e-06	2.0	1866	1e-6	2.8	2856	6e-8
15	0.1	553	1e-06	0.3	1008	1e-6	0.09	480	9e-8
16	0.1	245	2e-07	0.08	238	1e-7	0.02	62	9e-9
17	0.9	2269	8e-07	3.8	9918	9e-7	62.	220018	5e-7
18	122.	228135	9e-06[a]	116.	220038	2e-6	94.	220018	3e-3
19	0.6	2034	7e-16[b]	2.3	8136	6e-7	0.2	942	7e-8

[a]For this matrix, the residual defined by $\|C^TCv_1 - \sigma_1^2 v_1^2\|$ is 2e-15, while the computed singular value is 1e-12. The singular values returned by `jdsvd` is 5e-10. The singular values returned by `irlba` is 6e-7.
[b]For this matrix, the residual defined by $\|C^TCv_1 - \sigma_1^2 v_1^2\|$ is 3e-14, while the computed singular value is 9e-15. The singular values returned by `jdsvd` is 1e-14. The singular values returned by `irlba` is 4e-16.

singular vector computed by (2.12) has actually converged to a satisfactory level. Nevertheless, we list the computed singular values and the eigenvalue residuals in the footnotes. We note that, without preconditioning, `svdifp` converges much more slowly than the ones with preconditioning, and it appears that the additional iterations have resulted in a substantially reduction of the singular triplet residual. We do not expect this to be the case in general.

It appears that all three methods are comparable in convergence with each method outperforming in some problems. For non-square matrices, `irlba` has the best results outperforming in most problems. Note that `svdifp` without preconditioning is simply the restarted Lanczos method with the LOBPCG-type subspace enhancement. On the other hand, `irlba` is also essentially the Lanczos method, but, with the implicit restart, it uses a larger projection subspace with the same number of matrix-vector multiplications in each restart. Therefore, `irlba` may be expected to outperform `svdifp` without preconditioning in most cases. We also note that the performance of `svdifp` (Table 5.4) is significantly improved by preconditioning (Table 5.3). Several difficult problems with slow convergence are solved fairly easily after preconditioning. With a drop tolerance of 10^{-3}, the RIF preconditioner appears to produce a good preconditioner that also has a relatively small number of fill-ins. Indeed, the number of non-zeros in L (Table 5.3) is typically 2 to 3 times that of C (Table 5.2).

TABLE 5.5
Example 5.3. CPU: CPU time, nnz: non-zeros of R or L and U, Res: $\|[Cv_1 - \sigma_1 u_1; C^T u_1 - \sigma_1 v_1]\|/\|C\|_1$.

	`svdifp`			`svds`			`lansvd`		
No.	CPU	nnz	Res	CPU	nnz	Res	CPU	nnz	Res
				Square Matrix					
1	0.05	83918	1e-16	0.09	193650	4e-15	0.04	83918	2e-13
2	0.1	249160	6e-17	0.1	259562	5e-16	0.09	249160	7e-14
3	0.01	29165	2e-15	0.04	99351	5e-16	0.01	29165	3e-11
4	0.01	35267	9e-13	0.05	69421	1e-15	0.02	35267	3e-10
5	0.1	196083	4e-16	0.2	439407	4e-15	0.08	196083	3e-12
6	0.06	142050	5e-15	0.1	279930	4e-14	0.06	142050	4e-13
7	0.01	8561	9e-13	0.03	52239	5e-17	0.01	8561	2e-15
8	0.01	32816	2e-16	0.05	49971	3e-16	0.02	32816	2e-13
				Rectangular Matrix					
9	0.01	2974	2e-13	-	-	-	0.01	2974	4e-11
10	0.01	9209	1e-12	-	-	-	0.01	9209	2e-10
11	0.8	1514019	1e-14	-	-	-	0.6	1514019	7e-16
12	0.06	48470	2e-12	-	-	-	0.05	48470	2e-13
13	0.4	313320	8e-11	-	-	-	0.3	313320	8e-15
14	0.6	505993	8e-16	-	-	-	0.3	505993	2e-12
15	0.02	30975	1e-17	-	-	-	0.02	30975	4e-14
16	0.4	263226	9e-12	-	-	-	0.2	263226	8e-11
17	54.7	550793	1e-10	-	-	-	15.6	550793	1e-16
18	10.2	2046096	5e-2[a]	-	-	-	-	-	-
19	2.3	7336	5e-17[b]	-	-	-	-	-	-

[a]For this matrix, the residual defined by $\|C^T Cv_1 - \sigma_1^2 v_1^2\|$ is 8e-17, while the computed singular value is 2e-17.
[b]For this matrix, the residual defined by $\|C^T Cv_1 - \sigma_1^2 v_1^2\|$ is 3e-15, while the computed singular value is 3e-16.

EXAMPLE 5.3. In this example, we compare `svdifp` with `svds` and `lansvd`. For computing the smallest singular value, `svds` is based on applying ARPACK [30] to M^{-1} or the shift-and-invert matrix $(M - \mu I)^{-1}$. `lansvd` computes the QR factorization by `R = qr(C,0)` in MATLAB and then computes the largest singular value of R^{-1} by the Lanczos bidiagonalization algorithm. For comparison, we use `R = qr(C,0)` as the preconditioner for `svdifp`. This approach runs into difficulty if R is singular or nearly singular. Indeed, `lansvd` breaks down in such situations (problems 18 and 19). An advantage with `svdifp` is that R is only used as a preconditioner and its accuracy only affects the speed of convergence but not the accuracy of the computed singular values. Therefore, we can simply perturb the zero or nearly zero diagonal entries of R to deal with its singularity. For singular or nearly singular R, it is important to use a column pivoting in the QR factorization but MATLAB's `R = qr(C,0)` employs a column approximate minimum degree permutation to minimize fill-ins. For this test, if the resulting R is nearly singular, we compute a QR factorization by `[~,R,e] = qr(C,0)`, which appears to employ a column pivoting. We then set the diagonal elements of R that are less than the threshold $\sqrt{\mathbf{u}}\|R\|_1$ to the threshold to construct a preconditioner for `svdifp`.

All three codes require no additional input parameters other than the matrix, but we set the initial vector to the same random vector for all of them. We run the programs until convergence as determined by the algorithms themselves. We compare the residual `Res` defined by (5.1), the CPU time, as well as the number of non-zeros used in the factorizations

TABLE 5.6
Example 5.4: 5 largest singular values of matrix `lp_ganges`. *σ_k: singular value, μ: shift used for preconditioning, MV: # of matrix-vector multiplications, Res: $\|[Cv_1 - \sigma_1 u_1; C^T u_1 - \sigma_1 v_1]\|/\|C\|_1$.*

	preconditioning			no preconditioning	
σ_k	μ	MV	Res	MV	Res
3.9908	3.9926	91	3e-12	443	5e-11
3.9906	3.9907	91	2e-14	289	9e-11
3.9895	3.9900	91	1e-13	531	7e-11
3.9894	3.9895	91	5e-13	641	4e-11
3.9892	3.9893	91	4e-12	1103	6e-11

(`nnz`). For `svdifp` and `lansvd`, `nnz` is the number of non-zeros in R, and for `svds`, it is the total non-zeros in L and U of the LU-factorization of M.

The results are given in Table 5.5. All three methods perform comparably for square matrices. `svds` with the zero shift fails for all non-square matrices because of the singularity of M, which is marked by "-" in the table. Even using a small nonzero shift, `svds` usually converges to the eigenvalue 0 rather than σ_1. `svdifp` and `lansvd` can both solve non-square problems with comparable performances. However, `lansvd` can fail for matrices that are nearly rank deficient (problems 18 and 19, marked by "-") because of the inversion of a singular or nearly singular matrix R. On the other hand, `svdifp` does not suffer from a similar problem because R is slightly perturbed to be used as a preconditioner. Overall, `svdifp` appears most robust in this setting.

Finally, we consider `svdifp` for computing several of the largest singular values with deflation. With the shifts chosen inside the spectrum now, RIF constructs an LDL^T factorization for an indefinite matrix $C^TC - \mu I$. So, this also demonstrates the capability of RIF to work with indefinite matrices.

EXAMPLE 5.4. We consider `svdifp` with and without preconditioning in computing the 5 largest singular values of the matrix 15 (`lp_ganges`) in Table 5.2. In both cases, we set the termination threshold to $1 \cdot 10^{-10}$, and the number of outer iterations to 10000. To compute the largest singular value, `svdifp` adaptively chooses a shift for preconditioning; see Section 4. When computing the next largest singular value, the mean of the largest and the second largest singular values of the projection matrix constructed in computing the previous largest singular value is used as the shift to compute an RIF preconditioner. Then, `svdifp` proceeds with a deflated preconditioned iteration. Note that the second largest singular value of the projection matrix is a lower bound of the singular value to be computed and the mean value should provide a better estimate. The same procedure is used for the additional singular values.

We present the results with and without preconditioning for the 5 largest singular values in Table 5.6. We list the number of matrix-vector multiplications (by C or C^T) used for each singular value, the residual `Res` obtained, and in the preconditioned case, the shift μ used. We note that both methods can compute the singular values correctly, while preconditioning by RIF significantly accelerates the convergence of `svdifp`. In particular, the shifted matrix is indefinite now but with the modest indefiniteness in computing a few extreme singular values, RIF results in a very effective preconditioner.

6. Concluding remarks.

We have presented an inverse-free preconditioned Krylov subspace algorithm for computing a few of the extreme singular values of a rectangular matrix. The robust incomplete factorization (RIF) has been adapted to efficiently construct preconditioners for the shifted matrix $C^TC - \mu I$. A preliminary MATLAB implementation has been developed and is demonstrated to be very competitive compared to other existing programs in both settings of using preconditioners or shift-and-invert. A major disadvantage of our approach or the C^TC formulation in general appears to be the potential difficulty in computing left singular vectors corresponding to tiny singular values. This is a problem that we plan to further study. We also plan to refine the MATLAB program `svdifp` and make it available for download in the near future.

Acknowledgment. We would like to thank Michiel Hochstenbach, Patrick Quillen, and Lothar Reichel for some illuminating discussions. In particular, we thank Michiel Hochstenbach for providing us with the code for `jdsvd` and his numerous comments on the `svdifp` code, Michele Benzi and Miroslav Tůma for providing us with their code for RIF, and Lingfei Wu for testing our code `svdifp` [44] and providing numerous comments. We would also like to thank the anonymous referees for many valuable comments that have significantly improved the paper.

REFERENCES

[1] J. Baglama, D. Calvetti, and L. Reichel, *Algorithm 827: irbleigs: a MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix*, ACM Trans. Math. Software, 29 (2003), pp. 337–348.
[2] ———, *IRBL: an implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems*, SIAM J. Sci. Comput., 24 (2003), pp. 1650–1677.
[3] J. Baglama and L. Reichel, *Augmented implicitly restarted Lanczos bidiagonalization methods*, SIAM J. Sci. Comput., 27 (2005), pp. 19–42.
[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.
[5] Z.-Z. Bai, I. S. Duff, and A. J. Wathen, *A class of incomplete orthogonal factorization methods. I. Methods and theories*, BIT, 41 (2001), pp. 53–70.
[6] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, J. Comput. Phys., 182 (2002), pp. 418–477.
[7] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comp., 22 (2000), pp. 1318–1332.
[8] M. Benzi and M. Tůma, *A robust incomplete factorization preconditioner for positive definite matrices*, Numer. Linear Algebra Appl., 10 (2003), pp. 385–400.
[9] ———, *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM J. Sci. Comput., 25 (2003), pp. 499–512.
[10] M. W. Berry, *Large scale sparse singular value computations*, Int. J. Supercomputing Appl., 6 (1992), pp. 13–49.
[11] R. Boisvert, R. Pozo, K. Remington, B. Miller, and R. Lipman, *Matrix Market*, National Institute of Standards and Technology, 1996. Available at http://math.nist.gov/MatrixMarket/.
[12] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), Art. 1 (25 pages).
[13] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[14] T. Ericsson and A. Ruhe, *The spectral transformation Lánczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems*, Math. Comp., 35 (1980), pp. 1251–1268.
[15] D. R. Fokkema, G. L. G. Sleijpen, and H. A. Van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.
[16] A. George and J. Liu, *Householder reflectors versus Givens rotations in sparse orthogonal decompositions*, Linear Algebra Appl., 88/89 (1987), pp. 223–238.
[17] 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.
[18] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.
[19] G. H. Golub and Q. Ye, *Inexact inverse iteration for generalized eigenvalue problems*, BIT, 40 (2000), pp. 671–684.
[20] ———, *An inverse free preconditioned Krylov subspace method for symmetric generalized eigenvalue problems*, SIAM J. Sci. Comput., 24 (2002), pp. 312–334.

[21] M. E. HOCHSTENBACH, *A Jacobi-Davidson type SVD method*, SIAM J. Sci. Comput., 23 (2001), pp. 606–628.
[22] ———, *Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems*, BIT, 44 (2004), pp. 721–754.
[23] ———, Private communications, 2014.
[24] 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.
[25] A. V. KNYAZEV, *A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace*, in Numerical Treatment of Eigenvalue Problems, J. Albrecht, L. Collatz, P. Hagedorn and W. Velte, eds., Internat. Ser. Numer. Math., Vol. 96 , Birkhäuser, Basel, 1991, pp. 143–154.
[26] A. V. KNYAZEV, *Preconditioned eigensolvers—an oxymoron?*, Electron. Trans. Numer. Anal., 7 (1998), pp. 104–123.
http://etna.mcs.kent.edu/vol.7.1998/pp104-123.dir
[27] ———, *Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.
[28] E. KOKIOPOULOU, C. BEKAS, AND E. GALLOPOULOS, *Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization*, Appl. Numer. Math., 49 (2004), pp. 39–61.
[29] R. M. LARSEN, *Lanczos bidiagonalization with partial reorthogonalization*, Technical report, DAIMI PB-357, Department of Computer Science, University of Aarhus, Aarhus, Denmark.
[30] R. LEHOUCQ, D. SORENSON, AND C. YANG, *ARPACK Users' Guides. Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.
[31] N. LI AND Y. SAAD, *MIQR: a multilevel incomplete QR preconditioner for large sparse least-squares problems*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 524–550.
[32] J. H. MONEY AND Q. YE, *Algorithm 845: EIGIFP: a MATLAB program for solving large symmetric generalized eigenvalue problems*, ACM Trans. Math. Software, 31 (2005), pp. 270–279.
[33] R. B. MORGAN, *Computing interior eigenvalues of large matrices*, Linear Algebra Appl., 154/156 (1991), pp. 289–309.
[34] Y. NOTAY, *Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem*, Numer. Linear Algebra Appl., 9 (2002), pp. 21–44.
[35] A. T. PAPADOPOULOS, I. S. DUFF, AND A. J. WATHEN, *A class of incomplete orthogonal factorization methods II: implementation and results*, BIT, 45 (2005), pp. 159–179.
[36] P. QUILLEN AND Q. YE, *A block inverse-free preconditioned Krylov subspace method for symmetric generalized eigenvalue problems*, J. Comput. Appl. Math., 233 (2010), pp. 1298–1313.
[37] Y. SAAD, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, Manchester, 1992.
[38] G. L. G. SLEIJPEN AND H. A. VAN DER VORST, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.
[39] D. C. SORENSEN, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.
[40] A. STATHOPOULOS, Y. SAAD, AND K. WU, *Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods*, SIAM J. Sci. Comput., 19 (1998), pp. 227–245.
[41] H. A. VAN DER VORST, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.
[42] S. VARADHAN, M. W. BERRY, AND G. H. GOLUB, *Approximating dominant singular triplets of large sparse matrices via modified moments*, Numer. Algorithms, 13 (1996), pp. 123–152.
[43] X. WANG, K. A. GALLIVAN, AND R. BRAMLEY, *CIMGS: an incomplete orthogonal factorization preconditioner*, SIAM J. Sci. Comput., 18 (1997), pp. 516–536.
[44] L. WU AND A. STATHOPOULOS, *Primme_svds: A preconditioned svd solver for computing accurately singular triplets of large matrices based on the primme eigensolver*, Preprint on ArXiV, http://arxiv.org/pdf/1408.5535.pdf, 2014.
[45] Q. YE, *An adaptive block Lanczos algorithm*, Numer. Algorithms, 12 (1996), pp. 97–110.