Numerical Algorithms manuscript No.
(will be inserted by the editor)

Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods

James Baglama · Vasilije Perović · Jennifer Picucci

Received: date / Accepted: date

Abstract Presented are new hybrid restarted Lanczos bidiagonalization methods for the computation of a few of the extreme singular triplets of very large matrices. Restarting is carried out either by a thick–restarted scheme with Ritz vectors or explicitly with iterative refined Ritz vectors. Several criteria are used to determine which restarted process is to be used. Also presented, are MATLAB codes that implement the described algorithms along with numerous examples demonstrating our methods are competitive with other available routines.

Keywords (Partial) Singular value decomposition · Iterative method · Large-scale computation · Refined Ritz · Lanczos bidiagonalization
Mathematics Subject Classification (2010): 65F15, 65F50, 15A18

1 Introduction

The *singular value decomposition* (SVD) of matrix $A \in \mathbb{R}^{\ell \times n}$ $(\ell \geq n)$[1] is a factorization of the form

$$A = U\Sigma V^T \tag{1}$$

where $U = [u_1, \ldots, u_n] \in \mathbb{R}^{\ell \times n}$ and $V = [v_1, \ldots, v_n] \in \mathbb{R}^{n \times n}$ have orthonormal columns and $\Sigma = \operatorname{diag}\left(\sigma_1, \sigma_2, \ldots, \sigma_n\right) \in \mathbb{R}^{n \times n}$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

Submitted to the editors xxxxxx.

J. Baglama · V. Perović · J. Picucci
Department of Mathematics and Applied Mathematical Sciences, University of Rhode Island, Kingston, RI 02881
E-mail: jbaglama@uri.edu, perovic@uri.edu, jenniferpicucci@uri.edu

J. Picucci
U.S. Army Engineer Research and Development Center, Vicksburg, MS
E-mail: Jennifer.r.picucci@usace.army.mil

[1] Otherwise replace A with A^T.

The σ_j's are the singular values of A, while u_j's and v_j's are the corresponding left and right singular vectors of A, respectively. Collectively, $\{\sigma_j, u_j, v_j\}$ is referred to as a *singular triplet* of A. From (1), for $0 < s \le n$, we have

$$AV_s = U_s \Sigma_s, \qquad A^T U_s = V_s \Sigma_s, \tag{2}$$

where $\Sigma_s = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_s) \in \mathbb{R}^{s\times s}$, $U_s = [u_1, \ldots, u_s] \in \mathbb{R}^{\ell\times s}$, and $V_s = [v_1, \ldots, v_s] \in \mathbb{R}^{n\times s}$; when $s < n$ we refer to the factorization (2) as a *partial singular value decomposition* of A, or s-PSVD for short.

The primary focus of this paper is on computing a small number of singular triplets, let's say k, corresponding to the largest singular values and associated vectors, while using as little memory as possible. In other words, we are interested in computing $\{\sigma_j, u_j, v_j\}_{j=1}^k$ such that

$$Av_j = \sigma_j u_j, \qquad A^T u_j = \sigma_j v_j, \qquad j = 1, 2, \ldots, k\,. \tag{3}$$

Some of the earliest work in this direction can be traced to the landmark paper by Golub and Kahan [10], where the authors showed how singular triplets can be computed efficiently and in a numerically stable way by what is now known as the Golub-Kahan-Lanczos (GKL) bidiagonalization procedure.

Today, SVD is one of the main computational methods with numerous applications, e.g., dimension reduction, Principal Component Analysis (PCA) [24], genomics [1,3], data mining, data visualization, machine learning, and pattern recognition [8,31]. Matrices arising from these applications are often very large, sparse and only accessible via matrix-vector routines which makes it impractical for the computation of all singular triplets. Fortunately, with these matrices one is typically interested in computing only a few of the largest (or smallest) singular triplets – this has spurred a considerable amount of research and software development, see e.g., [4,5,9,12,20,21,25,26,27,28,41] and the references therein.

One of the features shared by many of the referenced routines is the vital role played by the GKL procedure [10]. Recall that for some starting unit vector p_1 (and $q_1 := Ap_1$), this procedure creates orthonormal bases for the Krylov subspaces,

$$\begin{aligned}
\mathbb{K}_m(A^TA, p_1) &= \operatorname{span}\left\{p_1,\, A^TAp_1,\, \left(A^TA\right)^2 p_1,\, \ldots,\, \left(A^TA\right)^{m-1} p_1\right\},\\
\mathbb{K}_m(AA^T, q_1) &= \operatorname{span}\left\{q_1,\, AA^Tq_1,\, \left(AA^T\right)^2 q_1,\, \ldots,\, \left(AA^T\right)^{m-1} q_1\right\},
\end{aligned} \tag{4}$$

using only matrix-vector products with A and A^T while avoiding explicitly creating the matrices A^TA and AA^T. This makes the process ideal for very large problems. The GKL procedure at step m yields the m-GKL factorization,

$$AP_m = Q_m B_m\,, \tag{5}$$

$$A^T Q_m = P_m B_m^T + f e_m^T = \begin{bmatrix} P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} B_m^T \\ \beta_m e_m^T \end{bmatrix}, \tag{6}$$

where $P_m = [p_1, \ldots, p_m] \in \mathbb{R}^{n\times m}$ and $Q_m = [q_1, \ldots, q_m] \in \mathbb{R}^{\ell\times m}$ have orthonormal columns which form bases for Krylov subspaces (4) respectively, the residual vector $f \in \mathbb{R}^n$ satisfies $P_m^T f = 0$, $\beta_m = \|f\|$, and $p_{m+1} = f/\beta_m$. Further, e_m is the m^{th} axis vector of appropriate dimension and

$$B_m := \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \alpha_3 & \beta_3 & \\ & & & \ddots & \ddots \\ & & & & \ddots & \beta_{m-1} \\ & & & & & \alpha_m \end{bmatrix} \in \mathbb{R}^{m\times m} \tag{7}$$

is an upper bidiagonal matrix. Approximations of the singular triplets of A can be obtained from the singular triplets of B_m. Observe that when the norm of the residual vector f is small, the singular values of B_m are close to the singular values of A (exact when f vanishes) and the associated singular vectors are computed using the basis vectors of the Krylov subspaces, see Section 2 for details. However, these approximations are typically poor for modest values of m, hence either requiring m to be increased or the starting vector p_1 to be modified (explicitly or implicitly) and the GKL process restarted. Considering that the matrix A is of large scale and assuming prohibitive memory limitations, increasing m to a suitable value to get acceptable approximations is not an option. Thus, much of the research, including this paper, revolves around developing different restarting schemes for the GKL process. Note that there are already several notable routines that do this [4,5,20,21,25,26], particularly the thick–restarted GKL routine in [4] which plays a key role in this paper.

In [4], Baglama and Reichel exploited the mathematical equivalence for symmetric eigenvalue computations of the implicitly restarted Arnoldi (Lanczos) method of Sorensen [35] and the thick–restarting scheme of Wu and Simon [40], as described in [29], and applied it to a restarted GKL procedure. Their thick–restarted GKL routine turns out to be a simple and computationally fast method for computing a few of the extreme singular triplets of large matrices that is less sensitive to propagated round-off errors; for a brief review of this scheme see Section 2. However, the routine struggles when the dimension, m, of the Krylov subspaces is memory limited and kept relatively small in relationship to the number of desired singular triplets k, see the examples in Section 5. Recently, in the context of symmetric eigenvalue computation, the authors overcame this memory restriction by creating a hybrid restarted Lanczos method that combines thick–restarting with Ritz vectors with a new technique, iteratively refined Ritz vectors [2]. The thick–restarted part was carried out as described in [40] and when certain criteria were met, the routine switched to restarting with a linear combination of iteratively refined Ritz vectors. In [2], the authors showed that the scheme of thick–restarting of Wu and Simon was not available with refined or iteratively refined Ritz vectors. Furthermore, in [2] an alternate scheme was introduced in which, based on the relationships first proposed by Sorensen [35] and later outlined in detail by Morgan [29], the iteratively refined Ritz vectors are linearly combined and then used to restart the process. The constants were chosen in such a way that the linear combination of the iteratively refined Ritz vectors resembles a restart, in a somewhat asymptotic sense, of thick–restarting, see [2, Sec. 6] for details.

It is well–known that the refined Ritz vectors can provide better eigenvector approximations than the Ritz vectors, see [18,22] for details. But in a restarted scheme, "better" approximation is only a part of the overall need and an efficient restarting scheme is also required. One approach was given in [16], where "refined" shifts are used in the implicitly restarted Arnoldi method. In the context of SVD, this approach was further extended [20,21] resulting in an implicitly restarted GKL procedure for computing singular triplets. In this paper, we present another approach where we extend the restarted hybrid iterative refined scheme [2] to the GKL procedure for computing singular triplets.

In the context of the symmetric eigenvalue problem, the authors in [2] consider an iterative refined Ritz scheme in which the refined process is repeated until convergence. This process has the benefit of eliminating part of the refined Ritz residuals and aiding in the ability to create a linear combination to resemble thick–restarting, all while producing a "smaller" norm. A brief review of the iterative refined Ritz scheme is provided in Section 3 though for a thorough discussion and results we refer the reader to [2].

To make the connection between the symmetric eigenvalue problem and the SVD of $A \in \mathbb{R}^{\ell\times n}$ more explicit, consider the matrices

$$A^TA \in \mathbb{R}^{n\times n} \qquad \text{and} \qquad C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \in \mathbb{R}^{(\ell+n)\times(\ell+n)}. \tag{8}$$

We refer to A^TA as the normal matrix or system and C as the augmented matrix or system. The eigenvalues of A^TA are the squares of singular values of A, while the associated eigenvectors of A^TA are the corresponding right singular vectors of A, i.e., $A^TAv_j = \sigma_j^2 v_j$. When $\sigma_j \neq 0$, the left singular vectors can be computed as $u_j = (1/\sigma_j)Av_j$. In the case of the augmented system C, its eigenvalues are $\pm\sigma_j$ as well as $\ell - n$ zero eigenvalues. The eigenvectors of C associated with $\pm\sigma_j$ are $\frac{1}{\sqrt{2}}[u_j; \pm v_j]$, where $\{\sigma_j, u_j, v_j\}$ is a singular triplet of A.

Multiplying equation (5) from the left by A^T produces the Lanczos tridiagonal decomposition of the normal matrix A^TA, namely

$$A^TAP_m = P_m B_m^T B_m + \alpha_m f_m e_m^T = \begin{bmatrix} P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} B_m^T B_m \\ \alpha_m \beta_m e_m^T \end{bmatrix}. \tag{9}$$

Similarly, in the case of matrix C, after performing $2m$ steps of the standard Lanczos algorithm with the starting vector $[0\,; p_1] \in \mathbb{R}^{\ell+n}$ we have a $2m \times 2m$ tridiagonal projection matrix, which when followed by an odd-even permutation gives the following Lanczos factorization [11, Sec. 10.4.3] [25]

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} Q_m & 0 \\ 0 & P_m \end{bmatrix} = \begin{bmatrix} Q_m & 0 & 0 \\ 0 & P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \\ \beta_m e_m^T & 0 \end{bmatrix}. \tag{10}$$

Considering the Lanczos factorization relationships (9) and (10), the results and properties related to the hybrid iterative refined Ritz scheme in [2] are carried over to the methods developed in the subsequent sections. Although our development is focused on the largest singular values, it can be applied to computing the smallest singular values and associated vectors.

The paper is organized as follows. The thick–restarted scheme with Ritz vectors is reviewed in Section 2 while a new development of iteratively refined Ritz vectors computed either on the normal system (9) or the augmented system (10) can be found in Section 3. In Section 4, we describe our new hybrid methods and present two algorithms for computing singular triplets. Numerical examples are presented in Section 5 followed by conclusions in Section 6.

Throughout this paper $\|\cdot\|$ denotes the Euclidean vector norm or the associated induced matrix norm. I_k is used to denote the $k\times k$ identity matrix while I_{k_1,k_2}, with $k_1 \geq k_2$, denotes the first k_2 columns of I_{k_1}; when the size is clear from the context we simply write I. When useful and for ease of presentation we utilize MATLAB's syntax for constructing block matrices. An expression of the form $\xi := \eta$ (resp., $\xi =: \eta$) is used to denote that ξ *is defined to be equal to* η (resp., η *is defined to be equal to* ξ). In order to distinguish among numerous SVD computations and to help the reader, throughout the paper we adopt the *convention* that superscripts (rz), $(rf\text{-}\star)$, and $(it\text{-}\star)$ correspond to the computations involving Ritz, refined Ritz, and iteratively refined Ritz values/vectors, respectively; here $\star \in \{n,\, a\}$ denotes that (iteratively) refined Ritz are computed with respect to either the normal or the augmented systems (8). Finally, when a formula is developed and used in different settings, we use a "generic" superscript (..) (see Sections 2-3).

2 Thick–restarted GKL process with Ritz vectors

In order to establish the notation, as well as for the sake of completeness, we briefly review the method of thick–restarting with Ritz vectors. We note that, although not presented here and can be used in our scheme, thick–restarting can also be carried out with harmonic Ritz vectors, see [4] for a thorough discussion and details.

The starting point for thick–restarting is the observation that once the m-GKL factorization (5)-(6) of A is computed, then singular values of A can be approximated by singular values of B_m. Let the s-PSVD of B_m from (7) be

$$B_m V_s^{(\mathrm{rz})} = U_s^{(\mathrm{rz})} \Sigma_s^{(\mathrm{rz})}, \qquad B_m^T U_s^{(\mathrm{rz})} = V_s^{(\mathrm{rz})} \Sigma_s^{(\mathrm{rz})}, \tag{11}$$

where $U_s^{(\mathrm{rz})} = [u_1^{(\mathrm{rz})}, \ldots, u_s^{(\mathrm{rz})}] \in \mathbb{R}^{m\times s}$ and $V_s^{(\mathrm{rz})} = [v_1^{(\mathrm{rz})}, \ldots, v_s^{(\mathrm{rz})}] \in \mathbb{R}^{m\times s}$ have orthonormal columns and $\Sigma_s^{(\mathrm{rz})} = \mathrm{diag}\left(\sigma_1^{(\mathrm{rz})}, \ldots, \sigma_s^{(\mathrm{rz})}\right) \in \mathbb{R}^{s\times s}$ such that $\sigma_1^{(\mathrm{rz})} \geq \sigma_2^{(\mathrm{rz})} \geq \cdots \geq \sigma_s^{(\mathrm{rz})} \geq 0$. Define $\tilde{P}_s := P_m V_s^{(\mathrm{rz})}$ and $\tilde{Q}_s := Q_m U_s^{(\mathrm{rz})}$, where P_m and Q_m are as in (5) and (6). Then from (5), (6), and (11) it follows that

$$A\tilde{P}_s = AP_m V_s^{(\mathrm{rz})} = Q_m B_m V_s^{(\mathrm{rz})} = Q_m U_s^{(\mathrm{rz})} \Sigma_s^{(\mathrm{rz})} = \tilde{Q}_s \Sigma_s^{(\mathrm{rz})} =: \tilde{Q}_s \tilde{B}_s\,. \tag{12}$$

Similarly,

$$A^T\tilde{Q}_s = A^T Q_m U_s^{(\mathrm{rz})} = P_m B_m^T U_s^{(\mathrm{rz})} + f e_m^T U_s^{(\mathrm{rz})} = P_m V_s^{(\mathrm{rz})} \Sigma_s^{(\mathrm{rz})} + f(e_m^T U_s^{(\mathrm{rz})}),$$

$$= \left[\tilde{P}_s \; p_{s+1}\right] \left[\begin{array}{c} \Sigma_s^{(\mathrm{rz})} \\ \hline \rho_1 \; \dots \; \rho_s \end{array}\right] =: \left[\tilde{P}_s \; p_{s+1}\right] \tilde{B}_{s,s+1}^T, \tag{13}$$

where $p_{s+1} = f/\|f\|$ and $\rho_j = \|f\| U_s^{(\mathrm{rz})}(m,j)$. Note that the pair of factorizations (12)-(13) can be extended with p_{s+1} as the starting vector to obtain a new factorization similar to the m-GKL factorization (5)-(6); the noted difference is in the structure of B_m which is given by

$$B_m = \begin{bmatrix} \left[\tilde{B}_{s,s+1}\right] & & & 0 \\ & \alpha_{s+1} & \beta_{s+1} & \\ & & \ddots & \ddots \\ & & \ddots & \beta_{m-1} \\ 0 & & & \alpha_m \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{14}$$

Remark 1 The pairs of factorizations (5)-(6) (with B_m as in (7) or (14)) and (12)-(13) play a central role in this paper. As such, throughout the rest of this paper, we refer to (12)-(13) and (5)-(6) as an *s-GKL* and an *m-GKL* factorizations, respectively. Note that due to the structure of matrices $\tilde{B}_s$ (12) and $\tilde{B}_{s,s+1}$ (13), the pair (12)-(13) is not a GKL factorization in the classical sense, though it can be transformed into one [38]. The algorithmic details for computation of the factorizations (5)-(6) and (12)-(13) are standard in the literature – e.g., see [4, Algorithm 2.1] and the subsequent discussion regarding different reorthogonalization strategies.

Once the m-GKL factorization (5)-(6) is computed, the s-PSVD factorization of B_m, with $k \le s < m$, can be used to initially approximate k singular triplets $\{\sigma_j, u_j, v_j\}$ of A, $j = 1,\dots,k$. Depending how good these approximations are, one can restart this process by first computing the s-GKL factorization (12)-(13) and extending it to the m-GKL (5)-(6) with B_m as in (14), until convergence.

We use the notation $\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\}$ to denote an approximation to the k desired singular triplets of A, where $\sigma_j^{(..)}$, $u_j^{(..)}$, and $v_j^{(..)}$ are taken from the methods described in this paper. For example, when using Ritz values and vectors we write

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} = \{\sigma_j^{(\mathrm{rz})}, Q_m u_j^{(\mathrm{rz})}, P_m v_j^{(\mathrm{rz})}\}. \tag{15}$$

The convergence is established by using the following residual equation that is derived from the Lanczos factorization (10),

$$resAug_j^{(..)} = \sqrt{\|B_m v_j^{(..)} - \sigma_j^{(..)} u_j^{(..)}\|^2 + \|B_m^T u_j^{(..)} - \sigma_j^{(..)} v_j^{(..)}\|^2 + (e_m^T u_j^{(..)})^2 \beta_m^2}, \tag{16}$$

where $\beta_m = \|f\|$. Note that (16) can be simplified when using Ritz approximation (15) to $resAug_j^{(\mathrm{rz})} = |e_m^T u_j^{(\mathrm{rz})}|\beta_m$. Likewise, a residual equation can be computed from the Lanczos factorization (9)

$$resNor_j^{(..)} = \sqrt{\|B_m^T B_m v_j^{(..)} - (\sigma_j^{(..)})^2 v_j^{(..)}\|^2 + (\alpha_m e_m^T v_j^{(..)})^2 \beta_m^2}\,.$$

Note that if $B_m v_j^{(..)} = \sigma_j^{(..)} u_j^{(..)}$, then $resNor_j^{(..)} = \sigma_j^{(..)} resAug_j^{(..)}$. Finally, independent of the restarted scheme used, convergence of an approximate triplet is tested via (16) and the condition

$$resAug_j^{(..)} \leq tol \cdot \|A\|\,, \tag{17}$$

where *tol* is a user specified tolerance and $\|A\|$ is approximated by the largest singular value of B_m over all iterations.

3 Refined and Iterative Refined Ritz vectors

In 1997, Jia proposed to use *refined Ritz* vectors in place of Ritz vectors as eigenvector approximations of a matrix M [15]. More specifically, for a given approximate eigenvalue μ_j of M, Jia's method looks to minimize $\|Mz_j - \mu_j z_j\|$ for a unit vector z_j from a given subspace $\mathcal{W}$, i.e.,

$$\min_{z_j \in \mathcal{W}\,,\, \|z_j\|=1} \|Mz_j - \mu_j z_j\|. \tag{18}$$

In [15] it was shown that on the subspace $\mathcal{W}$ an approximate eigenpair using the refined Ritz vector produced a "smaller" residual norm than an eigenpair approximation with the Ritz pair. Since then, the notion of "refined vectors" has produced a significant amount of research in many directions, see e.g., [2, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 25, 30] and references therein.

More recently, in [2] we introduced the idea of *iterative refined Ritz* values/vectors for the symmetric eigenvalue problem, where the approximate eigenvalue in the refined scheme is replaced with the latest computed refined Ritz value until convergence.

Through numerical examples in [2] it was demonstrated that when memory was limited and only iterative refined Ritz vectors were used to restart the method there was potential for either slow or no convergence. Similar behavior is also observed in this context, see Example 1. As a way to overcome these challenges, a hybrid method was developed that uses thick–restarted with Ritz vectors and under certain criteria it restarts with a linear combination of iterative refined Ritz vectors.

In this paper, we extend the idea of iterative refined values/vectors to the GKL process and develop new hybrid schemes for computing singular triplets. Considering the relationships of the Lanczos factorizations (9) and (10) and symmetric matrices $A^T A$ and $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$, respectively, we were able to leverage multiple results from [2], though still several nontrivial adaptations were required (see Sections 3.1-3.2). There are several refined schemes as applied to the matrix C that have been considered, e.g., [20,21]. More specifically, the refined Ritz scheme in [20] uses the lower bidiagonal Lanczos process [32] while the scheme in [21] utilizes the GKL process and computes refined harmonic Ritz values/vectors using the augmented system (10). Both schemes [20,21] implemented restarting by utilizing the refined process to gain "shifts" that are then used in an implicitly restarted GKL algorithm. Other implicitly restarted GKL methods worth mentioning include [25] where the authors utilized the lower bidiagonal Lanczos process on the related system AA^T while using Ritz or harmonic Ritz values as "shifts", and the method in [5] that used Leja points as "shifts" from the normal equations (9). What differentiates work in this paper from these methods, is that our primary focus is not on computing "shifts" but rather on a *hybrid scheme* that restarts the GKL process either through thick–restarting with Ritz or explicitly restarting with a linear combination of iterative refined Ritz vectors.

3.1 Refined and Iterative Refined on normal system

Our development of the iterative refined Ritz values/vectors naturally starts with the normal system (9). To that end, let $M = A^TA$ and $\mathcal{W} = \mathbb{K}_m(A^TA, p_1)$ in equation (18) and define

$$T_{m+1,m} := \begin{bmatrix} B_m^T B_m \\ \alpha_m \beta_m e_m^T \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}. \tag{19}$$

For each approximate eigenvalue μ_j of A^TA compute the smallest singular value $\sigma\!\downarrow_j^{\text{(rf-n)}}$ and associated unit singular vectors of $(T_{m+1,m} - \mu_j I_{m+1,m})$, i.e.,

$$(T_{m+1,m} - \mu_j I_{m+1,m}) v_j^{\text{(rf-n)}} = \sigma\!\downarrow_j^{\text{(rf-n)}} w_j, \tag{20}$$

$$(T_{m+1,m} - \mu_j I_{m+1,m})^T w_j = \sigma\!\downarrow_j^{\text{(rf-n)}} v_j^{\text{(rf-n)}}, \tag{21}$$

where $v_j^{\text{(rf-n)}} \in \mathbb{R}^m$ and $w_j \in \mathbb{R}^{m+1}$. Then from (5),(6), and (9) it follows that

$$\min_{\substack{z_j \in \mathbb{K}_m(A^TA,p_1)\\ \|z_j\|=1}} \|A^TAz_j - \mu_j z_j\| = \|(T_{m+1,m} - \mu_j I_{m+1,m}) v_j^{\text{(rf-n)}}\| = \sigma\!\downarrow_j^{\text{(rf-n)}} \tag{22}$$

and the *refined Ritz vector* z_j for μ_j is defined as $z_j := P_m v_j^{\text{(rf-n)}}$. The approximate eigenvalue of A^TA associated with the refined Ritz vector z_j is selected as the Rayleigh quotient

$$\sigma_j^{\text{(rf-n)}\,2} = z_j^T A^T A z_j = v_j^{\text{(rf-n)}\,T} B_m^T B_m v_j^{\text{(rf-n)}} = \|B_m v_j^{\text{(rf-n)}}\|^2, \tag{23}$$

and the approximate *refined singular triplet on the normal system* for A is given by

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{\text{(rf-n)}}, Q_m u_j^{\text{(rf-n)}}, P_m v_j^{\text{(rf-n)}}\}, \tag{24}$$

where $u_j^{\text{(rf-n)}} = B_m v_j^{\text{(rf-n)}} / \sigma_j^{\text{(rf-n)}}$.

The initial approximate eigenvalue μ_j in equations (20)-(22) can be taken as the Ritz value ${\sigma_j^{\text{(rz)}}}^2$ (11). Then the iterative refined Ritz process iteratively refines the approximation, by taking the output approximation, $\sigma_j^{\text{(rf-n)}}$ (23), setting $\mu_j = {\sigma_j^{\text{(rf-n)}}}^2$, and re–computing refined vectors $v_j^{\text{(rf-n)}}$, via (20)-(21) until convergence. This process produces a nonnegative, decreasing and hence convergent sequence ${\sigma{\downarrow}_j^{(..)}}^{(i)}$, see [2, Thm. 5.1]; Algorithm 1 outlines this process.

Algorithm 1 Iterative Refined

```
 1: Input: $T_{m+1,m} \in \mathbb{R}^{(m+1)\times m}$ (19) or $T_{2m+1,2m} \in \mathbb{R}^{(2m+1)\times 2m}$ (39) and $\{\mu_j\}_{j=1}^k$.
 2: Output: $\{\sigma_j^{\text{(it-n)}}, u_j^{\text{(it-n)}}, v_j^{\text{(it-n)}}\}_{j=1}^k$ and $\hat{\sigma}{\downarrow}_j^{\text{(it-n)}}$ or $\{\sigma_j^{\text{(it-a)}}, u_j^{\text{(it-a)}}, v_j^{\text{(it-a)}}\}_{j=1}^k$ and $\hat{\sigma}{\downarrow}_j^{\text{(it-a)}}$.
 3: for $j = 1, 2, \ldots, k$ do
 4:   for $i = 1, 2, \ldots, maxitref$ do
 5:     if normal system then
 6:       Compute ${v_j^{\text{(rf-n)}}}^{(i)}, w_j^{(i)}$, and ${\sigma{\downarrow}_j^{\text{(rf-n)}}}^{(i)}$ (20) and (21);
 7:       ${\sigma_j^{\text{(rf-n)}}}^{(i)} := \|B_m {v_j^{\text{(rf-n)}}}^{(i)}\|$ (23);
 8:       if converge then
 9:         $\sigma_j^{\text{(it-n)}} := {\sigma_j^{\text{(rf-n)}}}^{(i)}, v_j^{\text{(it-n)}} := {v_j^{\text{(rf-n)}}}^{(i)}, u_j^{\text{(it-n)}} := B_m v_j^{\text{(it-n)}} / \sigma_j^{\text{(it-n)}}$, $\hat{\sigma}{\downarrow}_j^{\text{(it-n)}} := {\sigma{\downarrow}_j^{\text{(rf-n)}}}^{(i)}$;
10:         Break;
11:       end if
12:       $\mu_j := ({\sigma_j^{\text{(rf-n)}}}^{(i)})^2$;
13:     else
14:       Compute $x_j^{(i)}, y_j^{(i)}, {w_x}_j^{(i)}, {w_y}_j^{(i)}, {w_z}_j^{(i)}$, and ${\sigma{\downarrow}_j^{\text{(rf-a)}}}^{(i)}$ (40) and (41);
15:       ${\sigma_j^{\text{(rf-a)}}}^{(i)} := 2{x_j^{(i)}}^T B_m y_j^{(i)}$ (43);
16:       if converge and $|\|x_j^{(i)}\| - 1/\sqrt{2}| \leq \sqrt{eps}$ then
17:         $\sigma_j^{\text{(it-a)}} := {\sigma_j^{\text{(rf-a)}}}^{(i)}$, $v_j^{\text{(it-a)}} := {y_j}^{(i)} / \|{y_j}^{(i)}\|$, $u_j^{\text{(it-a)}} := x_j^{(i)} / \|x_j^{(i)}\|$, $\hat{\sigma}{\downarrow}_j^{\text{(it-a)}} := {\sigma{\downarrow}_j^{\text{(rf-a)}}}^{(i)}$;
18:         Break;
19:       end if
20:       $\mu_j := {\sigma_j^{\text{(rf-a)}}}^{(i)}$;
21:     end if
22:   end for
23: end for
```

There are several options for the convergence check (steps 8 and 16) in Algorithm 1, e.g., $|{\sigma_j^{(..)}}^{(i)} - {\sigma_j^{(..)}}^{(i-1)}| / |{\sigma_j^{(..)}}^{(i)}| < eps$, where eps is machine epsilon; the additional requirement on $\|x_j^{(i)}\|$ in step 16 is discussed in Section 3.2. While using finite arithmetic, stagnation can occur and we propose including an additional check to exit when detected. We identify stagnation as failed convergence. The initial view of Algorithm 1 (for loop *maxitref*) may appear to be computationally expensive, however when the matrix B_m is kept very small, the cost is negligible in comparison to the cost of the matrix–vector products when the order of A is very large. We include computational times for numerical examples in Section 5. When m is larger or as the overall scheme converges, we found that fewer iterations are needed and the iterative refined vectors did not differ much from the refined vectors. However, it should be noted again that the main focus of this paper is on using a very small subspaces, where differences are readily observed. Therefore, using Algorithm 1 with initial approximate eigenvalues $\mu_j = {\sigma_j^{(\text{rz})}}^2$, we obtain the approximate *iterative refined Ritz singular triplet on the normal system* for A as

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{(\text{it-n})}, Q_m u_j^{(\text{it-n})}, P_m v_j^{(\text{it-n})}\}. \tag{25}$$

Using the m-GKL factorization and the refined Ritz singular approximation (24), together with equations (20)-(21), give us

$$AP_m v_j^{(\text{rf-n})} = Q_m B_m v_j^{(\text{rf-n})} \;=\; \sigma_j^{(\text{rf-n})} Q_m u_j^{(\text{rf-n})}, \tag{26}$$

$$\begin{aligned} A^T Q_m u_j^{(\text{rf-n})} &= P_m B_m^T u_j^{(\text{rf-n})} + f e_m^T u_j^{(\text{rf-n})}, \qquad (27)\\ &= \sigma_j^{(\text{rf-n})} P_m v_j^{(\text{rf-n})} + \sigma\!\downarrow_j^{(\text{rf-n})} / \sigma_j^{(\text{rf-n})} \left[P_m \; p_{m+1}\right] r_j, \end{aligned}$$

where $r_j = w_j - ([v_j^{(\text{rf-n})}; 0]^T w_j)[v_j^{(\text{rf-n})}; 0]$. Multiplying (26) by A^T on the left yields the following relation

$$A^T A P_m v_j^{(\text{rf-n})} = {\sigma_j^{(\text{rf-n})}}^2 P_m v_j^{(\text{rf-n})} + \sigma\!\downarrow_j^{(\text{rf-n})} \left[P_m \; p_{m+1}\right] r_j. \tag{28}$$

If Algorithm 1 is used to compute the iterative refined Ritz value and vectors we have the output satisfying,

$$(T_{m+1,m} - {\sigma_j^{(\text{it-n})}}^2 I_{m+1,m}) v_j^{(\text{it-n})} = \hat{\sigma}\!\downarrow_j^{(\text{it-n})} \hat{w}_j, \tag{29}$$

$$(T_{m+1,m} - {\sigma_j^{(\text{it-n})}}^2 I_{m+1,m})^T \hat{w}_j = \hat{\sigma}\!\downarrow_j^{(\text{it-n})} v_j^{(\text{it-n})}, \tag{30}$$

and since ${\sigma_j^{(\text{it-n})}}^2 = {v_j^{(\text{it-n})}}^T B_m^T B_m v_j^{(\text{it-n})}$ we have from (29) $[v_j^{(\text{it-n})}; 0]^T \hat{w}_j = 0$. Analogous to equations (26)-(27) with iterative refined Ritz singular approximation (25) we have,

$$AP_m v_j^{(\text{it-n})} = Q_m B_m v_j^{(\text{it-n})} \;=\; \sigma_j^{(\text{it-n})} Q_m u_j^{(\text{it-n})} \tag{31}$$

$$\begin{aligned} A^T Q_m u_j^{(\text{it-n})} &= P_m B_m^T u_j^{(\text{it-n})} + f e_m^T u_j^{(\text{it-n})} \qquad (32)\\ &= \sigma_j^{(\text{it-n})} P_m v_j^{(\text{it-n})} + \hat{\sigma}\!\downarrow_j^{(\text{it-n})} / \sigma_j^{(\text{it-n})} \left[P_m \; p_{m+1}\right] \hat{w}_j \end{aligned}$$

and after multipling (31) by A^T

$$A^T A P_m v_j^{(\text{it-n})} = {\sigma_j^{(\text{it-n})}}^2 P_m v_j^{(\text{it-n})} + \hat{\sigma}\!\downarrow_j^{(\text{it-n})} \left[P_m \; p_{m+1}\right] \hat{w}_j. \tag{33}$$

Applying [2, Eqns. (5.5) and (5.12)] to Lanczos relationships (28) and (33) shows that

$$\hat{\sigma}\!\downarrow_j^{(\text{it-n})} = resNor_j^{(\text{it-n})} \leq resNor_j^{(\text{rf-n})} \leq resNor_j^{(\text{rz})}. \tag{34}$$

Equation (34) shows that the iterative refined Ritz with respect to the normal residual on the same Krylov subspace $\mathbb{K}_m(A^TA, p_1)$ are better approximations, however an effective restart process that "improves" the next generated Krylov subspace is still needed. Equations (26)-(28) and (31)-(33) show that the refined Ritz and iterative refined Ritz vectors, respectively, are not all multiples of the same residual vector, see [2, Thm. 4.3] in context of Lanczos factorization and the symmetric eigenvalue problem. Therefore the thick–restarted scheme presented in Section 2 is not available. However, one can still explicitly restart the GKL algorithm with a linear combination. We first utilize that the approximations are taken from basis vectors and perform a single iteration of the GKL algorithm that avoids a matrix–vector product with A and A^T as follows.

$$\begin{array}{l}
\text{1. Given } \bar{v} = \sum_{j=1}^{k} c_j v_j^{(..)} \text{ set } \beta_0 = \|\bar{v}\| \text{ and } \bar{v} = \bar{v}/\beta_0 \\
\text{2. Let } \bar{u} = B_m\bar{v} \text{ set } \alpha_1 = \|\bar{u}\| \text{ and } \bar{u} = \bar{u}/\alpha_1 \\
\text{3. Set } f = P_m(B_m^T\bar{u} - \alpha_1\bar{v}) + f e_m^T\bar{u} \text{ and } \beta_1 = \|f\| \\
\text{4. Set } p_1 = P_m\bar{v},\ p_2 = f/\beta_1,\ q_1 = Q_m\bar{u}
\end{array} \tag{35}$$

The steps in (35) yield the following 1-GKL factorization

$$Ap_1 = q_1\alpha_1\,, \tag{36}$$

$$A^Tq_1 = \left[p_1\,,\, p_2\right]\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix}, \tag{37}$$

where GKL method can be restarted with p_2. It is worth noting for $k = 1$ and $\bar{v} = v_1^{(\text{rf-n})}$ or $\bar{v} = v_1^{(\text{it-n})}$, equations (36)-(37) are the same as equations (26)-(27) or (31)-(32), respectively. For $k > 1$ the coefficients c_j in (35) can be chosen several ways and greatly impact convergence. For example, for eigenvalue problems Saad [33] suggests using residual norms which was also used for the refined Ritz algorithm [15, Alg. 1]. In [2] an alternate approach for iterative refined vectors modeled after Morgan [29] was used to eliminate part of the residual vector as the next Krylov subspace is built. Morgan [29] showed that for Ritz vectors and carefully chosen constants c_j that parts of the residual vector is eliminated when multiplied by A in the next iteration to build out the Krylov subspace, which resulted in the same final subspace as when implementing Sorensen's implicitly restarted method [35]. Unfortunately, this equivalence is not present here, though not all is lost. It turns out that we can still eliminate part of the residual. This requires solving a small $(k-1)\times k$ homogeneous system of equations (38) for coefficients c_j

$$\begin{bmatrix} e_m^T v_1^{(\text{it-n})} & \dots & e_m^T v_k^{(\text{it-n})} \\ {\sigma_1^{(\text{it-n})}}^{2(i-2)} e_m^T B_m^T B_m v_1^{(\text{it-n})} & \dots & {\sigma_k^{(\text{it-n})}}^{2(i-2)} e_m^T B_m^T B_m v_k^{(\text{it-n})} \end{bmatrix} i > 1, \tag{38}$$

we refer the reader to [2, Sec. 6] for details.

3.2 Refined and Iterative Refined on augmented system

We now turn our attention to developing notions of refined and iterative refined Ritz values/vectors on the augmented system. We start by letting $M = C$ and $\mathcal{W} = \mathbb{K}_{2m}(C, [0; p_1])$ in equation (18) and define

$$T_{2m+1,2m} := \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \\ \beta_m e_m^T & 0 \end{bmatrix} \in \mathbb{R}^{(2m+1)\times 2m}. \tag{39}$$

For each initial eigenvalue approximation μ_j of C compute the smallest singular value $\sigma\downarrow_j^{\text{(rf-a)}}$ and associated unit singular vectors of $(T_{2m+1,2m} - \mu_j I_{2m+1,2m})$,

$$(T_{2m+1,2m} - \mu_j I_{2m+1,2m}) \begin{bmatrix} x_j \\ y_j \end{bmatrix} = \sigma\downarrow_j^{\text{(rf-a)}} \begin{bmatrix} w_{x_j} \\ w_{y_j} \\ w_{z_j} \end{bmatrix}, \tag{40}$$

$$(T_{2m+1,2m} - \mu_j I_{2m+1,2m})^T \begin{bmatrix} w_{x_j} \\ w_{y_j} \\ w_{z_j} \end{bmatrix} = \sigma\downarrow_j^{\text{(rf-a)}} \begin{bmatrix} x_j \\ y_j \end{bmatrix}, \tag{41}$$

where $x_j, y_j, w_{x_j}, w_{y_j} \in \mathbb{R}^m$ and w_{z_j} is a scalar. Then it follows that

$$\min_{\substack{z_j \in \mathbb{K}_{2m}(C,[0;p_1]) \\ \|z_j\|=1}} \|Cz_j - \mu_j z_j\| = \|(T_{2m+1,2m} - \mu_j I_{2m+1,2m}) \begin{bmatrix} x_j \\ y_j \end{bmatrix}\| = \sigma\downarrow_j^{\text{(rf-a)}} \tag{42}$$

and the *refined Ritz vector* z_j for μ_j is defined as $z_j := [\,Q_m x_j\,;\,P_m y_j]$. Analogous to the case of the normal system, the approximate eigenvalue of C associated with refined Ritz vector z_j is selected as the Rayleigh quotient

$$\sigma_j^{\text{(rf-a)}} = z_j^T C z_j = \begin{bmatrix} x_j \\ y_j \end{bmatrix}^T \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \end{bmatrix} \begin{bmatrix} x_j \\ y_j \end{bmatrix} = 2x_j^T B_m y_j\,, \tag{43}$$

and the approximate *refined singular triplet on the augmented system* for A is given by

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{\text{(rf-a)}}, Q_m u_j^{\text{(rf-a)}}, P_m v_j^{\text{(rf-a)}}\}\,, \tag{44}$$

where $u_j^{\text{(rf-a)}} = x_j/\|x_j\|$ and $v_j^{\text{(rf-a)}} = y_j/\|y_j\|$. Similar to (28) for the normal system, but this time applied to the Lanczos factorization (10) for the augmented system C, we have the following equality

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} Q_m x_j \\ P_m y_j \end{bmatrix} = \sigma_j^{\text{(rf-a)}} \begin{bmatrix} Q_m x_j \\ P_m y_j \end{bmatrix} + \sigma\downarrow_j^{\text{(rf-a)}} \begin{bmatrix} Q_m & 0 & 0 \\ 0 & P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} r_{xj} \\ r_{yj} \\ r_{zj} \end{bmatrix}, \tag{45}$$

where $r_{zj} = w_{z_j}$ is a scalar, $r_{yj} = w_{yj} - [x_j\,;\,y_j]^T [w_{xj}\,;\,w_{y_j}] y_j \in \mathbb{R}^m$, and $r_{xj} = w_{xj} - [x_j\,;\,y_j]^T [w_{xj}\,;\,w_{y_j}] x_j \in \mathbb{R}^m$. Given the relationship between the eigenvalues of C and the singular values of A, we can start Algorithm 1 with the initial approximation μ_j in equations (40)-(42) as the Ritz value $\sigma_j^{(\text{rz})}$. This now gives us an approximate *iterative refined Ritz singular triplet on the augmented system* for A as

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{(\text{it-a})}, Q_m u_j^{(\text{it-a})}, P_m v_j^{(\text{it-a})}\}. \tag{46}$$

For convenience, consider the unscaled output vectors of $u_j^{(\text{it-a})}$ and $v_j^{(\text{it-a})}$ from Algorithm 1 as the last iteration vectors $\hat{x}_j := x_j^{(i)}$ and $\hat{y}_j := y_j^{(i)}$, respectively. Therefore, analogous to (29)-(30) and (33) we have the output from Algorithm 1 that satisfies

$$(T_{2m+1,2m} - \sigma_j^{(\text{it-a})} I_{2m+1,2m}) \begin{bmatrix} \hat{x}_j \\ \hat{y}_j \end{bmatrix} = \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \begin{bmatrix} \hat{w}_{x_j} \\ \hat{w}_{y_j} \\ \hat{w}_{z_j} \end{bmatrix} \tag{47}$$

$$(T_{2m+1,2m} - \sigma_j^{(\text{it-a})} I_{2m+1,2m})^T \begin{bmatrix} \hat{w}_{x_j} \\ \hat{w}_{y_j} \\ \hat{w}_{z_j} \end{bmatrix} = \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \begin{bmatrix} \hat{x}_j \\ \hat{y}_j \end{bmatrix}, \tag{48}$$

where $[\hat{x}_j\,; \hat{y}_j]^T\; [\hat{w}_{x_j}\,; \hat{w}_{y_j}] = 0$, $\hat{x}_j, \hat{y}_j, \hat{w}_{x_j}, \hat{w}_{y_j} \in \mathbb{R}^m$, and $\hat{w}_{z_j}$ is a scalar and when applied to the Lanczos factorization (10) gives us the following

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} Q_m \hat{x}_j \\ P_m \hat{y}_j \end{bmatrix} = \sigma_j^{(\text{it-a})} \begin{bmatrix} Q_m \hat{x}_j \\ P_m \hat{y}_j \end{bmatrix} + \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \begin{bmatrix} Q_m & 0 & 0 \\ 0 & P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} \hat{w}_{x_j} \\ \hat{w}_{y_j} \\ \hat{w}_{z_j} \end{bmatrix}. \tag{49}$$

Similar to (34), the relationships (45) and (49) together with [2, Eqns. (5.5) and (5.12)] applied to symmetric matrix C imply that

$$\hat{\sigma}_{\downarrow j}^{(\text{it-a})} = resAug_j^{(\text{it-a})} \leq resAug_j^{(\text{rf-a})} \leq resAug_j^{(\text{rz})}. \tag{50}$$

Equation (50) shows that the iterative refined Ritz with respect to the augmented residual on the same Krylov subspace $\mathbb{K}_{2m}(C, [0; p_1])$ are better approximations. But relation (50) is derived with respect to the unscaled vectors $x_j, y_j, \hat{x}_j, \hat{y}_j$. Unlike the singular vectors computed from the eigenvectors of C, the norms $\|x_j\|$, $\|y_j\|$ $\|\hat{x}_j\|$, and $\|\hat{y}_j\|$ are not necessarily equal to the common value $1/\sqrt{2}$, especially during the onset of the overall routine. However, these norms do approach $1/\sqrt{2}$ as approximations improve and we use it as a part of a convergence requirement in Algorithm 1. This requirement is reasonable by observing that from the iterative process of Algorithm 1 and equations (39), (40), and (43) it follows that

$$x_j^{(i)} = 1/\sigma_j^{(\text{rf-a})\,(i-1)} \left(B_m y_j^{(i)} - \sigma_{\downarrow j}^{(\text{rf-a})\,(i)} w_{x_j}^{(i)} \right). \tag{51}$$

When the iterative refine process converges and $\hat{x}_j := x_j^{(i)}$, then we have $\sigma_j^{(\text{rf-a})\,(i-1)} = \sigma_j^{(\text{rf-a})\,(i)} = \sigma_j^{(\text{it-a})} = 2\hat{x}_j^T B_m \hat{y}_j$ and

$$\hat{x}_j = 1/\sigma_j^{\text{(it-a)}}\left(B_m\hat{y}_j - \hat{\sigma}_{\downarrow j}^{\text{(it-a)}}\hat{w}_{x_j}\right), \tag{52}$$

$$\|\hat{x}_j\|^2 = 1/2 - \hat{\sigma}_{\downarrow j}^{\text{(it-a)}}/\sigma_j^{\text{(it-a)}}\hat{x}_j^T\hat{w}_{x_j}. \tag{53}$$

If $\hat{\sigma}_{\downarrow j}^{\text{(it-a)}} = 0$, then we have the desired property and convergence (see (50)). If $\hat{\sigma}_{\downarrow j}^{\text{(it-a)}} \neq 0$, then from (40) and (43) we have the relationship $\hat{x}_j^T\hat{w}_{x_j} = -\hat{y}_j^T\hat{w}_{y_j}$. After multiplying (47) by $[\hat{w}_{x_j}\,;\,0\,;\,0]^T$ and using $B_m\hat{w}_{x_j} - \sigma_j^{\text{(it-a)}}\hat{w}_{y_j} = \hat{\sigma}_{\downarrow j}^{\text{(it-a)}}\hat{y}_j$ from (48), we obtain

$$|\|\hat{x}_j\|^2 - 1/2| = (\hat{\sigma}_{\downarrow j}^{\text{(it-a)}}/\sigma_j^{\text{(it-a)}})^2|\|\hat{w}_{x_j}\|^2 - \|\hat{y}_j\|^2|/2 \;\leq\; (\hat{\sigma}_{\downarrow j}^{\text{(it-a)}}/\sigma_j^{\text{(it-a)}})^2, \tag{54}$$

where the inequality is established using the triangle inequality and the fact that $\|\hat{w}_{x_j}\| < 1$ and $\|\hat{y}_j\| < 1$. Through numerical examples, we have found that including $|\|x_j^{(i)}\| - 1/\sqrt{2}| \leq \sqrt{eps}$ with the convergence test in step 16 in Algorithm 1 resulted in a better performance in our hybrid algorithm for the augmented system.

Remark 2 We make the following observation from an asymptotic point of view of the iterative refined Ritz values/vectors on the augmented system. As the overall routine converges, it is expected for $\hat{\sigma}_{\downarrow j}^{\text{(it-a)}}$ in (49) to approach 0. As $\hat{\sigma}_{\downarrow j}^{\text{(it-a)}} \to 0$, from (52)-(54) we have that $\|\hat{x}_j\| \approx \|\hat{y}_j\| \approx 1/\sqrt{2}$, $u_j^{\text{(it-a)}} \approx 1/\sigma_j^{\text{(it-a)}}B_m v_j^{\text{(it-a)}}$, and $\sigma_j^{\text{(it-a)}} \approx \|B_m v_j^{\text{(it-a)}}\|$. Moreover, we start to see the residual relation (50) holding on the normalized vectors and the alignment with the iterative refined Ritz values/vectors on the normal system. Therefore, we use formulas (35) with $v_j^{(..)} := v_j^{\text{(it-a)}}$ to obtain the 1-GKL factorization (36)-(37) where GKL method can be restarted with p_2. Likewise, when $k > 1$, we can replace $v_j^{(..)} := v_j^{\text{(it-a)}}$ and $\sigma_j^{(..)} := \sigma_j^{\text{(it-a)}}$ and solve the homogeneous system (38) to restart with a linear combination of vectors. Although an alignment is eventually expected, there are convergence differences, see the numerical examples in Section 5.

We close this section with an example that illustrates that even though the refined and iterative refined values/vectors yield a "smaller" residual norm on the same Krylov subspace than Ritz values/vectors restarting with these "better" vectors in presence of small m value may not always yield a "better" Krylov subspace on the next iteration.

Example 1 For this and the subsequent example, we consider the diagonal matrix $A = \text{diag}(1{:}500)$ and the 262111×262111 matrix $A = \text{amazon0302}$ from [7]. We let $k = 1$ and $m = 2$ and search for the largest singular triplet with tolerance 10^{-6} while using (17) as a stopping criteria. For both matrices, we started by computing 2-GKL factorization with a random vector p_1, and then on the next restart p_1 was computed to be Ritz vector $P_m v_1^{\text{(rz)}}$, refined Ritz on normal system $P_m v_1^{\text{(rf-n)}}$, iterative refined Ritz on normal system $P_m v_1^{\text{(it-n)}}$, refined Ritz on augmented system $P_m v_1^{(\text{rf-a})}$, or iterative refined Ritz on augmented system $P_m v_1^{(\text{it-a})}$. For both matrices, we ran all five restart methods 10 times with a different random starting vector p_1. For each restart method, we chose to compute only the common Ritz norm, $resAug_1^{(\text{rz})}$, as a way to make the comparison easier, but also because the focus here is on measuring the overall convergence, i.e., the quality of the Krylov subspaces.

The results are presented in Figures 1a-1b which display the number of matrix–vector products (mvp) with A and A^T against $resAug_1^{(\text{rz})}$. From Figures 1a-1b it is evident that there is a wide range of convergence while the iterative refined values/vectors which yield a "smaller" residual norm, (50), demonstrate poor convergence or stagnation. Moreover, both figures show that the all refined methods are struggling at the beginning, especially with the amazon0302 matrix (see Figure 1b). This suggest that on a small subspace the refined methods are having difficulty capturing the needed components of the desired singular vector for restarting. Section 4 shows how this can be overcome. Although not displayed, and as expected, when we increased the value of m the differences between routines became smaller with all routines converging, e.g., for the diagonal matrix, when $m = 10$ all routines converged between about 300 and 380 matrix–vector products.

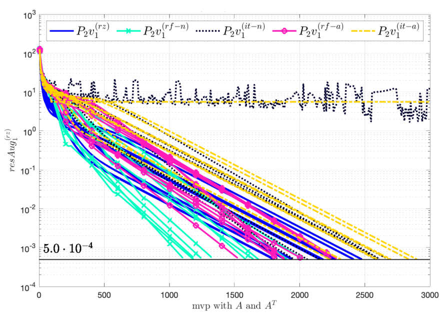
4 Hybrid Iterative Refined Algorithms

The poor convergence and stagnation reported for iterative refined Ritz vectors in Example 1 can be explained in part that the calculations of iterative refined Ritz vectors are more sensitive to converging to the next closest Ritz value during the iteration process. It is true that the refined Ritz also exhibit this behavior, but to a much lesser extent - causing slight jumps in residual curves at the beginning. This sensitivity of iterative refined Ritz vectors is the key for developing a hybrid method by signaling when the iterative refined vectors should *not be used* to restart the system.

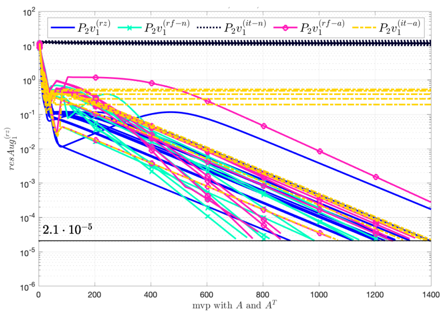
This now brings us to our first hybrid method for computing largest singular triplets which uses thick–restarting with Ritz vectors and when certain criteria are met it switches to restarting with iterative refined Ritz vectors on the normal or the augmented system.

The parameters to switch between thick–restarting and restarting with iterative refined vectors were chosen based on numerous experiments across a variety of problems. A careful balance is needed, since on the one side the iterative refined Ritz vectors can give a better approximation but with possible stagnation, while on the other side thick–restarted is a more efficient restarting scheme, but with not as good of approximations. Therefore, we first check the angle via the inner product between the desired iterative refined vector and the Ritz vector to determine that the refined process did not cause the vectors to deviate too far from each other. If the angle is acceptable, we use iterative refined Ritz vector(s) to restart. Numerous experiments suggest using

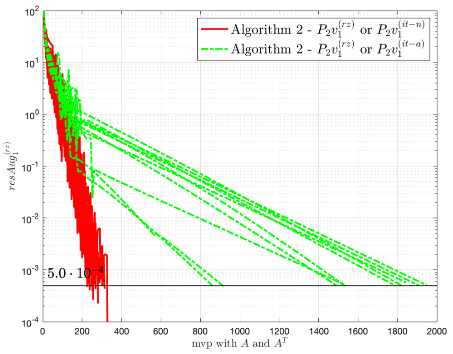
$$\min_{1\le j\le k} |v_j^{(\text{rz})^T} v_j^{(..)}| > 0.9\,, \tag{55}$$



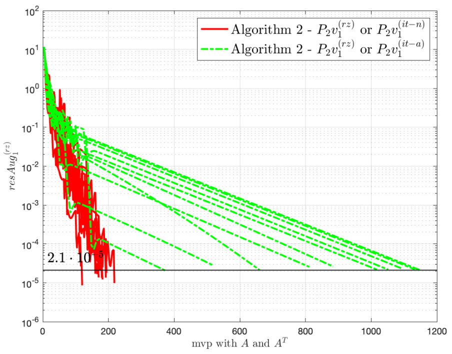
(a) Example 1: $A = \mathrm{diag}(1\!:\!500)$



(b) Example 1: $A =$ amazon0302 [7]



(c) Example 2: $A = \mathrm{diag}(1\!:\!500)$



(d) Example 2: $A =$ amazon0302 [7]

Fig. 1 Examples 1-2: Each line represents a start with a random vector and then a restart using the stated vector in the legend.

where $v_j^{(..)} := v_j^{(\text{it-n})}$ for the normal system and $v_j^{(..)} := v_j^{(\text{it-a})}$ for the augmented system. Although we have not encountered the following situation in practice, it is worth noting that it is possible that a Ritz vector may not have any accuracy from the same subspace even though the refined vector is arbitrarily close to the desired eigenvector, see [18, 23]. Since thick–restarted is the main routine with theoretical connection to implicitly restarted techniques and foundation for publicly available software, it is reasonable to assume that as the sequence of generated Krylov subspaces changes on each new iteration that the Ritz approximations will also change and improve.

Secondly, in order to ensure convergence and avoid missing singular triplets ($k > 1$), we also require the input value μ_j into Algorithm 1 to be the best approximation for singular value of A over all computed $\sigma_j^{(\text{rz})}$'s values thus far and to reject using restarting with iterative refined Ritz vectors if the current computed iterative refined values, $\sigma_j^{(\text{it-n})}$ or $\sigma_j^{(\text{it-a})}$, are not "better" than the past iteration's best approximation. For example, during a current iteration (iter) of Algorithm 2 we require in step 5 for the call to Algorithm 1 that

$$\mu_j = \max_{1 \leq \mathrm{i} \leq \mathrm{iter}} |\sigma_j^{(\mathrm{rz})^{(i)}}| \quad \text{for} \quad 1 \leq j \leq k \tag{56}$$

and for step 6

$$|\sigma_j^{(..)^{(\mathrm{iter})}}| \geq \max_{1 \leq \mathrm{i} \leq \mathrm{iter\text{-}1}} |\sigma_j^{(\mathrm{rz})^{(i)}}| \quad \text{for} \quad 1 \leq j \leq k\,, \tag{57}$$

where $\sigma_j^{(..)} := \sigma_j^{(\text{it-n})}$ for the normal system and $\sigma_j^{(..)} := \sigma_j^{(\text{it-a})}$ for the augmented system. When $k = 1$ we found that using (56) was a needed requirement for the best results, but encountered poor convergence results when enforcing (57) with $m = 2$. Additionally, due to a negligible computational cost, various convergence checks are performed at different stages of Algorithm 2, e.g., see steps 4, 7, and 13 – this allows for Algorithm 2 to exit at the right time and to avoid performing unnecessary expensive computations.

We note to the reader that Algorithm 2 is a simplification of the actual computations performed. For instance, in the thick–restarted step 14 in Algorithm 2 we compute s-GKL factorization where $s \geq k$ before restarting. The technique of including additional vectors ($> k$) is a very common strategy to accelerate the convergence in restarted methods. Similarly a gap strategy can also be used to accelerate the convergence by avoiding shifting too close to the desired spectrum. For example, in the implicitly shifted Lanczos bidiagonalization schemes, a relative gap strategy can be used to enhance convergence, see [6, 20, 21, 26] for details. Considering the connection between implicitly shifting with Ritz and thick–restarting, a simple gap strategy can also be used when deciding on adding additional vectors. We implemented the following straightforward and effective strategy for choosing $s \geq k$,

$$\begin{array}{l} s = k + n_c; \\ \text{if} \;\; \sigma_s - \sigma_{s+1} < \sigma_{s-1} - \sigma_s, \;\; s = s+1; \;\; \text{end} \\ s = \max(\text{floor}((m + n_c)/2), s); \\ \text{if} \;\; s >= m, \;\; s = m - 1; \;\; \text{end} \end{array} \tag{58}$$

where n_c is the number of converged singular triplets, see [39] for details and comparison of techniques. The strategy in (58) works well in this context, particularly when difference between k and m is kept relatively small. When restarting with iterative refined Ritz vectors, relations (58) were too aggressive and rarely satisfied the requirements (55) and (57) for all $s > k$ and therefore we always use k iterative refined Ritz vectors for restarting. However, using k iterative refined Ritz vectors to restart can cause an unfortunate increase in the residual norms measured by Ritz values/vectors, particularly when $k > 1$. This can be seen in part as negating the idea of the gap strategy mentioned above. Consequently, we do not restart consecutively with iterative refined Ritz vectors if the last restart with iterative refined Ritz vectors caused the residual norm of Ritz vectors/values to increase from the previous iteration.

Algorithm 2 Hybrid: Thick—Restarted – Restarted SVDS `(trrsvds)`

```
 1: Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating matrix-vector products with $A$ or $A^T$,
          $m$ : maximum size of GKL factorization,
          $k$ : number of desired singular triplets,
          $p_1$ : unit vector,
          $tol$ : tolerance for accepting computed approximate singular triple, cf. (17).
 2: Output: $k$ approximate singular triples $\{\sigma_j, u_j, v_j\}_{j=1}^{k}$ of $A$.
 3: Compute $m$-GKL factorization (5)-(6) with $B_m$ as in (7) or (14);
 4: Compute the SVD of $B_m$ (11) and check $1 \leq j \leq k$ (17) with (15);
 5: Compute $\{\sigma_j^{(..)}, u_j^{(..)}, v_j^{(..)}\}_{j=1}^{k}$ by Algorithm 1 with $\mu_j$ (56) for either the augmented
    system or the normal system;
 6: if all $\sigma_j^{(..)}$ converged and satisfy (55) and (57) then
 7:    Check $1 \leq j \leq k$ (17) with (25) or (46);
 8:    if $k > 1$ then
 9:       Compute $c_j$ from (38);
10:    end if
11:    Compute 1-GKL factorization (36)-(37);
12: else
13:    Check $1 \leq j \leq k$ (17) with (44) and $\mu_j$ (56);
14:    Compute $s$-GKL factorization (12)-(13) where $k \leq s < m$;
15: end if
16: Goto 3;
```

Example 2 This is a continuation of Example 1 and uses the same test matrices and parameters, except that now we use Algorithm 2 on two hybrid methods, restarting with $P_m v_1^{(\mathrm{rz})}$ and $P_m v_1^{(\text{it-n})}$ (iterative refined Ritz on normal system) and $P_m v_1^{(\mathrm{rz})}$ and $P_m v_1^{(\text{it-a})}$ (iterative refined Ritz on augmented system). Just as in Example 1 for both test matrices, we ran all hybrid methods 10 times with a different random starting vector p_1.

In Figures 1c-1d we collect the results, where the graphs display the number of matrix–vector products (mvp) with A and A^T against $resAug_1^{(\mathrm{rz})}$ for all routines. More specifically, for $A = \mathrm{diag}(1\!:\!500)$, Figure 1c shows that our hybrid method with iterative refined Ritz on normal system always converged between 210 and 315 matrix–vector products with respect to $resAug_1^{(\mathrm{rz})}$, compared to Example 1 where the best result is 1100 matrix–vector products. Similarly, for $A =$ amazon0302, Figure 1d shows the hybrid method with iterative refined Ritz on normal system always converged between 125 and 205 matrix–vector products with respect to $resAug_1^{(\mathrm{rz})}$ while comparable computation in Example 1 required about 700 matrix–vector products. This clearly illustrates that Algorithm 2 restarting with $P_m v_1^{(\mathrm{rz})}$ and $P_m v_1^{(\text{it-n})}$ performed significantly better than all restarted methods in Example 1. Furthermore, we emphasize that in comparison to Example 1, Algorithm 2 avoided stagnation which was one of the motivating factors for its development.

Remark 3 We note that in the context of Example 2, if iterative refined Ritz vectors were replaced with refined Ritz vectors in Algorithm 2, then we saw almost no performance increases over the results in Example 1 for restarting with refined Ritz vectors. This is attributed in part to the angle criteria (55) for switching being almost always satisfied, a similar observation was made in the context of eigenvalue computations in [2, Examples 5.3 and 6.2].

We conclude this section with a discussion of our second hybrid scheme, Algorithm 3, which can be viewed as a simple ($\approx$ 100 lines of MATLAB code) yet powerful variant of Algorithm 2. Motivated by the performance of Algorithm 2 in Example 2, for Algorithm 3 we use the standard restarted process (no thick–restarted techniques) where we fix the basis size at $m = 2$ and restart with either an iterative refined Ritz vector on the normal system, $P_m v_1^{(\text{it-n})}$, or a Ritz vector, $P_m v_1^{(\text{rz})}$. This has the added advantage of reducing the overall complexity and computational cost beyond matrix–vector products, namely not needing to reorthogonalize the basis vectors, a gap strategy (58), or solving homogeneous system (38) when $k > 1$, which potentially can become numerically ill-conditioned, see [2, Section 6]. Also, Algorithm 3 uses the "smallest" input matrix in the iterative scheme in Algorithm 1 further reducing the non–matrix–vector product computational cost.

Algorithm 3 requires a deflation strategy when computing $1 < k < m$ singular triplets. For the deflation ($k > 1$), our technique is simple and heavily motivated by the discussion in [36] – when singular vectors have been determined to converge, they are locked and not modified again while at the same time all subsequent computed basis vectors are orthogonalized against them (see step 4 in Algorithm 3). In our implementation, if the k largest singular triplets are to be computed subject to the user–specified tolerance tol, then the first $k-1$ singular triplets are computed and deflated with the tolerance $tol^{(d)} = 10^{-1} \cdot tol$. In comparison to the discussion in [36, Section 9] of the cascading approach, our choice of $tol^{(d)}$ is more restrictive when $k \leq 8$. Given that this paper primarily focuses on computing a small number of singular triplets, the choice of $tol^{(d)}$ for deflating vectors is reasonable, simple to implement, yet highly effective as evidenced by all numerical results in Section 5. It is worth noting that a more involved deflation procedure might be needed when k is larger or deflation fails, e.g., singular triplets can not be computed within the user-specified tolerance. For an outline of some alternative approaches to deflation we refer the reader to [36], while a more comprehensive discussion of deflation can be found in [34,37].

For the sake of completeness, we note that when deflation is performed in Algorithm 3 the computation of the residual in steps 7 and 13 is not straightforward and requires using the inner products from the Gram–Schmidt process between the converged singular vectors and the basis vectors in the GKL process. The monitoring of the inner products also permits an easy detection of a "locking problem" see [36, Lemma 1]. Therefore, in step 13 in Algorithm 3 we check that all residuals satisfy the user-specified tolerance tol. We note that in all numerical examples in Section 5, we used $tol^{(d)}$ and did not encounter any "locking problems" for small values of k.

Remark 4 Algorithm 2 which requires $k < m$ does not require any deflation or locking procedure for the handling of $k > 1$ singular triplets. That is, it implements a “non-locking” method [36], where singular triplets are updated on every iteration. This is the same method used in [4] and for a small number of desired singular triplets with a “reasonable” convergence tolerance is a very effective method. Implementation of additional locking strategies for either Algorithm 2 or Algorithm 3 is outside of the scope of this paper. Note that no significant advantages are expected when only a small number of singular triplets is desired, see remarks in [36,37].

Algorithm 3 Hybrid: Restarted Deflation (2×2) SVDS `(rd2svds)`

```
1: **Input:** $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating matrix-vector products with $A$ or $A^T$,
          $k$ : number of desired singular triplets,
          $p_1$ : unit vector,
          $tol$ : tolerance for accepting computed approximate singular triples, cf. (17),
          $tol^{(d)} < tol$ : tolerance for deflating $< k$ singular triples, cf. (17).
2: **Output:** $k$ approximate singular triples $\{\sigma_j, u_j, v_j\}_{j=1}^{k}$ of $A$.
3: $j := 1$;
4: Compute 2-GKL factorization where for $i = 1, 2, \dots, (j-1)$
   $P_2^T v_i = 0$, $f^T v_i = 0$, and $Q_2^T u_i = 0$;
5: Compute the largest singular triplet $\{\sigma_1^{(\text{rz})}, u_1^{(\text{rz})}, v_1^{(\text{rz})}\}$ of $B_2$ (11);
6: Compute $\{\sigma_1^{(\text{it-n})}, u_1^{(\text{it-n})}, v_1^{(\text{it-n})}\}$ by Algorithm 1 with $\mu_1$ (56);
7: **if** $j < k$ and (17) is satisfied with $tol^{(d)}$ using either (25) or (44) with $\mu_1$ (56); **then**
8:     $\{\sigma_j, u_j, v_j\}:=\{\sigma_1^{(\text{it-n})}, Q_2 u_1^{(\text{it-n})}, P_2 v_1^{(\text{it-n})}\}$ or $\{\sigma_j, u_j, v_j\}:=\{\sigma_1^{(\text{rf-a})}, Q_2 u_1^{(\text{rf-a})}, P_2 v_1^{(\text{rf-a})}\}$;
9:     Compute $f = f - (v_j^T f) v_j$;
10:    $p_1 := f/\|f\|$, $j := j + 1$;
11:    Goto 4;
12: **else**
13:    Check (17) with $tol$ using either (25) or (44) and $\mu_1$ (56);
14: **end if**
15: **if** $\sigma_1^{(\text{it-n})}$ converged and satisfies (55) **then**
16:    Compute 1-GKL factorization (36)-(37);
17: **else**
18:    Compute 1-GKL factorization (12)-(13);
19: **end if**
20: Goto 4;
```

5 Numerical Examples

In this section, we present MATLAB codes `trrsvds`[2] and `rd2svds`[2] which implement Algorithm 2 and Algorithm 3, respectively, along with several numerical examples that illustrate their performance. To that end, we compare our methods to six other routines: three publicly available MATLAB codes `irlba` [4][2,3], `svdifp` [28][4], and `GKD`[9][5], a publicly available MATLAB interfaced code `primme_svds`[41][6], and MATLAB's built–in functions `svds` and `eigs`, where

[2] Code available at: http://www.math.uri.edu/~jbaglama, retrieved on 4/10/22
[3] Code available at: http://www.netlib.org/numeralgo/na26.tgz, retrieved on 4/10/22
[4] Code available at: https://github.com/wildstone/SVDIFP, retrieved on 4/10/22
[5] Code available at: https://github.com/sgoldenCS/GKD, retrieved on 4/10/22
[6] Code available at: https://github.com/primme/primme, retrieved on 4/10/22

`eigs` is applied to the symmetric matrices A^TA and $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$ and the equivalent eigenvalue problems. Note that here we do not necessarily advocate using `eigs` on A^TA as a general purpose method nor do we choose comparison examples where it is known to perform poorly. We refer the reader to [37, Section 3.2] for a through investigation of using A^TA to compute singular triplets.

The MATLAB interfaced code `primme_svds` is part of a massive high performance C99 library PRIMME for computing eigenpairs and singular triplets and consists of numerous routines/techniques each with a different set of parameters. It is not possible for us to compare against all these options and thus we only provide a small sample of them while using default values for most of the parameters and only set the ones needed for fair comparison. More specifically, in all examples parameters are set to indicate that the problems are real and to use double precision. Also, the value *primme.method* is set to be the *default_min_matvecs*, since this is the measure we are comparing, and finally, the method is set to be *primme_svds_hybrid*. Likewise, the MATLAB `GKD` code also has many options and we continued to use the default parameter values except for *minRestart*. The default choice for *minRestart* caused the basis size to increase since *minRestart* must be less than than *maxBasis*. Because of such small basis sizes used in our examples we set *minRestart* to be equal to k and this provided very strong results for `GKD` as can be seen in Tables 3-5.

Routines `GKD`, `svdifp`, and `primme_svds` allow application of a preconditioner and can perform better when one is applied [9, 28, 41]. But the use of a preconditioner significantly increases the overall storage requirements, counter to this paper's primary goal of using as little storage as possible, and hence we do not apply a preconditioner. To quote the authors of [28], "`svdifp` without preconditioning is simply the restarted Lanczos method with the LOBPCG-type subspace enhancement."

The MATLAB code `irlba` implements a technique to include additional vectors for thick–restarted as a way of improving convergence, similar to our dynamic scheme (58). However, `irlba` instead utilizes a parameter *adjust* which is by default set at three and allows the parameter to internally increase by the number of converged singular triplets. If initially *adjust* and k exceed the size of basis, the basis size is increased. Because of this rigidity of parameter *adjust* at the start and the fact that in all of our examples the Lanczos basis is restricted to be as small as possible, we set *adjust* to zero instead of its default value three.

Now we turn our attention to `trrsvds` whose description of parameters and their default values are given in Table 1. To illustrate the different methods available for `trrsvds` via the parameter choices we use the notation `trrsvds([nor,aug])`. The first entry is either `nor` for the normal equations in the hybrid method to compute the iterative refined Ritz pairs (25) or `aug` for the augmented equations in the hybrid method to compute the iterative refined Ritz pairs (46).

With respect to reorthogonalization, `trrsvds` implements either one-sided full reorthogonalization or two-sided full reorthogonalization. If A is deter-

Table 1 The user specific parameters for `trrsvds`.

k	Number of desired singular values. Default: $k = 1$.
m	Number of Lanczos vectors. Default: $m = 2$ or $m = 15$ if $sigma$ ='SS' .
$maxit$	Maximum number of restarts. Default: $maxit = 2000$.
$maxitref$	Maximum number of iterations to find iterative refined Ritz singular values, see Algorithm 1. Default: $maxitref = 100$.
$method$	('nor', 'aug') which method to use. Default: $method$ ='nor'.
$reorth$	('one' or 'two') sided full reorthogonalization. Default: $reorth$ = 'one'.
$sigma$	('LS' or 'SS') location of singular values. Default: $sigma$ = 'LS'.
tol	Tolerance for convergence, (17). Default: $tol = \sqrt{eps}$.
p_1	Starting vector. If $\ell > n$ and $sigma$ ='SS' then $p_1 \in \mathbb{R}^\ell$ else $p_1 \in \mathbb{R}^n$. Default: $p_1 = randn(n, 1)$.

mined to be ill-conditioned, by monitoring the minimum and maximum singular values of B_m, then two-sided full reorthogonalization is used. Examples presented in this section with `trrsvds`, one-sided and two-sided full reorthogonalization yield about the same accuracy, and so we do not report both. It should be noted that the full reorthogonalization strategy increases the overall computational times when Lanczos basis is increased. Unlike `trrsvds`, reorthogonalization is not used in `rd2svds` since only one-step of the GKL process is used to build 2-GKL factorization. The routines `rd2svds` and `trrsvds` with basis size of only two vectors ($m = 2$) using hybrid method with normal equations and searching only for the largest singular triplets are mathematically equivalent, but they are slightly numerically different (see the results as reported in Examples 3-4 when $k = 1$ and $m = 2$).

For the purpose of comparing codes, we limit our analysis to either using the default values for the parameters or set the parameters so that they the provide fairest comparison with respect to our proposed methods. For all codes, we set the following common parameters: number of desired singular triplets k, a common random starting vector $p_1 \in \mathbb{R}^n$ or $[0\,; p_1] \in \mathbb{R}^{n+\ell}$ for the augmented system $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$, tolerance $tol = 10^{-6}$, and Lanczos basis maximum size m. Instead of a starting vector, routines `GKD`, `svdifp`, and `primme_svds` use an input matrix, and thus, for those routines we set the first column to be the common starting vector p_1 and the rest of the columns are set to be common among those three routines.

In regards to the other parameters, we set the tolerance for `svdifp` to be $tol \cdot \|A\|_2$. This parameter choice provided the same order of magnitude of the residuals computed by `svdifp` as well as the other routines in Examples 3-4. With respect to a common basis size similar to m in `trrsvds`, we identify the parameter in the other methods that represent the “storage” or basis size. Depending on a routine and a coding style, this parameter may be restricted (e.g., `eigs(C)` and `svds` require $m \geq k + 2$) or additional storage may be included for calculations. We assume that for all methods the parameter that represents “storage” is comparable to the basis size m in `ttrsvds` and is therefore represented by m in and Tables 3-5. However, given the complexities and propriety of some of the codes this may not always be the case.

Remark 5 Let $k > 1$ be an arbitrary but fixed number of desired singular triplets. Recall that `ttrsvds` computes those k triplets with respect to the basis parameter $m \geq k$ and requires storage of $2m + 1$ vectors. On the other hand, executing steps 4–19 in Algorithm 3 requires storage of 5 vectors, namely p_1, p_2, q_1, q_2, and f, that get constantly overwritten. `rd2svds` also requires an additional storage for $2(k - 1)$ converged left and right singular vectors. In case when $m = k + 1$ in `ttrsvds`, then `ttrsvds` and `rd2svds` have the same storage requirements, making them the most suitable for a direct comparison. Thus, we report results for `rd2svds` in Tables 4-5 under the size $m = k + 1$.

In all examples and for all codes except `svdifp`, matrices A and A^T are only accessed by calling a function whose inputs are x and a parameter designating which matrix-vector product, Ax or $A^T x$, is to be the output. `svdifp` requires user to input the matrix A. The recorded value mvp in the examples is the total number of times Ax and $A^T x$ are computed. When the augmented system C (8) is used, to save memory space, it is never explicitly formed; the input vector is split and the calculation is only performed on Ax and $A^T x$. All numerical examples were carried out using MATLAB version R2021a on a MacBook Pro 2.6 GHz 6-Core Intel Core i7 processor and 16 GB (2667 MHz) of memory using operating system macOS Big Sur. Machine epsilon is $\epsilon = 2.2 \cdot 10^{-16}$. In Tables 3-5, "N/A" is used to denote that the method is **n**ot **a**vailable for the specified choice of parameters, "N/R" stands for **n**ot **r**ecorded and is used when a method alters parameters making it unfair for comparison, and finally "N/C" denotes the routine did **n**ot **c**onverge in the allotted (default) number of iterations – note that we did not modify the parameters to get the routine to work (e.g., increase the default setting for maximum number of iterations). The recorded cpu times displayed in Tables 3-5 are in seconds and recorded using MATLAB's tic-toc command. Here we note that since `primme_svds` is a MATLAB interfaced code, the recorded times are expected to be less than the all MATLAB syntax codes. Finally, it is worth highlighting that the performance of the methods in our comparisons also depends on the machine architecture, the author's coding style, the design/purpose of the routines, and numerical implementation. Our MATLAB codes included here are only an illustration of the presented methods and the comparison is only meant to show the methods in this paper are competitive to other existing routines.

Example 3 In this example we investigate the performance of routines `ttrsvds` and `rd2svds` when computing k largest singular triplets of six different matrices, where $k = 1, 2, 3, 4$ and m is varied from $(k+1)$ to $(k+3)$. More specifically, we compare performance of `rd2svds`, `trrsvds(nor)`, and `trrsvds(aug)`, with the methods `eigs(C)`, `eigs(`$A^T A$`)`, `irlba`, `svdifp`, `svds`, `primme_svds`, and `GKD`. The test matrices we used for the comparison are $A = \text{diag}(1{:}500)$ and the five matrices listed in Table 2 from the SuiteSparse Matrix Collection [7].

The mvp and cpu times are displayed in Tables 4-5 for different combinations of k and m. It is easy to see from the Tables 4-5 that our proposed routines are competitive. Moreover, Tables 4-5 also demonstrate that all of our three methods have converged for all m and k values – particularly of note is the case when $m = k + 1$ in which case `rd2svds` performs excellently, while that was not even an option for majority of the other routines.

In summary, given a wide range in sizes of the test matrices, together with the varied proximity among the largest singular values (see Table 2), Example 3 shows that the methods developed in this paper are particularly competitive when using small m relative to the number of desired singular triplets k.

Table 2 Test matrices used for the examples from the SuiteSparse Matrix Collection [7]

Matrix	illc1033	JP	amazon0302	Rucci1	relat9
# Rows	1033	87616	262111	1977885	12360060
# Cols	320	67320	262111	109900	549336
Non-zeros	4719	13734559	1234877	7791168	38955420
Kind	Least Squares	Tomography	Directed Graph	Least Squares	Combinatorial
σ_1	2.1444	4223.1	21.218	7.0687	21.626
σ_2	2.1042	4019.3	21.136	6.9853	20.417
σ_3	2.0855	3872.8	20.027	6.9625	18.666
σ_4	2.0574	3819.2	19.277	6.8895	18.61

Example 4 For our final example, we compute the largest singular triplet for the matrix kmerV1r, currently the second largest in order in the SuiteSparse Matrix Collection [7] (kmerV1r is a square matrix with 214005017 rows and 465410904 nonzero entries). This is also one of the largest matrices that was able to be loaded into MATLAB allowing all of the routines to successfully compute the largest singular triplet and has pushed the bounds of the machine architecture used. Table 3 displays the results for computing the largest singular triplet of kmerV1r with $m = 2, 3$. The largest singular value was computed by all routines as $\sigma_1 = 6.5035$ within the desired tolerance. As seen in Table 3, for $m = 2$ our MATLAB codes `ttrsvds` and `rd2svds` all converged within 45 minutes, the fastest, `rd2svds`, converging in about 31 minutes making it highly competitive with the other routines.

6 Conclusions

This paper extends the hybrid concept in [2] recently applied to the symmetric eigenvalue problem to the GKL process. The new restarted hybrid GKL methods combine thick–restarting with Ritz vectors or with a judiciously chosen linear combination of iterative refined Ritz vectors. Numerical examples show our methods to be competitive with other publicly available codes, particularly when there are limited memory requirements.

Acknowledgements The authors are thankful to the anonymous referee whose suggestions helped improve the paper. The work of Vasilije Perović was partially supported by the Faculty Career Enhancement Grant 2020-2021 through the University of Rhode Island.

Table 3 Example 4: mvp counts and total cpu times for computing the largest singular triplet for the matrix kmerV1r with $m = 2, 3$. For $m = 2$ the method `GKD` increased m to $m = 3$ and hence is not reported N/R.

Method	m	mvp	cpu
`rd2svds`	2	72	1867s
	–	–	–
`trrsvds(nor)`	2	66	2032
	3	82	2939s
`trrsvds(aug)`	2	80	2698s
	3	66	2287s
`eigs(C)`	N/A	–	–
	3	274	22686s
`eigs(`A^TA`)`	N/A	–	–
	3	91	1868s

Method	m	mvp	cpu
`irlba`	2	138	4554s
	3	90	3122s
`svdifp`	2	115	9276s
	3	81	7669
`svds`	N/A	–	–
	3	206	15091s
`primme_svds`	N/A	–	–
	3	64	1975s
`GKD`	N/R	–	–
	3	58	5282s

Declarations

The authors declare that they have no conflict of interest. Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

References

1. Alter, O., Brown, P.O., Botstein, D.: Singular value decomposition for genome-wide expression data processing and modeling. Proceedings of the National Academy of Sciences **97**(18), 10101–10106 (2000)
2. Baglama, J., Bella, T., Picucci, J.: Hybrid iterative refined method for computing a few extreme eigenpairs of a symmetric matrix. SIAM Journal on Scientific Computing **43**(5), S200–S224 (2021).
3. Baglama, J., Kane, M., Lewis, B., Poliakov, A.: Efficient thresholded correlation using truncated singular value decomposition. arXiv preprint arXiv:1512.07246 (2015)
4. Baglama, J., Reichel, L.: Augmented implicitly restarted Lanczos bidiagonalization methods. SIAM Journal on Scientific Computing **27**(1), 19–42 (2005)
5. Baglama, J., Reichel, L.: An implicitly restarted block Lanczos bidiagonalization method using Leja shifts. BIT Numerical Mathematics **53**(2), 285–310 (2013)
6. Baglama, J., Richmond, D.J.: Implicitly restarting the LSQR algorithm. Electronic Transactions on Numerical Analysis **42**, 85–105 (2014)
7. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software **38**(1), 1–25 (2011)
8. Eldén, L.: Matrix methods in data mining and pattern recognition. SIAM (2007)
9. Goldenberg, S., Stathopoulos, A., Romero, E.: A Golub–Kahan Davidson method for accurately computing a few singular triplets of large sparse matrices. SIAM Journal on Scientific Computing **41**(4), A2172–A2192 (2019)
10. Golub, G., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis **2**(2), 205–224 (1965)
11. Golub, G.H., Van Loan, C.F.: Matrix Computations, fourth edn. The Johns Hopkins University Press (2013)
12. Hochstenbach, M.E.: Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems. BIT Numerical Mathematics **44**(4), 721–754 (2004)

13. Hochstenbach, M.E.: Generalizations of harmonic and refined Rayleigh-Ritz. Electronic Transactions on Numerical Analysis **20**, 235–252 (2005)
14. Hochstenbach, M.E., Sleijpen, G.L.: Harmonic and refined Rayleigh–Ritz for the polynomial eigenvalue problem. Numerical Linear Algebra with Applications **15**(1), 35–54 (2008)
15. Jia, Z.: Refined iterative algorithms based on Arnoldi's process for large unsymmetric eigenproblems. Linear Algebra and its Applications **259**, 1–23 (1997)
16. Jia, Z.: Polynomial characterizations of the approximate eigenvectors by the refined Arnoldi method and an implicitly restarted refined Arnoldi algorithm. Linear Algebra and its Applications **287**(1-3), 191–214 (1999)
17. Jia, Z.: The refined harmonic Arnoldi method and an implicitly restarted refined algorithm for computing interior eigenpairs of large matrices. Applied Numerical Mathematics **42**(4), 489–512 (2002)
18. Jia, Z.: Some theoretical comparisons of refined Ritz vectors and Ritz vectors. Science in China Series A: Mathematics **47**(1), 222–233 (2004)
19. Jia, Z.: The convergence of harmonic Ritz values, harmonic Ritz vectors and refined harmonic Ritz vectors. Mathematics of Computation **74**(251), 1441–1456 (2005)
20. Jia, Z., Niu, D.: An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition. SIAM Journal on Matrix Analysis and Applications **25**(1), 246–265 (2003)
21. Jia, Z., Niu, D.: A refined harmonic Lanczos bidiagonalization method and an implicitly restarted algorithm for computing the smallest singular triplets of large matrices. SIAM Journal on Scientific Computing **32**(2), 714–744 (2010)
22. Jia, Z., Stewart, G.W.: An analysis of the Rayleigh–Ritz method for approximating eigenspaces. Mathematics of Computation **70**(234), 637–647 (2001)
23. Jiang, W., Wu, G.: A thick-restarted block Arnoldi algorithm with modified Ritz vectors for large eigenproblems. Computers & Mathematics with Applications **60**(3), 873–889 (2010)
24. Jolliffe, I.: Principal component analysis. Wiley Online Library (2005)
25. Kokiopoulou, E., Bekas, C., Gallopoulos, E.: Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization. Applied Numerical Mathematics **49**, 39–61 (2004). DOI 10.1016/j.apnum.2003.11.011
26. Larsen, R.: Combining implicit restart and partial reorthogonalization in Lanczos bidiagnalization, 2001
27. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. SIAM (1998)
28. Liang, Q., Ye, Q.: Computing singular values of large matrices with an inverse-free preconditioned Krylov subspace method. Electronic Transactions on Numerical Analysis **42**, 197 (2014)
29. Morgan, R.B.: On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. Mathematics of Computation **65**(215), 1213–1230 (1996)
30. Niu, D., Yuan, X.: An implicitly restarted lanczos bidiagonalization method with refined harmonic shifts for computing smallest singular triplets. Journal of Computational and Applied Mathematics **260**, 208–217 (2014)
31. Olney, A.M.: Large-scale latent semantic analysis. Behavior research methods **43**(2), 414–423 (2011)
32. Paige, C.C., Saunders, M.A.: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software (TOMS) **8**(1), 43–71 (1982)
33. Saad, Y.: Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. Linear Algebra and its Applications **34**, 269–295 (1980)
34. Saad, Y.: Numerical Methods for Large Eigenvalue Problems: Revised Edition. Society for Industrial and Applied Mathematics (2011)
35. Sorensen, D.C.: Implicit application of polynomial filters in a k-step Arnoldi method. SIAM Journal on Matrix Analysis and Applications **13**(1), 357–385 (1992)
36. Stathopoulos, A.: Locking issues for finding a large number of eigenvectors of hermitian matrices. Tech. rep., Citeseer (2005)
37. Stewart, G.W.: Matrix Algorithms: Volume II: Eigensystems. Society for Industrial and Applied Mathematics (2001). DOI 10.1137/1.9780898718058

38. Stoll, M.: A Krylov–Schur approach to the truncated SVD. Linear Algebra and its Applications **436**(8), 2795–2806 (2012)
39. Wu, K., Simon, H.: Dynamic restarting schemes for eigenvalue problems. Tech. rep., Lawrence Berkeley National Lab., CA (US) (1999)
40. Wu, K., Simon, H.: Thick-restart Lanczos method for large symmetric eigenvalue problems. SIAM Journal on Matrix Analysis and Applications **22**(2), 602–616 (2000)
41. Wu, L., Romero, E., Stathopoulos, A.: Primme_svds: A high-performance preconditioned svd solver for accurate large-scale computations. SIAM Journal on Scientific Computing **39**(5), S248–S271 (2017)

Table 4 Example 3: mvp counts and total cpu times for matrices diag(1:500), illc1033, and JP. For $m = k + 1$ the methods `eigs(C)`, `eigs(`$A^T A$`)`, and `svds` were N/A and the method `GKD` was N/R since it increased m to $m = k + 2$, therefore those methods are omitted in the table when $m = k + 1$. `rd2svds` is only reported for $m = k + 1$.

	diag(1:500)				illc1033				JP			
	m=k+1											
Method	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4
`rd2svds`	276 0.07s	412 0.04s	686 0.06s	796 0.06s	120 0.03s	192 0.03s	298 0.03s	416 0.03s	54 0.80s	144 2.20s	212 3.30s	278 4.30s
`trrsvds(nor)`	286 0.15s	2172 0.39s	1006 0.23s	1086 0.32s	114 0.10s	436 0.13s	304 0.09s	208 0.05s	58 1.03s	136 2.33s	292 5.08s	214 3.80s
`trrsvds(aug)`	1390 0.17s	964 0.25s	946 0.35s	1192 0.55s	126 0.03s	282 0.06s	252 0.08s	570 0.19s	70 1.14s	94 1.55s	264 4.58s	202 3.57s
`irlba`	N/C	N/C	N/C	N/C	364 0.04s	708 0.05s	424 0.03s	728 0.05s	122 1.97s	148 2.37s	436 7.32s	210 3.54s
`svdifp`	423 0.05s	546 0.03s	801 0.02s	1042 0.02s	223 0.03s	306 0.02s	353 0.01s	422 0.01s	83 1.42s	132 2.22s	209 3.52s	282 4.78s
`primme_svds`	N/A	N/A	N/A	458 0.02s	N/A	N/A	N/A	188 0.01s	N/A	N/A	N/A	130 2.11s

	diag(1:500)				illc1033				JP			
	m=k+2											
Method	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4
`trrsvds(nor)`	310 0.05s	1102 0.13s	442 0.07s	612 0.13s	106 0.03s	228 0.04s	168 0.03s	164 0.04s	66 1.06s	94 1.60s	130 2.24s	94 1.57s
`trrsvds(aug)`	310 0.03s	622 0.10s	422 0.11s	504 0.16s	106 0.02s	154 0.03s	154 0.04s	124 0.04s	54 0.88s	82 1.33s	164 2.90s	86 1.49s
`eigs(C)`	N/C	N/C	N/C	N/C	690 0.07s	528 0.03s	584 0.05s	406 0.02s	242 4.08s	292 4.98s	558 10.1s	300 9.67s
`eigs(`$A^T A$`)`	1147 0.03s	706 0.02s	693 0.02s	N/C	199 0.01s	168 0.01s	185 0.01s	120 0.004s	79 1.26s	94 1.48s	149 2.39s	98 1.55s
`irlba`	1146 0.07s	960 0.06s	674 0.04s	1132 0.07s	198 0.02s	206 0.02s	162 0.01s	142 0.01s	78 1.22s	78 1.26s	118 1.94s	80 1.32s
`svdifp`	291 0.01s	478 0.02s	719 0.01s	960 0.02s	147 0.01s	262 0.02s	309 0.01s	408 0.01s	69 1.16s	126 2.12s	209 3.53s	252 4.46s
`svds`	N/C	N/C	N/C	N/C	N/C	N/C	N/C	212 0.02s	206 3.36s	208 3.42s	210 3.57s	202 3.50s
`primme_svds`	218 0.02s	N/A	404 0.02s	454 0.02s	118 0.01s	N/A	148 0.01s	176 0.01s	48 0.80s	N/A	106 1.74s	118 1.88s
`GKD`	212 0.04s	321 0.06s	368 0.05s	435 0.07s	112 0.03s	113 0.03s	130 0.03s	153 0.03s	42 0.73s	69 1.16s	94 1.68s	107 2.02s

	diag(1:500)				illc1033				JP			
	m=k+3											
Method	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4
`trrsvds(nor)`	386 0.05s	772 0.08s	402 0.05s	392 0.06s	112 0.02s	116 0.02s	148 0.02s	128 0.02s	58 0.96s	84 1.35s	102 1.72s	80 1.34s
`trrsvds(aug)`	406 0.06s	412 0.05s	354 0.07s	386 0.10s	102 0.01s	154 0.02s	166 0.03s	158 0.04s	54 0.87s	78 1.25s	78 1.32s	82 1.39s
`eigs(C)`	N/C	1732 0.06s	1270 0.04s	N/C	504 0.03s	402 0.02s	384 0.02s	286 0.01s	140 2.35s	188 3.19s	278 5.59s	166 2.82s
`eigs(`$A^T A$`)`	681 0.02s	506 0.01s	417 0.01s	654 0.02s	149 0.01s	134 0.004s	131 0.004s	112 0.003s	49 0.78s	72 1.13s	97 1.55s	80 1.25s
`irlba`	800 0.04s	680 0.03s	486 0.02s	426 0.02s	146 0.01s	158 0.01s	136 0.01s	130 0.01s	62 0.97s	70 1.11s	84 1.38s	70 1.17s
`svdifp`	251 0.01s	446 0.01s	681 0.03s	908 0.02s	123 0.003s	216 0.01s	297 0.03s	390 0.01s	59 0.99s	126 2.12s	189 3.36s	278 5.02s
`svds`	N/C	N/C	N/C	N/C	210 0.02s	208 0.02s	214 0.02s	236 0.02s	164 2.72s	212 3.63s	176 3.01s	136 2.29s
`primme_svds`	178 0.01s	242 0.02s	386 0.02s	366 0.02s	64 0.01s	104 0.01s	144 0.01s	156 0.01s	46 0.77s	64 1.04s	102 1.64s	110 1.75s
`GKD`	208 0.03s	325 0.05s	352 0.05s	421 0.05s	92 0.02s	113 0.02s	130 0.03s	161 0.03s	38 0.68s	69 1.20s	102 1.90s	101 1.80s

Table 5 Example 3: mvp counts and total cpu times for matrices amazon0302, Rucci1, and relat9. For $m = k + 1$ the methods `eigs(C)`, `eigs(`A^TA`)`, and `svds` were N/A and the method `GKD` was N/R since it increased m to $m = k+2$, therefore those methods are omitted in the table when $m = k + 1$. `rd2svds` is only reported for $m = k + 1$.

	amazon0302				Rucci1				relat9			
	m=k+1											
Method	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4
`rd2svds`	148 0.64s	220 0.97s	290 1.40s	318 1.90s	126 3.40s	194 5.70s	290 9.50s	438 17.0s	66 20.3s	114 42.0s	242 96.0s	386 170s
`trrsvds(nor)`	158 0.75s	136 0.83s	232 1.96s	114 1.11s	132 4.64s	978 41.6s	234 11.0s	916 64.0s	60 21.4s	104 38.0s	1216 677s	1088 729s
`trrsvds(aug)`	888 4.16s	104 0.70s	168 1.46s	136 1.48s	166 5.78s	846 37.7s	346 20.1s	1000 72s	76 27.5s	90 38.5s	1036 592s	1460 981s
`irlba`	1236 6.27s	150 1.15s	186 1.57s	160 1.92s	456 14.1s	1280 56.8s	636 34.2s	1330 81.0s	102 35.8s	102 46.6s	1442 803s	N/C
`svdifp`	179 1.37s	168 1.36s	233 2.10s	342 3.40s	183 9.20s	414 20.5s	489 25.2s	592 31.8s	79 40.2s	138 71.1s	441 245s	542 314s
`primme_svds`	N/A	N/A	N/A	148 1.06s	N/A	N/A	N/A	288 5.44s	N/A	N/A	N/A	224 58.8s

	amazon0302				Rucci1				relat9			
	m=k+2											
Method	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4
`trrsvds(nor)`	138 0.64s	86 0.49s	144 1.01s	92 0.81s	122 3.56s	306 10.1s	230 9.31s	540 40.5s	62 19.9s	74 27.0s	640 413s	490 354s
`trrsvds(aug)`	142 0.65s	78 0.48s	142 1.00s	86 0.86s	114 3.39s	210 9.91s	222 9.24s	482 34.9s	58 19.0s	64 27.1s	764 512s	402 297s
`eigs(C)`	N/C	290 2.18s	218 1.88s	342 4.04s	886 31.2s	N/C	N/C	N/C	202 82.3s	208 100s	N/C	N/C
`eigs(`A^TA`)`	651 2.46s	94 0.43s	85 0.43s	104 0.63s	255 3.26s	212 2.76s	365 5.23s	612 8.90s	71 16.5s	68 15.8s	N/C	464 109s
`irlba`	650 3.18s	92 0.61s	94 0.77s	92 0.93s	254 6.92s	264 9.08s	320 15.1s	596 38.0s	70 22.5s	66 25.7s	926 564s	464 314s
`svdifp`	105 0.85s	158 1.41s	259 2.53s	312 4.68s	141 6.99s	334 17.3s	439 23.7s	552 44.9s	69 35.6s	134 72.0s	389 225s	492 418s
`svds`	206 1.32s	208 1.64s	210 1.98s	212 2.28s	N/C	N/C	N/C	N/C	150 63.4s	208 104s	N/C	N/C
`primme_svds`	96 0.46s	N/A	110 0.80s	146 0.98s	98 1.74s	N/A	202 3.84s	292 5.20s	46 11.8s	N/A	180 47.3s	226 56.6s
`GKD`	90 0.74s	73 0.89s	98 1.44s	105 1.77s	92 4.94s	171 11.9s	194 15.3s	307 29.8s	40 24.0s	63 43.6s	326 260s	221 200s

	amazon0302				Rucci1				relat9			
	m=k+3											
Method	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4	k=1	k=2	k=3	k=4
`trrsvds(nor)`	110 0.62s	74 0.42s	96 0.65s	76 0.77s	118 3.52s	232 8.35s	262 9.30s	348 20.7s	52 16.7s	62 21.9s	348 233s	328 251s
`trrsvds(aug)`	84 0.47s	72 0.44s	76 0.56s	88 0.73s	126 3.85s	202 6.85s	158 5.83s	350 23.5s	58 20.0s	60 21.7s	338 238s	320 251s
`eigs(C)`	284 2.14s	222 1.59s	182 1.49s	218 2.04s	572 22.8s	624 24.6s	724 31.9s	1160 58.6s	128 55.9s	150 65.2s	N/C	988 539s
`eigs(`A^TA`)`	89 0.41s	76 0.33s	79 0.38s	84 0.44s	177 2.29s	188 2.59s	209 2.95s	356 5.02s	49 11.4s	60 14.1s	513 119s	268 62.7s
`irlba`	446 2.22s	76 0.48s	80 0.60s	76 0.69s	188 4.94s	210 6.48s	210 7.53s	206 8.94s	56 17.4s	60 21.5s	322 206s	332 237s
`svdifp`	99 0.89s	186 1.84s	237 3.55s	306 5.08s	123 6.26s	296 16.0s	405 32.8s	530 46.0s	67 35.8s	126 70.5s	345 292s	502 455s
`svds`	188 1.50s	166 1.54s	192 2.13s	412 3.44s	N/C	N/C	N/C	410 18.3s	126 61.7s	212 115s	N/C	412 199s
`primme_svds`	64 0.35s	76 0.48s	112 0.75s	122 0.85s	100 1.80s	148 2.75s	202 3.80s	230 4.33s	46 12.0s	66 17.7s	164 42.9s	182 47.7s
`GKD`	80 0.67s	73 0.76s	96 1.50s	105 1.52s	92 4.93s	171 11.4s	190 14.7s	283 24.2s	40 23.4s	63 41.6s	258 189s	231 207s