ITERATIVE METHODS FOR COMPUTING A FEW EIGENPAIRS OR SINGULAR TRIPLETS OF LARGE SPARSE MATRICES

Jennifer R. Picucci
University of Rhode Island, jenniferpicucci@uri.edu

Follow this and additional works at: https://digitalcommons.uri.edu/oa_diss

Recommended Citation

This Dissertation is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Dissertations by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons@etal.uri.edu.
ITERATIVE METHODS FOR COMPUTING A FEW EIGENPAIRS OR
SINGULAR TRIPLETS OF LARGE SPARSE MATRICES

BY

JENNIFER R. PICUCCI

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
APPLIED MATHEMATICS

UNIVERSITY OF RHODE ISLAND
2021
DOCTOR OF PHILOSOPHY DISSERTATION

OF

JENNIFER R. PICUCCI

APPROVED:

Dissertation Committee:

Major Professor    James Baglama

Vasilije Perović

Richard Vaccaro

Brenton DeBouf
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2021
ABSTRACT

This thesis presents new hybrid restarted Lanczos methods for computing eigenpairs and singular triplets of large matrices. Our methods combine thick-restarting with Ritz or harmonic Ritz vectors with iteratively refined Ritz vectors to compute a few of the extreme eigenpairs of symmetric matrices or singular triplets of rectangular matrices. The refined process improves the (harmonic) Ritz values/vectors yielding better approximations, i.e., this process results in a “smaller” residual norm compared to just using Ritz/harmonic vectors. The iterative refined process we developed improves the refined values/vectors by using a scheme, where we replace the approximate eigenvalue/singular value in the original refined scheme with the latest computed refined Ritz value until convergence. The thick-restarting schemes are superior in reference to efficiency to other restarted schemes, but are not available when using refined or iterative refined Ritz vectors. Therefore, we developed hybrid restarted methods that switch between the efficient thick-restarted scheme and restarting with a linear combination of “the better approximating” iterative refined Ritz vectors. Our developed methods have shown to be very effective on small subspaces, i.e., when memory is limited. We provide many theoretical results and numerical examples.
ACKNOWLEDGMENTS

I would first like to thank my amazing thesis advisor, Dr. James Baglama. I cannot express enough gratitude for Dr. Baglama’s patience and encouragement throughout the past few years in order to complete this research while I was working full time. I know I have frequently been pulled in more directions than anticipated which delayed our research more than planned.

For their time and commitment to my success, I would also like to thank my doctoral committee. I am grateful to Dr. Vasilije Perović for not only serving on my committee, but also for being my graduate Linear Algebra instructor here at URI. Dr. Perović was an enjoyable lecturer on the subject and passed his enthusiasm for the subject to anyone in attendance. I am also very appreciative of Dr. Richard Vaccaro from the Electrical Engineering Department for accepting my request to serve on my committee and Dr. Christopher Baxter, from the Ocean Engineering Department, for agreeing to be the chair of my doctoral defense. Lastly, I would like to thank Dr. Tom Bella for assisting in the writing and development of the first journal article in this thesis.

Returning back to school after working in government research for 10 years was harder than I ever imagined. I would not have succeeded without the friends I discovered in the Mathematics graduate Department while here at URI. These individuals helped me through every stage of this process, especially Eric Peterson and Sarah Van Beaver in joining forces to study for our comprehensive exams. They provided not only sounding boards for discussion but many many hours of much needed laughs as well. It goes without saying that this thesis was an enormous adventure to take on while working; as such, this is an achievement I am very proud of accomplishing. This effort would not have been possible without everyone in my life who has helped me along the way, my Fellowship of the Ring.

I am very thankful for the opportunity to complete Long Term Training through
my work at the U.S. Army Corps of Engineers, Engineer Research and Development Center. This allowed me to take a full year away from work responsibilities to complete my course work and oral examination. Without the ability to focus solely on academic responsibilities I don’t believe I could have accomplished everything necessary to finish my research and graduate. My friends Mihan and Oliver Taylor, Leigh and Neil Williams, and Pam Kinnebrew from ERDC were a constant support holding me up when the effort threatened to knock me down.

Last, but never least, I need to thank my family. They endured constant explanations and discussions into mathematics that gave them headaches. They patiently supported me when I stayed up all night to write and doodled matrices and equations on napkins at parties. I know this entire process seemed crazy to them but they helped me along solely because it was important to me. I cannot begin to tell them how much that means to me.
PREFACE

This thesis will be presented in manuscript format. In Chapter 1 we provide the motivation and goals for this research along with a synopsis of the following chapters. Chapter 2 is the first manuscript, published on May 4, 2021 to the SIAM Journal on Scientific Computing. Chapter 3 is the second manuscript submitted July 30, 2021 to Numerical Algorithms. Chapter 4 provides the conclusions for this research and Appendix A contains the MATLAB codes used for numerical examples.

Dr. James Baglama was involved in the idea, design, and writing, of both research papers and codes. Dr. Tom Bella and Dr. Vasilije Perović assisted in providing ideas, comments and writing of the first and second research papers, respectively.

A list of references used for the respective manuscripts is provided at the end of each chapter.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>PREFACE</td>
<td>v</td>
</tr>
<tr>
<td>Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Lanczos Method</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Thick–Restarted Lanczos with Ritz vectors</td>
<td>11</td>
</tr>
<tr>
<td>2.4 Refined Ritz vectors</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Iterative Refined Ritz vectors</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Hybrid Methods</td>
<td>26</td>
</tr>
<tr>
<td>2.6.1 Hybrid Thick–Restarted and Iterative Refined Ritz Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>2.7 Numerical Examples</td>
<td>33</td>
</tr>
<tr>
<td>2.8 Conclusions</td>
<td>38</td>
</tr>
<tr>
<td>3 Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods</td>
<td>44</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>46</td>
</tr>
<tr>
<td>3.2 Thick-restarted GKL process with Ritz or harmonic Ritz vectors</td>
<td>52</td>
</tr>
<tr>
<td>3.2.1 Thick-restarting with Ritz vectors</td>
<td>52</td>
</tr>
<tr>
<td>3.2.2 Thick-restarting with harmonic Ritz vectors</td>
<td>53</td>
</tr>
<tr>
<td>3.2.3 GKL and thick-restarted GKL algorithms</td>
<td>57</td>
</tr>
<tr>
<td>3.3 Refined and Iterative Refined Ritz vectors</td>
<td>60</td>
</tr>
<tr>
<td>3.3.1 Refined and Iterative Refined on normal system</td>
<td>62</td>
</tr>
<tr>
<td>3.3.2 Refined and Iterative Refined on augmented system</td>
<td>68</td>
</tr>
<tr>
<td>3.4 Hybrid Iterative Refined Algorithms</td>
<td>73</td>
</tr>
<tr>
<td>3.5 Numerical Examples</td>
<td>82</td>
</tr>
<tr>
<td>3.6 Conclusions</td>
<td>93</td>
</tr>
<tr>
<td>4 Conclusions</td>
<td>111</td>
</tr>
<tr>
<td>A Appendix: MATLAB Code</td>
<td>113</td>
</tr>
<tr>
<td>A.1 MATLAB function trreigs(varargin)</td>
<td>113</td>
</tr>
<tr>
<td>A.2 MATLAB function rd2svds(A,m,n,O,k,tol)</td>
<td>133</td>
</tr>
<tr>
<td>A.3 MATLAB function trrsvds(varargin)</td>
<td>138</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example 2.6.1 output for Lin from the SuiteSparse Matrix Collection</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>Examples 2.5.3 and 2.6.2 output for all algorithms</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>Example 3.3.1 results</td>
<td>73</td>
</tr>
<tr>
<td>4</td>
<td>Example 3.4.1 results</td>
<td>79</td>
</tr>
<tr>
<td>5</td>
<td>Example 3.5.1.a (largest singular triplets) for $A = \text{diag}(1:500)$</td>
<td>94</td>
</tr>
<tr>
<td>6</td>
<td>Example 3.5.1.b (largest singular triplets) for $A = \text{diag}(1:500)$</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix illc1033</td>
<td>96</td>
</tr>
<tr>
<td>8</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix dwt_1242</td>
<td>97</td>
</tr>
<tr>
<td>9</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix well1850</td>
<td>98</td>
</tr>
<tr>
<td>10</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix pde2961</td>
<td>99</td>
</tr>
<tr>
<td>11</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix Kemelmacher</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix JP</td>
<td>101</td>
</tr>
<tr>
<td>13</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix amazon0302</td>
<td>102</td>
</tr>
<tr>
<td>14</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix Rucci1</td>
<td>103</td>
</tr>
<tr>
<td>15</td>
<td>Example 3.5.2 (largest singular triplets) for the matrix relat9</td>
<td>104</td>
</tr>
<tr>
<td>16</td>
<td>Example 3.5.4 (smallest singular values) for the matrix well1850</td>
<td>105</td>
</tr>
<tr>
<td>17</td>
<td>Example 3.5.4 (smallest singular values) for the matrix lp_ganges</td>
<td>106</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example 2.5.1 Ritz residual norms $\beta_3</td>
<td>e_3^T y_1</td>
</tr>
<tr>
<td>2</td>
<td>Example 2.5.2 Algorithm 3 results with $\mu_1 = \theta_1 = 2.1520$</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Numerical Examples for all presented codes</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>User specified parameters for $\texttt{trrsvds}$</td>
<td>83</td>
</tr>
<tr>
<td>5</td>
<td>Test matrices used for examples</td>
<td>87</td>
</tr>
<tr>
<td>6</td>
<td>Example 3.5.3 results</td>
<td>91</td>
</tr>
</tbody>
</table>
CHAPTER 1
Introduction

The motivation for this thesis pertains specifically to my job at the U.S Army Corps of Engineers, Engineer Research and Development Center (ERDC) where I am a research Mathematician. At ERDC, I have spent over 10 years working with a seismic sensor system and its data processing algorithms. These sensors use a classification method which looks for characteristics that eliminate each option and deem whatever is left as the activity detected. This can cause misclassifications for unknown items not in the system database. If those misclassifications are classified as a threat activity this creates the potential for unnecessary resources to be expended in order to investigate the event. For this reason, we were looking for a new way to add confidence bounds to each of the items in the classification database. This would allow for items too far outside that bound to be given the label unknown instead of whichever classification option was left at the end of the process.

The method we chose to explore the use of confidence bounds was that of Principal Component Analysis (PCA) by way of Partial Singular Value Decomposition (PSVD). PCA has been used in multiple fields to filter outliers of a population based on specific data features. For the seismic sensor, we sometimes have a delay in classification of vehicles if they are on a heavily rutted road as the potholes can seem like digging. With PCA these signals from vehicles even with potholes in the road should fall outside a chosen boundary line and can be correctly classified more quickly. But in order to implement PCA we needed to develop a fast PSVD method. As these system may not have sufficient storage capabilities, in addition to increased speed this new method is required to run using limited data and therefore, the focus of this thesis is on the development of a PSVD method.
While studying and developing a new PSVD method we came across an ability to add an iterative cycle to improve the refined scheme developed by Jia [2] for a faster method. We quickly realized that a more concrete theoretical basis was needed to justify our new algorithms. While developing these foundations we explored the connection of the SVD of a matrix and the equivalent symmetric eigenvalue problems. More specifically, if \( A \in \mathbb{R}^{m \times n} \), then \( A^T A \) is a symmetric matrix which we refer to as the normal system. The eigenvalues, \( \sigma_j^2 \) of \( A^T A \) are the squares of singular values of \( A \), while the associated eigenvectors, \( v_j \) of \( A^T A \) are the corresponding right singular vectors of \( A \). The left singular vectors, \( u_j \) are computed as \( u_j = \frac{1}{\sigma_j} Av_j \), \( \sigma_j \neq 0 \). A similar link can be found with the symmetric matrix \( C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \), which we refer to as the augmented system. For \( C \), the eigenvectors corresponding to eigenvalues \( \pm \sigma_j \) are \( \frac{1}{\sqrt{2}} [u_j^T, \pm v_j^T]^T \), where \( \{\sigma_j, u_j, v_j\} \) is a singular triplet of \( A \). These connections allowed us to use the simpler framework of the symmetric case to prove our iterative process converges and why it provides a “smaller” residual, i.e., better approximation, than that of the refined method but would not work very well when used as starting vectors for restarting methods. This led us to the development of the hybrid portion of our new method and its implementation.

Chapter 2 contains the journal article, *Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix*. In this paper, we first provide background on thick-restarted with Ritz vectors and the refined Ritz methods along with a new way to represent the refined residual vectors under the Lanczos relationship. We then present our new hybrid algorithm along with convergence results and justifications for the use of linear combinations of iterative refined Ritz vectors. Lastly in this paper, we provide numerical results to compare the new method to multiple other methods currently available.

In Chapter 3, we make the connection from symmetric eigenvalue problems to
PSVD and introduce the second journal article *Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods*. It extends the hybrid method from the first paper to rectangular matrices by utilizing the Golub Kahan Lanczos Bidiagonalization (GKLB) method for explicitly computing the PSVD. This journal article uses the theoretical foundation provided in the first article along with relationships of the normal and augmented systems to build the needed methods to develop algorithms that are useful in the future for my work applications. We conclude with numerical results. Chapter 4 follows with a Conclusion section summing up the results from the entire thesis. Finally, Appendix A includes the MATLAB codes used in the numerical examples.

**List of References**


CHAPTER 2

Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix

James Baglama, Tom Bella, and Jennifer Picucci
Department of Mathematics
University of Rhode Island, Kingston, Rhode Island 02881-0816, USA

This manuscript was published to the SIAM Journal on Scientific Computing on May 4, 2021 and can be found at https://doi.org/10.1137/20M1344834.

Keywords:
restarted Lanczos method, eigenvalue computation, singular value, refined Ritz, thick–restarted.

AMS
65F15, 65F50, 15A18
Abstract We developed a hybrid restarted Lanczos method that combines thick-restarting with Ritz vectors with iteratively refined Ritz vectors to compute a few of the extreme eigenvalues and associated eigenvectors of a large sparse symmetric matrix $A$. The iterative refined Ritz vectors use a scheme, where we replace the approximate eigenvalue in the original refined scheme with the latest computed refined Ritz value until convergence. The thick-restarting schemes have shown to be superior to most other schemes, particularly restarted schemes of linear combinations. However, the simple thick-restarting Lanczos scheme is not available when using refined or iterative refined Ritz vectors. Instead, we use a hybrid restarted scheme that switches between thick-restarted with Ritz vectors and restarting with a judiciously chosen linear combination of iterative refined Ritz vectors. We provide some theoretical results and several computed examples.
2.1 Introduction

Large sparse symmetric eigenvalue problems

\[ Ax = \lambda x \quad A \in \mathbb{R}^{n \times n} \] (1)

are some of the most important and profoundly studied areas in numerical linear algebra. Although these problems are numerically attractive, they can exhibit computational challenges for even the best modern routines, e.g. clustering of eigenvalues, matrix sizes (memory constraints), and orthogonality. The importance of these problems and computational challenges have spurred a considerable amount of research, e.g.\,[1, 2, 6, 7, 14, 22, 23, 27, 29, 30] and references within. The goal of this paper is twofold: the development of a new, simple algorithm that uses as little storage as possible to compute a few of the extreme eigenpairs of a large sparse symmetric matrix \( A \), and provide some insightful results on the (iterative) refined Ritz scheme. Since \( A \) is so large, we assume its factorization is not feasible and only the evaluation of matrix–vector products with the matrix \( A \) is available.

The motivation for the paper starts with the pioneering algorithm of Sorensen [22] for symmetric matrices called the Implicitly Restarted Lanczos (IRL) method (non–symmetric case is referred to as Implicitly Restarted Arnoldi (IRA) method). The IRL method is a restarted Krylov subspace method that implicitly modifies the starting vector \( p_1 \) with an accelerating polynomial. The accelerating polynomial is determined by a specific selection of zeros of the polynomial, also called shifts. There are several choices for shifts; the IRL method in [22] uses Ritz values as shifts. The choice of shifts is crucial for performance of the IRL method and there have been investigations into other choices, e.g. Leja points [1] and harmonic Ritz [16]. In 1997, Jia [6] introduced the concept of refined Ritz vectors. The idea is for a given approximate eigenvalue \( \mu \) of \( A \), to minimize \( \|Az - \mu z\| \) for a unit vector \( z \) from the current working Krylov subspace. Refined Ritz vectors often provide better eigenvector ap-
proximations than the Ritz vectors, see analysis \[9, 11\] for details. Since refined Ritz will be the basis of our new method, a thorough discussion is presented in Section 2.4. In [7], Jia applied the refined concept in combination with the IRA procedure, referred to as Implicitly Restarted Refined Arnoldi method (IRRA). Morgan \[15\] showed the equivalence of the IRA method with Ritz values as shifts with augmenting the Krylov subspace by certain Ritz vectors. Wu and Simon \[27\] exploited this idea in the symmetric case, and by using the property that all of the Ritz vectors are multiples of same residual vector, they created a simple augmented method, called the “Thick–Restarted Lanczos method.” The method is mathematically equivalent to the IRL method and avoids the need for the implicitly shifted QR algorithm. Similarly, Stewart \[26\] presents the Krylov–Schur method for the non–symmetric case and showed its equivalence to, and numerical superiority over, the IRA method.

The key feature needed with the thick–restarting Lanczos method with Ritz vectors is that the resulting space remains a Krylov subspace, which is possible since the Ritz vectors are all multiples of a single residual vector, \[15\]. The simple thick–restarting scheme by Wu and Simon \[27\] with refined Ritz vectors in place of Ritz vectors is not possible, because refined Ritz vectors are not all multiples of a single residual vector, see Theorem 2.4.3 in Section 2.4. Furthermore, as discussed in \[9, 11, 18\] on the relationship between refined Ritz and Ritz vectors they are not parallel unless refined Ritz vectors equal eigenvectors. We present a similar result for the context here, see Theorems 2.4.2 and 2.5.2.

To improve approximations to desired eigenpairs and decrease the refined Ritz residuals norm, we introduced an iterative scheme, where we replace the approximate eigenvalue in the refined scheme with the latest computed refined Ritz value until convergence, Section 2.5. This process has the added benefit of eliminating part of the refined Ritz residuals. The resulting residual vector has convenient qualities and a
“smaller” norm. However, like refined Ritz vectors, the iterative refined Ritz vectors are not all multiples of a single residual vector, see Section 2.5.

Morgan showed in [15] that the IRA method developed by Sorensen can be implemented by using a starting vector with a cleverly chosen linear combination of the desired Ritz vectors. We use a similar linear combination when restarting with the iterative refined Ritz vectors. The idea is to inherit similar, beneficial, restarting properties.

However, when using only refined Ritz or iterative refined Ritz vectors for restarting we observed examples of stagnation, erratic convergence, or very slow convergence. Slow convergence is exacerbated with restarted methods when using a low dimensional subspace and/or clustered eigenvalues, see Examples 5.2 and 5.3 in Section 2.5. Thick–restarting with Ritz vectors also exhibits very slow convergence when using a low dimensional subspace. Therefore, we implemented a hybrid method Section 2.6 that depends on certain criteria for switching from thick–restarting with Ritz vectors to restarting with a linear combination of iterative refined Ritz vectors. We observed through numerical experiments that although switching from thick–restarted Ritz to a linear combination of iterative refined results in a temporary undesirable spike in the norms of the residuals, this often relieves the stagnation/slow convergence, resulting in an overall faster convergence. This has the added benefit of being able to use a small Krylov subspace, see Figure 2.

Hybrid methods have been used previously for combining other forms of eigenvector approximations. For example, a comparable method is a block hybrid method that was proposed in [12] in which thick–restarting is performed by “modified” Ritz vectors, computed over a block Krylov subspace. Their block hybrid algorithm uses a power method with refined Ritz vectors “stitched” together with thick–restarting with modified Ritz vectors. Although absent with their code, we do provide an example
in Section 2.7 on the symmetric matrix experiment presented in their paper.

It should be noted that the Jacobi–Davidson method and extensions thereof are competitive for the computation of a few eigenvalues and associated eigenvectors, particularly when a known preconditioner is available; see [5, 21, 24, 23, 29] for descriptions of such methods. We do provide some comparisons in Section 2.7.

Throughout this paper $\| \cdot \|$ denotes the Euclidean vector norm or the associated induced matrix norm. When useful and for ease of presentation we will utilize MATLAB notation.

The subsequent parts of the paper are organized as follows. We begin with a background of restarted Lanczos method Section 2.2, thick–restaring with Ritz vectors in Section 2.3, and refined Ritz pairs in Section 2.4. Some theoretical results and relationships on refined Ritz pairs are included in Section 2.4. In Section 2.5 we describe our new strategy for iterative refined Ritz vectors and provide several theoretical results, motivation, and relationships. Our new hybrid method for computing the extreme eigenpairs is presented in Section 2.6. Numerical examples are presented in Section 2.7 and conclusions follow in Section 2.8.

2.2 Lanczos Method

For this discussion, we describe a Lanczos method that is modeled after the algorithms presented in [17, 27]. Given a unit vector $p_1$, we define the Krylov subspace

$$\mathbb{K}_m(A, p_1) = \text{span}\{p_1, Ap_1, A^2p_1, \ldots, A^{m-1}p_1\}.$$  \hspace{1cm} (2)

Application of $m$ steps of the MLan(0) Algorithm 2.1 given below with the initial starting vector $p_1$ applied to the symmetric matrix $A$ generates a sequence of $m$ orthogonal vectors $p_j \in \mathbb{R}^n$ that form an orthonormal basis for the Krylov subspace (2). This algorithmic process forms the well–known Lanczos decomposition,

$$AP_m = P_mT_m + f e^T_m.$$  \hspace{1cm} (3)
where the Lanczos vectors
\[ P_m = [p_1, p_2, \ldots, p_m] \in \mathbb{R}^{n \times m} \] (4)
satisfy \( P_m^T P_m = I_m \), and \( f \in \mathbb{R}^n \), referred to as the residual vector, satisfies \( P_m^T f = 0 \).

The matrix \( I_m \) denotes the identity matrix of order \( m \) and the vector \( e_m \in \mathbb{R}^m \) consists of the last column of \( I_m \). The matrix \( T_m \) is tridiagonal of the form
\[
T_m = \begin{bmatrix}
\alpha_1 & \beta_1 & 0 \\
\beta_1 & \alpha_2 & \beta_2 \\
& \ddots & \ddots & \ddots \\
0 & \cdots & \beta_{m-1} & \alpha_m \\
\end{bmatrix} \in \mathbb{R}^{m \times m}. \tag{5}
\]

For brevity we give a more general version of MLan(\( \ell \)) that will be used in Section 2.3.

**Algorithm 2.1 MLan(\( \ell \))**

1: **Input:** \( A \in \mathbb{R}^{n \times n} \): symmetric matrix,
\( p_1, \ldots, p_{\ell+1} \in \mathbb{R}^{n \times (\ell+1)} \): orthonormal vector(s) \( \square \),
\( T_{\ell+1} \in \mathbb{R}^{\ell+1 \times \ell+1} \): if \( \ell > 0 \) input matrix \( \square \),
\( m \): maximum number of Lanczos vectors.

2: **Output:** \( P_m = [p_1, p_2, \ldots, p_m] \in \mathbb{R}^{n \times m} \): orthogonal matrix \( \square \) or \( \square \),
\( T_m \in \mathbb{R}^{m \times m} \): if \( \ell = 0 \) tridiagonal matrix \( \square \), if \( \ell > 0 \) matrix \( \square \),
\( f \in \mathbb{R}^n \): residual vector.

3: **for** \( j = \ell + 1 : m \) **do**
4: \( f := Ap_j \)
5: **if** \( j > \ell + 1 \) **then** \( f := f - p_{j-1} \beta_j \)
6: **if** \( j = \ell + 1 \) and \( \ell > 0 \) **then** \( f := f - P_{\ell+1} T_{\ell+1}(\ell+1: \ell)^T \)
7: \( \alpha_j := f^T p_j \) and \( f := f - p_j \alpha_j \)
8: Reorthogonalization: \( f := f - P_j (P_j^T f) \)
9: **if** \( j < m \) **then** \( \beta_j := \|f\| \) and \( p_{j+1} := f / \beta_j \)

For our discussion, we will assume that we can perform \( m - \ell \) steps of Algorithm 2.1 to generate the Lanczos tridiagonal decomposition \( \square \) or, later, \( \square \). In practice, near-breakdowns can occur, i.e. \( \beta_j \approx 0 \) for some \( j \) (step 9 Algorithm 2.1), see strategies such as those described in \( \square \) to continue the Lanczos process. Since
m in our discussion is of a modest size and the desired number of eigenpairs k is small, a near-breakdown without convergence is rare and therefore, we will assume for the following discussion, $T_m$ is unreduced. The reorthogonalization step introduced in step 8 of Algorithm 2.1 is a simple strategy to help maintain orthonormality among independent vectors for modest values $m \ll n$. More robust reorthogonalization strategies, e.g. selective or partial are described in the literature, see e.g. [17, 20, 27].

Let $\{\theta_j, y_j\}_{j=1}^m$ be the m eigenpairs of the tridiagonal matrix $T_m$ with the desired k eigenpairs appearing as the leading entries, i.e. $\{\theta_j, y_j\}_{j=1}^k$. Define $x_j := P_m y_j$. Then $\theta_j$ and $x_j$ are commonly referred to as a Ritz value and a Ritz vector of $A$, respectively, or simply a Ritz pair. It follows from (3) that

$$ Ax_j - \theta_j x_j = AP_m y_j - \theta_j P_m y_j = (P_m T_m - \theta_j P_m) y_j + f e_m^T y_j = f e_m^T y_j, \quad (6) $$

and therefore the norms of the residual errors for the Ritz pairs $\{\theta_j, x_j\}$ satisfy

$$ \|Ax_j - \theta_j x_j\| = \beta_m |e_m^T y_j| \quad (7) $$

where $\beta_m = \|f\|$. For numerical experiments, we use a stopping criterion

$$ \beta_m |e_m^T y_j| \leq \epsilon \|A\| \quad (8) $$

where $\epsilon$ is a user specified tolerance and $\|A\|$ is approximated by the eigenvalue of largest magnitude over all iterations thus far of the computed matrices $T_m$. For a given number $k$ of desired eigenpairs we have convergence when the norm (7) for all $j = 1, \ldots, k$ is less than $\epsilon \|A\|$. 

2.3 Thick–Restarted Lanczos with Ritz vectors

We next briefly describe the method of thick–restarting with Ritz vectors as outlined in [27] which is needed in subsequent sections. For a thorough description, we refer the reader to [25, 26, 27, 29] and the references within.
From (6), we have for $j = 1, \ldots, k$

$$Ax_j = \theta_j x_j + p_{m+1} \bar{\beta}_j$$  \hspace{1cm} (9)

where $p_{m+1} = f / \beta_m$ and $\bar{\beta}_j = \beta_m e_m^T y_j$. Define

$$\bar{P}_k := [x_1, \ldots, x_k] \quad \text{and} \quad \bar{P}_{k+1} := [\bar{P}_k, p_{m+1}].$$  \hspace{1cm} (10)

Then we have

$$A \bar{P}_k = \bar{P}_{k+1} \bar{T}_{k+1,k}$$  \hspace{1cm} (11)

where

$$\bar{T}_{k+1,k} = \begin{bmatrix} \theta_1 & 0 \\ \theta_2 & \theta_k \\ \vdots & \vdots \\ 0 & \bar{\beta}_k \end{bmatrix}.$$  \hspace{1cm} (12)

The factorization (11) can easily be extended to $m$ vectors via MLan($k$) Algorithm 2.1 by noticing that

$$\bar{P}_{k+1}^T A \bar{P}_{k+1} = \bar{T}_{k+1} = \begin{bmatrix} \theta_1 & 0 & \bar{\beta}_1 \\ \theta_2 & \bar{\beta}_2 \\ \vdots & \vdots \\ 0 & \bar{\beta}_1 & \bar{\beta}_2 \end{bmatrix} \hspace{1cm} (13)$$

where the last diagonal element $\alpha_{k+1}$ is computed in step 7 in the MLan($k$) Algorithm 2.1. Step 6 in MLan($k$) Algorithm 2.1 is used to ensure the orthogonalization of the newly computed Lanczos vector against $\bar{P}_k$ (10), c.f. Algorithm 3 in [27]. After $m - k$ steps of MLan($k$) Algorithm 2.1 we have,

$$A \bar{P}_m = \bar{P}_m \bar{T}_m + \bar{f} e_m^T,$$  \hspace{1cm} (14)

where

$$\bar{P}_m = [\bar{P}_{k+1}, p_{k+2}, \ldots, p_m] \in \mathbb{R}^{n \times m},$$  \hspace{1cm} (15)
satisfies \( \bar{P}_m^T \bar{P}_m = I_m \), and \( \bar{f} \in \mathbb{R}^n \) satisfies \( \bar{P}_m^T \bar{f} = 0 \) and

\[
\bar{T}_m = \begin{bmatrix}
\bar{T}_{k+1} & \beta_{k+1} & 0 \\
\beta_{k+1} & \alpha_{k+2} & \beta_{k+2} & & & \\
& \ddots & \ddots & \ddots & \ddots \\
0 & & & \ldots & \beta_{m-1} & \alpha_m
\end{bmatrix} \in \mathbb{R}^{m \times m}. \tag{16}
\]

**Algorithm 3.1** Thick–Restarted Lanczos

1. **Input:** \( A \in \mathbb{R}^{n \times n} \): symmetric matrix,
   
   \( p_1 \in \mathbb{R}^n \): orthonormal vector,

   \( k \) : number of desired eigenpairs of \( A \),

   \( m \) : maximum number of Lanczos vectors.

2. **Output:** \( \{ \theta_j, x_j \}_{j=1}^k \) approximate eigenpairs of \( A \).

3: Compute factorization (3) by MLan(0) Algorithm 2.1

4: Compute the \( k \) desired eigenpairs \( \{ \theta_j, y_j \}_{j=1}^k \) of \( T_m \) (5) or \( \bar{T}_m \) (16)

5: Check convergence (7)

6: Set up (11) and (13) and call MLan(\( k \)) Algorithm 2.1 to get factorization (14)

7: Goto 4

### 2.4 Refined Ritz vectors

We briefly describe and provide some results on refined Ritz vectors and values.

For a thorough description, we refer the reader to [6, 7, 9, 11]. The refined Ritz vector \( z_j \) for an approximate desired eigenvalue \( \mu_j \) of \( A \) is computed by

\[
\min_{z_j \in \mathbb{K}_m(A, p_1)} \| Az_j - \mu_j z_j \|,
\]

\[
\tag{17}
\]

Since \( z_j \in \mathbb{K}_m(A, p_1) \), we can represent it in terms of the Lanczos vectors (4), i.e. \( z_j = P_m w_j \) for some unit vector \( w_j \). Given \( \beta_m = \| f \| \), define

\[
P_{m+1} := [P_m, p_{m+1}] \in \mathbb{R}^{n \times (m+1)}, \quad T_{m+1,m} := \begin{bmatrix} T_m \\ \beta_m e_m^T \end{bmatrix} \in \mathbb{R}^{(m+1) \times m}, \tag{18}
\]

where \( p_{m+1} := f / \beta_m \). Let \( I_{m+1,m} \) denote the first \( m \) columns of the identity matrix of order \( m + 1 \). For each \( \mu_j \) compute the smallest singular value \( \sigma_{ij} \) and associated
unit singular vectors of \((T_{m+1,m} - \mu_j I_{m+1,m})\) i.e.

\[
(T_{m+1,m} - \mu_j I_{m+1,m})v_j = \sigma_j u_j
\]  

(19)

\[
(T_{m+1,m} - \mu_j I_{m+1,m})^T u_j = \sigma_j v_j.
\]  

(20)

We refer to the following unit vectors \(v_j \in \mathbb{R}^m\) as the right singular vector and \(u_j \in \mathbb{R}^{m+1}\) as the left singular vector associated with the smallest singular value \(\sigma_j\).

Therefore,

\[
\min_{z_j \in \mathcal{K}_j(A,P_m)} \|Az_j - \mu_j z_j\| = \min_{\|w_j\|=1} \|AP_m w_j - \mu_j P_m w_j\| = \|P_{m+1}(T_{m+1,m} - \mu_j I_{m+1,m})w_j\| = \|(T_{m+1,m} - \mu_j I_{m+1,m})v_j\| = \sigma_j.
\]  

(21)

Then the refined Ritz vector \(z_j\) for \(\mu_j\) is defined as \(z_j := P_m v_j\). The approximate eigenvalue can be selected as the Ritz value \(\theta_j\) or as the Rayleigh quotient \(\rho_j = z_j^T A z_j = v_j^T T_m v_j\). Jia [7] suggested using \(\rho_j\) in place of \(\theta_j\) as it may be more accurate.

Setting \(\mu_j = \theta_j\) and using (7), we have from [7, 9] that

\[
\|Az_j - \rho_j z_j\| \leq \|Az_j - \theta_j z_j\| \leq \|Ax_j - \theta_j x_j\| = \beta_m |e_{m+1}^T v_j|.
\]  

(22)

If \(\|Ax_j - \theta_j x_j\| \neq 0\), then \(\|Az_j - \theta_j z_j\| < \|Ax_j - \theta_j x_j\|\). The following discussion establishes the left inequality for \(\mu_j\) not necessarily \(\mu_j = \theta_j\) and shows when a strict inequality exists. This sets the foundation for the subsequent section on iterative refined Ritz.

Using \(\{\rho_j, z_j\}\) as the approximation, we have the following. Equate the first \(m\) rows of (19) and the last row to obtain

\[
(T_m - \mu_j I_m) v_j = \sigma_j u_j (1:m)
\]  

(23)

\[
\beta_m e_{m+1}^T v_j = \sigma_j u_j (m+1)
\]  

(24)
and left multiply (23) by $v_j^T$ to obtain

$$\mu_j - \rho_j = -\sigma_j v_j^T u_j (1:m). \quad (25)$$

Then using the relationships (23)–(25) we have,

$$A z_j = A P_m v_j = P_m T_m v_j + f e_m^T v_j$$

$$= \rho_j z_j + \sigma_j P_{m+1} (u_j - ([v_j; 0]^T u_j)[v_j; 0]) \quad (26)$$

where $[v_j; 0] \in \mathbb{R}^{m+1}$. Using (26) we have,

$$\|A z_j - \rho_j z_j\| = \sigma_j \|P_{m+1} u_j - ([v_j; 0]^T u_j) P_{m+1} [v_j; 0]\|$$

$$= \sigma_j \|u_j - ([v_j; 0]^T u_j)[v_j; 0]\|$$

$$\leq \sigma_j. \quad (27)$$

The inequality in (27) comes from using the orthogonal projection, as summarized in Lemma 2.8.1 in the appendix. A strict inequality $\|A z_j - \rho_j z_j\| < \sigma_j$ exists if $\mu_j \neq \rho_j$ then $[v_j; 0]^T u_j \neq 0$ in (25), also see Lemma 2.8.1.

Notice from (25) and (26) we have

$$A z_j = A P_m v_j = (\rho_j - \sigma_j v_j^T u_j (1:m)) z_j + \sigma_j P_{m+1} u_j$$

$$= \mu_j z_j + \sigma_j P_{m+1} u_j \quad (28)$$

and $\|A z_j - \mu_j z_j\| = \sigma_j \|P_{m+1} u_j\| = \sigma_j$. Therefore, we have

$$\|A z_j - \rho_j z_j\| \leq \|A z_j - \mu_j z_j\| \quad (29)$$

and if $\mu_j \neq \rho_j$ then $\|A z_j - \rho_j z_j\| < \|A z_j - \mu_j z_j\|$. The relationship and convergence properties of the refined Ritz vector $z_j$ and Ritz vector $x_j$ are described in detail in [9, 11, 18]. In particular, they show, when $\sigma_j = 0$, the vectors $z_j$ and $x_j$ are parallel to an eigenvector of $A$ and $\rho_j = \theta_j$ is an exact eigenvalue of $A$. It is remarked in [18] that for a special, non–symmetric case, that $z_j$ and $x_j$ can be parallel, even though
\( \sigma_j \neq 0 \). However, this is not the case in the context here for the symmetric problem, as Theorem 2.4.2 below demonstrates.

**Corollary 2.4.1.** Let \( T_m (5) \) be unreduced and \( \beta_m \neq 0 \). Given the singular value relationships (19) and (20) we have

i.) \( \sigma_j \neq 0 \) and 

ii.) \( u_j \neq \pm e_{m+1} \).

**Proof.** i.) To show that \( \sigma_j \neq 0 \), we argue via contradiction. Let \( \sigma_j = 0 \), then from (23) and (24) we have,

\[
(T_m - \mu_j I_m)v_j = 0
\]

\[
\beta_m e_m v_j = 0.
\]

Given \( \beta_m \neq 0 \), we have from (31) \( e_m^T v_j = 0 \) and since \( T_m \) is unreduced, \( e_m^T v_j \neq 0 \), c.f. [17, Theorem 7.9.3].

ii.) To show that \( u_j \neq \pm e_{m+1} \), we also argue via contradiction. Let \( u_j = \pm e_{m+1} \).

Then from (19) and (20) we have,

\[
(T_m - \mu_j I_m)v_j = 0
\]

\[
\pm \beta_m e_m = \sigma_j v_j.
\]

Given \( \beta_m \neq 0 \), we have from (33) that \( v_j = \pm e_m \). Then from (32), we have \( \beta_{m-1} = 0 \). Contradiction to \( T_m \) being unreduced.

**Theorem 2.4.2.** Let \( T_m (5) \) be unreduced and \( \beta_m \neq 0 \). Given the singular value relationships (19) and (20) with \( \mu_j = \theta_j \). Then \( z_j \neq \pm x_j \).

**Proof.** To show that \( z_j \neq \pm x_j \), we argue via contradiction. Let \( x_j = \pm z_j \) then \( y_j = \pm v_j \). From (23) and Corollary 2.4.1 we have

\[
0 = (T_m - \theta_j I_m)y_j = \pm (T_m - \theta_j I_m)v_j = \pm \sigma_j u_j(1:m) \neq 0.
\]
Therefore, the refined Ritz and Ritz vectors do not coincide until \( \sigma_j = 0 \).

Define

\[
r_j := u_j - ([v_j; 0]^T u_j)[v_j; 0]
\]

and notice that the refined Ritz vector and residual vectors satisfy

\[
z_j^T (P_{m+1} r_j) = 0.
\]

This is a requirement for restarting. Then from (26) we have similar to (9) for \( j = 1, \ldots, k \),

\[
A z_j = \rho_j z_j + \sigma_j P_{m+1} r_j.
\]

However, the setup of the thick–restarted method as described in Section 2.3 cannot be used with \( \{\rho_j, z_j\} \) in this context, since the residual vectors \( \sigma_j P_{m+1} r_j \) are not multiples of each other for different refined Ritz pairs \( \{\rho_j, z_j\} \). This is shown in Theorem 2.4.3. Below is a needed relationship before establishing the result. Notice from (19), (25), and (35) that

\[
\begin{bmatrix}
T_m - \rho_j I_m \\
\beta_m e_m^T
\end{bmatrix} v_j = \sigma_j r_j
\]

and if \( \{\rho_j, v_j\} \) is not an eigenpair of \( T_m \) then \( \sigma_j (u_j(1:m) - (v_j^T u_j(1:m)) v_j) \neq 0 \).

**Theorem 2.4.3.** Given refined Ritz pairs \( \{\rho_{j_1}, z_{j_1}\} \) and \( \{\rho_{j_2}, z_{j_2}\} \) with \( \rho_{j_1} \neq \rho_{j_2} \) satisfying (36) and \( \{\rho_{j_1}, v_{j_1}\} \) and \( \{\rho_{j_2}, v_{j_2}\} \) not eigenpairs of \( T_m \), we have

\( \sigma_{j_1} P_{m+1} r_{j_1} \neq \gamma \sigma_{j_2} P_{m+1} r_{j_2} \) for some scalar \( \gamma \).

**Proof.** To show that \( \sigma_{j_1} P_{m+1} r_{j_1} \neq \gamma \sigma_{j_2} P_{m+1} r_{j_2} \) for some scalar \( \gamma \), we argue via contradiction. Let \( \sigma_{j_1} P_{m+1} r_{j_1} = \gamma \sigma_{j_2} P_{m+1} r_{j_2} \) then from multiplying (36) with \( j = j_2 \) from the left by \( z_{j_1}^T \) and then by \( z_{j_2}^T \) when \( j = j_1 \) gives the relationships

\[
v_{j_1}^T T_m v_{j_2} - \rho_{j_2} v_{j_1}^T v_{j_2} = 0
\]

and

\[
v_{j_2}^T T_m v_{j_1} - \rho_{j_1} v_{j_2}^T v_{j_1} = 0.
\]

Since \( \rho_{j_1} \neq \rho_{j_2} \) this implies

\[
v_{j_2}^T T_m v_{j_1} = 0 \quad \text{and} \quad v_{j_2}^T v_{j_1} = 0.
\]

From (37) we have

\[
(T_m - \rho_{j_1} I_m) v_{j_1} = \gamma (T_m - \rho_{j_2} I_m) v_{j_2} \neq 0 \quad \text{and} \quad \beta_m e_m^T v_{j_1} = \gamma \beta_m e_m^T v_{j_2}.
\]

Therefore,

\[
0 < v_{j_1}^T (T_m - \rho_{j_1} I_m)(T_m - \rho_{j_1} I_m) v_{j_1} = \gamma v_{j_1}^T (T_m - \rho_{j_1} I_m)(T_m - \rho_{j_2} I_m) v_{j_2} = \gamma v_{j_1}^T T_m^2 v_{j_2}.
\]
For $v_{j1}$ from (19) and (20) we have 

$((T_m - \mu_{j1} I_m)(T_m - \mu_{j1} I_m) + \beta_m^2 e_m e_m^T)v_{j1} = \sigma_{j1}^2 v_{j1}$

and after left multiplying by $v_{j2}^T$ we get $\gamma v_{j1}^T T_m^2 v_{j2} = -\beta_m^2 \gamma^2 (e_m v_{j2})^2 < 0$.

A restarted technique such as setting the starting vector $p_1$ in Algorithm 2.1 as a linear combination of $k$ desired approximate refined Ritz vectors can be used, see [6]. We propose a different linear combination with constants chosen in a similar fashion outlined in [15]. Before presenting our restarted scheme, we will first focus on adjusting the refined Ritz vectors to reduce the residual norm. We also include some results and motivational remarks.

2.5 Iterative Refined Ritz vectors

Considering the refined Ritz pair $\{\rho_j, z_j\}$ may provide a better approximation by having a “smaller” norm (22) when used together, we propose iteratively refining the approximation. That is, set $\mu_j = \rho_j$ in (17) and re–compute the refined Ritz vectors as stated in (21) with the updated $\mu_j$. This process can be repeated and creates an iterative scheme with a sequence of refined Ritz pairs, $\{\rho_j(i), z_j(i)\}$ for $i = 1, 2, \ldots$. The process terminates with a refined Ritz pair $\{\hat{\rho}_j, \hat{z}_j\}$ that has favorable properties. We will refer to $\{\hat{\rho}_j, \hat{z}_j\}$, as the iterative refined Ritz value and vector respectively, and collectively as the iterative refined Ritz pair. Algorithm 5.1 outlines the computational process which we follow up with remarks. To show convergence, and hence termination of the iterative scheme, we establish via Theorem 2.5.1 that the nonnegative sequence $\sigma_j(i)$ computed from this process is bounded, decreasing, and hence converges. Let $\hat{\sigma}_j$ be the value to which the sequence $\sigma_j(i)$ converges.

**Theorem 2.5.1.** Let $\mu_j = \rho_j^{(i-1)}$ with $\rho_j^{(0)} = \theta_j$, $z_j^{(i)} = P_m v_j^{(i)}$, and $\rho_j^{(i)} = z_j^{(i)} A z_j^{(i)}$ for $i = 1, 2, \ldots$. Then the computed smallest singular values $\sigma_j^{(i)}$ for $i = 1, 2, \ldots$ from the equations (19) and (20) to solve (17) is a nonnegative bounded decreasing sequence and hence converges.
Proof. We compute

\[ 0 \leq \sigma_j^{(i+1)} = \sigma_j^{(i+1)} \| u_j^{(i+1)} \| = \| \sigma_j^{(i+1)} u_j^{(i+1)} \| \]

\[ = \| (T_{m+1,m} - \rho_j^{(i)} I_{m+1,m}) v_j^{(i+1)} \| \]

\[ \leq \| (T_{m+1,m} - \rho_j^{(i)} I_{m+1,m}) v_j^{(i)} \|. \]

We have the inequality in (38) since \( v_j^{(i+1)} \) satisfies the minimization property in (21) when \( \mu_j = \rho_j^{(i)} \). Note that \( v_j^{(i)} \) satisfies the minimization property in (21) when \( \mu_j = \rho_j^{(i-1)} \), therefore we have

\[ (T_{m+1,m} - \rho_j^{(i-1)} I_{m+1,m}) v_j^{(i)} = \sigma_j^{(i)} u_j^{(i)} \]

\[ T_{m+1,m} v_j^{(i)} = \sigma_j^{(i)} u_j^{(i)} + \rho_j^{(i-1)} I_{m+1,m} v_j^{(i)}. \]

From (25) we have

\[ \rho_j^{(i)} - \rho_j^{(i-1)} = \sigma_j^{(i)} v_j^{(i)^T} u_j^{(i)} \]

Plugging (39) and (40) into (38) and continuing (38) we have

\[ 0 \leq \sigma_j^{(i+1)} \leq \| T_{m+1,m} v_j^{(i)} - \rho_j^{(i)} I_{m+1,m} v_j^{(i)} \| \]

\[ = \sigma_j^{(i)} \| u_j^{(i)} - ([v_j; 0]^{(i)^T} u_j^{(i)}) [v_j; 0]^{(i)} \| \]

\[ \leq \sigma_j^{(i)}. \]

The last inequality in (41) comes from using Lemma 2.8.1. A strict inequality exists if \((v_j; 0)^{(i)^T} u_j^{(i)} \neq 0\), see Lemma 2.8.1

Notice from (27) and (41) we have

\[ \| A z_j^{(i+1)} - \rho_j^{(i+1)} z_j^{(i+1)} \| \leq \sigma_j^{(i+1)} \]

\[ \leq \sigma_j^{(i)} \| u_j^{(i)} - ([v_j; 0]^{(i)^T} u_j^{(i)}) [v_j; 0]^{(i)} \| \]

\[ = \| A z_j^{(i)} - \rho_j^{(i)} z_j^{(i)} \|. \]
which implies \( \{\rho_{ij}^{(i+1)}, z_{ij}^{(i+1)}\} \) can be a better approximation than \( \{\rho_{ij}^{(i)}, z_{ij}^{(i)}\} \). If \( \sigma_{ij}^{(i)} = 0 \) for some \( i \) we have from (42) that \( \{\rho_{ij}^{(i)}, z_{ij}^{(i)}\} \) is an exact eigenpair of \( A \). We assume for the remainder of the section that \( 0 < \hat{\sigma}_{ij} \leq \sigma_{ij}^{(i)} \).

Another point of view of the equations (19) and (20) is as an eigenvalue problem. In this context, define \( H(\mu_j) := (T_m - \mu_j I_m)(T_m - \mu_j I_m) + \beta^2 e_m e^T_m \) where \( \mu_j = \rho_j = v_j^T T_m v_j \) is a function of the unit vector \( v_j \). Therefore, we have

\[
H(v_j) v_j = \sigma_{ij}^2 v_j
\]  
(43)

and we are searching for the smallest eigenvalue \( \sigma_{ij}^2 \) and associated unit eigenvector \( v_j \). The vector \( u_j \) can be obtained by

\[
u_j = 1/\sigma_{ij}(T_{m+1,m} - \mu_j I_{m+1,m})v_j.
\]  
(44)

Equation (43) is referred to as an eigenvector–dependent nonlinear eigenvalue problem (NEPv) and the most commonly used routine for solving (43) is the simple self-consistent field (SCF) iteration process, see [3] and reference within.

The computation of iterative refined values is outlined in the Iterative Refined Algorithm 5.1. We assume \( m \ll n \) and the computational time required per iteration for Algorithm 5.1 is negligible in comparison to the computational time required for a matrix–vector product with \( A \) when \( n \) is very large.

**Algorithm 5.1 Iterative Refined**

1. **Input:** \( T_{m+1,m} \in \mathbb{R}^{m+1 \times m} \) (18) and \( \{\mu_j\}_{j=1}^k \)
2. **Output:** \( \{\hat{\rho}_j, \hat{v}_j, \hat{u}_j, \hat{\sigma}_{ij}\}_{j=1}^k \)
3. **for** \( j = 1, 2, \ldots, k \) **do**
4. **for** \( i = 1, 2, \ldots, maxit \) **do**
5. Compute \( v_{ij}^{(i)}, u_{ij}^{(i)}, \) and \( \sigma_{ij}^{(i)} \) using either (19) and (20) or (43) and (44)
6. Set \( \rho_j^{(i)} := v_j^{(i)T} T_m v_j^{(i)} \)
7. Check convergence
8. Set \( \mu_j := \rho_j^{(i)} \)

There are several options for step 7 in Algorithm 5.1 on checking convergence. For example, convergence can be checked by \( |\rho_j^{(i)} - \rho_j^{(i-1)}|/|\rho_j^{(i)}| < eps \), where \( eps \)
is machine epsilon. Some heuristics for stopping are provided in [3] and references within when using the SCF iteration to solve NEPv (43). It should be noted that stagnation can occur while using finite arithmetic and we propose including a check to exit when detected to avoid unnecessary iterations. As the restarted hybrid method presented in section 2.6 converges we notice via numerical experiments the number of iterations in Algorithm 5.1 reduce quickly to only a handful.

We see from Theorem 2.5.1 and the relationship $\sigma_j^2 = v_j^T H(v_j)v_j$ that the output $\{\hat{\rho}_j, \hat{v}_j, \hat{u}_j, \hat{\sigma}_j\}$ from Algorithm 5.1 satisfies,

\begin{align}
(T_{m+1,m} - \hat{\rho}_j I_{m+1,m})\hat{v}_j &= \hat{\sigma}_j \hat{u}_j \\
(T_{m+1,m} - \hat{\rho}_j I_{m+1,m})^T \hat{u}_j &= \hat{\sigma}_j \hat{v}_j
\end{align}

(45) (46)

where $\hat{\rho}_j = \hat{v}_j^T T_m \hat{v}_j$. Equate the first $m$ rows of (45) and left multiply by $\hat{v}_j^T$ to get

$$\hat{\sigma}_j \hat{v}_j^T \hat{u}_j (1: m) = 0. \quad (47)$$

Using (35), (36), and (47) we have

$$A \hat{z}_j = \hat{\rho}_j \hat{z}_j + \hat{\sigma}_j P_{m+1} \hat{u}_j \quad (48)$$

where $\hat{z}_j = P_m \hat{v}_j$ and $\hat{z}_j^T P_{m+1} \hat{u}_j = 0$. Notice from (22), (42), and (48)

$$\hat{\sigma}_j = \|A \hat{z}_j - \hat{\rho}_j \hat{z}_j\| \leq \|Az_j - \rho_j z_j\| \leq \|Az_j - \theta_j z_j\| \leq \|Ax_j - \theta_j x_j\|. \quad (49)$$

Notice that in floating point arithmetic we have $\hat{\sigma}_j \hat{v}_j^T \hat{u}_j (1: m) \approx 0$ and should be included in equation (48) when used in computer codes, c.f. (36). It is not included in establishing subsequent results and equations, i.e. we assume (47) holds.

Although the results listed in [9], [11], [18] were developed for refined Ritz and Ritz pairs when $\mu_j = \theta_j$, we see from the relationships (42) and (49) that similar results apply for iterative refined Ritz. In particular, we have when $\hat{\sigma}_j = 0$, vectors $\hat{z}_j$ and $x_j$ are parallel to an eigenvector of $A$ and $\hat{\rho}_j = \theta_j$ is an exact eigenvalue of $A$. Notice
that Corollary 2.4.1 does not depend on \( \mu_j = \theta_j \), however Theorem 2.4.2 depended on setting \( \mu_j = \theta_j \), therefore we state the result in this context for \( \hat{z}_j \).

**Theorem 2.5.2.** Let \( T_m (5) \) be unreduced and \( \beta_m \neq 0 \). Given the singular value relationships (45) and (46), then \( \hat{z}_j \neq \pm x_j \).

**Proof.** To show that \( \hat{z}_j \neq \pm x_j \), we argue via contradiction. Let \( x_j = \pm \hat{z}_j \) then \( y_j = \pm \hat{v}_j \) and \( \hat{\rho}_j = \hat{v}_j^T T_m \hat{v}_j = y_j^T T_m y_j = \theta_j \). From (45) and Corollary 2.4.1 we have

\[
0 = (T_m - \theta_j I_m)y_j = \pm (T_m - \hat{\rho}_j I_m)\hat{v}_j = \pm \hat{\sigma}_j \hat{u}_j(1:m) \neq 0 \quad (50)
\]

Therefore, the iterative refined Ritz and Ritz vectors do not coincide unless \( \hat{\sigma}_j = 0 \). Theorem 2.4.3 also does not depend on \( \mu_j = \theta_j \) and the result holds in this context. That is, the set-up of the thick–restarted method as described in Section 2.3 cannot be used with \( \{\hat{\rho}_j, \hat{z}_j\} \) in this context, since the residual vectors \( \sigma_j P_{m+1} \hat{r}_j \) are not multiples of each other for different iterative refined pairs \( \{\hat{\rho}_j, \hat{z}_j\} \). The iterative refined Ritz pair have a “smaller” residual norm and possess similar properties to the refined Ritz pair. The following examples provide motivation on our restarting technique.

**Example 2.5.1** Let \( A \) be the \( 4 \times 4 \) symmetric matrix,

\[
A = \begin{bmatrix}
9 & 1 & -2 & -1 \\
1 & 8 & -3 & -2 \\
-2 & -3 & 7 & -1 \\
1 & -2 & -1 & 6
\end{bmatrix}.
\quad (51)
\]

The eigenvalues of \( A \) are 12, 9, 6, 3. Using MLan(0) Algorithm 2.1 with \( p_1 = \frac{1}{2}[1 \ 1 \ 1 \ 1]^T \) and \( m = 3 \), gives a tridiagonal matrix \( T_3 \) with eigenvalues \( \theta_1 = 11.7913, \theta_2 = 7.4755, \theta_3 = 3.0239, \) and \( \beta_3 = 1.8035 \). We have for the largest eigenpair,

\[
\hat{\sigma}_1 = 0.831397 < \sigma_1 \|r_1\| = 0.831400 < \beta_3 \|e_3^T y_1\| = 0.885392 \quad (52)
\]
where $\mu_1 = \theta_1$ in Algorithm 5.1. Equation (52) shows the residual norm with iterative refined pair is “smaller”, with similar results for the other eigenpairs. In practice, the matrix $A$ is very large and restarting is required. A restarted Lanczos method depends on many things for successful complementation, one of which is a “good” (re)starting vector. For a fair comparison, Table 1 displays the Ritz residual norms $\beta_3 |e_3^T y_1|$ associated with $\{\theta_1, y_1\}$ for $A$ where we set the (re)starting vector $p_1$ in MLan(0) Algorithm 2.1 on the next restart to be the computed iterative refined Ritz $\hat{v}_1$, refined Ritz $P_3 v_1$, and Ritz vector $P_3 y_1$. Also, included in Table 1 is the sine of the angle for each vector $P_3 \hat{v}_1$, $P_3 v_1$, and $P_3 y_1$ with the desired eigenvector $x$ associated with the largest eigenvalue 12 of the matrix $A$. It should be noted that Krylov subspaces associated with each column in Table 1 are different, since they depend on the starting vector. However, we do see from Table 1 that using the iterative refined Ritz vector $P_3 \hat{v}_1$ we are able to obtain a smaller Ritz residual norm on each restart and a starting vector “closer” to the desired eigenvector. Although the results for iterative refined Ritz norms and angles may appear to be only marginally smaller than refined Ritz norms, in a restarted scheme for large matrices this can be significant.

Table 1. Example 2.5.1. Displays the Ritz residual norms $\beta_3 |e_3^T y_1|$ (7) associated with $\{\theta_1, y_1\}$ for different (re)starting vector $p_1$ in MLan(0) Algorithm 2.1 on the next restart. Also, displays the sine of the angle for each vector with the eigenvector $x$ associated with the largest eigenvalue 12.

<table>
<thead>
<tr>
<th>Restart</th>
<th>Iterative refined Ritz</th>
<th>Refined Ritz</th>
<th>Ritz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\beta_3</td>
<td>e_3^T y_1</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$\sin \angle(x, P_3 \hat{v}_1)$</td>
<td>$\sin \angle(x, P_3 v_1)$</td>
<td>$\sin \angle(x, P_3 y_1)$</td>
</tr>
<tr>
<td>1</td>
<td>2.4589 $\cdot 10^{-2}$</td>
<td>2.4820 $\cdot 10^{-2}$</td>
<td>6.6286 $\cdot 10^{-2}$</td>
</tr>
<tr>
<td>2</td>
<td>7.0237 $\cdot 10^{-4}$</td>
<td>7.1591 $\cdot 10^{-4}$</td>
<td>2.1557 $\cdot 10^{-3}$</td>
</tr>
<tr>
<td>3</td>
<td>1.8391 $\cdot 10^{-5}$</td>
<td>1.8918 $\cdot 10^{-5}$</td>
<td>1.5244 $\cdot 10^{-4}$</td>
</tr>
<tr>
<td>4</td>
<td>5.2531 $\cdot 10^{-7}$</td>
<td>5.4567 $\cdot 10^{-7}$</td>
<td>4.9572 $\cdot 10^{-6}$</td>
</tr>
</tbody>
</table>

Example 2.5.1 is a good illustrative example that shows that we can get better results with iterative refined Ritz vectors. However, this example’s features are ideal
with well–separated eigenvalues of $A$ and large $T_m$ ($m = 3$ compared with $n = 4$) that yields a good approximation to all eigenpairs on the first iteration. In practice, $A$ will be very large, $m \ll n$, and the corresponding generated Krylov subspace yielding a poor approximation to the desired eigenpairs. When $m$ is kept very small we experienced very poor results, often with stagnation. This can be contributed in part to an overall poor approximation from the Krylov subspace and the computational process of refined or iterative refined Ritz values. The following example illustrates this undesirable scenario.

**Example 2.5.2** Let $A$ be the $3 \times 3$ symmetric matrix,

$$A = \begin{bmatrix} 0.0025 & 0.0485 & 0 \\ 0.0485 & 2.1509 & 2.3 \\ 0 & 2.3 & 0 \end{bmatrix}. \quad (53)$$

The eigenvalues of $A$ are $3.6149, 2.4989 \cdot 10^{-3}$, and $-1.4640$. Using MLan(0) Algorithm 2.1 on matrix $A$ with $p_1 = [1\ 0\ 0]^T$ and $m = 2$, yields a matrix $T_{3,2}$ that consist of the first 2 columns of $A$, with eigenvalues of $T_{3,2}(1:2,1:2)$ as $\theta_1 = 2.1520$ and $\theta_2 = 0.0014$. We used Algorithm 5.1 with $\mu_1 = \theta_1 = 2.1520$ for computing refined Ritz and iterative refined Ritz pairs. Table 2a below displays the output of the refined Ritz $\rho_1$ and iterative refined $\hat{\rho}_1$ values and the sine of the angles between the eigenvector $x$ associated with the largest eigenvalue $3.6149$, and the eigenvectors $y_1$ and $y_2$, associated with $\theta_1$ and $\theta_2$, respectively. We see that refined Ritz pair $\{\rho_1, v_1\}$ and iterative refined Ritz pair $\{\hat{\rho}_1, \hat{v}_1\}$ are “closer” to $\{\theta_2, y_2\}$ than $\{\theta_1, y_1\}$ even though approximations set $\mu_1 = \theta_1$. This can cause a restarted method with refined Ritz or iterative refined Ritz to stagnate or converge slowly. See Table 2b where we set the (re)starting vector $p_1$ in MLan(0) Algorithm 2.1 on the next restart to be the computed iterative refined Ritz $P_2 \hat{v}_1$, refined Ritz $P_2 v_1$, and Ritz vector $P_2 y_1$. The initial iterative refined Ritz pair is very close to the undesired value $\{\theta_2, y_2\}$ and the process cannot recover, causing stagnation. The initial refined Ritz pair is not as close, and does recover, however the overall convergence is a lot slower than
using Ritz vector to restart. Also, the computed refined Ritz residual $\alpha_1 \| r_1 \|$ for the method that restarts with refined Ritz vector was only marginally better, e.g. on restart 4 with refined Ritz vector method we have $\alpha_1 \| r_1 \| = 5.3081 \cdot 10^{-5}$ (compared to $\beta_2 | e_2^T y_1 | = 5.3121 \cdot 10^{-5}$). In this example, notice that the initial Ritz vector is significantly closer to the desired eigenvector $x$ and restarting with Ritz vector converges a lot faster.

Table 2. Example 2.5.2.

<table>
<thead>
<tr>
<th>Restart</th>
<th>Iterative Refined Ritz $\rho_1 = 0.0017$</th>
<th>Refined Ritz $\rho_1 = 0.0655$</th>
<th>Ritz</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin $\angle(x, P_2 \hat{v}_1)$</td>
<td>sin $\angle(y_1, \hat{v}_1)$</td>
<td>sin $\angle(y_2, v_1)$</td>
<td>sin $\angle(x, P_2 v_1)$</td>
</tr>
<tr>
<td>1.000</td>
<td>0.0120</td>
<td>0.9999</td>
<td>0.9904</td>
</tr>
</tbody>
</table>

(a) Display results when using Algorithm 5.1 with $\mu_1 = \theta_1 = 2.1520$ applied to $T_{3,2}$. Also, displays the sine of the angle for each vector with the eigenvector $x$ associated with the largest eigenvalue 3.6149.

(b) Displays the Ritz residual norms $\beta_2 | e_2^T y_1 |$ associated with $\{ \theta_1, y_1 \}$ for different (re)starting vector $p_1$ in Mlan(0) Algorithm 2.1 on the next restart. Also, displays the sine of the angle for each vector with the eigenvector $x$ associated with the largest eigenvalue 3.6149.

Although the example 2.5.2 is contrived, it illustrates what can happen. The next example, further illustrates the problem.

**Example 2.5.3** Let $A = \text{diag}(1:500)$ be a $500 \times 500$ diagonal matrix. We are searching for the largest eigenpair. We set $m = 2$ and the (re)starting vector $p_1$ in Mlan(0) Algorithm 2.1 on the next restart to be the computed Ritz vector $P_2 y_1$, refined Ritz $P_2 v_1$, or iterative refined Ritz $P_2 \hat{v}_1$. We ran the example 100 times with a different beginning random vector $p_1$. The results are presented in Figure 2a, Figure 2b and Figure 2c and show that restarting with iterative refined Ritz $P_2 \hat{v}_1$ or refined
Ritz $P_2v_1$ caused stagnation, or erratic behavior and slow convergence. Restarting with Ritz vector, Figure 2a was not erratic, but convergence was slow.

Example 2.5.1 showed that when Krylov subspace was a fairly “good” subspace that restarting with iterative refined Ritz vector provided the better results. However, examples 2.5.2 and 2.5.3 demonstrated the pitfalls of using a conventional restarting scheme with just (iterative) refined Ritz vectors when the Krylov subspace was a “poor” subspace. Although examples 2.5.2 and 2.5.3 may show that the iterative refined Ritz vectors to be problematic, the iterative refined Ritz vectors highlight, more so than refined Ritz vectors, when they should not be used to restart. Using this information and that restarting with Ritz vectors convergence was not erratic (although slow), we developed a hybrid method that switches, depending on some parameters, between thick-restarting with Ritz vectors and restarting with iterative refined Ritz vectors and show this combination overcomes the stagnation and erratic behavior producing a faster overall converging method.

2.6 Hybrid Methods

The hybrid method developed here uses thick-restarted as the main routine and, under certain conditions, switches to restarting with iterative refined Ritz vectors. The iterative refined Ritz vectors from a “good” Krylov subspace can be better approximations, but a trade off is a restarted scheme that loses the benefits of thick-restarted which are crucial and traced back to the IRL method. Using only the thick-restarted with a small $m$ value converges very slowly or not at all. We have found that when the Krylov subspace is “good” that switching to restarting with iterative refined Ritz vectors, even for a few iterations, results in a faster overall convergence.

For $k = 1$ (single eigenpair) restating equations (9) and (48) for the Ritz pair
\{x_1, \theta_1\} and the iterative refined Ritz pair \{\hat{z}_1, \hat{\rho}_1\}

\[ Ax_1 = [x_1, p_2] \begin{bmatrix} \theta_1 \\ \beta_1 \end{bmatrix} \quad \text{where } p_2 = f/\beta_m \tag{54} \]

\[ A\hat{z}_1 = [\hat{z}_1, p_2] \begin{bmatrix} \hat{\rho}_1 \\ \hat{\sigma}_1 \end{bmatrix} \quad \text{where } p_2 = P_{m+1}\hat{u}_1. \tag{55} \]

Depending on certain parameters for switching described in Section 2.6.1, we can restart by calling the MLan(1) Algorithm 2.1 with starting vector \(p_2\). This is slightly different than a restarted Lanczos method, by using relationships in equations (54) and (55) to avoid a matrix–vector product with \(A\) on each restart.

The hybrid method for finding \(k > 1\) eigenpairs has added challenges. The thick–restarted algorithm is set up to compute \(k \geq 1\) eigenpairs, however the iterative refined Ritz does not fit this structure. Therefore, we implement a standard restart technique with a starting vector \(p_1\) constructed as a linear combination of \(k\) iterative refined Ritz vectors

\[ p_1 = \sum_{j=1}^{k} c_j \hat{z}_j. \tag{56} \]

This was the set up for the refined Ritz algorithm [6, Algorithm 1] where the combination of the refined Ritz vectors for the starting vectors \(p_1\) were constructed based on coefficients \(c_j\) chosen as described in Saad [19]. In our development, we chose the coefficients \(c_j\) in a similar way to the description outlined by Morgan [15]. In simplest terms, for Ritz vectors \(x_j\), constants \(c_j\) are chosen for a starting vector \(p_1 = c_1 x_1 + \ldots + c_k x_k\) to eliminate the coefficients \(\beta_m e_m^T y_j\) multiplying the common residual vector \(p_{m+1}\). That is, the choice of coefficients removes \(p_{m+1}\) when the starting vector \(p_1\) is multiplied by \(A\) in the next iteration to build out the Krylov subspace. The coefficients can be determined by solving a certain homogenous \(k - 1 \times k\) linear system. It was then proven by Morgan with these specially chosen coefficients \(c_j\) that the span\(\{p_1, Ap_1, \ldots, A^{k-1}p_1\} = \text{span}\{x_1, x_2, \ldots, x_k\}\) which is the same subspace resulting from implementing Sorensen’s IRA method [22]. We refer the reader
to [13] for specific details and theoretical results. Building on this idea, we can, in a similar way, remove some of the coefficients associated with \( p_{m+1} \) for iterative refined vectors. The rationale on implementing this technique is to have a restarted method that may inherit similar convergence benefits of the IRL method. We did observe fast convergence, even with removing only some of the coefficients associated with \( p_{m+1} \).

From equations (24), (48), and (56) we have

\[
Ap_1 = \sum_{j=1}^{k} c_j (\hat{\rho}_j P_m \hat{v}_j + \beta_m e_m^T \hat{v}_j p_{m+1} + \hat{\sigma}_j P_m \hat{u}_{j(1:m)}).
\]

(57)

Therefore, we select coefficients \( c_j \) such that \( \beta_m e_m^T \hat{v}_j c_j = 0 \) which removes \( p_{m+1} \) from (57). If we ignore \( \hat{\sigma}_j P_m \hat{u}_{j(1:m)} \) in (57) and all future occurrences, as we multiple (57) by \( A \) we obtain a similar \( k - 1 \times k \) homogenous system linear system to the one presented in [13, Section 3] where \( i^{th} \) row is represented as,

\[
\beta_m \left[ \hat{\rho}_1^{-1} e_m^T \hat{v}_1 \quad \hat{\rho}_2^{-1} e_m^T \hat{v}_2 \quad \cdots \quad \hat{\rho}_k^{-1} e_m^T \hat{v}_k \right].
\]

(58)

If we leave \( \hat{\sigma}_j P_m \hat{u}_{j(1:m)} \) in (57) and continue with \( p_{m+1} \) removed in (57), we have,

\[
A^2 p_1 = \sum_{j=1}^{k} c_j (\hat{\rho}_j^2 P_m \hat{v}_j + \hat{\rho}_j \hat{\sigma}_j P_m \hat{u}_{j(1:m)} + \beta_m \hat{\rho}_j e_m^T \hat{v}_j p_{m+1} + \hat{\sigma}_j A P_m \hat{u}_{j(1:m)})
\]

(59)

where

\[
\hat{\sigma}_j A P_m \hat{u}_{j(1:m)} = \hat{\sigma}_j P_m T_m \hat{u}_{j(1:m)} + \beta_m (e_m^T T_m \hat{v}_j - \hat{\rho}_j e_m^T \hat{v}_j) p_{m+1}.
\]

(60)

Collecting terms multiplying \( p_{m+1} \) in (59) and (60) we have,

\[
(\beta_m \hat{\rho}_j e_m^T \hat{v}_j + \beta_m (e_m^T T_m \hat{v}_j - \hat{\rho}_j e_m^T \hat{v}_j)) c_j = \beta_m e_m^T T_m \hat{v}_j c_j.
\]

(61)

We therefore select coefficients \( c_j \) such that \( \beta_m e_m^T T_m \hat{v}_j c_j = 0 \) which removes \( p_{m+1} \) from (59). If we ignore \( \hat{\sigma}_j P_m T_m \hat{u}_{j(1:m)} \) in (60) and all future occurrences, as we multiply (57) by \( A \) we obtain the following \( k - 1 \times k \) homogenous system linear system where coefficient matrix has the same first row as in (58) and is represented as,

\[
\beta_m \left[ e_m^T \hat{v}_1 \quad e_m^T \hat{v}_2 \quad \cdots \quad e_m^T \hat{v}_k \right] i > 1.
\]

(62)
Notice from (45) that as $\hat{\sigma}_j$ approaches zero, we expect $T_m \hat{v}_j$ to approach $\hat{\rho}_j \hat{v}_j$ and (62) would become like the homogenous system (58). If we leave $\hat{\sigma}_j P_m T_m \hat{u}_j(1:m)$ in (60) and continue to remove $p_{m+1}$ we get the $k - 1 \times k$ homogenous linear system where coefficient matrix has the same first two rows as in (62) and is represented as,

$$
\beta_m \begin{bmatrix}
\hat{v}_1 \\
\hat{v}_2 \\
\vdots \\
\hat{v}_k
\end{bmatrix} +
\begin{bmatrix}
\hat{\rho}_j^{-2} e_m^T T_m \hat{v}_1 + s_1 \\
\hat{\rho}_2^{-2} e_m^T T_m \hat{v}_2 + s_2 \\
\vdots \\
\hat{\rho}_k^{-2} e_m^T T_m \hat{v}_k + s_k
\end{bmatrix} i > 2
$$

(63)

where

$$
s_j = \hat{\sigma}_j \sum_{\ell=3}^i \hat{\rho}_j^{-\ell} e_m^T T_m^{\ell-2} \hat{u}_j(1:m) \quad 1 \leq j \leq k.
$$

(64)

Likewise, as $\hat{\sigma}_j$ approaches zero, we expect $T_m \hat{v}_j$ to approach $\hat{\rho}_j \hat{v}_j$ and $s_j$ to approach zero, hence (63) would also become like the homogenous system (58). The matrices become more complicated and ill–conditioned as we include more terms for eliminating the coefficients multiplying $p_{m+1}$. We do assume that $k$ is small and have observed similar results with all three systems, but more consistent results over a wide range of problems when using (62) or (63). We solved the $k - 1 \times k$ homogenous linear system by finding the null space vector using the singular value decomposition. When a column becomes numerically zero, indicating an iterative refined Ritz vector has converged, we remove that column and the last row of the matrix and compute the null space vector of the reduced matrix. We then replace the corresponding coefficient with the norm of the iterative refined Ritz residual vector before creating the linear combination for restart.

Notice that when restarting with the linear combination of iterative refined Ritz vectors, a single matrix–vector product can be saved per iteration by utilizing the relationship (57) before restarting. Setting $p_1 = p_1/\|p_1\|$ and setting $\tilde{f}$ to be right side of the equality in (57) multiplied by $1/\|p_1\|$ we have $A p_1 = \tilde{f}$, $\tilde{\alpha}_1 = p_1^T \tilde{f}$ and $\tilde{f} = \tilde{f} - p_1 \tilde{\alpha}_1$, $\tilde{\beta}_1 = \|\tilde{f}\|$ and

$$
A p_1 = [p_1, p_2] \begin{bmatrix} \tilde{\alpha}_1 \\ \tilde{\beta}_1 \end{bmatrix}
$$

(65)
where $p_2 = \tilde{f}/\tilde{\beta}_1$. The MLaN(1) Algorithm 2.1 can be continued with $p_2$. Algorithm 6.1 outlines the hybrid method.

### 2.6.1 Hybrid Thick–Restarted and Iterative Refined Ritz Algorithm

Hybrid Thick–Restarted and Iterative Refined Ritz Algorithm 6.1 presents the main algorithm of the paper. Algorithm 6.1 starts with the efficient thick–restarted routine. We provide parameters as to when restarting with iterative refined Ritz vectors can be used. The parameters were chosen from numerous experiments on a variety of problems. A careful balance is needed, since the iterative refined Ritz vectors can give a better approximation when the Krylov subspace is “good”, but thick–restarted is a more efficient restarting scheme, but often has slower convergence. Also, as illustrated in Example 5.2 in Section 2.5, the iterative refined Ritz pair may be “closer” to a different Ritz pair of $T_m$ than the originally sought after Ritz pair. Therefore, since $m$ is kept small relative to $k$ we suggest using thick–restarted for the beginning iterations to build a more accurate approximation subspace, i.e. until

$$\max_{1 \leq j \leq k} |\tilde{\beta}_j| \leq \epsilon^{0.1} \|A\|$$

where $\epsilon$ is a user input tolerance for overall convergence and $\tilde{\beta}_j$ is from [9]. We then check the angle via inner product between $\hat{z}_j$ and $x_j$, i.e. between $\hat{v}_j$ and $y_j$. If the angle is acceptable we use iterative refined Ritz vector(s) to restart. Numerous experiments suggest using $\min_{1 \leq j \leq k} |y_j^T \hat{v}_j| > 0.9$. However, when $k > 1$ these conditions alone do not always prevent missing eigenvalues. One solution is to also require the input value $\mu_j$ into Iterative Refined Algorithm 5.1 to be the best approximation eigenvalue of $A$ over all computed $\theta_j$’s values thus far and to reject using restarting with iterative refined Ritz vectors if the current computed $\hat{\rho}_j$ are not “better” than the past iteration’s best approximation. For example, during a current iteration (iter) of Algorithm 6.1 if searching for the $k$ largest in magnitude eigenpairs, we require in step 5 for the call to Algorithm 5.1 that

$$\mu_j = \max_{1 \leq i \leq \text{iter}} |\theta_j^{(i)}| \quad \text{for} \quad 1 \leq j \leq k \quad (66)$$

where $\mu_j$ is the best approximation eigenvalue of $A$ over all computed $\theta_j$’s values thus far.
and for step 7
\[ |\hat{\rho}_j^{(\text{iter})}| \geq \max_{1 \leq i \leq \text{iter}-1} |\theta_j^{(i)}| \quad \text{for} \quad 1 \leq j \leq k. \quad (67) \]

Similar requirements are made for other desired extreme eigenvalue locations. When \( k = 1 \) we found that using (66) was a needed requirement for the best results, but encountered poor convergence results when enforcing (67) with \( m = 2 \), see Examples 7.4 and 7.5 Section 2.7. The following example illustrates the methods presented.

**Algorithm 6.1** Hybrid Thick–Restarted and Iterative Refined Ritz

1: **Input:** \( A \in \mathbb{R}^{n \times n} \): symmetric matrix,  
\( p_1 \in \mathbb{R}^n \): orthonormal vector,  
\( \epsilon \): usr specified tolerance,  
\( \delta_1 \): usr specified tolerance on when switching can start  
\quad \text{(recommended: } \delta_1 := \epsilon^{0.1})\),  
\( \delta_2 \): usr specified tolerance on when vectors are “close”  
\quad \text{(recommended: } \delta_2 := 0.9)\),  
\( k \): number of desired eigenpairs of \( A \),  
\( m \): maximum number of Lanczos vectors.

2: **Output:** \( k \) approximate desired eigenpair(s) of \( A \).

3: Compute factorization (3) by MLan(0) Algorithm 2.1

4: Compute the \( k \) desired eigenpair \( \{\theta_j, y_j\}_{j=1}^k \) of \( T_m (5) \) or \( \bar{T}_m (16) \)

5: Compute \( \{\hat{\rho}_j, \hat{v}_j, \hat{u}_j, \hat{\sigma}_j\}_{j=1}^k \) by Iterative Refined Algorithm 5.1 with e.g. \( \mu_j \) (66)

6: Check convergence (8) or (68)

7: if all \( \hat{\rho}_j \) converged in Algorithm 5.1 and satisfy e.g. (67) then

8: if \( \max_{1 \leq j \leq k} |\hat{\beta}_j| \leq \delta_1 \|A\| \) and \( \min_{1 \leq j \leq k} |x_j^T \hat{z}_j| > \delta_2 \) then

9: if \( k > 1 \) then compute \( c_j \) from (68), (62) or (63)

10: Restart with iterative refined (55) or (65) via MLan(1) Algorithm 2.1

11: else

12: Restart with Ritz (54) or (11) via MLan(1) Algorithm 2.1

13: Goto 4

**Example 2.6.1** Let \( A \) be the 256,000 \( \times \) 256,000 matrix \( Lin \) from the SuiteSparse Matrix Collection [4]. The largest in magnitude eigenvalue is 1063.63 and the next three largest in magnitude are 1063.32, 1063.31, and 1062.98. We set \( m = 15 \) and \( k = 1 \) and \( k = 4 \). We compared Thick–Restarted Ritz Algorithm 3.1 with Algorithm 6.1. For \( k = 1 \) we also included the restarted refined Ritz as described in [6] Algorithm...
Algorithm 6.1 converges faster than the other restarted methods. We did not include restarted refined Ritz for $k = 4$ as the method did not converge within 3000 matrix-vector products. We used the same starting vector for each routine, a normalized random vector and a stopping criteria $10^{-8} \|A\|$. For fair comparisons, we recorded the Ritz norm residual $\|Ax_1 - \theta_1 x_1\|$ for $k = 1$ and $\max \|Ax_i - \theta_i x_i\|$ for $k = 4$. For $k = 4$ we compared results using Algorithm 3.1 with Algorithm 6.1 using (58), (62), and (63). We also computed the coefficients in (56) using Saad’s method [19] indicated by Algorithm 6.1 (S) in the legend. The graphs in Figure 1 show that Algorithm 6.1 with (62) and (63) outperforms the other routines. Numerical comparison with this matrix with other routines is given in Example 2.7.3 in Section 2.7.

**Example 2.6.2** Revisiting Example 5.3 where $A = \text{diag}(1:500)$ a $500 \times 500$ diagonal matrix and $m = 2$. Figures 2a, 2b and 2c showed for 100 restarts we had stagnation, or erratic behavior and no convergence within 500 matrix-vector products using Ritz vector $P_{2y_1}$, refined Ritz $P_{2v_1}$ or iterative refined Ritz $P_{2\hat{v}_1}$ as the restarting vector in MLan(0) Algorithm 2.1. Figure 2c Algorithm 6.1, with 100 random restarts always converged within tolerance $10^{-8} \|A\|$ with no more than 170 matrix-vector products and typical convergence between 100 and 150 matrix-vector products. We also modified Algorithm 6.1 to call Algorithm 5.1 to compute refined...
Ritz vectors. All other parameters, in Algorithm 6.1 remained the same. Figure 2d displays the results and shows using refined Ritz vectors in place of iterative refined Ritz vectors performed poorly.

2.7 Numerical Examples

This section presents some numerical examples that illustrate the performance of Algorithm 6.1. For ease of comparisons we implemented Algorithm 6.1 in a MATLAB code called trreigs. We compare our method to the publicly available MATLAB code irbleigs\(^1\), the MATLAB interfaced code primme\(_eigs\)\(^2\) and MATLAB’s built-in function eigs. We refer the reader to the citations and noted websites for full details and descriptions of parameters. There are numerous selections and variety of combinations of parameters for each code. Some choices and combinations yield faster convergence than others. We cannot provide examples with all possible combinations. We used either the default values for the parameters or parameter choices that represent the fairest comparison with respect to Lanczos basis size and similarity with respect to the foundational Lanczos method for trreigs. All examples and methods used a common unit length vector with random entries that are normally distributed entries with zero mean.

The parameters for trreigs are based on Algorithm 6.1. For all examples we set \(\delta_1 := \epsilon^{0.1}, \delta_2 := 0.9\), and maximum iterations 100 for Algorithm 5.1. The number of eigenpairs \(k\), maximum size of the Lanczos basis \(m\), tolerance for convergence \(\epsilon\), and location of eigenvalues are set depending on the example. In addition to checking the Ritz residual norm (8) for termination, the code trreigs also checks

\[\|AP_m \hat{q}_j - \hat{\rho}_j P_m \hat{q}_j\| = \sqrt{(T_m \hat{q}_j - \hat{\rho}_j \hat{q}_j)^T (T_m \hat{q}_j - \hat{\rho}_j \hat{q}_j) + \beta_m^2 (e_m^T \hat{q}_j)^2} \leq \epsilon \|A\| \]  \(68\)

where \(\hat{q}_j\) is the \(j\)th column of the QR factorization of \(\hat{v}_1, \ldots, \hat{v}_k\). We use the QR

\(^1\)Code available at: http://www.math.uri.edu/~jbaglama
\(^2\)Code available at: https://github.com/primme/primme
factorization to ensure that the eigenvectors are orthogonal. Furthermore, to help avoid the pitfalls of Example 5.2, we only check (68) when \(|x^T_j \hat{z}_j| > \delta_2\). The technique of including additional vectors (> k) can greatly accelerate the convergence in restarted methods, like thick-restarting with Ritz vectors. There are many strategies for determining the number of restart vectors, see e.g. [25, 28]. A comparison of heuristic techniques is given in [28]. We implemented a simple but often effective strategy when the hybrid scheme uses thick-restarted, we restart with

\[ k = \max(\text{floor}(nc + (m - nc)/2), k) \]  

(69)

vectors where \(n_c\) is the number of converged desired approximate eigenvectors. However, using more than \(k\) vectors in the restarting scheme for iterative refined part was found to be counterproductive, often not satisfying the criteria in Algorithm 6.1 for switching. Since \(m \ll n\) the code \texttt{trreigs} uses full reorthogonalization as outlined in step 8 of Algorithm 2.1. This is a simple strategy, but can increase overall computational times.

The MATLAB code \texttt{irbleigs} is a block Lanczos method that uses the implicitly restarted formulas to apply Leja points as shifts. Given a Lanczos basis with \(m\) blocks, the method applies \(m\) Leja shifts via the implicit shift formulas until a single block of vectors is obtained and then restarts. Although the method utilizes implicit formulas for applying shifts to obtain a starting block, the overall structure can be considered an explicit restarted Lanczos method. For fair comparisons, \texttt{irbleigs} should be restricted to block size one, however that restriction often caused abnormally large number of matrix-vector products or no convergence. Therefore, in order to provide the fairest comparison with \texttt{irbleigs}, when appropriate, we recorded results with block size greater than one where the combination with the number of blocks is equal to the maximum size of the Lanczos basis \(m\). Block size and number of blocks for \texttt{irbleigs} are reported in Table 3 as (block size, number of blocks). The common
starting vector is used, where the rest of the starting block is filled in with random vectors. Besides the number $k$ of desired eigenpairs and tolerance for convergence we used the default settings for all other parameters.

MATLAB’s built-in function \texttt{eigs} used symmetric parameter true and Lanczos basis max size as $m$, and all default settings except the number $k$ of desired eigenpairs, convergence tolerance, and common starting vector.

The MATLAB interfaced code \texttt{primme\_eigs} uses the state-of-the-art high performance C99 library PRIMME for computing the eigenvalues and eigenvectors. This is an impressive, carefully designed code that includes numerous parameter settings, multiple routines/techniques, and options to include preconditioning. There are 15 choices for methods. For comparisons, we used the setting for method to be “default\_min\_matvecs” (referred to as \texttt{min\_mv} in Table\texttt{3}) which is the best method for heavy matrix-vector products and performed better than the default “dynamic” for Examples 7.1 to 7.4. On some $m$ choices for Example 7.5, “dynamic” performed better, therefore we reported the better results for Example 7.5 (“dynamic” is referred to as \texttt{dyn} in Table\texttt{3}). We also included the method “jdqmr\_etol” (referred to as \texttt{jd\_tol} in Table\texttt{3}). \texttt{primme\_eigs} allows the user to input preconditioners to accelerate convergence. We did not apply any preconditioners for the reported examples. We set the parameters “isreal” and “isdouble” to be true and used $k$ for desired eigenpairs and the common tolerance for convergence. Unless specified in the example, we used the default values for all other adjustable parameters. \texttt{primme\_eigs} allows the user to include any number of initial guesses to the eigenvectors, however we only set a starting vector to the routine to be the common starting vector used for all routines in that example. \texttt{primme\_eigs} allows the user to select the maximum size of the search subspace. We set the maximum size of the search subspace to be the common restrict value $m$ for the other routines. It should be noted, the storage requirement
and maximum size of the search space for \texttt{primme\_eigs} are not always equivalent, see \cite{23} Section 3.4.1).

All examples use the location of eigenvalues to be largest in magnitude. Example 7.1 also includes an example for smallest algebraic. In all examples, the matrix $A$ was only accessed by call to a function with input $x$ and output $Ax$. In the Table \ref{tab:cpu-times}, the cpu times are in seconds recorded using MATLAB’s tic-toc command. The row for error represents $\max \|Ax_i - \lambda_i x_i\|$ which was computed outside the routines with the outputted approximations. The references to (62) and (63) refer to the different matrices used to find the coefficients for \texttt{trreigs}. Similar to Example 6.1, using (58) reported inferior results and is not recorded. We finally remark that the performance of the methods in our comparisons also depends on the machine architecture, MATLAB coding style, and numerical implementation, (e.g. selective, partial or full reorthogonalization). The MATLAB code \texttt{trreigs} is only an illustration of Algorithm 6.1 and was not designed in the same fashion as the publicly/commercially developed codes. Nevertheless, the examples do show that Algorithm 6.1 can match or outperform the performance of the other methods. All numerical examples were performed on matrices from SuiteSparse Matrix Collection \cite{4} and all computations were carried out using MATLAB version R2019b on an iMac with 3.7Ghz Intel Core i5 processor and 32GB (2667 MHz) of memory using operating system macOS Mojave. Machine epsilon is $\epsilon = 2.2 \cdot 10^{-16}$.

\textbf{Example 2.7.1.} We considered two matrices, $2,680 \times 2,680$ \texttt{dwt2680} and $2,233 \times 2,233$ \texttt{lshp2233} that were used as numerical examples in \cite{8}. For \texttt{dwt2680} the author was seeking 5 dominant eigenvalues and for \texttt{lshp2233} the 5 smallest algebraic eigenvalues. Both examples used a stopping criteria of $10^{-6}$. The examples in \cite{8} compared several related methods, the implicitly restarted Arnoldi (IRA), the implicitly restarted refined Arnoldi (IRRA), and the implicitly restarted refined har-
monic Arnoldi (IRRHA). As a point of reference the best computed result for \textit{dwt2680} for the smallest used space \( m = 20 \), was for the IRRA with 244 mvp, \[8\] Table 7] and the best computed result for \textit{lshp2233} for the smallest used space \( m = 20 \), was for the IRRHA with 1333 mvp, \[8\] Table 6]. Table 4a displays the results for \textit{dwt2680} and Table 4b displays the results for \textit{lshp2233}. For both matrices and all methods we used \( k = 5 \) and \( \epsilon = 10^{-6} \). For \textit{dwt2680} we display results for \( m = 10, 20 \) and for \textit{lshp2233} for \( m = 20 \). The code \texttt{trreigs} displays the best results with respect to mvp for \textit{lshp2233} and when \( m = 10 \) for \textit{dwt2680} with comparable results when \( m = 20 \).

**Example 2.7.2.** We considered the 12,992 \( \times \) 12,992 matrix \textit{tuma2}. This was the only symmetric matrix that was used as a numerical example in \[12\] for finding the 6 dominant eigenvalues with a stopping criteria of \( 10^{-10} \). The example in \[12\] compared several related methods, thick–restarted block Arnoldi, modified thick–restarted block Arnoldi, a hybrid modified Ritz thick–restarted and refined block Arnoldi method, and the block Krylov-Schur algorithm \[30\]. As a point of reference the best computed result with respect to mvp for the smallest used space \( m = 18 \), was for the modified thick–restarted block Arnoldi with 1520 mvp, \[12\] Table 6]. Table 4c displays the results for \( k = 5, m = 10, 18 \) and \( \epsilon = 10^{-10} \). The code \texttt{trreigs} displays competitive results.

**Example 2.7.3** We considered the 256,000 \( \times \) 256,000 matrix \textit{Lin} that was used in Example 6.1 in Section 2.6.1. We are searching for largest eigenvalue(s) and associated vector(s). We compared the codes with \( k = 1, 4, m = 15, \) and \( \epsilon = 10^{-8} \). Table 4d displays the results. Notice that the results are significantly better than the recorded results in Fig. 1. This is due in part to using the strategy \[69\] for the thick–restarted scheme and incorporating stopping criteria \[68\]. The code \texttt{trreigs} displays the best results with respect to mvp when compared to \texttt{eigs} and \texttt{irbleigs} for both \( k = 1 \) and \( k = 4 \).
Example 2.7.4 We considered the $1,062,400 \times 1,062,400$ matrix $nlpkkt80$. We are searching for largest eigenvalue and associated eigenvector while using the smallest search space. We set $\epsilon = 10^{-6}$. eigs did not record convergence within 6000 matrix-products until $m = 10$. For $m$ equal to 3 for primme_eigs we set the parameters $maxPrevRetain = 1$ and $minRestartSize = 1$, otherwise they were set as the default values. The reported largest eigenvalue was 259.799. We include results for trreigs not requiring (67) restriction. This column is labeled 'FLT' under trreigs. Table 4e displays the results. The code trreigs with 'FLT' displays the best results with respect to mvp and smallest space $m = 2$.

Example 2.7.5 We considered the $214,005,017 \times 214,005,017$ matrix kmer_V1r. We are searching for largest eigenvalue and associated eigenvector while using the smallest possible search space. We set $\epsilon = 10^{-6}$ and used the smallest possible value for $m$ for each routine to get convergence within 200 matrix–vector products. The Matlab code, irbleigs did not converge for $m = 3, 4, 5$ and jdqmr_etol did not converge for $m = 3, 4$ and therefore are not reported. For $m$ equal to 3 for primme_eigs we set the parameters $maxPrevRetain = 1$ and $minRestartSize = 1$, otherwise they were set as the default values. We include results for trreigs not requiring (67) restriction. This column is labeled 'FLT' under trreigs. The reported largest eigenvalue was 6.50346. Table 4f displays the results. For the smallest space $m = 2$ the code trreigs with 'FLT' displays competitive results.

2.8 Conclusions

This paper presents a restarted hybrid method that combines thick–restarting with restarting with a linear combination of iterative refined Ritz vectors. The method does not require factorization of $A$, and can therefore be applied to very large problems. Numerical examples show the method to be competitive with other available codes with respect to matrix–vector products and storage required.
Appendix

Lemma 2.8.1. Given \( \|y\| = 1 \) and \( \|x\| = 1 \) then \( \|x - (x^T y)y\| \leq 1 \). Additionally, if \( x^T y \neq 0 \) then \( \|x - (x^T y)y\| < 1 \).

Proof. Follows from \( \|x - (x^T y)y\|^2 = 1 - (x^T y)^2 \) and \( 0 \leq (x^T y)^2 \leq \|x\|^2 \|y\|^2 = 1 \). \( \Box \)

List of References


Figure 2. Example 2.5.3 and 2.6.2. Matrix $A = \text{diag}(1 : 500)$ is a $500 \times 500$ diagonal matrix. We are searching for the largest eigenpair and set $m = 2$. The figures (a), (b), and (c) use MLan(0) Algorithm 2.1 where the (re)starting vector $p_1$ on the next restart to be Ritz vector $P_2y_1$, refined Ritz $P_2v_1$, and iterative refined Ritz $P_2\hat{v}_1$, respectively. The figure (d) Algorithm 6.1 but with refined Ritz vectors in place of iterative refined Ritz vectors and figure (e) uses Algorithm 6.1 as presented. The example is done 100 times for each figure with a different beginning random vector. Each line represent a start with a random vector and then a restart using the stated vector. The process was terminated at 500 matrix-vector products or when $\|Ax_1 - \theta_1x_1\| \leq 10^{-8}\|A\|$. Only Algorithm 6.1 as presented converged, figure (e).


Table 3. Numerical Examples

<table>
<thead>
<tr>
<th>m</th>
<th>treigs</th>
<th>20</th>
<th>irbleigs</th>
<th>20</th>
<th>eigs</th>
<th>20</th>
<th>primme_eigs</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvp</td>
<td>98</td>
<td>106</td>
<td>94</td>
<td>129</td>
<td>140</td>
<td>188</td>
<td>104</td>
<td>86</td>
</tr>
<tr>
<td>err</td>
<td>7.9×10^{-6}</td>
<td>7.6×10^{-6}</td>
<td>6.1×10^{-6}</td>
<td>9.4×10^{-7}</td>
<td>8.0×10^{-6}</td>
<td>7.7×10^{-6}</td>
<td>6.9×10^{-6}</td>
<td>2.3×10^{-6}</td>
</tr>
</tbody>
</table>

(a) Example 2.7.1. 2,680 × 2,680 matrix dwt2680 (k = 5). Largest in magnitude.

<table>
<thead>
<tr>
<th>m</th>
<th>treigs</th>
<th>18</th>
<th>irbleigs</th>
<th>18</th>
<th>eigs</th>
<th>18</th>
<th>primme_eigs</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvp</td>
<td>648</td>
<td>498</td>
<td>294</td>
<td>294</td>
<td>488</td>
<td>561</td>
<td>713</td>
<td>408</td>
</tr>
<tr>
<td>err</td>
<td>2.6×10^{-10}</td>
<td>2.2×10^{-10}</td>
<td>4.3×10^{-10}</td>
<td>4.3×10^{-10}</td>
<td>1.9×10^{-7}</td>
<td>1.2×10^{-8}</td>
<td>4.4×10^{-10}</td>
<td>4.8×10^{-10}</td>
</tr>
</tbody>
</table>

(b) Example 2.7.1. 2,233 × 2,233 matrix bshp2233 (k = 5, m = 20). Smallest algebraic.

<table>
<thead>
<tr>
<th>k</th>
<th>treigs</th>
<th>(3,5)</th>
<th>irbleigs</th>
<th>(3,5)</th>
<th>eigs</th>
<th>(4,5)</th>
<th>primme_eigs</th>
<th>(63)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvp</td>
<td>491</td>
<td>970</td>
<td>1129</td>
<td>1170</td>
<td>1953</td>
<td>839</td>
<td>1095</td>
<td>380</td>
</tr>
<tr>
<td>err</td>
<td>7.6×10^{-6}</td>
<td>9.5×10^{-6}</td>
<td>7.8×10^{-6}</td>
<td>7.9×10^{-6}</td>
<td>1.0×10^{-5}</td>
<td>8.3×10^{-6}</td>
<td>5.0×10^{-6}</td>
<td>9.7×10^{-6}</td>
</tr>
</tbody>
</table>

(c) Example 2.7.2. 12,992 × 12,992 matrix tunaf2 (k = 6). Largest in magnitude.

<table>
<thead>
<tr>
<th>m</th>
<th>treigs</th>
<th>20</th>
<th>irbleigs</th>
<th>20</th>
<th>eigs</th>
<th>20</th>
<th>primme_eigs</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvp</td>
<td>1098</td>
<td>692</td>
<td>1422</td>
<td>5800</td>
<td>2014</td>
<td>1507</td>
<td>742</td>
<td>1010</td>
</tr>
<tr>
<td>err</td>
<td>2.4×10^{-4}</td>
<td>2.6×10^{-4}</td>
<td>1.8×10^{-4}</td>
<td>2.6×10^{-4}</td>
<td>2.5×10^{-4}</td>
<td>2.2×10^{-4}</td>
<td>2.6×10^{-4}</td>
<td>2.3×10^{-4}</td>
</tr>
</tbody>
</table>

(d) Example 2.7.3. 256,000 × 256,000 matrix Lin (m = 15). Largest in magnitude.

<table>
<thead>
<tr>
<th>m</th>
<th>treigs</th>
<th>20</th>
<th>irbleigs</th>
<th>20</th>
<th>eigs</th>
<th>20</th>
<th>primme_eigs</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvp</td>
<td>98</td>
<td>70</td>
<td>131</td>
<td>132</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>err</td>
<td>6.4×10^{-6}</td>
<td>5.4×10^{-6}</td>
<td>6.3×10^{-6}</td>
<td>5.5×10^{-6}</td>
<td>5.5×10^{-6}</td>
<td>5.5×10^{-6}</td>
<td>4.6×10^{-6}</td>
<td>4.8×10^{-6}</td>
</tr>
</tbody>
</table>

(e) Example 2.7.4. 1,062,400 × 1,062,400 matrix nlpkkt80 (k = 1). Largest in magnitude.

<table>
<thead>
<tr>
<th>m</th>
<th>treigs</th>
<th>20</th>
<th>eigs</th>
<th>20</th>
<th>primme_eigs</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvp</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>err</td>
<td>6.4×10^{-6}</td>
<td>5.4×10^{-6}</td>
<td>6.3×10^{-6}</td>
<td>5.5×10^{-6}</td>
<td>5.5×10^{-6}</td>
<td>5.5×10^{-6}</td>
</tr>
</tbody>
</table>

(f) Example 2.7.5. 214,005,017 × 214,005,017 matrix kmer_V1r (k = 1). Largest in magnitude.
CHAPTER 3

Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods

James Baglama, Vasilije Perović, and Jennifer Picucci
Department of Mathematics and Applied Mathematical Sciences
University of Rhode Island, Kingston, Rhode Island 02881-0816, USA

This manuscript was submitted for publication to Numerical Algorithms on July 30, 2021.

Keywords: Singular value computation, partial singular value decomposition, iterative method, large-scale computation, Krylov subspace, Refined Ritz

Mathematics Subject Classification (2010): 65F15, 65F50, 15A18
Abstract This paper describes hybrid restarted Lanczos bidiagonalization methods for the computation of a few of the largest (or smallest) singular triplets of very large scale matrices. Restarting is carried out either by a thick–restarted scheme with Ritz or harmonic Ritz vectors or explicitly restarted with iterative refined Ritz vectors. Several criteria are used to determine which restarted scheme is to be used. The iterative refined Ritz vectors are computed using a scheme in which the refined process is repeated until convergence. In the context of the symmetric eigenvalue computation, the authors have shown that this iterative scheme converges and moreover it is highly effective when combined with thick–restarting in the cases when memory is limited. Given the connections with the Lanczos tridiagonal process, results are carried over in this context of computing singular triplets. Also presented, are MATLAB codes that implement the described algorithms along with numerous examples demonstrating our methods are competitive with other available routines.
3.1 Introduction

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

$$A = U \Sigma V^T$$

(70)

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 $\Sigma = \text{diag} (\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}$ such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

The $\sigma_j$’s are referred to as 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 (70), for $0 < s \leq n$, we have

$$AV_s = U_s \Sigma_s, \quad A^T U_s = V_s \Sigma_s,$$

(71)

where $\Sigma_s = \text{diag} (\sigma_1, \sigma_2, \ldots, \sigma_s) \in \mathbb{R}^{s \times s}$, $U_s = [u_1, \ldots, u_s] \in \mathbb{R}^{\ell \times s}$, and $V_s = [v_1, \ldots, v_s] \in \mathbb{R}^{n \times s}$; when $s < n$ we refer to the factorization (71) 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

$$Av_j = \sigma_j u_j, \quad A^T u_j = \sigma_j v_j, \quad j = 1, 2, \ldots, k,$$

(72)

or equivalently, computing a $k$-PSVD of $A$.

Despite the fact that the SVD of a matrix can be traced back to the 1870’s [47], it took nearly a century for it to become one of the most-widely used matrix factorizations. This is largely due to the seminal work by Golub and Kahan [13] \footnote{Otherwise replace $A$ with $A^T$.}.
creating the Golub-Kahan-Lanczos (GKL) bidiagonalization procedure where they showed that singular triplets can be computed efficiently and in a numerically stable way. Today, SVD is one of the main computational methods with numerous applications, e.g., dimension reduction, Principal Component Analysis (PCA), machine learning [28, 49, 50], genomics [1, 4, 43], data mining, data visualization, and detection of patterns [11, 18, 38]. Many of the matrices arising from these applications are typically very large, sparse and only accessible via matrix-vector routines which makes it impractical for the computation of the entire singular structure. Fortunately, in many cases one is only interested in a few largest (or smallest) singular triplets. The computation of only a few of the extreme singular triplets of very large sparse matrices has spurred a considerable amount of research and software, see e.g., [5, 6, 9, 12, 15, 24, 25, 29, 30, 31, 32, 33, 53] and the references therein.

One of the features shared by many of the referenced routines is the vital role played by the GKL procedure [13]. Recall that for some starting unit vector $p_1$ (and $q_1 := Ap_1$), this procedure creates orthonormal bases for the Krylov subspaces,

\begin{align}
\mathbb{K}_m(A^T A, p_1) &= \text{span}\left\{ p_1, A^T A p_1, (A^T A)^2 p_1, \ldots, (A^T A)^{m-1} p_1 \right\}, \\
\mathbb{K}_m(A A^T, q_1) &= \text{span}\left\{ q_1, A A^T q_1, (A A^T)^2 q_1, \ldots, (A A^T)^{m-1} q_1 \right\},
\end{align}

(73)

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

\begin{align}
A P_m &= Q_m B_m, \\
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},
\end{align}

(74) (75)

where the matrices $P_m = [p_1, \ldots, p_m] \in \mathbb{R}^{n \times m}$ and $Q_m = [q_1, \ldots, q_m] \in \mathbb{R}^{\ell \times m}$ have orthonormal columns which form bases for Krylov subspaces \((73)\) 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 $m^{\text{th}}$ axis vector of appropriate dimension and,

$$B_m := \begin{bmatrix}
\alpha_1 & \beta_1 \\
\alpha_2 & \beta_2 \\
\alpha_3 & \beta_3 \\
\vdots & \vdots \\
\beta_{m-1} & \alpha_m \\
\end{bmatrix} \in \mathbb{R}^{m \times m}$$  \hspace{1cm} (76)

is an upper bidiagonal matrix. Now 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 3.2 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. Therefore, much of the research, including this paper, revolves around developing different restarting schemes for the GKL process. Note though that there are already several notable routines that do this [5, 6, 24, 25, 29, 30], particularly the thick–restarting GKL routine in [5] which plays a vital role in this paper.

In [5], Baglama and Reichel exploited the mathematical equivalence for symmetric eigenvalue computations of the implicitly restarted Arnoldi (Lanczos) method of Sorensen [45] and the thick–restarting scheme of Wu and Simon [52] and applied it to a restarted GKL procedure; for details on the equivalences for eigenvalue computations and in the context of least squares see [34] and [7], respectively. Their thick–restarting 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 3.2. 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 3.5. 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 [52] 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. An alternate restarting scheme when using iteratively refined Ritz vectors was derived based on the relationships outlined by Morgan in [34]. Morgan showed in the case of Ritz values/vectors that implicitly restarting was equivalent to restarting with a certain linear combination of Ritz vectors. Therefore, in a similar way, we chose constants to linearly combine the iteratively refined Ritz vectors to restart the process. The idea is that this linear combination of the iteratively refined Ritz vectors will resemble 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 analysis [22, 26] 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 presented in [20], where Jia used “refined” shifts in the implicitly restarted Arnoldi method. In the context of SVD, this approach was extended by Jia and Niu resulting in an implicitly restarted GKL procedure for computing singular triplets [24, 25]. In this paper, we present another approach where we extend the restarted hybrid iterative refined scheme from [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 3.3 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, we consider the matrices

$$A^T A \in \mathbb{R}^{n \times n} \quad \text{and} \quad C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \in \mathbb{R}^{(\ell+n) \times (\ell+n)}.$$  \hfill (77)

We refer to $A^T A$ as the normal matrix or system and $C$ as the augmented matrix or system. The eigenvalues of the normal matrix $A^T A$ are the squares of singular values of $A$, while the associated eigenvectors of $A^T A$ are the corresponding right singular vectors of $A$, i.e., $A^T Av_j = \sigma_j^2 v_j$. When $\sigma_j \neq 0$, the left singular vectors can be computed as $u_j = (1/\sigma_j) A v_j$. We note that from a practical standpoint, good SVD algorithms based on the connection with $A^T A$ typically work on $A$ directly due to the fact that the explicit construction of $A^T A$ (or $AA^T$) is numerically unstable. In the case of the augmented matrix $C$, its eigenvalues are $\pm \sigma_j$ as well as $\ell - n$ zero eigenvalues. The eigenvectors of $C$ associated with $\pm \sigma_j$ are $\frac{1}{\sqrt{2}}[u_j^T, \pm v_j^T]^T$, where \{\sigma_j, u_j, v_j\} is a singular triplet of $A$.

Now observe that by multiplying equation (74) from the left by $A^T$ produces the Lanczos tridiagonal decomposition of the normal matrix $A^T A$, namely

$$A^T A P_m = P_m B_m^T B_m + \alpha_m f_m e_m^T = \begin{bmatrix} P_m & P_{m+1} \end{bmatrix} \begin{bmatrix} B_m^T B_m \\ \alpha_m \beta_m e_m^T \end{bmatrix}. \hfill (78)$$

Similarly, in the case of the augmented matrix $C$, after performing $2m$ steps of the standard Lanczos algorithm with the starting vector $[0^T, p_1^T]^T \in \mathbb{R}^{\ell+n}$ one obtains a
$2m \times 2m$ tridiagonal projection matrix, which when followed by an odd-even permutation gives the following Lanczos factorization \cite[Sec. 10.4.3]{14} \cite{29} 

\[
\begin{bmatrix}
0 & A \\
A^T & 0
\end{bmatrix}
\begin{bmatrix}
Q_m & 0 \\
0 & P_m
\end{bmatrix}
= 
\begin{bmatrix}
Q_m & 0 \\
0 & P_m
\end{bmatrix}
\begin{bmatrix}
0 & B_m \\
B_m^T & 0
\end{bmatrix}
+ 
\begin{bmatrix}
0 & 0 \\
f_m e_m^T & 0
\end{bmatrix},
\]

(79)

= 
\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}.

Finally, considering the Lanczos factorization relationships (78) and (79), the results and properties related to the hybrid iterative refined Ritz scheme in \cite{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, see Example 3.5.4. However, it should be noted that there are other routines with a specific focus on computing the smallest singular values, that utilize, for example preconditioners, factorization, or specific techniques, see e.g., \cite{25,29,33,53}.

The paper is organized as follows. The thick–restarted scheme with Ritz or harmonic Ritz vectors is given in Section 3.2 while a review of iteratively refined Ritz vectors computed either on the normal system (78) or the augmented system (79) can be found in Section 3.3. In Section 3.4 we describe our new hybrid methods and present two algorithms for computing singular triplets. Numerical examples are presented in Section 3.5 followed by conclusions in Section 3.6.

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 (colon and semicolon) for constructing block matrices. An expression of the form $\xi := \eta$ (resp., $\xi =: \eta$) is used to denote that $\xi$ is defined to be equal to $\eta$ (resp., $\eta$ is defined to be equal to $\xi$). In order to distinguish among numerous SVD computations and to
help the reader, throughout the paper we adopt the convention that superscripts \((rz)\), \((hm)\), \((rf - \ast)\), and \((it - \ast)\) correspond to the computations involving Ritz, harmonic Ritz, refined Ritz, and iteratively refined Ritz values/vectors, respectively; here \(\ast \in \{n, a\}\) denotes that (iteratively) refined Ritz are computed with respect to either the normal or the augmented systems \((77)\). Finally, when a formula is developed and used in different settings, we use a “generic” superscript \((..)\) (see Section 3.2.3).

3.2 Thick-restarted GKL process with Ritz or harmonic Ritz vectors

We briefly describe the method of thick-restarting with Ritz or harmonic Ritz vectors and refer the reader to [5] for a thorough discussion and details.

3.2.1 Thick-restarting with Ritz vectors

Let the \(s\)-PSVD of \(B_m\) from \((76)\) be given as

\[
B_m V_s^{(rz)} = U_s^{(rz)} \Sigma_s^{(rz)}, \quad B_m^T U_s^{(rz)} = V_s^{(rz)} \Sigma_s^{(rz)},
\]

\[(80)\]

where \(U_s^{(rz)} = [u_1^{(rz)}, \ldots, u_s^{(rz)}] \in \mathbb{R}^{m \times s}\) and \(V_s^{(rz)} = [v_1^{(rz)}, \ldots, v_s^{(rz)}] \in \mathbb{R}^{m \times s}\) have orthonormal columns and \(\Sigma_s^{(rz)} = \text{diag} \left( \sigma_1^{(rz)}, \ldots, \sigma_s^{(rz)} \right) \in \mathbb{R}^{s \times s}\) such that \(\sigma_1^{(rz)} \geq \sigma_2^{(rz)} \geq \cdots \geq \sigma_s^{(rz)} \geq 0\). Define \(\tilde{P}_s := P_m V_s^{(rz)}\) and \(\tilde{Q}_s := Q_m U_s^{(rz)}\), where \(P_m\) and \(Q_m\) are as in \((74)\) and \((75)\). Then from equations \((74)\), \((75)\), and \((80)\) it follows that

\[
A \tilde{P}_s = A P_m V_s^{(rz)} = Q_m B_m V_s^{(rz)} = Q_m U_s^{(rz)} \Sigma_s^{(rz)},
\]

\[
= \tilde{Q}_s \Sigma_s^{(rz)} =: \tilde{Q}_s \tilde{B}_s.
\]

\[(81)\]

Similarly,

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

\[
= \left[ \tilde{P}_s \ p_{s+1} \right] \left[ \begin{array}{c} \Sigma_s^{(rz)} \\ \rho_1 \ldots \rho_s \end{array} \right] =: \left[ \tilde{P}_s \ p_{s+1} \right] \tilde{B}_{s,s+1}^{T},
\]

\[(82)\]

where \(p_{s+1} = f/\|f\|\) and \(\rho_j = \|f\| U_s^{(rz)}(m, j)\). Note that the pair of factorizations \((81)-(82)\) can be extended via Algorithm 6.2 with \(p_{s+1}\) as the starting vector to obtain
a new factorization similar to the \( m \)-GKL factorization (74)-(75); the noted difference is in the structure of \( B_m \) which is given by

\[
B_m = \begin{bmatrix}
\tilde{B}_{s,s+1} & 0 \\
\alpha_{s+1} & \beta_{s+1} & \ddots & \ddots \\
0 & \ddots & \ddots & \beta_{m-1} \\
\alpha_m & & & \beta_m
\end{bmatrix} \in \mathbb{R}^{m \times m}.
\] (83)

One can then continue the overall process of computing approximate singular triplets of \( A \), by computing the \( s \)-PSVD of \( B_m \) in (83), setting up equations (81)-(82) and extending them via Algorithm 6.2 to an analog of (74)-(75); Algorithm 6.3 outlines this process.

**Remark.** The pairs of factorizations (74)-(75) (with \( B_m \) as in (76) or (83)) and (81)-(82) play a central role in this paper. As such, throughout the rest of this paper, we refer to (81)-(82) and (74)-(75) as an \( s \)-GKL and an \( m \)-GKL factorizations, respectively. Note that due to the structure of matrices \( \tilde{B}_s \) (81) and \( \tilde{B}_{s,s+1} \) (82), the pair (81)-(82) is not a GKL factorization in the classical sense, though it can be transformed into one \[48\].

### 3.2.2 Thick–restarting with harmonic Ritz vectors

Thick-restarting for the GKL process can also be performed with harmonic Ritz vectors \[5\]. Over the last several decades, the topics of harmonic Ritz values/vectors and their applications have been extensively studied, see e.g., \[3, 5, 6, 7, 8, 15, 16, 17, 21, 23, 25, 29, 31, 35, 36, 39\] and the references therein. Harmonic Ritz values/vectors are typically used when the focus is on approximating the smallest (or interior) singular triplets. In our numerical examples with the hybrid scheme and limited memory restrictions (see Section 3.5), we observed that the harmonic Ritz also gave very good results when searching for the largest singular triplet often performing better than thick-restarting with Ritz vectors. This may not be as surprising when one considers
the connection between thick-restarting and implicitly shifting – see [7, 35] for discussions on the equivalency of shifting by unwanted harmonic Ritz values (smallest when searching for largest) is the same as augmenting with wanted harmonic Ritz vectors.

In the rest of this section, we provide a brief outline on thick-restarting with harmonic Ritz vectors and refer the reader to [5] for additional details. Our development utilizes the connection of GKL factorization with the normal system $A^T A$.

Starting with the factorization (78) and the assumption that $B_m$ is nonsingular, we consider the symmetric matrix $B_mB_m^T + \beta_m^2 e_m e_m^T$. If $(\theta_j, g_j)$ is an eigenpair of $B_mB_m^T + \beta_m^2 e_m e_m^T$, that is,

$$
(B_mB_m^T + \beta_m^2 e_m e_m^T)g_j = \theta_j g_j,
$$

(84)

then $(\theta_j, P_mB_m^{-1}g_j)$ defines a harmonic Ritz pair of $A^T A$. It turns out that one can avoid having to explicitly construct $B_mB_m^T$ in order to compute harmonic Ritz pairs of $A^T A$ by considering the related matrix $B_{m,m+1}$ given by

$$
B_{m,m+1} := \begin{bmatrix} B_m & \beta_m e_m \end{bmatrix} \in \mathbb{R}^{m \times (m+1)}.
$$

(85)

Then for any $s < m$, the $s$-PSVD of $B_{m,m+1}$ is given by

$$
B_{m,m+1}\hat{V}_s = U_s^{(hm)}\Sigma_s^{(hm)}, \quad B_{m,m+1}^T U_s^{(hm)} = \hat{V}_s \Sigma_s^{(hm)},
$$

(86)

where $U_s^{(hm)} = [u_1^{(hm)}, \ldots, u_s^{(hm)}] \in \mathbb{R}^{m \times s}$ and $\hat{V}_s = [\hat{v}_1, \ldots, \hat{v}_s] \in \mathbb{R}^{(m+1) \times s}$ have orthonormal columns and $\Sigma_s^{(hm)} = \text{diag}(\sigma_1^{(hm)}, \ldots, \sigma_s^{(hm)}) \in \mathbb{R}^{s \times s}$ such that $\sigma_1^{(hm)} \geq \sigma_2^{(hm)} \geq \cdots \geq \sigma_s^{(hm)} > 0$.

In order to set up the thick-restarted routine with $s$ harmonic Ritz vectors one can exploit the relationship of the $s$-PSVD of $B_{m,m+1}$ in (86) and the following standard symmetric eigenvalue problem

$$
B_{m,m+1}B_{m,m+1}^T U_s^{(hm)} = (B_mB_m^T + \beta_m^2 e_m e_m^T) U_s^{(hm)} = U_s^{(hm)}(\Sigma_s^{(hm)})^2,
$$

(87)
where the diagonal entries of \((\Sigma_s^{(hm)})^2\) are the \(s\) harmonic Ritz values \(\theta_j\) in (84). Then it follows directly from the properties of block matrix multiplication and definition \(V_s^{(hm)} := B_m^{-1}U_s^{(hm)}\) that

\[
P_mV_s^{(hm)} = P_mB_m^{-1}U_s^{(hm)} = P_m \left[I \beta_m B_m^{-1}e_m\right] \hat{V}_s (\Sigma_s^{(hm)})^{-1},
\]

(88)

where

\[
\hat{V}_s = \begin{bmatrix}
B_m^{-1}U_s^{(hm)} & -\beta_m B_m^{-1}e_m \\
0 & 1
\end{bmatrix} \begin{bmatrix} I_s \\ e_m^T \hat{V}_s \end{bmatrix}.
\]

(89)

Note that the columns \(P_m v_j^{(hm)}\) of \(P_m V_s^{(hm)}\), for \(j = 1, \ldots, s\), are the harmonic Ritz vectors of \(A^T A\) associated with \(\theta_j\) in (84). Using (78), the definition of \(V_s^{(hm)}\), and (87), the residual vector for the harmonic Ritz pair \((\theta_j, P_m v_j^{(hm)})\) is given as,

\[
r_j = A^T A P_m v_j^{(hm)} - \theta_j P_m v_j^{(hm)} = e_m^T B_m v_j^{(hm)} (f - \beta_m^2 P_m B_m^{-1} e_m),
\]

\[
= e_m^T B_m v_j^{(hm)} \beta_m (p_{m+1} - \beta_m P_m B_m^{-1} e_m),
\]

\[
= e_m^T u_j^{(hm)} \beta_m \begin{bmatrix} P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} -\beta_m B_m^{-1} e_m \\ 1 \end{bmatrix}.
\]

(90)

From (90) we see that all of the residual vectors \(r_j\) are just multiplies of the same vector \(r^{(hm)}\), where

\[
r^{(hm)} := p_{m+1} - \beta_m P_m B_m^{-1} e_m.
\]

(91)

With the aim of creating formulas analogous to (81) and (82) we notice from (88) and (91) that

\[
[P_m V_s^{(hm)} \Sigma_s^{(hm)} r^{(hm)}] = [P_m \ p_{m+1}] \begin{bmatrix} B_m^{-1}U_s^{(hm)} \Sigma_s^{(hm)} & -\beta_m B_m^{-1}e_m \\ 0 & 1 \end{bmatrix}.
\]

(92)

In order to have the columns in (92) be orthonormal, we compute the QR-decomposition of the far right matrix in (92) as

\[
\begin{bmatrix} B_m^{-1}U_s^{(hm)} & -\beta_m B_m^{-1}e_m \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \hat{V}_s^{(hm)} \\ 0 \end{bmatrix} \hat{R}_{s+1},
\]

(93)

where \(\hat{V}_s^{(hm)} \in \mathbb{R}^{m \times s}\), \((\hat{V}_s^{(hm)})^T \hat{V}_s^{(hm)} = I_s\), \(\hat{v} \in \mathbb{R}^{m+1}\), \(\hat{v}(1:m)^T \hat{V}_s^{(hm)} = 0\), \(\hat{R}_{s+1} \in \mathbb{R}^{(s+1) \times (s+1)}\) is upper triangular, and define \(\hat{R}_s := \hat{R}_{s+1}(1:s,1:s)\). Similarly to our
discussion on thick–restarted with Ritz vectors, we redefine \( \tilde{P}_s := P_m \hat{V}_s^{(hm)} \) and \( \tilde{Q}_s := Q_m U_s^{(hm)} \), which together with (74) and (93) give us the following relation analogous to (81)

\[
A \tilde{P}_s = AP_m \hat{V}_s^{(hm)} = Q_m B_m \hat{V}_s^{(hm)} = Q_m U_s^{(hm)} \Sigma_s^{(hm)} \tilde{R}_s^{-1}
\]

\[
= \tilde{Q}_s \Sigma_s^{(hm)} \tilde{R}_s^{-1} =: \tilde{Q}_s \tilde{B}_s.
\]

(94)

Now in order to create an equation analogous to (82) we first notice from equations (75), (86), (89), and (93) that,

\[
A^T \tilde{Q}_s = A^T Q_m U_s^{(hm)} = P_m B_m^T U_s^{(hm)} + f e_m^T U_s^{(hm)}
\]

\[
= [P_m \, p_m + 1] B_{m,m+1}^{(hm)} U_s^{(hm)} = [P_m \, p_m + 1] \hat{V}_s \Sigma_s^{(hm)}
\]

\[
= [P_m \, p_m + 1] \begin{bmatrix} \hat{V}_s^{(hm)} \\ 0 \end{bmatrix} \begin{bmatrix} I_s \\ e_{m+1}^T \hat{V}_s \end{bmatrix} \Sigma_s^{(hm)}
\]

\[
= [P_m \, p_m + 1] \begin{bmatrix} \hat{V}_s^{(hm)} \\ 0 \end{bmatrix} \begin{bmatrix} \tilde{R}_{s+1} \Sigma_s^{(hm)} \\ e_{m+1}^T \hat{V}_s \Sigma_s^{(hm)} \end{bmatrix}.
\]

Next from (94) it follows that \( (\tilde{P}_s A^T \tilde{Q}_s)^T = \tilde{Q}_s^T A \tilde{P}_s = \tilde{B}_s \) and we define

\[
[\rho_1 \ldots \rho_s] := e_{s+1}^T \tilde{R}_{s+1} \begin{bmatrix} \Sigma_s^{(hm)} \\ e_{m+1}^T \hat{V}_s \Sigma_s^{(hm)} \end{bmatrix}, \quad p_{s+1} := P_m \hat{v}(1:m) + \hat{v}(m + 1)p_{m+1}.
\]

Returning back to the expression for \( A^T \tilde{Q}_s \) we obtain an equation analogous to (82) as

\[
A^T \tilde{Q}_s = [P_m \, p_m + 1] \begin{bmatrix} \hat{V}_s^{(hm)} \\ 0 \end{bmatrix} \begin{bmatrix} \tilde{B}_s^T \\ \rho_1 \ldots \rho_s \end{bmatrix}
\]

\[
= [P_m \hat{V}_s^{(hm)} \, p_{s+1}] \tilde{B}_{s,s+1}^T =: [\tilde{P}_s \, p_{s+1}] \tilde{B}_{s,s+1}^T.
\]

(95)

Finally, in the spirit of Remark 3.2.1 equations (94)-(95) are also called an \( s \)-GKL factorization which can then be extended with a starting vector \( p_{s+1} \) via Algorithm 6.2 to obtain an \( m \)-GKL factorization like (74)-(75), where the \( B_m \) in the factorization will have same structure as in (83) and \( \tilde{B}_{s+1} \) is now defined in (95). Then the overall process of computing approximate singular triplets can be continued in the same fashion as for thick–restarted with Ritz vectors, which is outlined in Algorithm 6.3.
3.2.3 GKL and thick-restarted GKL algorithms

We now outline algorithms that implement developments from Sections 3.2.1 and 3.2.2. Algorithm 6.2 determines an \( m \)-GKL factorization given either a starting unit vector \( p_1 \) or the \( s \)-GKL factorizations (81)-(82) or (94)-(95). In Algorithm 6.2 we assume that the \( m \)-GKL factorization can be computed, i.e., \( \alpha_j > 0 \) and \( \beta_j > 0 \); for a brief discussion on this requirement see [5] and references within. Also note that in order to avoid loss of orthogonality, lines 11 and 16 in Algorithm 6.2 implement reorthogonalization. There are several reorthogonalization strategies cited in the literature, e.g., see [30] for a discussion on partial reorthogonalization or [44] where only the columns of \( P_m \) or \( Q_m \) are reorthogonalized for matrices that are not too ill-conditioned. Given that our overall scheme in this paper maintains a small value \( m \), in line 16 we implement full reorthogonalization only on the “short” vectors (columns of \( P_m \)) for matrices that are not too ill-conditioned and switch to reorthogonalization of columns of \( Q_m \) and \( P_m \) when it is determined the matrix is ill-conditioned; in Section 3.5 this is referred to as one-sided and two-sided reorthogonalization, respectively.

To denote an approximation to the \( k \) desired singular triplets \( \{\sigma_j, u_j, v_j\} \) of \( A \) from the Krylov subspaces (73), we use notation \( \{\sigma_j^{(\cdot)}, Q_m u_j^{(\cdot)}, P_m v_j^{(\cdot)}\} \), where \( \sigma_j^{(\cdot)} \), \( u_j^{(\cdot)} \), and \( v_j^{(\cdot)} \) are taken from the described methods in this paper. For example, based on Section 3.2.1 we can use Ritz values and vectors to write

\[
\{\sigma_j^{(\cdot)}, Q_m u_j^{(\cdot)}, P_m v_j^{(\cdot)}\} = \{\sigma_j^{(rz)}, Q_m u_j^{(rz)}, P_m v_j^{(rz)}\}. \tag{96}
\]

In case of the harmonic Ritz approximations and using \( \hat{V}_k^{(hm)} \) from (93), we first compute the \( k \) approximations to the singular values

\[
\hat{s}_j^{(hm)} = u_j^{(hm)T} B_m \hat{v}_j^{(hm)} \quad j = 1, \ldots, k, \tag{97}
\]
Algorithm 6.2 Golub-Kahan-Lanczos (GKL) Process

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 output decomposition,
2: unit vector \( p_1 \) or \( s \)-GKL factorizations (81)-(82) or (94)-(95).
3: 
4: **Output:** \( m \)-GKL factorizations (74)-(75), with \( B_m \) as defined in (76), (83), or (95).
5: \( q_{s+1} := A p_{s+1} \);
6: 
7: if \( s > 0 \) then
8: \( q_{s+1} := q_{s+1} - \tilde{Q}_s (\tilde{Q}_s^T q_{s+1}) \);
9: \( \alpha_{s+1} := \| q_{s+1} \| ; \ q_{s+1} := q_{s+1} / \alpha_{s+1} \);
10: \( Q_{s+1} := [\tilde{Q}_s \ q_{s+1}]; \ P_{s+1} := [\tilde{P}_s \ p_{s+1}] \);
11: 
12: for \( j = (s + 1) : m \) do
13: \( f := A^T q_j - \alpha_j p_j \);
14: 
15: Reorthogonalization (“short”): \( f := f - P_j (P_j^T f) \);
16: 
17: if \( j < m \) then
18: \( \beta_j := \| f \| ; \ p_{j+1} := f / \beta_j \);
19: \( P_{j+1} := [P_j \ p_{j+1}] \);
20: 
21: \( q_{j+1} := A p_{j+1} - \beta_j q_j \);
22: 
23: Reorthogonalization (“long”): \( q_{j+1} := q_{j+1} - Q_j (Q_j^T q_{j+1}) \);
24: 
25: \( \alpha_{j+1} := \| q_{j+1} \| ; \ q_{j+1} := q_{j+1} / \alpha_{j+1} ; \ Q_{j+1} := [Q_j \ q_{j+1}] \);
and use
\[
\{\sigma_j^{(-)}, Q_m u_j^{(-)}, P_m v_j^{(-)}\} = \{\hat{s}_j^{(hm)}, Q_m u_j^{(hm)}, P_m \hat{v}_j^{(hm)}\}. \tag{98}
\]

**Remark.** It is important to note the difference in how approximations \((96)\) and \((98)\) are chosen. By analogy with \((96)\), \(\{\sigma_j^{(hm)}, Q_m u_j^{(hm)}, P_m v_j^{(hm)}\}\) could have been chosen as an approximation in \((98)\). But that could pose a problem since the singular vector approximations \(\{P_m v_j^{(hm)}\}\) need not be pairwise orthonormal. Therefore, in cases when more than one singular triplet is desired \((k > 1)\), for the final exit of our routine we additionally require that the both sets of vectors \(\{u_j^{(-)}\}_{j=1}^k\) and \(\{v_j^{(-)}\}_{j=1}^k\) are orthonormal. We also update \(\sigma_j^{(-)}\), such that \(\sigma_j^{(-)} = u_j^{(-)T} B_m v_j^{(-)}\).

Convergence in both methods is established by using the following residual equation that is derived from the Lanczos factorization \((79)\) on the augmented matrix \(C = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}\),

\[
resAug_j^{(-)} = \sqrt{\| A P_m v_j^{(-)} - \sigma_j^{(-)} Q_m u_j^{(-)} \|^2 + \| A^T Q_m u_j^{(-)} - \sigma_j^{(-)} P_m v_j^{(-)} \|^2}, \tag{99}
\]

\[
= \sqrt{\| B_m v_j^{(-)} - \sigma_j^{(-)} u_j^{(-)} \|^2 + \| B_m^T u_j^{(-)} - \sigma_j^{(-)} v_j^{(-)} \|^2 + (e_m^T u_j^{(-)})^2 \beta_m^2},
\]

where \(\beta_m = \|f\|\). Note that \((99)\) can be simplified when using Ritz approximation \((96)\) to \(resAug_j^{(rz)} = e_m^T u_j^{(rz)} \beta_m\). Likewise, a residual equation can be computed from the Lanczos factorization \((78)\) on the normal matrix \(A^T A\)

\[
resNor_j^{(-)} = \| A^T A P_m v_j^{(-)} - (\sigma_j^{(-)})^2 v_j^{(-)} \|,
\]

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

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 \((99)\) and the condition

\[
resAug_j^{(-)} \leq tol \cdot \|A\|, \tag{100}
\]

59
where \( \text{tol} \) is a user specified tolerance and \( \|A\| \) is approximated by the largest singular value of \( B_m \) over all iterations.

**Algorithm 6.3** Thick–restarted GKL with Ritz or harmonic Ritz vectors

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,
2. \( k \) : number of desired singular triplets,
3. \( p_1 \) : unit vector,
4. \( \text{tol} \) : tolerance for accepting computed approximate singular triple, cf. (100).
5. **Output:** \( k \) approximate singular triplets of \( A \) \( \{ \sigma_j, u_j, v_j \}_{j=1}^k \).
6. Compute \( m \)-GKL factorization with Algorithm 6.2;
7. Compute the \( s \)-PSVD of \( B_m \) (80) with \( k \leq s < m \);
8. Check convergence of \( k \) desired triplets (100) with (96);
9. **if** thick–restarted with Ritz **then**
10. Compute \( s \)-GKL factorization (81)–(82);
11. Goto 3;
12. **else**
13. Compute the \( s \)-PSVD of \( B_{m,m+1} \) (86);
14. Check convergence (100) with (98);
15. Compute \( s \)-GKL factorization (94)–(95);
16. Goto 3;

3.3 **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 \( M \) [19]. More specifically, for a given approximate eigenvalue \( \mu_j \) of \( M \), Jia’s method looks to minimize \( \|Mz_j - \mu_jz_j\| \) for a unit vector
\( z_j \) from a given subspace \( W \), i.e.,

\[
\min_{z_j \in W, \|z_j\|=1} \|Mz_j - \mu_j z_j\|. \tag{101}
\]

It was shown that on the subspace \( W \), refined Ritz vectors \( z_j \) provided better eigenvector approximations than the Ritz vectors. Moreover, 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, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27, 29, 37] and the 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; for complete details on this development and related theoretical properties see [2]. An important and subtle result regarding iterative refined or refined Ritz vectors is that they are \textit{not} scalar multiples of the same residual vector and as such do not fit naturally into the thick–restarted scheme developed by Wu and Simon [52]. Through numerical examples in [2] we also 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. This behavior can, in part, be explained by the refined process (minimization of the norm (101) on a small subspace) possibly favoring the next closest eigenvalue, see [2, Ex. 5.2]. 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. We note that if the iterative refined vectors were replaced with refined vectors in this hybrid scheme, the overall method did not perform as well, see [2, Ex. 5.3 and 6.2].

In this paper, we extend the idea of iterative refined values/vectors to the GKL process and develop a similar hybrid scheme for computing singular triplets. Con-
sidering the relationships of the Lanczos factorizations (78) and (79) and symmetric matrices $A^T A$ and $C = [A^T 0 0]$, respectively, the properties in [2] are mostly carried over though some nontrivial adaptations were necessary (e.g., see Section 3.3.2). It is worth noting that refined schemes for computing singular triplets using the Lanczos bidiagonal method with matrix $C$ have been considered in [24, 25]. More specifically, the refined Ritz scheme in [24] uses the lower bidiagonal Lanczos process [40] while the scheme in [25] utilizes the GKL process and computes refined harmonic Ritz values/vectors using the augmented system (79). Both schemes [24, 25] 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 [29] 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 [6] that used Leja points as “shifts” from the normal equations (78). In contrast to these methods, the primary focus of our method in this paper is not on computing “shifts” but rather on a hybrid restarting scheme that restarts the GKL process either through thick-restarting with Ritz/harmonic Ritz or explicitly restarting with linear combination of iterative refined Ritz vectors.

3.3.1 Refined and Iterative Refined on normal system

Our development of the iterative refined Ritz values/vectors naturally starts with the normal system (78). To that end, let $M = A^T A$ and $\mathcal{W} = \mathbb{K}_m(A^T A, p_1)$ in equation (101) and define

$$T_{m+1,m} := \begin{bmatrix} B_m^T B_m \\ \alpha_m \beta_m e_m^T \end{bmatrix} \in \mathbb{R}^{(m+1) \times m}. \quad (102)$$

For each approximate eigenvalue $\mu_j$ of $A^T A$ compute the smallest singular value $\sigma_{(n-n)}^j$ and associated unit singular vectors of $(T_{m+1,m} - \mu_j I_{m+1,m})$, i.e.,
\[(T_{m+1,m} - \mu_j I_{m+1,m}) v_j^{(rf-n)} = \sigma_j^{(rf-n)} w_j, \quad (103)\]
\[(T_{m+1,m} - \mu_j I_{m+1,m})^T w_j = \sigma_j^{(rf-n)} v_j^{(rf-n)}, \quad (104)\]

where \(v_j^{(rf-n)} \in \mathbb{R}^m\) and \(w_j \in \mathbb{R}^{m+1}\). Then from (74), (75), and (78) it follows that
\[
\min_{z_j \in \mathbb{R}^m \setminus A^T A p_1} \| A^T A z_j - \mu_j z_j \| = \| (T_{m+1,m} - \mu_j I_{m+1,m}) v_j^{(rf-n)} \| = \sigma_j^{(rf-n)} \quad (105)
\]
and the refined Ritz vector \(z_j\) for \(\mu_j\) is defined as \(z_j := P_m v_j^{(rf-n)}\). The approximate eigenvalue of \(A^T A\) associated with the refined Ritz vector \(z_j\) is selected as the Rayleigh quotient
\[
\sigma_j^{(rf-n)} = z_j^T A^T A z_j = v_j^{(rf-n)}^T B_m^T B_m v_j^{(rf-n)} = \| B_m v_j^{(rf-n)} \|^2, \quad (106)
\]
and the approximate refined singular triplet on the normal system for \(A\) is given by
\[
\{ \sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)} \} = \{ \sigma_j^{(rf-n)}, Q_m u_j^{(rf-n)}, P_m v_j^{(rf-n)} \}, \quad (107)
\]
where \(u_j^{(rf-n)} = B_m v_j^{(rf-n)} / \sigma_j^{(rf-n)}\).

The initial approximate eigenvalue \(\mu_j\) in equations (103)-(105) can be taken as the Ritz value \(\sigma_j^{(rz)}\) (80). Alternately, \(\mu_j\) can be chosen as harmonic Ritz value \(\sigma_j^{(hm)}\) (86) in which case the output from the refined process is referred to as refined harmonic Ritz. For the purpose of streamlining exposition, in this section we initially set approximate eigenvalues \(\mu_j = \sigma_j^{(rz)}\) and refer the reader to [21, 23] for theoretical considerations on refined harmonic Ritz. The iterative refined Ritz process iteratively refines the approximation, by taking the output approximation, \(\sigma_j^{(rf-n)}\) (106), setting \(\mu_j = \sigma_j^{(rf-n)}\), and re–computing refined vectors \(v_j^{(rf-n)}\), via (103)-(104) until convergence. This process produces a nonnegative, decreasing and hence convergent sequence \(\sigma_j^{(..)(i)}\), see [2, Thm. 5.1]; Algorithm 6.4 outlines this process.

There are several options for the convergence check (steps 8 and 16) in Algorithm 6.4, e.g., \(|\sigma_j^{(..)(i)} - \sigma_j^{(..)(i-1)}| / |\sigma_j^{(..)(i)}| < eps\), where eps is machine epsilon; the
Algorithm 6.4 Iterative Refined

1: **Input:** $T_{m+1,m} \in \mathbb{R}^{(m+1) \times m}$ (102) or $T_{2m+1,2m} \in \mathbb{R}^{(2m+1) \times 2m}$ (122) and $\{\mu_j\}_{j=1}^k$.

2: **Output:** $\{\sigma_{j}^{(i-t-n)}(it-n), u_{j}^{(i-t-n)}, v_{j}^{(i-t-n)}\}_{j=1}^k$ and $\hat{\sigma}_{j}^{(i-t-n)}$ or $\{\sigma_{j}^{(i-t-a)}(it-a), u_{j}^{(i-t-a)}, v_{j}^{(i-t-a)}\}_{j=1}^k$ and $\hat{\sigma}_{j}^{(i-t-a)}$.

4: for $j = 1, 2, \ldots, k$ do
5:   for $i = 1, 2, \ldots, \text{maxitref}$ do
6:     if normal system then
7:        Compute $v_{j}^{(rf-n)}(i), w_{j}^{(i)},$ and $\sigma_{j}^{(rf-n)}(i)$ (103) and (104);
8:        $\sigma_{j}^{(rf-n)} := \|B_{m}v_{j}^{(rf-n)}(i)\|$ (106);
9:        if converge then
10:           $\sigma_{j}^{(i-t-n)} := \sigma_{j}^{(rf-n)}(i), v_{j}^{(i-t-n)} := v_{j}^{(rf-n)}(i), u_{j}^{(i-t-n)} := B_{m}v_{j}^{(i-t-n)}/\sigma_{j}^{(i-t-n)}, \hat{\sigma}_{j}^{(i-t-n)} := \hat{\sigma}_{j}^{(rf-n)}(i);$
11:           Break;
12:     end
13:     else
14:        Compute $x_{j}^{(i)}, y_{j}^{(i)}, w_{x_{j}}^{(i)}, w_{y_{j}}^{(i)}, w_{z_{j}}^{(i)},$ and $\sigma_{j}^{(i-t-a)}(i)$ (123) and (124);
15:        $\sigma_{j}^{(i-t-a)} := 2x_{j}^{(i)}B_{m}y_{j}^{(i)}$ (126);
16:        if converge and $\|x_{j}^{(i)}\| - 1/\sqrt{2} \leq \sqrt{\text{eps}}$ then
17:           $\sigma_{j}^{(i-t-a)} := \sigma_{j}^{(i-t-a)}(i), v_{j}^{(i-t-a)} := y_{j}^{(i)}/\|y_{j}^{(i)}\|, u_{j}^{(i-t-a)} := x_{j}^{(i)}/\|x_{j}^{(i)}\|, \hat{\sigma}_{j}^{(i-t-a)} := \hat{\sigma}_{j}^{(i-t-a)}(i);$
18:           Break;
19:        end
20:     end
21:     $\mu_{j} := (\sigma_{j}^{(i-t-a)}(i))^{2};$
22: end
additional requirement on $\|x_j^{(i)}\|$ in step 16 is discussed in Section 3.3.2. 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 6.4 (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 3.5. 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 6.4 with initial approximate eigenvalues $\mu_j = \sigma_j^{(r_{n}))2$, we obtain the approximate iterative refined Ritz singular triplet on the normal system for $A$ as

$$\{\sigma_j^{(-)}, Q_m u_j^{(-)}, P_m v_j^{(-)}\} = \{\sigma_j^{(it-n)}, Q_m u_j^{(it-n)}, P_m v_j^{(it-n)}\}. \quad (108)$$

Using the $m$-GKL factorization and the refined Ritz singular approximation (107), together with equations (103)-(104), give us

$$A P_m v_j^{(it-n)} = Q_m B_m v_j^{(it-n)} = \sigma_j^{(it-n)} Q_m u_j^{(it-n)}, \quad (109)$$

$$A^T Q_m u_j^{(it-n)} = P_m B_m^T u_j^{(it-n)} + f e_m^T u_j^{(it-n)}, \quad (110)$$

$$= \sigma_j^{(it-n)} P_m v_j^{(it-n)} + \alpha_j^{(it-n)}/\sigma_j^{(it-n)} [P_m p_{m+1}] r_j,$$

where $r_j = w_j - ([v_j^{(it-n)}; 0]^T w_j)[v_j^{(it-n)}; 0]$. Multiplying (109) by $A^T$ on the left yields the following relation

$$A^T A P_m v_j^{(it-n)} = \sigma_j^{(it-n)2} P_m v_j^{(it-n)} + \alpha_j^{(it-n)} [P_m p_{m+1}] r_j. \quad (111)$$
If Algorithm 6.4 is used to compute the iterative refined Ritz value and vectors we have the output satisfying,

\[(T_{m+1,m} - \sigma_j^{(it-n)^2} I_{m+1,m}) v_j^{(it-n)} = \hat{\sigma}_j^{(it-n)} \hat{w}_j,\]  
\[(T_{m+1,m} - \sigma_j^{(it-n)^2} I_{m+1,m})^T \hat{w}_j = \hat{\sigma}_j^{(it-n)} v_j^{(it-n)},\]

and since \(\sigma_j^{(it-n)^2} = v_j^{(it-n)^T} B_m B_m v_j^{(it-n)}\) we have from (112) \([v_j^{(it-n)^T}; 0]^T \hat{w}_j = 0\). Analogous to equations (109)-(110) with iterative refined Ritz singular approximation (108) we have,

\[AP_m v_j^{(it-n)} = Q_m B_m v_j^{(it-n)} = \sigma_j^{(it-n)} Q_m u_j^{(it-n)} \]
\[A^T Q_m u_j^{(it-n)} = P_m B_m^T u_j^{(it-n)} + f e_m^T u_j^{(it-n)} = \sigma_j^{(it-n)} P_m v_j^{(it-n)} + \hat{\sigma}_j^{(it-n)} / \sigma_j^{(it-n)} [P_m \ p_{m+1}] \hat{w}_j \]

and after multiplying (114) by \(A^T\)

\[A^T A P_m v_j^{(it-n)} = \sigma_j^{(it-n)^2} P_m v_j^{(it-n)} + \hat{\sigma}_j^{(it-n)} [P_m \ p_{m+1}] \hat{w}_j.\]  

Applying [2] Eqns. (5.5) and (5.12) to Lanczos relationships (111) and (116) shows that

\[\hat{\sigma}_j^{(it-n)} = resNor_j^{(it-n)} \leq resNor_j^{(it-n)} \leq resNor_j^{(it-n)}.\]  

Equation (117) shows that the iterative refined Ritz with respect to the normal residual on the same Krylov subspace, \(K_m(A^T A, p_1)\) are better approximations, however an effective restart process that “improves” the next generated Krylov subspace is still needed. Equations (109)-(111) and (114)-(116) 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 3.2 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.

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

The steps in (118) yield the following 1-GKL factorization

$$Ap_1 = q_1 \alpha_1,$$  \hspace{1cm} (119)
$$A^T q_1 = [p_1, p_2] \left[ \begin{array}{c} \alpha_1 \\ \beta_1 \end{array} \right],$$  \hspace{1cm} (120)

where Algorithm 6.2 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 (119)-(120) are the same as equations (109)-(110) or (114)-(115), respectively. For $k > 1$ the coefficients $c_j$ in (118) can be chosen several ways and greatly impact convergence. For example for eigenvalue problems, Saad [41] suggests using residual norms, which was also used for the refined Ritz algorithm [19, Alg. 1]. In [2] an alternate approach for iterative refined vectors modeled after Morgan [34] was used to eliminate part of the residual vector as the next Krylov subspace is built. Morgan [34] 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 [45]. 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 (121) for coefficients $c_j$, we refer the reader to [2, Sec. 6] for details.

$$\begin{bmatrix}
\beta_m \\
\sigma_1^{(\text{it-n})} & e_m^T v_1^{(\text{it-n})} & \cdots & e_m^T v_k^{(\text{it-n})} \\
\sigma_2^{(\text{it-n})} & e_m^T B_m v_1^{(\text{it-n})} & \cdots & e_m^T B_m v_k^{(\text{it-n})} \\
\vdots & \vdots & \ddots & \vdots \\
\sigma_k^{(\text{it-n})} & e_m^T B_m B_m v_1^{(\text{it-n})} & \cdots & e_m^T B_m B_m v_k^{(\text{it-n})}
\end{bmatrix} \bar{c} = 0,$$  \hspace{1cm} (121)
3.3.2 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 $W = \mathbb{K}_{2m}(C, [0; p_1])$ in equation (101) and define

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

(122)

For each initial eigenvalue approximation $\mu_j$ of $C$ compute the smallest singular value $\sigma_{(rf-a)} \downarrow j$ and associated unit singular vectors of $(T_{2m+1,2m} - \mu_j I_{2m+1,2m})$, i.e.,

$$
(T_{2m+1,2m} - \mu_j I_{2m+1,2m}) \begin{bmatrix} x_j \\ y_j \end{bmatrix} = \sigma_{(rf-a)} \downarrow j \begin{bmatrix} w_{x_j} \\ w_{y_j} \\ w_{z_j} \end{bmatrix},
$$

(123)

$$
(T_{2m+1,2m} - \mu_j I_{2m+1,2m})^T \begin{bmatrix} w_{x_j} \\ w_{y_j} \\ w_{z_j} \end{bmatrix} = \sigma_{(rf-a)} \downarrow j \begin{bmatrix} x_j \\ y_j \end{bmatrix},
$$

(124)

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

$$
\min_{z_j \in \mathbb{K}_{2m}(C,[0;p_1])} \|Cz_j - \mu_j z_j\| = \|(T_{2m+1,2m} - \mu_j I_{2m+1,2m}) \begin{bmatrix} x_j \\ y_j \end{bmatrix}\| = \sigma_{(rf-a)} \downarrow j
$$

(125)

and the refined Ritz vector $z_j$ for $\mu_j$ 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

$$
\sigma_{(rf-a)} \downarrow j = z_j^T C z_j = \begin{bmatrix} x_j \\ y_j \end{bmatrix}^T \begin{bmatrix} 0 & B_m \\
B_m^T & 0
\end{bmatrix} \begin{bmatrix} x_j \\ y_j \end{bmatrix} = 2x_j^TB_m y_j,
$$

(126)

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

$$
\{\sigma_{\downarrow j}, Q_m u_{\downarrow j}, P_m v_{\downarrow j}\} = \{\sigma_{(rf-a)} \downarrow j, Q_m u_{(rf-a)} \downarrow j, P_m v_{(rf-a)} \downarrow j\},
$$

(127)

where $u_{(rf-a)} \downarrow j = x_j/\|x_j\|$ and $v_{(rf-a)} \downarrow j = y_j/\|y_j\|$. Similar to (111) for the normal system, but this time applied to the Lanczos factorization (79) for the augmented matrix $C$. 

68
we have the following relationship,

\[
\begin{bmatrix}
0 & A \\
A^T & 0
\end{bmatrix}
\begin{bmatrix}
Q_m x_j \\
P_m y_j
\end{bmatrix}
= \sigma_j^{(\text{rf-a})}
\begin{bmatrix}
Q_m x_j \\
P_m y_j
\end{bmatrix}
+ \sigma_j^{(\text{rf-a})}^{(\text{rf-a})}
\begin{bmatrix}
Q_m & 0 & 0 \\
P_m & P_{m+1}
\end{bmatrix}
\begin{bmatrix}
r_{xj} \\
r_{yj} \\
r_{zj}
\end{bmatrix},
\]

(128)

where \( r_{zj} = w_{zj} \) is a scalar, \( r_{yj} = w_{yj} - [x_j; y_j]^T[w_{xj}; w_{yj}]y_j \in \mathbb{R}^m \), and \( r_{xj} = w_{xj} - [x_j; y_j]^T[w_{xj}; w_{yj}]x_j \in \mathbb{R}^m \). Given the relationship between the eigenvalues of \( C \) and the singular values of \( A \), we can start Algorithm 6.4 with the initial approximation \( \mu_j \) in equations (123)-(125) 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

\[
\{\sigma_j^{(\text{it-a})}, Q_m u_j^{(\text{it-a})}, P_m v_j^{(\text{it-a})}\} = \{\sigma_j^{(\text{it-a})}, Q_m u_j^{(\text{it-a})}, P_m v_j^{(\text{it-a})}\}.
\]

(129)

For convenience, consider the unscaled output vectors of \( u_j^{(\text{it-a})} \) and \( v_j^{(\text{it-a})} \) from Algorithm 6.4 as the last iteration vectors \( \hat{x}_j := x_j^{(i)} \) and \( \hat{y}_j := y_j^{(i)} \), respectively. Therefore, analogous to (112)-(113) and (116), we have the output from Algorithm 6.4 that satisfies

\[
(T_{2m+1,2m} - \sigma_j^{(\text{it-a})} I_{2m+1,2m})
\begin{bmatrix}
\hat{x}_j \\
\hat{y}_j
\end{bmatrix}
= \hat{\sigma}_j^{(\text{it-a})}
\begin{bmatrix}
\hat{w}_{xj} \\
\hat{w}_{yj} \\
\hat{w}_{zj}
\end{bmatrix},
\]

(130)

\[
(T_{2m+1,2m} - \sigma_j^{(\text{it-a})} I_{2m+1,2m})^T
\begin{bmatrix}
\hat{w}_{xj} \\
\hat{w}_{yj} \\
\hat{w}_{zj}
\end{bmatrix}
= \hat{\sigma}_j^{(\text{it-a})}
\begin{bmatrix}
\hat{x}_j \\
\hat{y}_j
\end{bmatrix},
\]

(131)

where \([\hat{x}_j; \hat{y}_j]^T [\hat{w}_{xj}; \hat{w}_{yj}] = 0\), \( \hat{x}_j, \hat{y}_j, \hat{w}_{xj}, \hat{w}_{yj} \in \mathbb{R}^m \), and \( \hat{w}_{zj} \) is a scalar and when applied to the Lanczos factorization (79) gives us the following

\[
\begin{bmatrix}
0 & A \\
A^T & 0
\end{bmatrix}
\begin{bmatrix}
Q_m \hat{x}_j \\
P_m \hat{y}_j
\end{bmatrix}
= \hat{\sigma}_j^{(\text{it-a})}
\begin{bmatrix}
Q_m \hat{x}_j \\
P_m \hat{y}_j
\end{bmatrix}
+ \hat{\sigma}_j^{(\text{it-a})}^{(\text{it-a})}
\begin{bmatrix}
Q_m & 0 & 0 \\
P_m & P_{m+1}
\end{bmatrix}
\begin{bmatrix}
\hat{w}_{xj} \\
\hat{w}_{yj} \\
\hat{w}_{zj}
\end{bmatrix}.
\]

(132)

Similar to (117), the relationships (128) and (132) together with (2) Eqns. (5.5) and (5.12) applied to symmetric matrix \( C \) imply that

\[
\hat{\sigma}_j^{(\text{it-a})} = \text{resAug}^{(\text{it-a})}_j \leq \text{resAug}^{(\text{rf-a})}_j \leq \text{resAug}^{(\text{rz})}_j.
\]

(133)
Equation (133) shows that the iterative refined Ritz with respect to the augmented residual on the same Krylov subspace $K_{2m}(C, [0; p_1])$ are better approximations. But relation (133) 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 $\|x_j\|, \|y_j\|, \|\hat{x}_j\|, \|\hat{y}_j\|$ 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 6.4. This requirement is reasonable by observing that from the iterative process of Algorithm 6.4 and equations (122), (123), and (126) it follows that

$$x_j^{(i)} = 1/\sigma_j^{(rf-a)}(i-1) \left( B_m \hat{y}_j^{(i)} - \sigma_j^{(rf-a)}(i) w_x^{(i)} \right).$$

(134)

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

$$\hat{x}_j = 1/\sigma_j^{(it-a)} \left( B_m \hat{y}_j - \hat{\sigma}_j^{(it-a)} \hat{w}_x \right),$$

(135)

$$\|\hat{x}_j\|^2 = 1/2 - \hat{\sigma}_j^{(it-a)} / \sigma_j^{(it-a)} \hat{x}_j^T \hat{w}_x.$$

(136)

If $\hat{\sigma}_j^{(it-a)} = 0$, then we have the desired property and convergence (see (133)). If $\hat{\sigma}_j^{(it-a)} \neq 0$, then from (123) and (126) we have the relationship $\hat{x}_j^T \hat{w}_x = - \hat{y}_j^T \hat{w}_y$. After multiplying (130) by $[\hat{w}_x ; 0 ; 0]^T$ and using $B_m \hat{w}_x - \sigma_j^{(it-a)} \hat{w}_y = \hat{\sigma}_j^{(it-a)} \hat{y}_j$ from (131), we obtain

$$||\|\hat{x}_j\|^2 - 1/2|| \leq (\hat{\sigma}_j^{(it-a)} / \sigma_j^{(it-a)})^2 ||\|\hat{w}_x\|^2 - ||\hat{y}_j\|^2||/2,$$

(137)

where the inequality is established using the triangle inequality and the fact that $||\hat{w}_x|| < 1$ 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 6.4 resulted in a better performance in our hybrid algorithm for the augmented system.
Remark. 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 \( \hat{\sigma}_{j}^{(t-a)} \) in (132) to approach 0. As \( \hat{\sigma}_{j}^{(t-a)} \to 0 \), from (135)-(137) we have that \( \|\hat{x}_{j}\| \approx \|\hat{y}_{j}\| \approx 1/\sqrt{2} \), \( u_{j}^{(t-a)} \approx 1/\sigma_{j}^{(t-a)}B_{m}v_{j}^{(t-a)} \), and \( \sigma_{j}^{(t-a)} \approx \|B_{m}v_{j}^{(t-a)}\| \). Moreover, we start to see the residual relation (133) holding on the normalized vectors and the alignment with the iterative refined Ritz values/vectors on the normal system. Therefore, we use formulas (118) with \( v_{j}^{(-)} := v_{j}^{(t-a)} \) to obtain the 1-GKL factorization (119)-(120) where Algorithm 6.2 can be restarted with \( p_{2} \). Likewise, when \( k > 1 \), we can replace \( v_{j}^{(-)} := v_{j}^{(t-a)} \) and \( \sigma_{j}^{(-)} := \sigma_{j}^{(t-a)} \) and solve the homogeneous system (121) to restart with a linear combination of vectors. Although an alignment is eventually expected, there are convergence differences, see the numerical examples in Section 3.5.

The goal of the paper is the development of a restarted GKL scheme with a focus on keeping the value \( m \) relatively small in relationship with \( k \), e.g. \( m = k + 1 \). 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. Putting aside the efficiency of thick–restarted when \( k > 1 \) and focusing on restarting with a single vector, we see that the minimization process may not always coincide with the best restart vector. Even when refined vectors do yield better results, the results may not be profoundly significant. The following example illustrates this point.

Example 3.3.1 For this example, we look at two matrices, \( A = \text{diag}(1:500) \) a 500 \( \times \) 500 diagonal matrix and \( A \) being the 262111 \( \times \) 262111 amazon0302 matrix from the SuiteSparse Matrix Collection [10]. We let \( k = 1 \) and \( m = 2 \) and search for the largest singular triplet with tolerance \( 10^{-6} \) while using (100) as a stopping criteria.
We started Algorithm 6.2 with a random vector $p_1$ and then on the next restart of Algorithm 6.2 we computed $p_1$ to be Ritz vector $P_m v_1^{(rz)}$, refined Ritz on normal system $P_m v_1^{(rf-n)}$, iterative refined Ritz on normal system $P_m v_1^{(it-n)}$, refined Ritz on augmented system $P_m v_1^{(rf-a)}$, or iterative refined Ritz on augmented system $P_m v_1^{(it-a)}$. All methods for both examples ran 10 times with a different random starting vector $p_1$ where each method started with the same random vector. Since this example is focused on measuring the overall convergence, i.e., the quality of the Krylov subspace, and for ease of comparison, we only computed the common Ritz norm for each restart method, i.e., $resAug_1^{(rz)}$. The results are presented in Figure 3. The figures display the number of matrix vector products with $A$ and $A^T$ against the residual norm computed with Ritz value $\sigma_1^{(rz)}$ and vector $P_m v_1^{(rz)}$, i.e., $resAug_1^{(rz)}$ for all routines.

We see from the outputs in Figure 3 a wide range of convergence with typically (not always) restarting with refined Ritz vectors performing better than restarting with Ritz vectors. Figure 3 also shows the methods struggling at the beginning, especially with the amazon0302 example (see Figure 3b). This suggests that the methods are having difficulty on a small subspace, capturing the needed components of desired singular vector for restarting. Although not displayed, and as expected, when we increased the value of $m$ the differences between routines becomes 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.

The calculations of iterative refined Ritz vectors are more sensitive to converging to the next closest Ritz value during the iteration process causing stagnation. Figure 3b shows iterative refined Ritz on the normal system stagnating for all restarts. At first glance, this sensitivity appears to be a disadvantage for the iterative refined vectors, but it can in fact be used to more easily signal when the iterative refined vectors should not be used to restart the system. Exactly this reliance on sensitivity
is what led us to develop a new hybrid method in this paper. Our hybrid method uses thick–restarting with either Ritz or harmonic Ritz vectors and when certain criteria are met it switches to restarting with iterative refined Ritz vectors on the normal or the augmented system. This combination works very well when \( m \) is relatively small – for relevant discussion and numerical results on the same matrices but with our hybrid method Algorithm 6.5, see Example 3.4.1 and Figure 4.

### 3.4 Hybrid Iterative Refined Algorithms

We now present the main contributions of this paper, namely Algorithms 6.5–6.6. In the previous section, we saw that despite the fact that the iterative refined Ritz values/vectors can provide “better” approximations on a given Krylov subspace, using them to restart the GKL method need not yield better overall convergence and in fact can lead to stagnation (see Example 3.3.1). The stagnation or slow convergence is contributed in part to the use of a small subspace along with the iterative refined process causing the output approximation to be “closer” to the next closest Ritz value/vector. This apparent weakness turns out to be more of an advantage of the iterative refined Ritz values/vectors, since they can be used to more easily identify
when they are not a good choice to restart the GKL process. The implications of this are twofold: there ought to be a process to determine when the iterative refined vectors are good to be used to restart the GKL process and an alternate choice of restart vectors is needed. The thick–restarted methods described in Section 3.2 are a more efficient restarted GKL method with a theoretical connection to implicit restarting, but can not be used directly with (iterative) refined Ritz vectors. Consequently, we advocate to use a hybrid method that utilizes thick–restarting as the main routine and, under certain conditions described below, switches to restarting with iterative refined Ritz vectors.

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

$$\min_{1 \leq j \leq k} |v_j^{(it-a)} T v_j^{(-)}| > 0.9,$$  \hspace{1cm} (138)

where $v_j^{(-)} := v_j^{(it-n)}$ if using the normal system and $v_j^{(-)} := v_j^{(it-a)}$ if using 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 [22, 27]. Since thick–restarted is the main routine with theoretical connection to implicitly restarting techniques and foundations 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 \(\mu_j\) into Algorithm 6.4 to be the best approximation for singular value of \(A\) over all computed \(\sigma_j^{(rz)}\)’s values thus far and to reject using restarting with iterative refined Ritz vectors if the current computed iterative refined values, \(\sigma_j^{(it-n)}\) or \(\sigma_j^{(it-a)}\), are not “better” than the past iteration’s best approximation. For example, during a current iteration (iter) of Algorithm 6.5 we require in step 9 for the call to Algorithm 6.4 that

\[
\mu_j = \max_{1 \leq i \leq \text{iter}} |\sigma_j^{(rz)}(i)| \quad \text{for} \quad 1 \leq j \leq k
\]

and for step 10

\[
|\sigma_j^{(\text{iter})}| \geq \max_{1 \leq i \leq \text{iter-1}} |\sigma_j^{(rz)}(i)| \quad \text{for} \quad 1 \leq j \leq k,
\]

where \(\sigma_j^{(\cdot)} := \sigma_j^{(it-n)}\) for normal system and \(\sigma_j^{(\cdot)} := \sigma_j^{(it-a)}\) for augmented system. When \(k = 1\) we found that using \((139)\) was a needed requirement for the best results, but encountered poor convergence results when enforcing \((140)\) with \(m = 2\). Similar criteria is used for searching for the smallest singular triplets. Additionally, due to a negligible computational cost, various convergence checks are performed at different stages of Algorithm 6.5 e.g., see steps 4, 7, 11, and 17 – this allows for Algorithm 6.5 to exit at the right time and to avoid performing unnecessary expensive computations.

We note to the reader that Algorithm 6.5 is a simplification of the actual computations performed. For instance, in the thick–restarted steps 19 and 21 in Algorithm 6.5 we compute \(s\)-GKL factorization where \(s \geq k\) to restart Algorithm 6.3. 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 [7, 24, 25, 30] for details. Considering the connection between implicitly shifting with (harmonic) 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 \( s \geq k \),

\[
\begin{align*}
  s &= k + nc; \\
  \text{if } |\sigma_{s+1} - \sigma_s| < |\sigma_s - \sigma_{s-1}|, s &= s + 1; \text{ end} \\
  s &= \max(\floor{(m + nc)/2}, s); \\
  \text{if } s > m, s &= m - 1; \text{ end}
\end{align*}
\]

where \( n_c \) is the number of converged singular triplets, see [51] for details and comparison of techniques. The strategy in (141) works well in this context, particularly when difference between \( k \) and \( m \) is kept relatively small. When restarting with iterative refined Ritz vectors, relations (141) were too aggressive and rarely satisfied the requirements (138) and (140) 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 norm of Ritz vectors/values to increase from the previous iteration.

**Example 3.4.1** We now revisit Example 3.3.1 and still consider the same matrices \( A = \text{diag}(1:500) \) and \( A = \text{amazon0302} \). We continue to use the same values as specified in Example 3.3.1 namely \( k = 1, m = 2, \) and tolerance \( 10^{-6} \). But now we use Algorithm 6.5 on two hybrid methods, restarting with \( P_m v_1^{(rz)} \) and \( P_m v_1^{(it-n)} \) (iterative refined Ritz on normal system) and \( P_m v_1^{(rz)} \) and \( P_m v_1^{(it-a)} \) (iterative refined Ritz on augmented system). Similar to Example 3.3.1 we ran all methods in both
Algorithm 6.5 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 \( A^T \),
2: \( m \) : maximum size of GKL factorization,
3: \( k \) : number of desired singular triplets,
4: \( p_1 \) : unit vector,
5: \( \text{tol} \) : tolerance for accepting computed approximate singular triple, cf. (100).
6: Output: \( k \) approximate singular triplets \( \{\sigma_j, u_j, v_j\}_{j=1}^k \) of \( A \).
7: Compute \( m \)-GKL factorization with Algorithm 6.2
8: 9: Compute the SVD of \( B_m \) (80) and check \( 1 \leq j \leq k \) (100) with (96);
10: 11: if thick-restarted with harmonic Ritz then
12: 13: Compute the s-PSVD of \( B_{m,m+1} \) (86) where \( k \leq s < m \);
14: 15: Check \( 1 \leq j \leq k \) (100) with (98);
16: 17: Compute \( \{\sigma_j^{(\cdot)}, u_j^{(\cdot)}, v_j^{(\cdot)}\}_{j=1}^k \) by Algorithm 6.4 with \( \mu_j \) (139) for either the augmented system or the normal system;
18: if all \( \sigma_j^{(\cdot)} \) converged and satisfy (138) and (140) then
19: 20: if \( k > 1 \) then
21: 22: Compute \( c_j \) from (121);
23: 24: Compute 1-GKL factorization (119)-(120);
25: else
26: 27: if thick-restarted with Ritz then
28: 29: Compute s-GKL factorization (81)-(82) where \( k \leq s < m \);
30: else
31: 32: Compute s-GKL factorization (94)-(95) where \( k \leq s < m \);
33: Goto 3;
34:
examples, 10 times with a different random starting vector $p_1$ where each method started with a same random vector.

In Figure 4, we collect the results, where the graphs display the number of matrix vector products with $A$ and $A^T$ against the residual norm computed with Ritz value $\sigma_1^{(rz)}$ and vector $P_m v_1^{(rz)}$, i.e., $\text{resAug}_1^{(rz)}$ for all routines. More specifically, for the diagonal system, Figure 4a shows that our hybrid method with iterative refined Ritz on normal system always converged between 210 and 315 matrix–vector products with respect to $\text{resAug}_1^{(rz)}$, compared to Example 3.3.1 where the best result for diagonal matrix is 1100 matrix–vector products. Similarly, for amazon0302 matrix, Figure 4b shows the hybrid method with iterative refined Ritz on normal system always converged between 125 and 205 matrix–vector products with respect to $\text{resAug}_1^{(rz)}$ while comparable computation in Example 3.3.1 required about 700 matrix–vector products. This clearly illustrates that Algorithm 6.5 restarting with $P_m v_1^{(rz)}$ and $P_m v_1^{(it-n)}$ performed significantly better than all restarted methods in Example 3.3.1. Furthermore, we emphasize that in comparison to Example 3.3.1 Algorithm 6.5 avoided stagnation which was one of the motivating factors for its development.

**Remark.** We note that in the context of Example 3.4.1, if iterative refined Ritz vectors were replaced with refined Ritz vectors in Algorithm 6.5, then we saw almost no performance increases over the results in Example 3.3.1 for restarting with refined Ritz vectors. This is attributed in part to the angle criteria (138) 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, namely Algorithm 6.6 which can be viewed as a simple yet powerful variant of Algorithm 6.5. The main difference between these two hybrid schemes is the way they treat the case when multiple singular triplets are desired, i.e., $k > 1$. More specifically, Algo-
A = diag(1 : 500)

Algorithm 6.6 uses a deflation technique that requires the generated basis vectors for the Krylov subspaces to be orthogonal to the converged singular vectors. We do not advocate the inclusion of this technique in Algorithm 6.5 because Algorithm 6.5 is set up as a method that continuously updates the singular vectors regardless of their residual values. This process can be referred to as “soft” locking while the deflation process in Algorithm 6.6 can be referred to as “hard” locking – for more details on “soft” and “hard” locking we direct the reader to [46].

In case when \( k > 1 \), a typical implementation of a deflation technique searches for \( k \) singular triplets and as a singular triplet converges, “locks” the singular vectors, reduces the search space to avoid increasing memory, and orthogonalizes the subsequent generated basis vectors against the converged singular vectors, see [42] for details. Note however that if such an approach was to be implemented in Algorithm 6.5, then it would add “hard” locking to a routine which already effectively utilizes “soft” locking. Therefore, considering the compelling results in Example 3.4.1 for the normal system and a very small basis size, we deviate from a typical imple-
mentation and instead fix $k = 1$ and $m = 2$ and as a singular triplet converges, “lock” the singular vectors and orthogonalize the subsequent generated basis vectors against the converged singular vectors. The “large” memory requirement for such deflation as implemented in Algorithm 6.6 is only five vectors $(p_1, p_2, q_1, q_2, f)$ and the $2(k - 1)$ converged singular vectors. Notice that increasing $m > 2$, and the remarks on accelerating the convergence along with formula (141), would revert our implementation back to including thick–restarted and mixing locking techniques.

There are clear advantages to fixing $k = 1$ and $m = 2$, namely reducing the overall computational cost beyond matrix-vector products, e.g., for a majority of problems one does not need to reorthogonalize the basis vectors, though for very ill-conditioned problems a second orthogonalization is recommended. Moreover, the resulting algorithm is a simple yet powerful routine which is easy to understand and implement ($\approx 100$ lines of MATLAB code).

It should be noted though that if the space spanned by converged vectors is not computed accurately enough, then orthogonalizing against it could slow down the overall convergence. Therefore, a tolerance used to deflate converged vectors should be better than the user input desired tolerance for the first $(k - 1)$ singular triplets. Our numerical experiments and previous experiences, led us to set the tolerance for deflation to be the minimum between $\sqrt{\text{eps}}$ and $10^{-2} \cdot \text{tol}$, where $\text{tol}$ is the user input tolerance. When $k > 1$ and convergence is determined for deflation, we restart the method with the residual vector $f$ from the last iteration, orthogonalized against the converged right singular vector(s) and then proceed to orthogonalize all subsequent computed basis vectors against the converged singular vector(s), see step 4 in Algorithm 6.6.
Algorithm 6.6 Hybrid: Restarted Deflation $(2 \times 2)$ 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,

2: $p_1$ : unit vector,

3: $tol$ : tolerance for accepting computed approximate singular triple, cf. (100).

4: **Output:** $k$ approximate singular triplets $\{\sigma_j, u_j, v_j\}_{j=1}^k$ of $A$.

5: $j := 1$;

6: Compute 2-GKL factorization with Algorithm 6.2 where for $i = 1, 2, \ldots, (j - 1)$

7: $P^T v_i = 0$, $f^T v_i = 0$, and $Q^T u_i = 0$;

8: Compute the largest singular triplet $\{\sigma_{1}\text{(rz)}_1, u_{1}\text{(rz)}_1, v_{1}\text{(rz)}_1\}$ of $B_2$ (80);

9: Compute $\{\sigma_{1}\text{(it-n)}_1, u_{1}\text{(it-n)}_1, v_{1}\text{(it-n)}_1\}$ by Algorithm 6.4 with $\mu_1$ (139);

10: if $j < k$ and (100) is satisfied with $tol = \min\{\sqrt{\varepsilon n}, 10^{-2}tol\}$ using either (108) or (127) with $\mu_1$ (139); then

11: $\{\sigma_j, u_j, v_j\} := \{\sigma_{1}\text{(it-n)}_1, Q_2 u_{1}\text{(it-n)}_1, P_2 v_{1}\text{(it-n)}_1\}$ or $\{\sigma_j, u_j, v_j\} := \{\sigma_{1}\text{(it-n)}_1, Q_2 u_{1}\text{(it-n)}_1, P_2 v_{1}\text{(it-n)}_1\}$;

12: Compute $f = f - (v_j^T f)v_j$;

13: $p_1 := f/\|f\|$, $j := j + 1$;

14: Goto 4;

15: else

16: Check (100) with (108) or (127) and $\mu_1$ (139);

17: if $\sigma_{1}\text{(it-n)}$ converged and satisfies (138) then

18: Compute 1-GKL factorization (119)-(120);

19: else

20: Compute 1-GKL factorization (81)-(82);

21: Goto 4;
3.5 Numerical Examples

In this section, we present MATLAB codes \texttt{trrsvds} and \texttt{rd2svds} which implement Algorithm 6.5 and Algorithm 6.6 respectively. The numerical experiments in this section are specifically created to provide the reader with an insight into the developed methods and illustrate their performance. To that end, we compare our methods to five other routines: two publicly available MATLAB codes \texttt{irlba} [5] and \texttt{svdifp} [33] a publicly available MATLAB interfaced code \texttt{primme} [53] and MATLAB’s built–in functions \texttt{svds} and \texttt{eigs}, where \texttt{eigs} is applied to the matrix 
\[ C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \] 
and the equivalent eigenvalue problem (79).

Table 4 presents the parameters and default values used in \texttt{trrsvds}. Setting parameter \texttt{method} = 'thk' results in \texttt{trrsvds} implementing only thick–restarting with Ritz or harmonic Ritz vectors, i.e., Algorithm 6.3. In essence this is very similar to \texttt{irlba}, though there are some noted differences. Both methods include additional vectors as way of improving convergence, but while our method uses a dynamic scheme (141) \texttt{irlba} has the parameter \texttt{adjust} which is by default set at three. Furthermore, the \texttt{irlba} also increases the number of restart vectors by the number of converged singular triplets. Because of this rigidity of parameter \texttt{adjust} and the fact that in all our examples when searching for the largest singular triplets the Lanczos basis is restricted to be as small as possible, we set \texttt{adjust} to zero instead of its default value three.

For the parameter choices for the other codes, we refer the reader to the references, code documentation, and noted websites for full details and descriptions of parameters. Here, we only provide remarks on other routines’ parameters as needed for clarification. There are numerous selections and variety of combinations of pa-

\footnotesize{
\begin{itemize}
  \item \texttt{Code available at}: \url{http://www.math.uri.edu/~jbaglama}
  \item \texttt{Code available}: \url{http://www.netlib.org/numeralgo/na26.tgz} or \url{http://www.math.uri.edu/~jbaglama}
  \item \texttt{Code available at}: \url{https://github.com/wildstone/SVDIFP}
  \item \texttt{Code available at}: \url{https://github.com/primme/primme}
\end{itemize}
}
### Table 4. The user specific parameters for trrsvds.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roh</td>
<td>Four letter string (‘ritz’ or ‘harm’) specifying the use of either Ritz vectors or harmonic Ritz vectors for thick-restarting. Default value: roh = ’harm’ if sigma = ’SS’ and roh = ’ritz’ if sigma = ’LS’.</td>
</tr>
<tr>
<td>k</td>
<td>Number of desired singular values. Default value: k = 1.</td>
</tr>
<tr>
<td>m</td>
<td>Number of Lanczos vectors, i.e., the size of bidiagonal Lanczos matrix $B_m$. Default value: m = 2 if sigma = ’LS’ and m = 15 if sigma = ’SS’.</td>
</tr>
<tr>
<td>maxitref</td>
<td>Maximum number of iterations used to find iterative refined Ritz singular values, see Algorithm 6.4. Default value: maxitref = 100.</td>
</tr>
<tr>
<td>method</td>
<td>Three letter string (’nor’, ’aug’, or ’thk’) to determine which method to use. Default value: method = ’nor’.</td>
</tr>
<tr>
<td>reorth</td>
<td>Three letter string (’one’ or ’two’) specifying whether to use one-sided (’one’) or two sided (’two’) full re-orthogonalization. Default value: reorth = ’one’.</td>
</tr>
<tr>
<td>sigma</td>
<td>Two letter string (’LS’ or ’SS’) specifying the location of the desired singular values. ’LS’ for the largest singular values and ’SS’ for the smallest singular values. Default value: sigma = ’LS’.</td>
</tr>
<tr>
<td>tol</td>
<td>Tolerance used for convergence. Convergence is determined by equation (100). Default value: tol = $\sqrt{\varepsilon_p}$ (roughly 10$^{-8}$), where $\varepsilon_p$ is machine precision.</td>
</tr>
<tr>
<td>p1</td>
<td>An initial starting vector. If $\ell &gt; n$ and sigma = ’SS’ then $p_1 \in \mathbb{R}^\ell$ else $p_1 \in \mathbb{R}^n$. Default value: $p_1 = \text{randn}(n, 1)$.</td>
</tr>
</tbody>
</table>
parameters for each code. Some choices and combinations may yield faster convergence than others. We cannot provide examples with all possible combinations and only present comparisons to illustrate that the developed methods are competitive. For comparison codes we used either the default values for the parameters or parameter choices that represent the 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 \( p_1 \), tolerance \( tol \), and Lanczos basis maximum size \( m \). Instead of a starting vector, routines \texttt{svdifp} and \texttt{primme\_svds} use an input matrix whose \( i \)th column is the \( i \)th initial approximate right singular vector. Therefore, for those routines we set the first column to be the common starting vector \( p_1 \).

In regards to the other parameters, we set the tolerance for \texttt{svdifp} to be \( tol \cdot \| A \|_2 \). This parameter choice provided the same order of magnitude of the residuals computed by \texttt{svdifp} as well as the other routines, see Table 6 and captions in Figures 7-17. With respect to a common basis size similar to \( m \) in \texttt{ttrsvds}, 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, \texttt{eigs(C)} and \texttt{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 \texttt{ttrsvds} and is therefore represented by \( m \) in Figures 7-17. However, given the complexities and propriety of some of the codes this may not always be the case.

**Remark.** We now briefly discuss the required storage of our hybrid method \texttt{rd2svds} and how it compares to the basis parameter \( m \) in \texttt{ttrsvds} and how it is represented in the figures. Recall that for any \( k \leq m \), \texttt{ttrsvds} with basis size \( m \) requires storage of \( 2m + 1 \) vectors. On the other hand, \texttt{rd2svds} always works with the fixed basis size
of two and thus it only needs to store $5 + 2(k - 1)$ vectors, namely, $p_1, p_2, q_1, q_2, f$ and $2(k - 1)$ converged singular vectors. Therefore, when computing $k$ singular triplets \texttt{rd2svds} is only reported for size $m = k + 1$ in figures, since the number of stored vectors by \texttt{trrsvds} and \texttt{r2svds} is the same.

The program \texttt{svdifp} allows application of a preconditioner and can perform better when one is applied, see [33] for details. However, 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. Consequently, we did not apply a preconditioner. It should be noted, as stated by the authors of [33], “that \texttt{svdifp} without preconditioning is simply the restarted Lanczos method with the LOBPCG-type subspace enhancement.”

The MATLAB interfaced code \texttt{primme_svds} is part of a massive high performance C99 library PRIMME for computing eigenpairs and singular triplets. There are numerous parameter settings, multiple routines/techniques, and also has options to include preconditioning. We cannot possibly compare against all options, nor use preconditioners. Therefore, we only provide a small sample and try to only use default values. We only set the parameters needed for comparison. For all examples, we set the parameters to indicate that the problems are real and to use double precision. Also, the value \texttt{primme.method} is set to be the \texttt{default_min_matvecs}, since this is the measure we are comparing, and finally, the method is set to be \texttt{primme_svds_hybrid}. These choices provided very consistent results throughout all examples.

Since the goal of the paper is to use as little storage as possible and if a routine cannot be used with the default values and the fixed parameter for maximum basis size, we recorded the result as “N/A” - not available for that basis size. We did not try to modify parameters to get the routine to work for that basis size. Likewise, if a routine did not converge we recorded the result as “N/C” and also did not modify the
parameters to get the routine to work (e.g., increase the default setting for maximum number of iterations).

The code \texttt{trrsvds} implements the following reorthogonalization strategies: applies one-sided full reorthogonalization when matrix $A$ is fairly well conditioned and two-sided full reorthogonalization when $A$ is ill-conditioned. The condition number of $A$ is estimated with minimum and maximum singular values of the computed $B_m$ over all iterations thus far. Examples presented in this section with \texttt{trrsvds}, one- 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 \texttt{trrsvds}, reorthogonalization is not used in \texttt{rd2svds} since only one-step of the GKL process is used to build 2-GKL factorization. The routines \texttt{rd2svds} and \texttt{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 the examples when $k = 1$ and $m = 2$).

To illustrate the different methods available for \texttt{trrsvds} via the parameter choices we used the notation \texttt{trrsvds([nor,aug,thk],[ritz,harm])} in all of our examples. The first entry is either \texttt{nor} for the normal equations in the hybrid method to compute the iterative refined Ritz pairs \cite{108}, \texttt{aug} for the augmented equations in the hybrid method to compute the iterative refined Ritz pairs \cite{129} and \texttt{thk} for using only the thick–restarted method (non-hybrid), see Algorithm \ref{alg:6.3}. The second entry is either \texttt{ritz} using Ritz vectors with thick–restarted or \texttt{harm} using harmonic Ritz vectors with thick–restarted. For example, \texttt{trrsvds(nor,ritz)} uses the normal equations in the hybrid method to compute the iterative refined Ritz pairs and Ritz vectors when using thick–restarted.
The numerical examples are separated into four distinct parts. Example 3.5.1 displays the performance of \texttt{trsvds} and \texttt{rd2svds} for computing the largest singular triplets on a diagonal matrix with varying $k$ and $m$ values. The example provides useful illustrations and comparison of methods developed in the paper. Example 3.5.2 is a comparison of the codes on a variety of common test matrices from the SuiteSparse Matrix Collection \cite{10}, see Table 5. The example is focused on computing the largest singular triplets for varying $k$ and $m$ values. Example 3.5.3 is a comparison of the methods for finding largest singular triplet of one of the largest test matrices in the SuiteSparse Matrix Collection. Finally, Example 3.5.4 is focused on computing the smallest singular triplets on two commonly used test matrices from the SuiteSparse Matrix Collection.

Table 5. Test matrices used for the examples from the SuiteSparse Matrix Collection \cite{10}

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size</th>
<th>Non-zeros</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>illc1033</td>
<td>$1033 \times 320$</td>
<td>4719</td>
<td>Least Squares</td>
</tr>
<tr>
<td>lp_ganges</td>
<td>$1309 \times 1706$</td>
<td>6937</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>dwt_1242</td>
<td>$1242 \times 1242$</td>
<td>10426</td>
<td>Structural</td>
</tr>
<tr>
<td>well1850</td>
<td>$1850 \times 712$</td>
<td>8755</td>
<td>Least Squares</td>
</tr>
<tr>
<td>pde2961</td>
<td>$2961 \times 2961$</td>
<td>14585</td>
<td>2D/3D Problem</td>
</tr>
<tr>
<td>Kemelmacher</td>
<td>$28452 \times 9693$</td>
<td>100875</td>
<td>Graphics/Vision</td>
</tr>
<tr>
<td>JP</td>
<td>$87616 \times 67320$</td>
<td>13734559</td>
<td>Tomography</td>
</tr>
<tr>
<td>amazon0302</td>
<td>$262111 \times 262111$</td>
<td>1234877</td>
<td>Directed Graph</td>
</tr>
<tr>
<td>Rucci1</td>
<td>$1977885 \times 109900$</td>
<td>7791168</td>
<td>Least Squares</td>
</tr>
<tr>
<td>relat9</td>
<td>$12360060 \times 549336$</td>
<td>38955420</td>
<td>Combinatorial</td>
</tr>
<tr>
<td>kmerV1r</td>
<td>$214005017 \times 214005017$</td>
<td>465410904</td>
<td>Undir. Graph</td>
</tr>
</tbody>
</table>

In all examples and for all codes except \texttt{svdifp}, the matrix $A$ and $A^T$ are only accessed by call to a function with input $x$ and output $Ax$ or $A^T x$ with an input parameter indicating which product to perform. \texttt{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. To aid in computational times $A^T x$ is performed as
$(x^TA)^T$ to avoid constantly transposing a very large matrix. We also implemented this strategy in svdifp. When the matrix $C$ 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 $A^Tx$.

The recorded cpu times displayed in the examples are in seconds and recorded using MATLAB’s tic-toc command. Since primme_svds is a MATLAB interfaced code, the recorded times are expected to be less than the MATLAB codes. We finally remark 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 are only an illustration of the presented methods and the comparison are only meant to show the methods in this paper are competitive to other existing routines.

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 $\epsilon = 2.2 \cdot 10^{-16}$. For all examples we set the following: a common number of desired singular values, a common random starting vector, comparable parameters to represent the search space, and $tol = 10^{-6}$ ($10^{-10}$ tolerance is used only in Example 3.5.4). Finally, in all of the figures, “N/A” is used to denote that the method is not available for the specified choice of parameters, “N/C” denotes the routine did not converge in the allotted (default) number of iterations, and, unless otherwise stated, the number above the converged methods denotes the total cpu time in seconds obtained from the MATLAB’s tic-toc command.

**Example 3.5.1 (Largest Singular Triplets):**
Let $A = \text{diag}(1:500)$ be a $500 \times 500$ diagonal matrix.

**3.5.1.a.** We are searching for the the largest singular triplets for varying param-
eter choices in the routine \texttt{trrsvds}. We let $k = 1, 2, 3, 4$ and vary $m$ from $(k+1)$ to $(k+4)$. For comparison, we also included the similar hybrid MATLAB eigenvalue code \texttt{trreigs} applied to the equivalent eigenvalue problem for matrix $C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$. The matrix-vector products of $C$ are recorded as the sum of the matrix-vector products of $A$ and $A^T$. The code \texttt{trreigs} applies the hybrid restarted method using either the iterative refined Ritz vectors or thick-restarted with Ritz vectors to compute the extreme eigenpairs. The starting vector for \texttt{trreigs} was the same as \texttt{trrsvds} for the last 500 entries and zero values for the first 500 entries.

In Figure 5 we display our results. The recorded values for \texttt{rd2svds}, the matrix-vector products, and the number of restarts with iterative refined Ritz vectors are the total values necessary to get all of the desired values within the tolerance. Also, recall from Remark 3.5 that \texttt{rd2svds} is only reported for size $m = k + 1$. The values above the bars in Figure 5 correspond to the number of restarts with iterative refined Ritz vectors. As expected, from Figure 5 we observe that for small $m$ and small $k$ values many restarts are needed with iterative refined Ritz vectors. As $m$ and/or the number of desired singular triplets $k$ increase, the routine uses less restarts with iterative refined Ritz vectors. This can contributed in part to the criteria needed to be satisfied becomes more difficult. However, even a few restarts with iterative refined Ritz vectors can still improve overall convergence, particularly when $m$ is small. The method \texttt{trreigs} applied to $C$ did not perform as well for all $m$ values and $k > 1$ in comparison with \texttt{trrsvds}. This can be contributed in part to the larger spread of eigenvalues of $C$. Therefore, we excluded \texttt{trreigs} from the other examples and in turn used \texttt{eigs(C)} for comparisons. We have also excluded \texttt{trrsvds(thk,--)} from the other examples given its similarities with \texttt{irlba}.

In summary, this example is just one illustration that the hybrid method \texttt{trrsvds} is better suited for problems when searching for small number of singular triplets $k$.

\footnote{Code available at: \url{http://www.math.uri.edu/~jbaglama}}
given small $m$.

3.5.1.b. We continue with the same diagonal matrix and compare five methods from the paper, namely \texttt{rd2svds}, \texttt{trrsvds(nor,ritz)}, \texttt{trrsvds(aug,ritz)}, \texttt{trrsvds(nor,harm)}, and \texttt{trrsvds(aug,harm)}, with the methods \texttt{eigs(C)}, \texttt{irlba}, \texttt{svdifp}, \texttt{svds}, and \texttt{primme_svds}. Figure 6, which displays the result of this comparison, shows that all five methods from the paper converged for all $m$ and $k$ values, particularly of note is the case when $m = k + 1$. This example shows that the methods developed in the paper are particularly competitive when using small $m$ relative to the desired number of singular triplets $k$. As $m$ and $k$ values increase we see the routines in this paper remain fairly consistent and competitive among the other publicly available codes. This pattern continues for the rest of the examples.

Example 3.5.2 (Largest Singular Triplets):
For this example, we are using nine test matrices listed in Table 5, namely illc1033, dwt_1242, well1850, pde2961, Kemelmacher, JP, amazon0302, Rucci1, and relat9. We are searching for the the largest singular triplets and let $k = 1, 2, 3, 4$ and vary $m$ from $(k + 1)$ to $(k + 4)$. As in the previous example, we consider ten methods and display the results comparing matrix-vector products and cpu times for all routines in Figures 7-15. The four largest singular values and the order of the maximum residual errors for the routines can be found in captions of the respective figures. We see that all five methods from the paper compared very favorably against other routines for all $m$ and $k$ values. Moreover, the residual norms are all of the same order of magnitude. Given a wide range in sizes of the test matrices, together with the varied proximity among the largest singular values, these examples reinforce our previous observation — methods developed here are particularly competitive when using very small $m$ relative to the desired number of singular triplets.

Example 3.5.3 (Largest Singular Triplet):
Table 6. Example 3.5.3. Computing the largest singular triplet for matrix kmerV1r with \( m = 2, 3 \). The residual norm \( \sqrt{\|Av_1 - \sigma_1 u_1\|^2 + \|A^T u_1 - \sigma_1 v_1\|^2} \) was computed explicitly from the output from each routine.

<table>
<thead>
<tr>
<th>Method</th>
<th>( m )</th>
<th>mvp ( A ) and ( A^T )</th>
<th>mvp cpu</th>
<th>total cpu</th>
<th>residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd2svds</td>
<td>2</td>
<td>72</td>
<td>1770s</td>
<td>2478s</td>
<td>1.5 \cdot 10^{-6}</td>
</tr>
<tr>
<td>trrsvds(nor,ritz)</td>
<td>2</td>
<td>66</td>
<td>1652s</td>
<td>2598s</td>
<td>5.3 \cdot 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>82</td>
<td>1846s</td>
<td>3383s</td>
<td>1.7 \cdot 10^{-6}</td>
</tr>
<tr>
<td>trrsvds(aug,ritz)</td>
<td>2</td>
<td>80</td>
<td>1894s</td>
<td>3066s</td>
<td>6.3 \cdot 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>66</td>
<td>1467s</td>
<td>2645s</td>
<td>5.6 \cdot 10^{-6}</td>
</tr>
<tr>
<td>trrsvds(nor,harm)</td>
<td>2</td>
<td>74</td>
<td>1774s</td>
<td>2842s</td>
<td>5.5 \cdot 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>74</td>
<td>1654s</td>
<td>3105s</td>
<td>6.1 \cdot 10^{-6}</td>
</tr>
<tr>
<td>trrsvds(aug,harm)</td>
<td>2</td>
<td>92</td>
<td>2148s</td>
<td>3723s</td>
<td>6.3 \cdot 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>66</td>
<td>1475s</td>
<td>2977s</td>
<td>3.4 \cdot 10^{-6}</td>
</tr>
<tr>
<td>eigs(C)</td>
<td>N/A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>274</td>
<td>8995s</td>
<td>25712s</td>
<td>8.7 \cdot 10^{-6}</td>
</tr>
<tr>
<td>irlba</td>
<td>2</td>
<td>138</td>
<td>3124s</td>
<td>5473s</td>
<td>6.0 \cdot 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>90</td>
<td>1941s</td>
<td>3608s</td>
<td>3.5 \cdot 10^{-6}</td>
</tr>
<tr>
<td>svdifp</td>
<td>2</td>
<td>81</td>
<td>-</td>
<td>8049s</td>
<td>5.7 \cdot 10^{-6}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>75</td>
<td>-</td>
<td>8457</td>
<td>2.3 \cdot 10^{-6}</td>
</tr>
<tr>
<td>svds</td>
<td>N/A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>206</td>
<td>6407s</td>
<td>15232s</td>
<td>5.1 \cdot 10^{-6}</td>
</tr>
<tr>
<td>primme_svds</td>
<td>N/A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>64</td>
<td>1413s</td>
<td>2438s</td>
<td>3.6 \cdot 10^{-6}</td>
</tr>
</tbody>
</table>

For this example, we are computing the largest singular triplet for the matrix kmerV1r, currently the second largest in order in the SuiteSparse Matrix Collection. 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. This example pushed the bounds of the machine architecture used. Table 6 displays the results for computing the largest singular triplet for \( m = 2, 3 \). The largest singular value was computed by all routines as \( \sigma_1 = 6.5035 \). As seen in Table 6, for \( m = 2 \) our MATLAB codes trrsvds and rd2svds all converged in approximately one hour total time, the fastest converging in about 41 minutes. This is comparable to the MATLAB interfaced code primme_svds with \( m = 3 \) (not available at \( m = 2 \)). MATLAB’s
internal functions \texttt{eigs(C)} and \texttt{svds} were only available for \( m = 3 \) and needed more than 7 hours and 4 hours, respectively.

**Example 3.5.4 (Smallest Singular Triplets):**

Although the focus of the developed methods is not on computing the smallest singular triplets, we still include two examples. We consider two commonly used test matrices in the literature, \texttt{lp.ganges} and \texttt{well1850}. For this example, we let \( k = 1, 2, 3, 4 \) and vary the maximum basis size from 12 to 15 – it is well-known that when computing the smallest singular triplets a larger basis size is required, see e.g., \cite{5, 25}.

We start with some remarks. Our routine \texttt{rd2svds} could not be used, since it requires the basis to be fixed at 2. The routine \texttt{eigs(C)} could not be used with only matrix-vector products since the smallest singular triplets are interior values. \texttt{eigs} also requires a linear solver routine, i.e., \((A - \sigma I)x\). The routine \texttt{svds} also could not be used with only matrix-vector products. Moreover, \texttt{svds} requires the QR decomposition of the input matrix \( A \) and then searches for the largest singular triplets using linear solve routines \( R\backslash x \) and \( RT\backslash x \). Therefore, in Figures 16-17, N/A is used to report that these three methods are not available.

For the remaining seven methods, we set \( tol = 10^{-10} \) and display the results comparing matrix-vector products and cpu times in Figures 16-17. The four smallest singular values along with the maximum residual errors can be found in the captions of Figures 16-17. As we can see, all of the four available methods from this paper compared favorably against the other routines for all \( m \) and \( k \) values. It should be noted again that the routines \texttt{svdifp} and \texttt{primme Svds} allow use of preconditioners which have been shown to significantly improve their performance in comparison when no preconditioner is used, see \cite{33, 53}. Therefore, our presented results are not a reflection on the performance of those codes as developed. However, as previously stated, preconditioners (and factorization - QR) increase storage requirements, and
therefore in this context are not used.

In summary, even though our developed methods focused on the computation of the largest singular triplets, they proved to be competitive in case when smallest singular triplets are desired.

3.6 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 method combines thick-restarting with Ritz or harmonic Ritz vectors or with a judiciously chosen linear combination of iterative refined Ritz vectors. Numerical examples show the method to be highly competitive with other publicly available codes, particularly when there are limited memory requirements.
Figure 5. Example 3.5.1.a. (largest singular triplets) $A = \text{diag}(1:500)$ varying values of $k$ and basis size for MATLAB codes \texttt{rd2svds}, \texttt{trsvds} and \texttt{trreigs}(C). The value above the bars are the number of restarts with iterative refined Ritz vectors. For small $m$ and small $k$ values more restarts are required with iterative refined Ritz vectors.
Figure 6. Example 3.5.1.b (largest singular triplets) $A = \text{diag}(1:500)$ varying values of $k$ and $m$ for different MATLAB codes.
Figure 7. Example 3.5.2 (largest singular triplets) for the $1033 \times 320$ matrix $\text{ilc1033}$ for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-6}$. The four largest singular values are: $\sigma_1 = 2.1444$, $\sigma_2 = 2.1042$, $\sigma_3 = 2.0885$, and $\sigma_4 = 2.0574$. 
Figure 8. Example 3.5.2 (largest singular triplets) for the $1242 \times 1242$ matrix $dwt_{1242}$ for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-6}$, except for $\text{eigs(C)}$ it is $\approx 1.3 \cdot 10^{-5}$. The four largest singular values are: $\sigma_1 = 9.3912$, $\sigma_2 = 9.2379$, $\sigma_3 = 9.1552$, and $\sigma_4 = 9.0722$. 

97
Figure 9. Example 3.5.2 (largest singular triplets) for the 1850 × 712 matrix well1850 for varying values of $k$ and maximum (basis) size for different routines. For all methods, the maximum residual errors are on the order of $10^{-6}$. The four largest singular values are: $\sigma_1 = 1.7943$, $\sigma_2 = 1.7388$, $\sigma_3 = 1.7189$, and $\sigma_4 = 1.6828$. 
Figure 10. Example 3.5.2 (largest singular triplets) for the 2961×2961 matrix pde2961 for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-5}$, except for $rd2svds$ and $svds$ they were $\approx 9.8 \cdot 10^{-6}$. The four largest singular values are: $\sigma_1 = 10.378$, $\sigma_2 = 10.096$, $\sigma_3 = 9.8779$, and $\sigma_4 = 9.7856$. 

99
Figure 11. Example 3.5.2 (largest singular triplets) for the 28452 × 9693 matrix Kemelmacher for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-4}$. The four largest singular values are: $\sigma_1 = 240.58$, $\sigma_2 = 238.02$, $\sigma_3 = 236.17$, and $\sigma_4 = 235.35$. 
Figure 12. Example 3.5.2 (largest singular triplets) for the $87616 \times 67320$ matrix $JP$ for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-3}$. The four largest singular values are: $\sigma_1 = 4223.1$, $\sigma_2 = 4019.3$, $\sigma_3 = 3872.8$, and $\sigma_4 = 3819.2$. 
Figure 13. Example 3.5.2 (largest singular triplets) for the 262111 × 262111 matrix amazon0302 for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-5}$. The four largest singular values are: $\sigma_1 = 21.218$, $\sigma_2 = 21.136$, $\sigma_3 = 20.027$, and $\sigma_4 = 19.277$. 
Figure 14. Example 3.5.2 (largest singular triplets) for the $1977885 \times 109900$ matrix Rucci1 for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-6}$. The four largest singular values are: $\sigma_1 = 7.0687$, $\sigma_2 = 6.9853$, $\sigma_3 = 6.9625$, and $\sigma_4 = 6.8895$. 
Figure 15. Example 3.5.2 (largest singular triplets) for the $12360060 \times 549336$ matrix relat9 for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-5}$. The four largest singular values are: $\sigma_1 = 21.626$, $\sigma_2 = 20.417$, $\sigma_3 = 18.666$, and $\sigma_4 = 18.61$. 

104
Figure 16. Example 3.5.4 (smallest singular values) for the $1850 \times 712$ matrix well1850 for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-10}$. The four smallest singular values are: $\sigma_1 = 0.01612$, $\sigma_2 = 0.019113$, $\sigma_3 = 0.02316$, and $\sigma_4 = 0.030219$. 

<table>
<thead>
<tr>
<th>$k$</th>
<th>$m$</th>
<th>$\sigma_1$</th>
<th>$\sigma_2$</th>
<th>$\sigma_3$</th>
<th>$\sigma_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>0.01612</td>
<td>0.019113</td>
<td>0.02316</td>
<td>0.030219</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>0.01612</td>
<td>0.019113</td>
<td>0.02316</td>
<td>0.030219</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>0.01612</td>
<td>0.019113</td>
<td>0.02316</td>
<td>0.030219</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>0.01612</td>
<td>0.019113</td>
<td>0.02316</td>
<td>0.030219</td>
</tr>
</tbody>
</table>
Figure 17. Example 3.5.4 (smallest singular values) for the $1309 \times 1706$ matrix $\text{lp}_{\text{ganges}}$ for varying values of $k$ and $m$ for different routines. For all methods, the maximum residual errors are on the order of $10^{-10}$. The four smallest singular values are: $\sigma_1 = 1.8708 \cdot 10^{-4}$, $\sigma_2 = 0.10645$, $\sigma_3 = 0.16297$, and $\sigma_4 = 0.2079$. 
List of References


CHAPTER 4
Conclusions

Results of this thesis have increased the knowledge in the field of numerical linear algebra and led to promising new methods for solving eigenvalue and singular value problems. These new methods have allowed for greater speeds in calculation of the extreme eigenpairs and singular triplets while using very little storage space, and can be applied to pattern recognition, genomics, data visualization, signal classification, and a broad range of other real world problems.

The Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix from Chapter 2 presented our restarted hybrid method in which we combine thick-restarting with restarting with refined Ritz vectors in a linear combination but using our new iterative process to speed up convergence. We proved that the sequence of iterative refined Ritz vectors will converge and established the ability to implement a linear combination of these iterative refined vectors to improve the process. Our new method was seen to be competitive through numerical examples with other available codes while using limited memory.

Chapter 3, the Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods takes the hybrid concept in Chapter 2 and extends it from the symmetric eigenvalue problem to the singular value problem. The new hybrid GKL methods use thick-restarting with Ritz vectors or harmonic Ritz vectors combined with a linear combination of iterative refined Ritz vectors, computed either on the normal or augmented system. As these methods do not require factorization of $A$, they can be applied to not only the seismic sensor work but many other large data problems. When compared with existing MATLAB codes and other publicly available codes, our new GKL hybrid methods were seen to be very competitive, especially when using
very small basis sizes, i.e., limited memory.

Finally, the next steps will be to take the hybrid GKL code and apply them to multiple seismic sensor data sets. There may be a need for method adjustments as well as evaluations of which signals to begin the investigations. Once feature spaces can be expressed using their singular triplets they can be plotted using PCA. This will allow us to test whether the sensor data will cluster like that of the genome data in other studies and apply the desired confidence bounds. We expect the hybrid GKL codes to be fast and compact enough to implement in our sensor processing scheme in real time.
CHAPTER A

Appendix: MATLAB Code

A.1 MATLAB function trreigs(varargin)

function varargout = trreigs(varargin)

% TRREIGS: Computes the K extreme eigenvalue and associated eigenvector
% of a N x N symmetric matrix A.

% PROGRAM INFORMATION:
% -------------------
% ...
% ... = TRREIGS(A)
% ... = TRREIGS('AFUN',N)
% ... = TRREIGS(@(x)A(x),N)

% The first input value into TRREIGS can be a numeric matrix or a
% function/function handle that returns Y = A*X. If the input matrix
% is a function/function handle the second input value must be
% the size of the matrix A. The the input/output structure of M-file
% Y = AFUN(X,N).

% OUTPUT OPTIONS:
% -------------
% I.) TRREIGS(A)
% If convergence, displays the K desired eigenvalues.
% II.) D = TRREIGS(A)
% If convergence, returns K eigenvalues in the vector D.
% III.) [V,D] = TRREIGS(A)
% If convergence, returns a diagonal matrix D that contains the
% K eigenvalues along the diagonal and the matrix V contains the
% corresponding orthogonal eigenvectors, such that A*V = V*D.
% If TRREIGS reaches the maximum number of iterations before convergence
% then V = [ ] and D = [ ]. Use output option IV.).
% IV.) [V,D,FLAG] = TRREIGS(A)
This option returns the same structural output as (III) plus a two dimensional array FLAG that reports if the algorithm converges and the number of matrix-vector products. If FLAG(1) = 0 then this implies normal return: all eigenvalues have converged. If FLAG(1) = 1 then the maximum number of iterations have been reached before all desired eigenvalues have converged. FLAG(2) contains the number of matrix-vector products used by the code. If the maximum number of iterations are reached (FLAG(1) = 1), then the matrices V and D contain the last available approximations for the eigenpairs.

% INPUT OPTIONS:
% --------------
...
... = TRREIGS(A,OPTS) or TRREIGS('AFUN',N,OPTS) or TRREIGS(@(x)A(x),N,OPTS)
OPTS is a structure containing input parameters. The input parameters can be given in any order and can greatly influence convergence rates. The structure OPTS may contain some or all of the following input parameters. If parameter OPTS is missing or an input parameter in OPTS is not set, default value(s) are used. The string for the input parameters can contain upper or lower case characters.

% INPUT PARAMETER DESCRIPTION

% OPTS.COEFF When K > 1, COEFF is used to determine which matrix is use to compute the coefficients for the linear combination of iterative refined vectors. See Section 6 in the reference.

coeff = 1 -> matrix (6.5)
coeff = 2 -> Add one more term - matrix (6.9) (Default)
coeff = 3 -> Add all terms - matrix (6.10)
DEFAULT VALUE COEFF = 2

% OPTS.DELTA1 Used to determine when thick restarting with Ritz vectors can start switching to restarting with iterative refined vectors. The value should be related to the overall convergence tolerance. Therefore, we restrict the input to be in decimal form, 0 <= DELTA1 <= 1 and check for switching if the maximum residual norm of the desired Ritz pairs is <= TOL^DELTA1.
EXAMPLE: If TOL = 1d-8 and user would like to start switching when convergence has reached 25% of TOL or 1d-2, user inputs .25. The routine computes
TOL^(.25).
Too large and only thick-restarted will be done. Too small and stagnation
results from using a poor Krylov space.
|| A*V_RITZ - V_RITZ*D_RITZ ||_2 <= TOL^(DELTA1)*||A||_2.
|||A|||_2 is approximated by largest absolute Ritz value.
DEFAULT VALUE  DELTA1 = 0.1

OPTS.DELTA2  Used to determine when the Iterative refined vectors
are "close" enough to the Ritz vectors and can be
considered for restarting. Too large and only thick-restarted will be done.
Too small and stagnation may result from using iterative refined vectors
closer to the non-corresponding Ritz vector. See Example 5.2 in reference.
(al 1 <= j <= K) |(Refine vectors_j)^T*(Ritz vectors_j)| > DELTA2
Must be 0< DELTA2 < 1
DEFAULT VALUE  DELTA2 = 0.9

OPTS.FLT  Toggle to restrict using restarting with iterative refined Ritz vectors when
the current computed iterative refined Ritz values are better than the past
iterations best Ritz approximation. This is only used when K=1 and
should only be used when M is small. FLT = 1 uses
the restriction, FLT= 0 do not use restriction.
DEFAULT VALUE FLT = 1 if M > 2 and FLT = 0 when M = 2.

OPTS.K  Number of desired eigenvalues.
DEFAULT VALUE  K = 1

OPTS.M  Number of Lanczos vectors, i.e. size tridiagonal
matrix. Currently, full reorthogonalization is
used. Large M will increase non-matrix-vector product
CPU times.
DEFAULT VALUE  M = 2

OPTS.MAXIT  Maximum number of iterations, i.e. maximum number of Lanczos restarts.
DEFAULT VALUE  MAXIT = 1000

OPTS.MAXITREF  Maximum number of iterations used to find iterative
refined Ritz values, see Section 5 Algorithm 5.1 in
reference. Recommend 10% of MAXIT. Stagnation prevention
is in place. Require MAXITREF > =100.
DEFAULT VALUE  MAXITREF = 100
Two letter string specifying the location of the desired eigenvalues.

'LM' or 'LA' Largest Magnitude or Algebraic

'SA' Smallest Algebraic

DEFAULT VALUE SIGMA = 'LM'

Tolerance used for convergence. Convergence is determined when

\[ || A*V - V*D ||_2 \leq TOL*||A||_2 \]

\[ ||A||_2 \] is approximated by largest absolute Ritz value. V and D are either iterative refined or Ritz. If iterative refined V matrix is made orthogonal.

DEFAULT VALUE TOL = SQRT(EPS) (roughly 1d-8)

A matrix of starting vectors.

DEFAULT VALUE VO = randn

DATE MODIFIED: 12/03/2020

VER: 1.0

AUTHORS: J. Baglama, T, Bella, and J. Picucci

REFERENCE:


Too many output arguments requested.

if (nargout >= 4), error('ERROR: Too many output arguments.'); end

This MATLAB code is provided to illustrate Algorithm 6.1 and is NOT optimized for performance or set up for commercial use.

Any use beyond illustrate purposes (e.g. comparison for publications) requires consent of the authors.

*************************************************************************
* This MATLAB code is provided to illustrate Algorithm 6.1 and is NOT *
* optimized for performance or set up for commercial use. *
* Any use beyond illustrate purposes (e.g. comparison for publications) *
* requires consent of the authors. *
*************************************************************************
if nargin == 0, help trreigs, return, end

% Get matrix A. Check type (numeric or character) and dimensions.
if ischar(varargin{1}) || isa(varargin{1}, 'function_handle')
    sst_data = 3;
    if nargin == 1, error('ERROR: Need N (size of matrix A).'); end
    n = varargin{2};
    if ~isnumeric(n) || length(n) > 2
        error('ERROR: Second argument N must be a numeric value.');
    end
else
    sst_data = 2;
    n = size(varargin{1},1);
    if any(size(varargin{1}) ~= n), error('ERROR: Matrix A is not square.'); end
    if ~isnumeric(varargin{1}), error('ERROR: A must be a numeric matrix.'); end
end

% Square root of machine tolerance used in convergence testing.
sqteps = sqrt(eps);

% Set all input options to default values.
k=1; hybrid = 0; maxit = 1000; m=2;
maxitref=100; projt = 0; sigma = 'LM'; tol = sqteps;
delta2 = 0.9; deltai=0.1; coeff = 2; flt = 0; input_flt=[];

% Preallocate memory for large matrices.
v = spalloc(n,m,n*m); f = spalloc(n,1,n);

% Get input options from the data structure.
if nargin > 1 + ischar(varargin{1})
    options = varargin(sst_data:nargin);
    names = fieldnames(options);
    I = strmatch('COEFF',upper(names),'exact');
    if ~isempty(I), coeff = getfield(options,names{I}); end
    I = strmatch('DELTA1',upper(names),'exact');
    if ~isempty(I), deltai = getfield(options,names{I}); end
    I = strmatch('DELTA2',upper(names),'exact');
    if ~isempty(I), delta2 = getfield(options,names{I}); end
    I = strmatch('FLT',upper(names),'exact');
    if ~isempty(I), input_flt = getfield(options,names{I}); end
end
I = strmatch('K',upper(names),'exact');
if ~isempty(I), k = getfield(options,names{I}); end
I = strmatch('M',upper(names),'exact');
if ~isempty(I), m = getfield(options,names{I}); end
I = strmatch('MAXIT',upper(names),'exact');
if ~isempty(I), maxit = getfield(options,names{I}); end
I = strmatch('MAXITREF',upper(names),'exact');
if ~isempty(I), maxitref = getfield(options,names{I}); end
I = strmatch('SIGMA',upper(names),'exact');
if ~isempty(I), sigma = upper(getfield(options,names{I})); end
I = strmatch('TOL',upper(names),'exact');
if ~isempty(I), tol = getfield(options,names{I}); end
I = strmatch('V0',upper(names),'exact');
if ~isempty(I), v = getfield(options,names{I}); end
end

%***************************************************
% Check for some input errors in the data structure.
% ***** This is not an exhaustive check list. *****
%***************************************************

% Check that input values are numerical values.
if (~isnumeric(coeff) || ~isnumeric(delta1) || ~isnumeric(delta2) || ...
  ~isnumeric(k) || ~isnumeric(m) || ~isnumeric(maxit) || ...
  ~isnumeric(maxitref) || ~isnumeric(tol))
  error('ERROR: Incorrect type for input value(s) in the structure.');
end

% Check value of COEFF
if coeff ~=1 && coeff ~=2 && coeff ~=3 , error('ERROR: COEFF must 1, 2, or 3'); end

% Check value of DELTA1
if delta1<0 || delta1 > 1, error('ERROR: 0<= DELTA1 <= 1'); end

% Check value of DELTA2
if delta2<=0 || delta2 >= 1, error('ERROR: 0< DELTA2 < 1'); end

% Check value of FLT.
if isempty(input_flt) % User input.
  if input_flt ~=0 && input_flt ~=1, error('ERROR: FLT must 0 or 1'); end
flt = input_flt; % set to user input.

end

% Check value of K.
if k <= 0, error('ERROR: K must be a positive value.'); end
if k >= m, error('ERROR: K is too large compared to M.'); end

% Check value of M.
if m <= 1, error('ERROR: M must be greater than 1.'); end
if m > n, m = n; end

% Check FLT again.
if k > 1, flt = 1; end % override input, cannot be used when k > 1.
if isempty(input_flt) % no user input, check if m was changed
  if m > 2, flt = 1; end % set to default
end

% Check value of MAXIT
if maxit <= 0, error('ERROR: MAXIT must be a positive value.'); end

% Check value of MAXITREF
if maxitref <= 0, error('ERROR: MAXITREF must be a positive value.'); end
if maxitref > maxit, error('ERROR: MAXITREF should be less than MAXIT'); end
if maxitref < 100, error('ERROR: MAXITREF >= 100'); end

% Check value of SIGMA.
if isnumeric(sigma), error('ERROR: SIGMA must be SA, LM or LA.'); end
if length(sigma) ~= 2, error('ERROR: SIGMA must be SA, LM or LA.'); end
if strcmp(sigma,'SM') error('SM currently not supported. Recommend user use A\X and
 LM - output is 1/lambda');
end

% Check value of TOL.
% Set tolerance to machine precision if tol < eps.
if tol < eps, tol = eps; warning('Changing TOL to EPS'); end

% If starting vector V0 is not given then set starting vector V0 to be a
% (n x 1) matrix of normally distributed random numbers.
if nnz(v) == 0, v = randn(n,1); end

% Check starting vector V0.
if ~isnumeric(v), error('ERROR: Incorrect starting matrix V0.'); end
if ((size(v,1) ~= n) || (size(v,2) ~= 1))
    error('ERROR: Incorrect size of starting matrix V0.);
end

%--------------------------%
% END: PARSE INPUT VALUES. %
%--------------------------%

%-----------------------------------------------------------%
% BEGIN: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
%-----------------------------------------------------------%
% <- DO NOT MODIFY ->
dmax=zeros(m,1); % Initialize array for max. abs. value of all Ritz values
if strcmp(sigma,'SA'), dmax = Inf(m,1); end
if strcmp(sigma,'LA'), dmax = -Inf(m,1); end
f = zeros(n,1); % Initialize residual vector f.
iter = 1; % Main loop iteration count.
r_rf_0=zeros(m,1); % Used for iterative refined comparison from last iteration.
rztol=tol^(delta1); % Tolerance for checking on using iterative refined
Smax = 0; % Holds the maximum absolute value of all computed Ritz values.
T = zeros(m,m); % Initialize T matrix.
thick=0; % Indicate if thick restarted is to be used.
thk = 2; % Indicate starting point for Lanczos.

%---------------------------------------------------------%
% END: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
%---------------------------------------------------------%
%----------------------------------------------%
% BEGIN: INITIALIZATION - ONE STEP OF LANCZOS. %
%----------------------------------------------%
v(:,1) = v(:,1)/norm(v(:,1));
v(:,2) = matrixprod(varargin{1},v(:,1),n); mprod=1;
T(1,1) = v(:,2)'*v(:,1);
v(:,2) = v(:,2) - v(:,1)*T(1,1);
dotv = v(:,2)'*v(:,1); % Reorth step
v(:,2) = v(:,2) - v(:,1)*dotv;
T(1,1) = T(1,1) + dotv;
T(2,1) = norm(v(:,2)); T(1,2) = T(2,1);
if abs(T(2,1)) < eps*abs(T(1,1))
    error('ERROR: VO caused |T(2,1)| < eps|T(1,1)|-> Change VO. ');
end
v(:,2) = v(:,2)/T(2,1);

%--------------------------------------------%
% END: INITIALIZATION - ONE STEP OF LANCZOS. %
%--------------------------------------------%

%---------------------------%
% BEGIN: ITERATION PROCESS. %
%---------------------------%
while (iter <= maxit)

%-------------------------%
% BEGIN: LANCZOS PROCESS. %
%-------------------------%
for j=thk:m
    f = matrixprod(varargin{1},v(:,j),n); mprod=mprod+1;
    if thick == 1
        f = f - v(:,1:j-1)*T(j,1:j-1)';
    thick = 0;
    else
        f = f - v(:,j-1)*T(j,j-1);    
    end
    T(j,j) = f'*v(:,j);
    f = f - (v(:,j)*T(j,j));
    dotv = (f'*v(:,1:j))'; % Reorth step
    f = f - (v(:,1:j)*dotv);
    if norm(dotv)>sqrteps % 2nd reorth step if needed
        dotv2 = (f'*v(:,1:j))';
        dotv=dotv+dotv2;
        f = f - (v(:,1:j)*dotv2);
    end
for i=1:j, T(j,j) = T(j,j) + dotv(i); end
fnorm = norm(T);
if fnorm < max(Smax*tol,eps) && j>= k && iter > 1
    T = T(1:j,1:j); m=j; % resize T for exit;
    warning('Early termination in Lanczos - results may not be trusted');
    break;
end
if fnorm < max(Smax*eps,eps) && (j < k || iter == 1)
    error('ERROR: Lanczos breakdown. ');
end
if j < m
    v(:,j+1) = f/fnorm;
    T(j+1,j) = fnorm;
    T(j,j+1) = T(j+1,j);
end
end

%----------------------------------------------------------
% END: LANCZOS PROCESS. %
%----------------------------------------------------------

%------------------------------------------------------------------------
% BEGIN: COMPUTE/SORT EIGENVALS AND EIGENVECS OF T AND TEST CONVERGENCE. %
%------------------------------------------------------------------------
[x_rz, d_rz] = eig(T); d_rz = diag(d_rz);
Smax = max([Smax;abs(d_rz)]); % Use to app. ||A||_2
if strcmp(sigma,'LM')
    [~,I_eig] = sort(abs(d_rz));
    I_eig = I_eig(length(I_eig):-1:1);
    x_rz = x_rz(:,I_eig); d_rz = d_rz(I_eig);
    for j=1:k
        dmax(j) = max(dmax(j),abs(d_rz(j)));
        dmax(j) = sign(d_rz(j))*dmax(j);
    end
else if strcmp(sigma,'LA')
    [~,I_eig] = sort(d_rz);
    I_eig = I_eig(length(I_eig):-1:1);
    x_rz = x_rz(:,I_eig); d_rz = d_rz(I_eig);
    for j=1:k
        dmax(j) = max(dmax(j),d_rz(j));
    end
end

122
elseif strcmp(sigma,'SA')
    [~,I_eig] = sort(d_rz);
    x_rz = x_rz(:,I_eig); d_rz = d_rz(I_eig);
    for j=1:k
        dmax(j) = min(dmax(j),d_rz(j));
    end
    end

% Convergence check.
nc = 0; % number of convergence desired Ritz values.
res_r = ones(k,1); % reset residual norms.
for j=1:k
    res_r(j) = abs(x_rz(m,j))*fnorm;
    if res_r(j) < tol*Smax, nc = nc+1; end
end
if nc ==k || iter == maxit
    v=v*x_rz(:,1:k); d = diag(d_rz(1:k));
    break;
end

% Compute size matrix T.
Tsz = size(T,1);

%--------------------------------------------------------------------%  
% BEGIN: COMPUTE ITERATIVE REFINED AND DETERMINE RESTARTING VECTORS. %
%--------------------------------------------------------------------%  
% Compute Iterative refined Ritz values.
rconv=0;
if max(res_r(1)) < rztol*Smax
    [r_rf,x_rf,s_rf,u_rf,rconv] = refined_ritz(dmax(1:k),fnorm,T,maxitref,Tsz,k);
end
% rconv returns 0 or k. 0 indicates not all converged. rconv = k indicates all k converged
% Compute inner product between the k converged iterative refined Ritz vector(s)
% and Ritz vector(s). Also, check if iterative Ritz values are less than
% the previous Ritz values. If so, do not use.
ang_rz_rf = 0; decr = 0;
for i=1:rconv
  ang_rz_rf(i) = abs(x_rz(:,i)'*x_rf(:,i)); % computer the inner prod. to check angle
  if flt % if true - requires iter. refined to be better approx. prev. eig. of T.
    if strcmp(sigma,'LM') && abs(r_rf(i)) < abs(r_rf_0(i)), decr = 1; end
    if strcmp(sigma,'LA') && r_rf(i) < r_rf_0(i), decr = 1; end
    if strcmp(sigma,'SA') && r_rf(i) > r_rf_0(i), decr = 1; end
  end
  r_rf_0(i) = dmax(i);
end

% Find all inner products between iterative refined and Ritz > delta2
I = find(ang_rz_rf > delta2);

% Convergence check of iterative refined Ritz vector(s)
if length(I) == k
  norm_res=[]; % reset residual norms.
  [q_rf,~] = qr(x_rf,0); % QR to ensure orthogonal vectors
  nr = 0; % number of iter. ref. converged.
  for j=1:k
    Tq = T*q_rf(:,j) - r_rf(j)*q_rf(:,j);
    norm_res(j) = sqrt(Tq'*Tq + fnorm^2*q_rf(m,j)^2);
    if norm_res(j) < tol*Smax, nr = nr+1; end
  end
  if nr ==k
    v=v*q_rf(:,1:k); d = diag(r_rf(1:k));
    break;
  end
end

% Determine if restart with linear combination of Iterative refined Ritz
% or thick-restarted with Ritz vectors
if (length(I) == k && max(res_r(1)) < rztol*Smax && decr == 0)
  %------------------------------------------%
  % BEGIN: RESTART ITERATIVE REFINED SECTION %
  %------------------------------------------%
  % Re-compute residual vector of iterative refined Ritz. Done without QR and is
% different than convergence check since x_rf are not orthogonal.
% Needed for restart. res_rf is m x k "small" vector.
res_rf=[]; norm_res = []; % reset residual norms.
for j=1:k
    res_rf(:,j) = s_rf(j)*(u_rf(1:m,j) - x_rf(:,j)*(x_rf(:,j)'*u_rf(1:m,j)));
    norm_res(j) = sqrt(res_rf(:,j)'*res_rf(:,j) + s_rf(j)^2*u_rf(m+1,j)^2);
end

% Initialize the k coefficients c_rf for the linear combination of
% the k iterative refined Ritz vectors x_rf.
c_rf = ones(length(r_rf),1);
if k > 1
    Bc = zeros(k-1,k); Tem = T(m,1:m); em = zeros(m,1); em(m) = 1;

    % coeff matrix - (6.5)
    if coeff == 1
        Dc = diag(x_rf(m,1:k));
        for i=1:k-1
            for j=1:k
                Bc(i,j) = r_rf(j)^(i-1);
            end
        end
        Bc = Bc*Dc;
    end

    % Added one more term coeff matrix - (6.9)
    if coeff == 2
        Bc(1,1:k) = x_rf(m,1:k);
        for i=2:k-1
            for j=1:k
                Bc(i,j) = Tem*x_rf(:,j)*r_rf(j)^(i-2);
            end
        end
    end

    % Use all terms coeff matrix - (6.10)
    if coeff == 3
        Bc(1,1:k) = x_rf(m,1:k); % if k=2 => 1 x 2 matrix end
        if k >=3 % if k=3 => 2 x 3 matrix end
            Bc(2,1:k) = Tem*x_rf(1:m,1:k);
        end
    end
if k >= 4
    for i=3:k-1
        for j=1:k
            Bc(i,j) = Tem*x_rf(:,j)*r_rf(j)^(i-2);
            r_sum = 0;
            for jj = 3:i
                r_sum = r_sum + r_rf(j)^(i-jj)*em'*T^(jj-2)*u_rf(1:m,j);
            end
            Bc(i,j) = Bc(i,j) + s_rf(j)*r_sum;
        end
    end
end

% compute the solution of the k-1 x k homogeneous system using
% null. Following code is needed to avoid zero column(s) in Bc
% for numerically converged eigenvectors.
Iin = []; Iout=[]; % set variables for which columns are used.
Bc_max = max(abs(Bc),[],'all');
    for i=1:k % search for columns numerically zero.
        if norm(Bc(:,i),Inf) < sqrt(eps)*Bc_max
            Iout = [Iout i]; % references which columns to remove.
        else
            Iin = [Iin i];
        end
    end
if isempty(Iout) % remove numerically zero columns from Bc
    Bc = Bc(1:k-length(Iout)-1,Iin);
    c_rf(1:k) = zeros(k,1);
    c_rf_in = null(Bc); % call Matlab's null to solve system
    c_rf(Iin) = c_rf_in(:,1);
    c_rf(Iout) = norm_res(Iout); % Place zero entries back in sol.
else
    c_rf = null(Bc); % call Matlab's null to solve full system
end

%----------------------------------------------------------%
% BEGIN: ONE STEP OF LANCZOS WITHOUT MATRIX VECTOR PRODUCT %
\begin{verbatim}
% (See section 6 equations (6.2) & (6.12) in reference [1]) %
%-------------------------------------------------------------%
f_rf = 0;
% Used for computing the coefficients the residual vectors
for j=1:k
    \texttt{f}_\texttt{rf} = \texttt{f}_\texttt{rf} + \texttt{c}_\texttt{rf}(j)\times\texttt{x}_\texttt{rf}(m,j);
end
% Compute left and right vectors to avoid matrix-vector product
% \texttt{v}_\texttt{left} & \texttt{v}_\texttt{right} are \texttt{m x 1} "small" vectors.
v_left = x_rf*c_rf; v_right = x_rf*(r_rf.*c_rf)+res_rf*c_rf;
v_left_norm = norm(v_left);
v_left = v_left/v_left_norm;
v_right = v_right/v_left_norm;
% Compute the matrix \texttt{T}
T = zeros(2,2);
% Update Lanczos vectors \texttt{v(:,1)} \& \texttt{v(:,2)}. \texttt{f} is reused and avoids extra
% storage needs.
f = v*v_right + (f_rf/v_left_norm)*f;
v(:,1) = v*v_left;
T(1,1) = f'*v(:,1);
f = f - (v(:,1)*T(1,1));
dotv = (f'*v(:,1))'; % Reorth step
f = f - (v(:,1)*dotv);
T(1,1) = T(1,1) + dotv;
fnorm = norm(f);
v(:,2) = f/\texttt{fnorm};
T(2,1) = fnorm;
T(1,2) = T(2,1);
\texttt{thk}=2; \texttt{thick}=0; % set values to start Lanczos at step 2
%-------------------------------------------------------------%
% END: ONE STEP OF LANCZOS WITHOUT MATRIX VECTOR PRODUCT %
%-------------------------------------------------------------%
% END: RESTART ITERATIVE REFINED SECTION %
%-------------------------------------------------------------%
else
%-------------------------------------------------------------%
% BEGIN: THICK-RESTART RITZ SECTION %
%-------------------------------------------------------------%
ext
\end{verbatim}
Simple strategy to improve convergence.

\[
    k_{thk} = \max(\text{floor}(nc + (Tsz-nc)/2), k);
\]

if \( Tsz - 1 - k_{thk} < 0 \), \( k_{thk} = Tsz-1 \); end

Set up matrices and vectors for thick-restarted

\[
v = [v*x_rz(:,1:k_{thk}) f/fnorm];
\]

\[
T = \text{zeros}(k_{thk}+1,k_{thk}+1);
\]

\[
T(1:k_{thk}+1,1:k_{thk}) = [\text{diag}(d_rz(1:k_{thk})); x_rz(m,1:k_{thk})*fnorm];
\]

\[
T = \text{tril}(T,-1) + \text{tril}(T)'; \text{thk} = k_{thk}+1; \text{thick}=1;
\]

% END: THICK-RESTART RITZ SECTION

% Update the main iteration loop count.

\[
\text{iter} = \text{iter}+1;
\]

end % end main loop

% END: ITERATION PROCESS.

% BEGIN: OUTPUT RESULTS

% Output option I: Display eigenvalues only.

if (nargout == 0)
    if iter < maxit
        eigenvalues = \text{diag}(d)
    else
        eigenvalues =[]
    end
end

% Output option II: Set eigenvalues equal to output vector.

if (nargout == 1)
    if iter < maxit
```
varargout{1} = diag(d);
else
    varargout{1} =[];
end

% Output option III: Output diagonal matrix of eigenvalues and
% corresponding matrix of eigenvectors.
if (nargout == 2)
    if iter < maxit
        varargout{1} = v;
        varargout{2} = d;
    else
        varargout{1} =[];
        varargout{2} =[];
    end
end

% Output option IV: Output diagonal matrix of eigenvalues and
% corresponding matrix of eigenvectors and FLAG.
FLAG(1) = 0; FLAG(2) = mprod;
if iter >= maxit,FLAG(1) = 1; end
if nargout == 3
    varargout{1} = v;
    varargout{2} = d;
    varargout{3} = FLAG;
end

%---------------------%
% END: OUTPUT RESULTS %
%---------------------%

%---------------------------------------%
% BEGIN: COMPUTE REFINED RITZ ITERATION %
%---------------------------------------%

function [rho,v_min,s_min,u_min,rconv] = refined_ritz(D_ritz,R,T,maxitref,Tsz,k)
% Computes the Iterative refined Ritz values and vectors. Also, computes any needed vectors
% for computing residuals.
```
% Input:
% D_ritz - (K x 1) vector of eigenvalues of T (aka Ritz values).
% R - real number - norm of residual vector F.
% T - (TSZ x TSZ) Lanczos tridiagonal matrix.
% MAXITREF - Integer indicating the maximum number of iterations for the iterative Ritz values.
% TSZ - Integer indicates the size of the tridiagonal matrix.

% Output:
% RHO - (K x 1) vector of iterative refined Ritz values.
% V_MIN - (TSZ x K) Matrix of right singular values of [T; 0 R] and iterative refined Ritz vectors.
% S_MIN - (K x 1) vector of minimum singular values of [T; 0 R] associated with V_MIN and U_MIN.
% U_MIN - (TSZ+1 x K) Matrix of left singular values of [T; 0 R].
% Values are needed in computing residuals.
% RCONV - Integer indicate if all K iterative refined Ritz converge.
% Algorithm 5.1 in reference [1].
% DATE MODIFIED: 06/03/2020
% VER: 1.0

% Set up the augmented matrix [T; 0 R]
T_aug = zeros(Tsz+1,Tsz); T_aug(1:Tsz,1:Tsz) = T;
T_aug(Tsz+1:Tsz+1,Tsz) = R;

% Initialize values.
% Initialize min. singular values of [T; 0 R].
% Initialize right singular values of [T; 0 R].
% Initialize left singular values of [T; 0 R].
% Initialize convergence.
% Initialize rho value.
% Used to check for convergence during iterations.
% square root of machine precision - eps

% Compute k number of iterative refined Ritz values/vectors
for j = 1:k
Set difference in rho values to test for stagnation.

diff_rho_0 = -1; v_min_0 = zeros(Tsz,1);

% Iteration to compute the iterative refined Ritz values/vectors
for i=1:maxitref

  % Compute the SVD of [T; 0 R] - rho* I
  [U,S,V] = svd((T_aug-rho(j)*eye(Tsz+1,Tsz)),0);

  % Need the smallest singular triplet of [T; 0 R] - rho* I. Matlab
  % returns order of singular values largest to smallest.
  s_min(j) = S(Tsz,Tsz); v_min(:,j) = V(:,Tsz); u_min(:,j) = U(:,Tsz);

  % Compute the new rho = v_min'*T*v_min (aka refined Ritz value)
  rho(j) = v_min(:,j)'*T*v_min(:,j);

  % Compute the difference of previous rho to check for convergence
  diff_rho = abs( (rho_0(j) - rho(j))/rho(j) );

  % Check for convergence
  if (diff_rho < eps && abs(v_min(:,j)'*u_min(1:Tsz,j)) < sqrt(eps))...
      || s_min(j) < eps || norm(v_min_0 - v_min) < eps
      rconv(j) = 1; break;
  end

  % Check for stagnation to avoid too many unnecessary iterations. Care
  % must be taken to avoid the situation where rho(k) is still changing
  % very slightly at first and then an increase in change later. An
  % early termination due to stagnation with no convergegence
  % may avoid increase in change later that converges.
  if abs(diff_rho_0 - diff_rho)/diff_rho < eps, break; end % stagnate
  if i>= 10 && mod(i,10)==0 % update every 10 iterations to avoid early termination.
      diff_rho_0 = diff_rho;
  end

  % Update rho_0 and v_min_0 to test convergence.
  rho_0(j) = rho(j); v_min_0 = v_min;
end
After K iteration finish check to see if *all* K iterative Refined Ritz values have all converged. Reset rconv to return an integer value, rconv = k all converged and rconv = 0, not all converged.

if all(rconv == 1), rconv = k; else, rconv = 0; end

%-------------------------------------%
% END: COMPUTE REFINED RITZ ITERATION %
%-------------------------------------%

%-----------------------------%
% BEGIN: MATRIX-VECTOR PRODUCT. %
%-----------------------------%
function x = matrixprod(A,x,n)
% Computes the matrix vector products.

% Input:
% A - Matrix A.
% x - (N x 1) vector to multiply OP (operator).
% N - size of A.

% Output:
% x - (N x 1) Product of OP*X (operator).

% if ischar(A)
% x=feval(A,x,n); % does not accept input parameters
elseif (isa(A, 'function_handle'))
  x=A(x); % simple function handle
else
  x = A*x;
end
%-----------------------------%
% END: MATRIX-VECTOR PRODUCT. %
%-----------------------------%
A.2 MATLAB function rd2svds(A,m,n,O,k,tol)

function [U,S,V,FLAG] = rd2svds(A,m,n,O,k,tol)

% RD2SVDS: Computes the k largest singular value and associated singular vectors
% of a m x n matrix A such that A*V = U*S and A'*U = V*S, V'*V=I
% and U'*U = I and S is a diagonal matrix.

% CONVERGENCE: sqrt(|| A*v - u*s||^2 + || A'*u - v*s ||^2))<= tol*||A||
% where norm is the 2-norm and ||A|| is approximated by largest
% singular value of the projected matrix B over all iterations.

% INPUT:
% A - m x n numeric matrix A or an M-file ('Afunc'). If the m x n matrix A
% is a filename then y = Afunc(x,m,n,'transpose'). If transpose = 'F',
% then y = A*x. If transpose = 'T', then y = A'*x.
% m,n - size of the m x n matrix A
% P - n x 1 "right" starting vector & first column of Lanczos matrix P,
% no vector P = []
% k - number of desired singular triplets
% tol - user specified tolerance - if k > 1 then tolerance is changed to
% min(sqrt(eps), 1d-2*tol_user) for 1,...,k-1 and to tol for k
% singular triplet

% OUTPUT:
% U - m x k left approximate singular vectors
% V - n x k right approximate singular vectors
% S - k x k diagonal matrix of singular values
% FLAG - integer output to indicate convergence
% - 0 all k singular triplets converged within tol*||A||
% - 1 not all k singular triplets converged - output the < k converged
% singular triplets and the last iteration approximations.

% DATE MODIFIED: 6/22/21
% VER: 1.0

% AUTHORS:
% James Baglama email: jbaglama@uri.edu
% Vasilije Perovic email: perovic@uri.edu
% Jennifer Picucci email: jenniferpicucci@uri.edu

31
% Initialize values - maxit_outer and maxit_inner can be changed by user.
maxit_outer = 2000; maxit_inner = 100; 
srteps = eps^(1/2); f = zeros(n,1); FLAG(1) = 0;
tol_user = tol; V = zeros(n,k); U = zeros(m,k); S = zeros(k); B = zeros(2,2); Q = zeros(m,2);
if k > 1, tol = min(srteps,ld-2*tol_user); end; tol = max(tol,eps); numIterRef = 0;

% Begin iter_k for loop
for iter_k = 1:k

% Set values for each k value.
iter = 1; Smax = 0; if iter_k == k, tol = max(tol_user,eps); end

% Set up starting vector P(:,1)
if isempty(P), P = randn(n,1); end; if iter_k > 1, P = f - V*(f'*V)'; end
P = [P/norm(P) zeros(n,1)];

% Matrix-vector product, orthogonalization, and deflation to get Q(:,1)
if ischar(A), Q(:,1) = feval(A,P(:,1),m,n,'F'); else, Q(:,1) = A*P(:,1); end
if iter_k > 1, Q(:,1) = Q(:,1) - U*(Q(:,1)'*U)'; end; B(1,1) = norm(Q(:,1));
Q(:,1) = Q(:,1)*(1/B(1,1));

% Matrix-vector product, orthogonalization, and deflation to get P(:,2)
if ischar(A), P(:,2) = feval(A,Q(:,1),m,n,'T'); else, P(:,2) = (Q(:,1)'*A)' end
if iter_k > 1, P(:,2) = P(:,2) - V*(P(:,2)'*V)'; end
P(:,2) = P(:,2) - P(:,1)*B(1,1); B(1,2) = norm(P(:,2)); P(:,2) = P(:,2)*(1/B(1,2));

% Matrix-vector product, orthogonalization, and deflation to get Q(:,2)
if ischar(A), Q(:,2) = feval(A,P(:,2),m,n,'F'); else, Q(:,2) = A*P(:,2); end
if iter_k > 1, Q(:,2) = Q(:,2) - U*(Q(:,2)'*U)'; end
Q(:,2) = Q(:,2) - Q(:,1)*B(1,2); B(2,2) = norm(Q(:,2)); Q(:,2) = Q(:,2)*(1/B(2,2));

% Matrix-vector product, orthogonalization, and deflation to get f
if ischar(A), f = feval(A,Q(:,2),m,n,'T'); else, f = (Q(:,2)'*A)' end
if iter_k > 1, f = f - V*(f'*V)'; end; f = f - P(:,2)*B(2,2);
f = f - P*(f'*P)'; fnorm = norm(f);
% Begin main iter while loop
while (iter <= maxit_outer)

% Compute the largest singular value and associated vector of B.
[u_b,s_b,v_b] = svd(B); v_b = v_b(:,1); u_b = u_b(:,1); s_b = s_b(1,1);
Smax = max(Smax,s_b);

% Compute iterative refined Ritz value/vectors.
iter_refined = 0; rho = Smax^2; rho_0 = rho; rconv = 0; v_rf_0 = zeros(2,1);
T = B'*B; B_aug = [T; 0 B(2,2)*fnorm]; diff_rho_0 = -1;
for i=1:maxit_inner
    B_aug(1,1) = T(1,1) - rho; B_aug(2,2) = T(2,2) - rho;
    [u_rf,s_rf,v_rf] = svd(B_aug,0);
    v_rf = v_rf(:,2); s_rf = s_rf(2,2); rho = norm(B*v_rf)^2;
    diff_rho = abs((rho_0 - rho)/max(rho,rho_0));
    if (diff_rho < eps && abs(v_rf'*u_rf(1:2,2)) < sqrt(eps)) ... || norm(abs(v_rf_0) - abs(v_rf)) < eps || s_rf < eps
        u_rf = B*v_rf; alpha = norm(u_rf); u_rf = u_rf/alpha;
        rho = sqrt(rho); rconv = 1; break; % Converge iter. ref. exit.
    end
    if abs(diff_rho_0 - diff_rho)/max(diff_rho,eps) < eps, break; end
    % Stagnate exit
    if i >= 10 && mod(i,10) == 0, diff_rho_0 = diff_rho; end
    rho_0 = rho; v_rf_0 = v_rf;
end
if abs(v_b'*v_rf) > 0.9 && rconv == 1 && iter>1, iter_refined = 1; end

% Convergence check.
if ~iter_refined
    B_aug = [-Smax*eye(2,2) B; B' -Smax*eye(2,2); zeros(1,4)]; B_aug(5,2) = fnorm;
    [~,v] = svd(B_aug,0); v_c = v(3:4,4); v_c = v_c/norm(v_c);
    u_c = v(1:2,4); u_c = u_c/norm(u_c);
else
    v_c = v_rf; u_c = u_rf;
end
rho_c = u_c'*B*v_c;
norm_res = sqrt(norm(B*v_c - rho_c*u_c)^2 + norm(B'*u_c - rho_c*v_c)^2 + (u_c(2)*fnorm)^2);
if norm_res < tol*max(Smax,S(1,1)) && abs(v_b'*v_c) > 0.9, break; end
% Converge exit

if iter_refined

% Count number of iter. refined restarts.
numIterRef = numIterRef + 1;

% Restart with iterative refined Ritz value/vectors.
f = P*(B'*u_rf - alpha*v_rf) + f*u_rf(2); B=[]; B(1,1) = alpha;
P(:,1) = P*v_rf; Q(:,1) = Q*u_rf; f = f - (P(:,1)'*f)*P(:,1);
B(1,2) = norm(f); P(:,2) = f*(1/B(1,2));

else

% Restart with Ritz value/vectors
P = [P*v_b f*(1/fnorm)]; Q(:,1) = Q*u_b;
B(1,1) = s_b; B(1,2) = fnorm*u_b(2);
end

% Matrix-vector product, orthogonalization, and deflation to get Q(:,2)
if ischar(A), Q(:,2) = feval(A,P(:,2),m,n,'F'); else, Q(:,2) = A*P(:,2); end
if iter_k > 1, Q(:,2) = Q(:,2) - U*(Q(:,2)'*U)' end
Q(:,2) = Q(:,2) - Q(:,1)*(Q(:,2)'*Q(:,1));
B(2,2) = norm(Q(:,2)); Q(:,2) = Q(:,2)*(1/B(2,2));

% Matrix-vector product, orthogonalization, and deflation to get f
if ischar(A), f = feval(A,Q(:,2),m,n,'T'); else, f = (Q(:,2)'*A)'; end
if iter_k > 1, f = f - V*(f'*V)' end; f = f - P(:,2)*B(2,2);
fnorm = norm(f);

iter = iter+1;
end % end: iter while loop

% Deflation or output.
U(:,iter_k)= Q*u_c; V(:,iter_k) = P*v_c; S(iter_k,iter_k) = rho_c;
if iter >= maxit_outer
U = U(:,1:iter_k); V = V(:,1:iter_k); S = S(1:iter_k,1:iter_k);
FLAG(1) = 1; return;
end

FLAG(2) = numIterRef;

end % end: iter_k for loop
A.3 MATLAB function trrsvds(varargin)

function varargout = trrsvds(varargin)

% TRRSVDS: Computes the k largest or smallest singular values and associated
% singular vectors of a m x n matrix A such that A*V = U*S and
% A'*U = V*S, V'*V=I and U'*U = I and S is a diagonal matrix.

% PROGRAM INFORMATION:
% -------------------

% ... = TRRSVDS(A)
% ... = TRRSVDS('AFUN',m,n)

% The first input argument into TRRSVDS can be a numeric matrix A or
% a function. If the m x n matrix A is a function, 'Afunc,'
% then the structure must be y = Afunc(x,m,n,'transpose'). If transpose = 'F',
% then y = A*x. If transpose = 'T', then y = A'*x.

% OUTPUT OPTIONS:
% ---------------

% I.) TRRSVDS(A)
% If convergence, displays the k desired singular values.

% II.) S = TRRSVDS(A)
% If convergence, returns k singular values in the vector S.

% III.) [U,S,V] = TRRSVDS(A)
% If convergence, S is a diagonal matrix that contains the k desired
% singular values in descending order along the diagonal, the matrix
% V contains the corresponding "right" singular vectors, and U contains
% the corresponding "left" singular vectors such that that A*V = U*S,
% A'*U = V*S, V'*V = I, and U'*U = I. If TRRSVDS reaches the maximum
% number of iterations before convergence then U=[], S = [] and V = [].

% IV.) [U,S,V,STATS] = TRRSVDS(A)
% This option returns the same as (III) plus approximation of U,S,V if
% no convergence and a structure STATS that reports statistical
% information:
% STATS =
% numMatProds:  -> number of matrix-vector products with A and A^T
% timeMatProds:  -> total time computing products with A and A^T
% numIterRefRestart:  -> number of restarts with Iterative Refined vectors
% timeCompIterRef:  -> time computing iterative Refined vectors
% (independent of using them to restart)
% timeReorth:  -> time spent on full reorthogonalization
% timeTotal:  -> total time elapsed
% estimateSVmax:  -> estimate of the maximum singular value over all iterations
% estimateSVmin:  -> estimate of the maximum singular value over all iterations
% convergedKVals:  -> true if all k singular triplets converged, otherwise false
% outputestRitzorIterRef:  -> if converged, names which values were used to exit
% Ritz, Harmonic Ritz, Refined Ritz, or Iterative Refined Ritz.
% If the maximum number of iterations are reached before convergence of all k desired
% singular values, convergedKVals = 'False' then the matrices U, V, and S contain any
% singular triplets that have converged plus the last singular triplets
% approximation for the pairs that have not converged.
% INPUT OPTIONS:
% ---------------
% ... = TRRSVDS(A,OPTS) or TRRSVDS('AFUN',m,n,OPTS)
% OPTS is a structure containing input parameters. The input parameters can
% be given in any order and can greatly influence convergence rates. The structure OPTS
% may contain some or all of the following input parameters. If parameter OPTS is missing
% or an input parameter in OPTS is not set, default value(s) are used. The string for the
% input parameters
% can contain upper or lower case characters.
% INPUT PARAMETER    DESCRIPTION
% OPTS.RoH    Four letter string ( 'RITZ' or 'HARM' ) specifying the use of either Ritz
% vectors or harmonic Ritz vectors for thick-restarting.
% DEFAULT VALUE    RoH = 'HARM' if SIGMA = 'SS'
%                   RoH = 'RITZ' if SIGMA = 'LS' or if cond(B) > 1/sqrt(eps)
% OPTS.K    Number of desired singular values.
% DEFAULT VALUE    K = 1
% OPTS.M_B Number of Lanczos vectors, i.e. size bidiagonal Lanczos matrix. Full reorthogonalization is used. Large M_B will increase non-matrix-vector product CPU times.

DEFAULT VALUE M_B = 2 if SIGMA = 'LS'

DEFAULT VALUE M_B = 15 if SIGMA = 'SS'

% OPTS.MAXIT Maximum number of iterations, i.e. maximum number of restarts.

DEFAULT VALUE MAXIT = 2000

% OPTS.MAXITREF Maximum number of iterations used to find iterative refined Ritz singular values.

DEFAULT VALUE MAXITREF = 100

% OPTS.METHOD Three letter string ('NOR', 'AUG', or 'THK') to determine which method to use. Hybrid method - 'NOR' or 'AUG'. Determines how iterative refined Ritz values/vectors are computed in the hybrid method.

NOR - (Hybrid) Lanczos Bidiagonal decomposition of size m

AP_m = Q_mB_m and A'Q_m = P_mB_m' + f_me_m'

Equivalent eigenvalue system

A'AP_m = P_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_MB


Equivalent eigenvalue system

A'AP_m = P_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_MB

Compute iterative refined Ritz on [B*B; 0 B_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_MB

AUG - (Hybrid) Equivalent eigenvalue system

[0 A; A' 0][Q_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_mB_MB

[0 0; f_me_m' 0]

Compute iterative refined Ritz on [0 B; B' 0; norm(f) 0]

THK - (non-Hybrid) no iterative refined Ritz values are computed. Thick-restarted only with either Ritz vectors or harmonic Ritz vectors depending on parameter RoH value.

DEFAULT VALUE AUG = 'NOR'

% OPTS.REORTH Three letter string ('ONE' or 'TWO') specifying whether to use one-sided ('ONE') full reorthogonalization or two-sided ('TWO'). One-sided is performed only on the "short" vectors. Two-sided orthogonality is always used when cond(A) estimated by cond(B) > 1/sqrt(eps).

DEFAULT VALUE REORTH = 'ONE'

% OPTS.SIGMA Two letter string ('LS' or 'SS') specifying the location of the desired singular values.

'LS' Largest singular values and 'SS' Smallest singular values.
% DEFAULT VALUE SIGMA = 'LS'
% 
% OPTS.TOL  Tolerance used for convergence. Convergence is determined when
% MAX (SQRT(|| AV - US||^2 + || A'U - VS ||^2)) <= TOL*||A||.
% Norm is the two norm and [U,S,V] are approximated either by Ritz, harmonic
% Ritz, Refined Ritz, or Iterative Refined Ritz singular triplets. ||A|| is
% approximated by largest singular value of all projection matrices.
% DEFAULT VALUE  TOL = SQRT(EPS) (roughly 1d-8)
% 
% OPTS.P0  An n x 1 starting vector if m >= n and sigma = 'LS' or 'SS' and an m x 1
% starting vector if m < n and sigma = 'SS'. P0 is not explicitly created
% and use first column of matrix V.
% DEFAULT VALUE  P0 = randn(n,1)
% 
% DATE MODIFIED: 6/22/21
% VER: 1.0
% 
% AUTHORS:
% James Baglama email: jbaglama@uri.edu
% Vasilije Perovic email: perovic@uri.edu
% Jennifer Picucci email: jenniferpicucci@uri.edu
% 
% REFERENCES:
% 1. Baglama, J, Perovic, V, and Picucci, J, "Hybrid Iterative Refined Restarted
% Lanczos Bidiagonalization Method", 2021 submitted Numerical
% https://doi.org/10.1137/20M1344834

% Start timing count using the MATLAB tic command.
tStart = tic;

% Incorrect number of output arguments requested.
if (nargout > 4 || nargout==2 ), error('ERROR: Incorrect number of output arguments.'); end

%---------------------------------------------------%
% BEGIN: PARSE INPUT VALUES. %
%---------------------------------------------------%

141
if nargin == 0, help ttrsvds, return, end

% Matrix A is stored in varargin{1}. Check type (numeric or character) and dimensions.
if (isstruct(varargin{1})), error('A must be a matrix.'), end
if ischar(varargin{1})
  if nargin == 1, error('Need dimension M for matrix A.'); end
  m = varargin{2};
  if ~isnumeric(m) || length(m) ~= 1
    error('Second argument M must be a numeric value.');
  end
  if nargin == 2, error('Need dimension N for matrix A.'), end
  n = varargin{3};
  if ~isnumeric(n) || length(n) ~= 1
    error('Third argument N must be a numeric value.');
  end
else
  if ~isnumeric(varargin{1}), error('ERROR: A must be a numeric matrix.'), end
  [m,n] = size(varargin{1});
end

% Square root of machine tolerance used in convergence testing.
sqreps = sqrt(eps);

% Set all input options to default values.
k=1; maxit = 1000; m_b=[]; reorth='ONE'; method='NOR';
maxitref=100; sigma = 'LS'; tol = sqreps; V=[]; roh = [];

% Get input options from the data structure.
if nargin > 1 + 2+ischar(varargin{1})
  options = varargin(2+2+ischar(varargin{1})):nargin;
  names = fieldnames(options);
  I = strmatch('ROH',upper(names),'exact');
  if ~isempty(I), roh = upper(getfield(options,names{I})); end
  I = strmatch('K',upper(names),'exact');
  if ~isempty(I), k = getfield(options,names{I}); end
  I = strmatch('m_B',upper(names),'exact');
  if ~isempty(I), m_b = getfield(options,names{I}); end
end
I = strmatch('MAXIT',upper(names),'exact');  
if ~isempty(I), maxit = getfield(options,names{I}); end  
I = strmatch('MAXITREF',upper(names),'exact');  
if ~isempty(I), maxitref = getfield(options,names{I}); end  
I = strmatch('METHOD',upper(names),'exact');  
if ~isempty(I), method = upper(getfield(options,names{I})); end  
I = strmatch('REORTH',upper(names),'exact');  
if ~isempty(I), reorth = upper(getfield(options,names{I})); end  
I = strmatch('SIGMA',upper(names),'exact');  
if ~isempty(I), sigma = upper(getfield(options,names{I})); end  
I = strmatch('TOL',upper(names),'exact');  
if ~isempty(I), tol = getfield(options,names{I}); end  
I = strmatch('P0',upper(names),'exact');  
if ~isempty(I), V = getfield(options,names{I}); end  
end

%***************************************************
% Check for some input errors in the data structure.
% **** This is not an exhaustive check list. ******
%***************************************************

% Check that input values are numerical or char values.
if (~isnumeric(k) || ~isnumeric(m_b) || ~isnumeric(maxit) || ...  
    ~isnumeric(maxitref) || ~ischar(method)|| ~ischar(reorth) || ...  
    ~ischar(sigma) || ~isnumeric(tol))
    error('ERROR: Incorrect type for input value(s) in the structure.');
end

% Check value of MAXIT
if maxit <= 0, error('ERROR: MAXIT must be a positive value.'); end

% Check value of MAXITREF
if maxitref <= 0,    error('ERROR: MAXITREF must be a positive value.'); end  
if maxitref > maxit, error('ERROR: MAXITREF should be less than MAXIT'); end  
if maxitref == 1
    warning(['WARNING: MAXITREF = 1 - computing refined Ritz - not iterative refined  
    Ritz.']);
    warning(['WARNING: MAXITREF = 1 - not recommended with this routine.']);
end  
if maxitref < 100
warning(['WARNING: MAXITREF should be at least 100 to ensure Iterative Refined Ritz are computed.']);
end

% Check value of METHOD
if length(method) ~= 3, error('ERROR: METHOD must be NOR, AUG, or THK.'); end
if ('strcmp(method,'NOR') && 'strcmp(method,'AUG') && 'strcmp(method,'THK'))
    error('ERROR: METHOD must be NOR, AUG, or THK.');
end

% Check the values of REORTH
if length(reorth) ~= 3, error('ERROR: REORTH must be ONE or TWO.'); end
if ('strcmp(reorth,'ONE') && 'strcmp(reorth,'TWO')
    error('ERROR: REORTH must be ONE or TWO.');
end

% Check value of SIGMA.
if length(sigma) ~= 2, error('ERROR: SIGMA must be LS or SS'); end
if ('strcmp(sigma,'SS') && 'strcmp(sigma,'LS')
    error('ERROR: SIGMA must be LS or SS.');
end

% Interchange m and n so that size(A'A) = min(m,n). Avoids
% finding zero values when searching for the smallest singular values.
interchange = 0; if n > m && strcmp(sigma,'SS'), t=m; m=n; n=t; interchange = 1; end

% Determine value of m_b to use
if isempty(m_b)
    if strcmp(sigma,'LS'), m_b = 2; else, m_b = 15; end
end

% Preallocate memory for W and F. These matrices are full and resizing will cause
% an increase in cpu time.
W = zeros(m,m_b); F = zeros(n,1);

% If starting p0 is not given then set starting vector p0 to be a
% (n x 1) matrix of normally distributed random numbers.
% p0 is not explicitly created and use first column of matrix V.
if isempty(V)
    V = zeros(n,m_b); % Preallocate memory for all V.
280  \[ V(:,1) = \text{randn}(n,1); \]
281  else
282     \[ V(:,2:m_b) = \text{zeros}(n,m_b-1); \] % Preallocate memory for other columns of \( V \).
283  end
284
285  \% Check for input errors in the data structure for \( K, M_B, \) and \( TOL \)
286  if \( k \leq 0 \), error('ERROR: \( K \) must be a positive value.'), end
287  if \( k > \min(n,m) \), error('ERROR: \( K \) must be less than \( \min(n,m) \)'), end
288  if \( m_b \leq 1 \), error('ERROR: \( M_B \) must be greater than 1.'), end
289  if \( \text{tol} < 0 \), error('ERROR: \( TOL \) must be non-negative.'), end
290  if \( m_b >= \min(n,m) \)
291      \[ m_b = \text{floor}(\min(n,m)-0.1); \]
292      \% Warning if \( M_B \) is too large
293      warning(['Changing \( M_B \) to ',num2str(m_b)]);
294  end
295
296  if \( m_b - k - 1 < 0 \)
297      \[ k = m_b - 1; \]
298      \% Warning if \( K \) is too large
299      warning(['Changing \( K \) to ',num2str(k)]);
300  end
301  end
302  if \( m_b - k - 1 < 0 \)
303      \[ m_b = \text{ceil}(k+1+0.1); \]
304      \% Warning if \( M_B \) is too large
305      warning(['Changing \( M_B \) to ',num2str(m_b)]);
306      \% Warning if \( K \) is too large
307      warning(['Changing \( K \) to ',num2str(k)]);
308  end
309  if \( \text{isnumeric}(V) \), error('ERROR: Incorrect starting vector P0.'), end
310  if \( \text{size}(V,1) \neq n \), error('ERROR: Incorrect size of starting vector P0.'), end
311  \%
312  \% Check value of \( TOL \).
313  \% Set tolerance to machine precision if \( \text{tol} < \text{eps} \).
314  if \( \text{tol} < \text{eps} \), \( \text{tol} = \text{eps} \); warning('WARNING: Changing TOL to EPS'); end
315  \%
316  \% Determine which vectors to use for thick-restarting
317  if \( \text{isempty(roh)} \)
318    if strcmp(sigma,'LS'), roh = 'RITZ'; else, roh = 'HARM'; end
319  else
320    if length(roh) \neq 4, error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM');
321  end
312  \%
313  \% Set tolerance to machine precision if \( \text{tol} < \text{eps} \).
314  if \( \text{tol} < \text{eps} \), \( \text{tol} = \text{eps} \); warning('WARNING: Changing TOL to EPS'); end
315  \%
316  \% Determine which vectors to use for thick-restarting
317  if \( \text{isempty(roh)} \)
318    if strcmp(sigma,'LS'), roh = 'RITZ'; else, roh = 'HARM'; end
319  else
320    if length(roh) \neq 4, error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM');
if ~ischar(roh), error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM'); end
if (~strcmp(roh,'RITZ') &~strcmp(roh,'HARM') )
    error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM');
end

%--------------------------%
% END: PARSE INPUT VALUES. %
%--------------------------%

%-----------------------------------------------------------%
% BEGIN: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
%-----------------------------------------------------------%
% <- DO NOT MODIFY ->

% Set convergence values for testing to 0 or empty.
conv_rz_tol =0; conv_hm_tol =0; conv_it_tol=0; conv_hm_sqrt=0;
conv_it_sqrt = 0; num_conv_val=0; V_B_cv=[]; U_B_cv=[]; S_B_cv=[]; out_val = -1;
res_norm_rz=0;

% Set all values empty for computing singular triplets for matrix B
U_B_rz=[]; S_B_rz=[]; V_B_rz=[]; % Holds the singular triplets of B (aka Ritz)
U_B_hm1=[]; S_B_hm1=[]; V_B_hm1=[]; % Holds the singular triplets and intermediate steps for computing harmonic Ritz
R_hm=[]; V_B_hm2=[]; V_hm_last=[]; % singular values of B
s=[]; V_B_hm3 =[]; V_B_hm=[]; S_B_hm=[]; % singular values of B
U_rf=[]; S_rf=[]; V_rf=[]; s_min=[]; % Holds iterative refined Ritz singular values of B
W_rf=[]; rconv=0; U_rf1=[]; S_rf1=[]; % and values used for convergence tests.
V_rf1=[]; s_min1=[]; W_rf1=[]; rconv1=0;
U_B_it=[]; S_B_it=[]; V_B_it=[];

% Initialize array for max/min. singular value of B depending on sigma.
if strcmp(sigma,'SS'), dmax = Inf(m_b,1); end
if strcmp(sigma,'LS'), dmax = -Inf(m_b,1); end

% Initialization and description of local variables.
B = []; % Bidiagonal matrix.
Bsz =[]; % Size of the bidiagonal matrix (will be <= m_B)
delta = 0.9; % Used to determine when the Iter. refined Ritz singular vectors are "close" enough to the desired Ritz singular vectors
eps23 = eps^(2/3); % Two thirds of eps, used for Smax. Avoids using zero.
I=[]; % Used for indexing.
iter = 1; % Main loop iteration count.
k_org = k; % Set k_org to the original value of k. k may be adjusted during % iterations.
Fnorm=[]; % Two norm of the residual vector F.
Ritz = 1; % Toggle for Lanczos bidiagonalization on which vectors are used % for restarting
Smax = -Inf; % Holds the maximum value of all computed singular values of B % est. ||A||_2.
Smin = Inf; % Holds the minimum value of all computed singular values of B % est. cond(A).
SVTol = min(sqrteps,tol); % Tolerance to determine when Lanczos encounters linearly % dependent vectors.
S_rf_0=zeros(m_b,1); % Used for iterative refined comparsion from last iteration.

% Initialization of values used for STATS output.
mprod = 0; % Number of matrix vector products with A and A^T.
IterRefRestart = 0; % Number of restarts with Iter Refined Vectors.
timeMatProds=0; % Total time computing matrix vector products with A and A^T.
timeIterRef = 0; % Total time computing Iter. Ref. values.
timeReorth = 0; % Total time doing full reorthogonalization

%-----------------------------------------------% % END: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. % %-----------------------------------------------%

%---------------------------% % BEGIN: ITERATION PROCESS. % %---------------------------%
while (iter <= maxit)

% Compute the Lanczos bidiagonalization decomposition.
[V,W,F,B,mprod,timeMatProds,timeReorth] = ablanzbd(varargin{1},V,...
W,F,B,k,interchange,m_b,n,m,mprod,Ritz,SVTol*Smax,reorth,iter,...
timeMatProds,timeReorth);

% Reset k back to the original value k_org.
k = k_org;
% Determine the size of the bidiagonal matrix B.
Bsz = size(B,1);

% Compute the norm of the vector F.
Fnorm = norm(F);

% Compute singular triplets of B. MATLAB's svds orders the singular values
% largest to smallest.
[U_B_rz,S_B_rz,V_B_rz] = svd(B); S_B_rz = diag(S_B_rz);

% Estimate ||A|| using the largest singular value over all iterations
% and estimate the cond(A) using approximations to the largest and smallest
% singular values. If a small singular value is less than sqrt(eps) use only Ritz
% vectors for thick restarted and require two-sided reorthogonalization.
if iter==1
Smax = S_B_rz(1); Smin = S_B_rz(Bsz);
else
Smax = max(Smax,S_B_rz(1)); Smin = min(Smin,S_B_rz(Bsz));
end
Smax = max(eps23,Smax);
if Smin/Smax < sqrt(eps), reorth = 'TWO'; roh='RITZ'; end

% Re-order the singular values accordingly. MATLAB's SVD orders the
% singular values largest to smallest.
if strcmp(sigma,'SS')
 [~,I] = sort(S_B_rz,'ascend');
 U_B_rz = U_B_rz(:,I); V_B_rz = V_B_rz(:,I); S_B_rz = S_B_rz(I);
end

% If selected compute Harmonic Ritz values and vectors.
U_B_hm=[]; S_B_hm=[]; V_B_hm=[]; % Reset to empty.
if strcmp(roh,'HARM')
 R_hm = Fnorm*U_B_rz(Bsz,:);
 % Update the SVD of B to be the SVD of [B ||F||E_m].
[U_B_hm1,S_B_hm1,V_B_hm1] = svd([diag(S_B_rz) R_hm']); S_B_hm1 = diag(S_B_hm1);
if strcmp(sigma,'SS') % reorder if looking for smallest.
 [~,I] = sort(S_B_hm1,'ascend');
 U_B_hm1 = U_B_hm1(:,I); V_B_hm1 = V_B_hm1(:,I); S_B_hm1 = S_B_hm1(I);
end
U_B_hm1 = U_B_rz*U_B_hm1; U_B_hm = U_B_hm1;
V_B_hm2 = [[V_B_rz; zeros(1,Bsz)] flipud(eye(Bsz+1,1))]*V_B_hm1;
V_B_hm_last = V_B_hm2(Bsz+1,:); % Set equal to the last row of V_B_hm2.
s = Fnorm*(B\flipud(eye(Bsz,1)));
V_B_hm3 = V_B_hm2(1:Bsz,:) + s*V_B_hm2(Bsz+1,:);

% Compute the orthogonal harmonic to get the Rayleigh Quotient values
% to test for convergence.
[V_B_hm,R_Q] = qr(V_B_hm3,0);
D = diag(sign(diag(R_Q))); % Compute signs, so diag(R_Q) > 0 and signs of
V_B_hm = V_B_hm*D; % cols. V_B_hm & V_B_hm3 are same. Note: D*D = I.
S_B_hm = diag(U_B_hm'*B*V_B_hm); % Rayleigh Quotient values

% Need to reorder the Harmonic values based on Rayleigh Quotient.
if strcmp(sigma,'SS')
 [~,I] = sort(S_B_hm,'ascend');
else
 [~,I] = sort(S_B_hm,'descend');
end
U_B_hm = U_B_hm(:,I); V_B_hm = V_B_hm(:,I); S_B_hm = S_B_hm(I);
end

% Used later for calling iterative refined function
if strcmp(sigma,'LS')
 for j=1:k
 dmax(j) = max([dmax(j),S_B_rz(j)]);
 end
else
 for j=1:k
 dmax(j) = min([dmax(j),S_B_rz(j)]);
 end
end

% Convergence tests for Ritz and if computed Harmonic Ritz singular values/vectors.
res_norm_rz_pre = res_norm_rz;
[conv_rz_tol,conv_hm_tol,conv_it_tol,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt,V_B_cv,
 U_B_cv,S_B_cv,...
res_norm_rz,res_norm_hm,res_norm_it,out_val] = convtests(B,Bsz,Fnorm,tol,k_org,
U_B_rz,...
S_B_rz,V_B_rz,U_B_hm,S_B_hm,V_B_hm,[],[],[],Smax,sqrteps);
num_conv_val = max([num_conv_val,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt]);
% If all desired singular values converged then exit main loop.
if conv_rz_tol >= k_org || conv_hm_tol >= k_org || iter >= maxit, break, end

%--------------------------------------------------------------------%
% BEGIN: COMPUTE ITERATIVE REFINED AND DETERMINE RESTARTING VECTORS. %
%--------------------------------------------------------------------%

% Compute Iterative refined Ritz values.
% rconv = 0; % set the number of convergence iterative refined Ritz to 0.
if strcmp(method,'THK') && iter > 1
    IRstart = tic;
    [U_rf,S_rf,V_rf,rconv] = refined_ritz(dmax(1:k),Fnorm,abs(B(Bsz,Bsz)),B,maxitref,Bsz,
        k,method);
    timeIterRef = timeIterRef + toc(IRstart);
end

% rconv returns 0 or k. 0 indicates not all converged. rconv = k indicates all k converged
% Compute inner product between the k converged iterative refined Ritz vector(s)
% and Ritz vector(s). Also, check if iterative refined Ritz values are less than
% the previous Ritz values. If so, do not use iterative refined Ritz
% vectors to restart.
ang_rz_rf = 0; decr = 0;
if rconv > 0
    ang_rz_rf = abs(diag(V_B_rz(:,1:k)'*V_rf(:,1:k))); % compute the innner prod. to check
    % angle
    for i=1:k_org
        if k_org > 1 || (k_org == 1 && Bsz > 2) % Requires iter. refined to be better approx.
            % prev. vals.
            if strcmp(sigma,'LS') && (S_rf(i) + eps < S_rf_0(i)), decr = 1; end
            if strcmp(sigma,'SS') && (S_rf(i) > S_rf_0(i) + eps), decr = 1; end
        end
    end
end

% Reset S_rf_0 to be prev. dmax values for next iteration comparision
S_rf_0 = dmax;

% Find all inner products between iterative refined Ritz and Ritz >
% delta = 0.9
I = find(ang_rz_rf > delta);

% Convergence check.
if length(I) == k && decr == 0
    U_rf1 = U_rf; V_rf1 = V_rf; % if converged use iterative refined Ritz
    iterrefined = 1;
else
    % Use refined Ritz on augmented system - when all iterative refined Ritz
    % failed to converge.
    [U_rf1,S_rf1,V_rf1,rcv1] = refined_ritz(dmax(1:k),Fnorm,abs(B(Bsz,Bsz)),B,1,Bsz,k,'AUG');
    iterrefined = 0;
end

[V_B_it,V_R] = qr(V_rf1(:,1:k),0); % QR to ensure orthogonal vectors
D = diag(sign(diag(V_R))); % Compute signs and set diag(R) > 0
V_B_it = V_B_it*D; % cols. V_B_it & V_rf1 are same. Note: D*D = I
[U_B_it,U_R] = qr(U_rf1(:,1:k),0); % QR to ensure orthogonal vectors
D = diag(sign(diag(U_R))); % Compute signs and set diag(R) > 0
U_B_it = U_B_it*D; % cols. U_B_it & U_rf1 are same. Note: D*D = I
S_B_it = diag(U_B_it'*B*V_B_it); % Rayleigh Quotient values - no need to reorder

% Double check vectors are close to Ritz vectors.
[-,R_ch] = qr(V_B_rz(:,1:k)'*V_B_it(:,1:k),0);
if abs(prod(diag(R_ch))) > sqrt(eps)
    % Convergence tests for Ritz singular value and to determine if to compute Iter. Refined.
    [conv_rz_tol,conv_hm_tol,conv_it_tol,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt,V_B_cv,U_B_cv,
     S_B_cv,...
     res_norm_rz,res_norm_hm,res_norm_it,out_val] = convtests(B,Bsz,Fnorm,tol,k,U_B_rz,...
     S_B_rz,V_B_rz,[],[],[],U_B_it,S_B_it,V_B_it,Smax,sqrt(eps));
    num_conv_val = max([num_conv_val,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt]);
end

% If all desired singular values converged then exit main loop.
if conv_it_tol >= k_org, break, end

% Do not restart with iterative refined Ritz vectors if no change
% between input and output values.
if rconv > 0
    all_conv = 0;
    for i=1:k_org
...
if abs(S_rf(i) - dmax(i)) < eps*10, all_conv = all_conv +1; end
end
if all_conv >= k_org, rconv = 0; end
end

% Do not restart consecutively with iterative refined Ritz vectors if last restart with
% iterative refined Ritz vectors caused norm of Ritz values/vectors to increase.
if length(I) == k && rconv > 0 && decr == 0 && ~strcmp(method,'THK') && Ritz == 0
    if max(res_norm_rz) < max(res_norm_rz_pre), Ritz = 1; end
end

% Check to determine if restart with iterative refined Ritz or thick-restart
if length(I) == k && rconv > 0 && decr == 0 && ~strcmp(method,'THK') && (Ritz ~= 0 || k == 1)

% Count the number of restarts with Iterative Refined.
IterRefRestart = IterRefRestart + 1;

%------------------------------------------%
% BEGIN: RESTART ITERATIVE REFINED SECTION %
%------------------------------------------%

% Find the coefficients c_rf to setp linear combination.
% See reference [1] for details.
% Follows the equivalent eigenvalue problem.
% A'T*AV = V*(B'T*B) + B(Bsz,Bsz)*f*e'T
% Therefore, the approximate eigenpair is (S_rf^2,V_rf)
% Re-compute residual error of iterative refined vectors.
c_rf = ones(length(S_rf),1);
if k > 1

% For restarting need the square of singular value - i.e. eigenvalue of T = B'T*B.
S_rf = S_rf.^2; T = B'*B;

% Compute norm of residual for A'T*AV = V*(B'T*B) + B(Bsz,Bsz)*f*e'T
for i=1:k
    res_norm_it(i,1) = sqrt(norm((T - S_rf(i)*eye(Bsz))*V_rf(:,i))^2 + (abs(B(Bsz,Bsz))
        *Fnorm*abs(V_rf(Bsz,i)))^2);
% Initialize the k coefficients c_rf for the linear combination of the k iterative refined Ritz singular vectors V_rf.
Bc = zeros(k-1,k); Tem = T(Bsz,1:Bsz); Bc(1,1:k) = V_rf(Bsz,1:k);
for i=2:k-1
  for j=1:k
    Bc(i,j) = Tem*V_rf(:,j)*S_rf(j)^(i-2);
  end
end

% Compute the solution of the k-1 x k homogeneous system using null. Following code is needed to avoid zero column(s) in Bc for numerically converged eigenvectors of B'*B.
Iin = []; Iout=[]; % set variables for which columns are used.
Bc_max = max(abs(Bc),[],'all');
for i=1:k % search for columns numerically zero.
  if norm(Bc(:,i),Inf) < sqrt(eps*Bc_max)
    Iout = [Iout i]; % references which columns to remove.
  else
    Iin = [Iin i];
  end
end
if ~isempty(Iout) % remove numerically zero columns from Bc
  Bc = Bc(1:k-length(Iout)-1,Iin);
  if ~isempty(Bc)
    c_rf(1:k) = zeros(k,1);
    c_rf_in = null(Bc); % call Matlab's null to solve system
    if ~isempty(c_rf_in), c_rf(Iin) = c_rf_in(:,1); end
    c_rf(Iout) = res_norm_it(Iout); % Place zero entries back in sol.
  else
    c_rf = res_norm_it(1:k);
  end
else
  c_rf = null(Bc); % call Matlab's null to solve full system
end

% The coefficients c_rf have been computed at this point.
% Compute the starting vector
v_bar = c_rf(1)*V_rf(:,1) + ... + c_rf(k)*V_rf(:,k)
v_bar = V_rf*c_rf; v_bar = v_bar/norm(v_bar);

% Compute u_bar = B*v_bar and B(1,1).
% Now have A*V*v_bar = W*u_bar*alpha (avoids computing A*V*v_bar)
% u_bar = B*v_bar; alpha = norm(u_bar); u_bar = u_bar/alpha;

% Compute F = A^T*W*u_bar - alpha without using A^T*W*u_bar
F = V*(B'*u_bar - alpha*v_bar) + F*u_bar(Bsz);

% Reset B matrix.
B=[]; B(1,1) = alpha;

% Updated first column of V. Do not resize - slows down code.
V(:,1) = V*v_bar;

% Updated first column of W. Do not resize - slows down code.
W(:,1) = W*u_bar;

% Reorthogonalize the residual vector (independent of REORTH)
ORstart = tic;
F = F - (V(:,1)'*F)*V(:,1);
timeReorth = timeReorth + toc(ORstart);

% Compute the norm of F, scale and update V and B matrices for restarting.
beta = norm(F); V(:,2) = F/beta; B(1,2) = beta;

% Set the Ritz for which vectors restarting
% Lanczos bidiagonalization decomposition
% Ritz = 0 => iterative refined Ritz vectors
% Ritz = 1 => thick-restarting with harmonic or Ritz
Ritz = 0;
%----------------------------------------%
% END: RESTART ITERATIVE REFINED SECTION %
%----------------------------------------%
else
%----------------------------------------%
% BEGIN: THICK-RESTART RITZ SECTION %
%----------------------------------------%
% Simple strategy to improve convergence. Adjust k value.
k = k_org + num_conv_val;
if k < Bsz-1 && k > 1 % used for small values of Bsz
    if abs(S_B_rz(k+1) - S_B_rz(k)) < abs(S_B_rz(k-1) - S_B_rz(k))
        k = k+1;
    end
end
k = max(floor((Bsz+num_conv_val)/2),k); % used for large Bsz
if k >= Bsz, k = Bsz - 1; end

% Use Harmonic Ritz vectors for thick-restarting
if strcmp(roh,'HARM')
    % Vectors are not orthogonal.
    [V_B_hm,R] = qr([ [V_B_hm3(:,1:k); zeros(1,k)] [-s; 1] ],0);
    V(:,1:k+1) = [V F/Fnorm]*V_B_hm;

    % Update and compute the K x K+1 part of B.
    B = diag(S_B_hm1(1:k))*triu((R(1:k+1,1:k)+ R(:,k+1)*V_B_hm_last(1:k))');
    W(:,1:k) = W*U_B_hm1(:,1:k);
else
    % Use Ritz vectors for thick-restarting
    R = Fnorm*U_B_rz(Bsz,:); F = F/Fnorm;
    V(:,1:k+1) = [V*V_B_rz(:,1:k) F];
    B = [diag(S_B_rz(1:k)), R(1:k)'];
    W(:,1:k) = W*U_B_rz(:,1:k);
end

% Set the Ritz for which vectors restarting
% Lanczos bidiagonalization decomposition
% Ritz = 0 => iterative refined Ritz vectors
% Ritz = 1 => thick-restarting with harmonic or Ritz
Ritz = 1;
% Update the main iteration loop count.
iter = iter+1;

end % end main loop

%-------------------------%
% END: ITERATION PROCESS. %
%-------------------------%

%-----------------------%
% BEGIN: OUTPUT RESULTS %
%-----------------------%

% Output option I: Display singular values only.
if (nargout == 0)
  if iter >= maxit
    disp('Maximum number of iterations exceeded.');
    disp('Use option IV for best estimate.');
    SingularValues=[]
  else
    SingularValues = S_B_cv(1:k_org)
  end
end

% Output option II: Set singular values equal to output vector.
if (nargout == 1)
  if iter >= maxit
    disp('Maximum number of iterations exceeded.');
    disp('Use option IV for best estimate.');
    varargout{1}=[]
  else
    varargout{1} = S_B_cv;
  end
end
% Output option III and IV: Output singular triplets (U,S,V)
if nargout > 1
    if iter >= maxit && nargout == 3
        disp('Maximum number of iterations exceeded.');
        disp('Use option IV for best estimate.');
        varargout{1} = [];
        varargout{2} = [];
        varargout{3} = [];
    else
        if interchange
            varargout{1} = V*V_B_cv(:,1:k_org);
            varargout{2} = diag(S_B_cv(1:k_org));
            varargout{3} = W*U_B_cv(:,1:k_org);
        else
            varargout{1} = W*U_B_cv(:,1:k_org);
            varargout{2} = diag(S_B_cv(1:k_org));
            varargout{3} = V*V_B_cv(:,1:k_org);
        end
    end
% Output options IV: Output singular triplets (U,S,V) and STATS
if nargout == 4
    STATS.numMatProds = mprod;
    STATS.timeMatProds = timeMatProds;
    STATS.numIterRefRestart = IterRefRestart;
    STATS.timeCompIterRef = timeIterRef;
    STATS.timeReorth = timeReorth;
    STATS.timeTotal = toc(tStart);
    STATS.estimateSVmax = Smax;
    STATS.estimateSVmin = Smin;
    if iter >= maxit
        STATS.convergedKVals = 'FALSE';
    else
        STATS.convergedKVals = 'TRUE';
    end
if out_val == 1
    STATS.outputestRitzorIterRef = 'RITZ SING.';
    STATS.maxnormres = max(res_norm_rz);
elseif out_val == 2
    STATS.outputestRitzorIterRef = 'HARM SING.';
    STATS.maxnormres = max(res_norm.hm);
elseif out_val == 3
    if iterrefined

STATS.outputestRitzorIterRef='ITER. REF. SING';
else
    STATS.outputestRitzorIterRef='REF. SING';
end
STATS.maxnormres = max(res_norm_it);
elseif out_val == -1
    STATS.outputestRitzorIterRef='ERROR';
    STATS.maxnormres = 'ERROR';
end
varargout{4}=STATS;
end

%-----------------%
% END: OUTPUT RESULTS %
%-----------------%
end

%------------------------------------------------%
% BEGIN: LANCZOS BIDIAGONALIZATION DECOMPOSITION %
%------------------------------------------------%
function [V,W,B,mprod,timeMatProds,timeReorth] = ablanzbd(A,V,W,F,B,K,interchange,
m_b,n,m,mprod,Ritz,SVTol,reorth,iter,timeMatProds,timeReorth)
% Computes the Lanczos bidiagonalization decomposition
% A*V = W*B
% A'*W = V*B' + F*E^T
% with full reorthogonalization.
% If the m x n matrix A is a function, 'Afunc',
% then the structure must be y = Afunc(x,m,n,'transpose'). If transpose = 'F',
% then y = A*x. If transpose = 'T', then y = A'*x.
% James Baglama
% DATE: 6/22/21

% Initialization of main loop count J.
J = 1;
% Normalize starting vector.
if iter == 1
    V(:,1) = V(:,1)/norm(V(:,1)); B=[];
else
    if Ritz, J = K+1; else, J = 2; end
end

% Matrix A product with vector(s) V, (W=A*V).
Astart = tic;
if interchange
    if ischar(A)
        W(:,J) = feval(A,V(:,J),m,n,'T');
    else
        W(:,J) = (V(:,J)'*A)';
    end
else
    if ischar(A)
        W(:,J) = feval(A,V(:,J),m,n,'F');
    else
        W(:,J) = A*V(:,J);
    end
end
timeMatProds = timeMatProds + toc(Astart);

% Count the number of matrix vector products.
mprod = mprod + 1;

% Input vectors are singular vectors and AV(:,J) which must be orthogonalized.
if iter ~= 1
    W(:,J) = orthog(W(:,J),W(:,1:J-1));
    W(:,J) = orthog(W(:,J),W(:,1:J-1));
end

% Compute the norm of W.
S = norm(W(:,J));
% Check for linearly dependent vectors.
if S <= SVTol
    W(:,J) = randn(size(W,1),1);
    W(:,J) = orthog(W(:,J),W(:,1:J-1));
    W(:,J) = W(:,J)/norm(W(:,J));
    S = 0;
else
    W(:,J) = W(:,J)/S;
end

% Begin of main iteration loop for the block Lanczos bidiagonalization decomposition.
while (J <= m_b)
    % Matrix A' product with vector(s), (F = A'*W).
    Astart = tic;
    if interchange
        if ischar(A)
            F = feval(A,W(:,J),m,n,'F');
        else
            F = A*W(:,J);
        end
    else
        if ischar(A)
            F = feval(A,W(:,J),m,n,'T');
        else
            F = (W(:,J)'*A)';
        end
    end
    timeMatProds = timeMatProds + toc(Astart);
    mprod = mprod + 1;

    % One step of the Gram-Schmidt process.
    F = F - V(:,J)*S;

    % Second step to maintain strong local orthogonality
    S2 = F'*V(:,J); F = F - V(:,J)*S2; S = S + S2;

    % Always perform full reorthogonalization step of "Short vectors".
ORstart = tic;
if J > 1, F = orthog(F,V(:,1:J-1)); end
timeReorth = timeReorth + toc(ORstart);

if J+1 <= m_b

% Compute the norm of F.
R = norm(F);

% Check for linearly dependent vectors.
if R <= SVTol
    F = randn(size(V,1),1);
    F = orthog(F,V(:,1:J));
    V(:,J+1) = F/norm(F);
    R = 0;
else
    V(:,J+1) = F/R;
end

% Compute bidiagonal matrix B.
if isempty(B)
    B = [S R];
else
    B = [B zeros(J-1,1); zeros(1,J-1) S R];
end

% Matrix A product with vector(s), (W=A*V).
Astart = tic;
if interchange
    if ischar(A)
        W(:,J+1) = feval(A,V(:,J+1),m,n,'T');
    else
        W(:,J+1) = (V(:,J+1)'*A)';
    end
else
    if ischar(A)
        W(:,J+1) = feval(A,V(:,J+1),m,n,'F');
    else
        W(:,J+1) = A*V(:,J+1);
    end
end
% Count the number of matrix vector products.
mprod = mprod + 1;

% One step of the Gram-Schmidt process.
W(:,J+1) = W(:,J+1) - W(:,J)*R;

% Second step for local orthogonality
R2 = W(:,J+1)'*W(:,J); W(:,J+1) = W(:,J+1) - W(:,J)*R2; R = R + R2;

% Full Reorthogonalization step. "Long vectors"
ORstart = tic;
if ( iter == 1 || strcmp(reorth,'TWO') )
    if J > 1
        W(:,J+1) = orthog(W(:,J+1),W(:,1:J-1));
    end
end
timeReorth = timeReorth + toc(ORstart);

% Compute the norm of W.
S = norm(W(:,J+1));

% Check for linearly dependent vectors.
if S <= SVTol
    W(:,J+1) = randn(size(W,1),1);
    W(:,J+1) = orthog(W(:,J+1),W(:,1:J));
    W(:,J+1) = W(:,J+1)/norm(W(:,J+1));
    S = 0;
else
    W(:,J+1) = W(:,J+1)/S;
end

else

% Add last element to matrix B
B = [B; zeros(1,J-1) S];

end
% Update iteration count.
J = J + 1;
end
end

function y = orthog(y,X)
% Simple re-orthogonalization of vector y against columns of matrix X.
for i=1:size(X,2)
dotY = y'*X(:,i);
y = y - X(:,i)*dotY;
end
end

%----------------------------------------------%
% END: LANCZOS BIDIAGONALIZATION DECOMPOSITION %
%----------------------------------------------%

%--------------------------%
% BEGIN: CONVERGENCE TESTS %
%--------------------------%

function [conv_rz_tol,conv_hm_tol,conv_it_tol,conv_rz_sqrt,conv_hm_sqrt,...
conv_it_sqrt,V_B_cv,U_B_cv,S_B_cv,res_norm_rz,res_norm_hm,res_norm_it,
out_val] = ...
convtests(B,Bsz,Fnorm,tol,k,U_B_rz,S_B_rz,V_B_rz,U_B_hm,...
S_B_hm,V_B_hm,U_B_it,S_B_it,V_B_it,Smax,sqrteps)
% This function checks the convergence of singular triplets

% Convergence check for all input values. %
% %

% Use the Lanczos bidiagonalization decomposition relationship %
% % A V = W B
% % A' T W = V E' T + F E' T


163
% \| A v - s u \|^{2} = \| A V v - s W u \|^{2} = \| W B v - s W u \|^{2} \\
% = \| V B^T u + F E^T u - s V v \|^{2} \\
% \| A^T u - s v \|^{2} = \| V B^T u + F E^T u - s V v \|^{2} \\
% \| B^T u - s v \|^{2} + \| F E^T u \|^{2} \\
% RITZ \Rightarrow B v = s u \text{ and } B^T u = s v \text{ and} \\
% \| A v - s u \| = 0 \\
% \| A^T u - s v \| = \| E^T u \| \| F \| \)

% Initialize output values.
conv_rz_tol = 0; conv_hm_tol=0; conv_it_tol=0;
conv_rz_sqrt = 0; conv_hm_sqrt=0; conv_it_sqrt=0;
V_B_cv=[]; U_B_cv=[]; S_B_cv=[];
res_norm_hm = ones(k,1); res_norm_it = ones(k,1);

% Compute the residual for Ritz values
res_norm_rz = (abs(U_B_rz(Bsz,1:k))*Fnorm)';
for i=1:k
  if res_norm_rz(i,1) < tol*Smax, conv_rz_tol = conv_rz_tol+1; end
  if res_norm_rz(i,1) < sqrteps*Smax, conv_rz_sqrt = conv_rz_sqrt+1; end
end

% Compute the residual for Harmonic Ritz values.
if ~isempty(S_B_hm)
  for i=1:k
    res_norm_1(i,1) = norm(B*V_B_hm(:,i) - S_B_hm(i)*U_B_hm(:,i))^2;
    res_norm_2(i,1) = norm(B'*U_B_hm(:,i) - S_B_hm(i)*V_B_hm(:,i))^2 + (U_B_hm(Bsz,i)*Fnorm)^2;
    res_norm_hm(i,1) = sqrt(res_norm_1(i,1)+res_norm_2(i,1));
  end
end

% James Baglama
% DATE: 3/30/21
if res_norm_hm(i,1) < tol*Smax, conv_hm_tol = conv_hm_tol+1; end
if res_norm_hm(i,1) < sqrt(eps)*Smax, conv_hm_sqrt = conv_hm_sqrt+1; end
end
end

% Compute the residual for Iterative Ritz values.
if ~isempty(S_B_it)
    for i=1:k
        res_norm_1(i,1) = norm(B*V_B_it(:,i) - S_B_it(i)*U_B_it(:,i))^2;
        res_norm_2(i,1) = norm(B'*U_B_it(:,i) - S_B_it(i)*V_B_it(:,i))^2 + (U_B_it(Bsz,i) * Fnorm)^2;
        res_norm_it(i,1) = sqrt(res_norm_1(i,1)+res_norm_2(i,1));
        if res_norm_it(i,1) < tol*Smax, conv_it_tol = conv_it_tol+1; end
        if res_norm_it(i,1) < sqrt(eps)*Smax, conv_it_sqrt = conv_it_sqrt+1; end
    end
end

% Output values.
[~,I] = max([conv_rz_tol conv_hm_tol conv_it_tol]);
if I == 1, out_val = 1; U_B_cv = U_B_rz; V_B_cv = V_B_rz; S_B_cv = S_B_rz; end
if I == 2, out_val = 2; U_B_cv = U_B_hm; V_B_cv = V_B_hm; S_B_cv = S_B_hm; end
if I == 3, out_val = 3; U_B_cv = U_B_it; V_B_cv = V_B_it; S_B_cv = S_B_it; end

%------------------------%
% END: CONVERGENCE TESTS %
%------------------------%

%---------------------------------------%
% BEGIN: COMPUTE REFINED RITZ ITERATION %
%---------------------------------------%

function [u,rho,v_min,rconv] = refined_ritz(D_ritz,R,alpha,B,maxitref,Bsz,k,method)
% Computes the Iterative refined Ritz values and vectors.

% INPUT:
% D_RITZ - (K x 1) vector of approx. singular values.
% R - real number - norm of residual vector F.
% ALPHA - real number - last diagonal element of B -> B(Bsz,Bsz).
% B - Bsz x Bsz bidiagonal matrix
% MAXITREF - Integer indicating the maximum number of iterations for the iterative
% set to 1 and the refined Ritz are computed - used in convergence testing.
% BSZ - Integer indicates the size of the tridiagonal matrix.
% K - number of Iterative refined Ritz values/vectors to compute
% METHOD - NOR - Equivalent eigenvalue system
% AP_m = Q_mB_m and A'Q_m = P_m B_m' + f_me_m'
% Equivalent eigenvalue system
% A'AP_m = P_m B_m'B_m + B_m(m,m)f_me_m'
% Compute iterative refined Ritz on [B'*B; 0 B_m(m,m)*norm(f)]
% AUG - Equivalent eigenvalue system
% [0 A; A' 0][Q_m 0; 0 P_m] = [Q_m 0; 0 P_m][0 B; B' 0]+[ 0 0; f_me_m' 0]
% Compute iterative refined Ritz on [0 B; B' 0; norm(f) 0]
% OUTPUT:
% U - (BSZ x K) matrix of left singular vectors:
% METHOD
% NOR => U = B*V_min; U = U/norm(U);
% AUG => U = V_min(1:Bsz/2,Bsz); U = U/norm(U);
% RHO - (K x 1) vector of iterative refined Ritz values.
% V_MIN - (BSZ x K) matrix of right singular values
% METHOD
% AUG => V_MIN = V_MIN(Bsz/2+1:Bsz,Bsz); V_MIN = V_MIN/norm(V_MIN);
% RCONV - Integer indicate if all K iterative refined Ritz converge.
% - 0 not all converged - do not use
% - K all converged
% DATE MODIFIED: 5/10/2021
% VER: 1.0

% Set up the augmented matrix [B'*B; 0 alpha*R]
if strcmp(method,'NOR')
B_aug = zeros(Bsz+1,Bsz); B_aug(1:Bsz,1:Bsz) = B'*B;
B_aug(Bsz+1,Bsz) = alpha*R;

166
% Initialize rho value.
rho = D_ritz.^2;

else
    % Set up the augmented matrix [0 B; B' 0; R 0]
    B_aug = [zeros(Bsz) B; B' zeros(Bsz); zeros(1,2*Bsz)];
    B_aug(2*Bsz+1,Bsz) = R;

    % Initialize rho value.
    rho = D_ritz;
end

% Initialize values.
s_min = zeros(k,1); % Intialize min. singular values of B_aug.
v_min = zeros(Bsz,k); % Intialize right singular vectors of B_aug.
u = zeros(Bsz,k); % Intialize left singular vectors of B_aug.
rconv = zeros(k,1); % Intialize convergence.
rho_0 = rho; % Used to check for convergence during iterations.
sqrteps = sqrt(eps); % square root of machine precision - eps
v_zero = zeros(Bsz,1); % Set starting vector to zero.

% Change size for augmented system.
if strcmp(method,'AUG'), Bsz = 2*Bsz; end

% Compute k number of iterative refined Ritz values/vectors.
% Compute in reverse order. Less likely k, k-1, .. converge - avoids
% unneeded iteration. Exit if any iterative refined Ritz fail to converge.
for j = k:-1:1
    % Set difference in rho values to test for stagnation.
    diff_rho_0 = -1; v_min_0 = v_zero;

    % Iteration to compute the iterative refined Ritz values/vectors
    for i=1:maxitref
        % Compute the SVD of B_aug - rho* I
        [U,S,V] = svd((B_aug-rho(j)*eye(Bsz+1,Bsz)),0);
% Need the smallest singular triplet of B_{aug} - \rho* I. Matlab
% returns order of singular values largest to smallest.
if strcmp(method,'NOR')
    s_min(j) = S(Bsz,Bsz); v_min(:,j) = V(:,Bsz);
    % Compute the new \rho = v_min'*B'*B*v_min = \|B*v_min\|^2
    rho(j) = norm(B*v_min(:,j))^2; diff1 = 0;
else
    s_min(j) = S(Bsz,Bsz);
    v_min(:,j) = V(Bsz/2+1:Bsz,Bsz); v_min(:,j) = v_min(:,j)/norm(v_min(:,j));
    % Compute the new \rho = V(:,Bsz)'*[0 B; B' 0]*V(:,Bsz)
    rho(j) = V(:,Bsz)'*B_{aug}(1:Bsz,:)’*V(:,Bsz);
    % check norm singular vectors are close to 1/sqrt(2)
    diff1 = abs(norm(V(Bsz/2+1:Bsz,Bsz)) - 1/sqrt(2));
end

% Compute the difference of previous \rho to check for convergence
den = max([rho_0(j),rho(j), eps]);
diff_rho = abs( (rho_0(j) - rho(j))/den );
% Check for convergence
if (((((diff_rho < eps && abs(V(:,Bsz)'*U(1:Bsz,Bsz)) < sqrteps)...) || s_min(j) < eps || norm(abs(v_min_0) - abs(v_min(:,j))) < eps) && diff1 < sqrteps) || maxitref == 1)
    rconv(j) = 1;
    if strcmp(method,'NOR')
        u(:,j) = B*v_min(:,j); u(:,j) = u(:,j)/norm(u(:,j));
        rho(j) = sqrt(rho(j));
    else
        u(:,j) = V(1:Bsz/2,Bsz);
        u(:,j) = u(:,j)/norm(u(:,j));
    end
end
% Check for stagnation to avoid too many unnecessary iterations. Care
% must be taken to avoid the situation where rho(k) is still changing
% very slightly at first and then an increase in change later. An
% early termination due to stagnation with no convergence
% may avoid increase in change later that converges.
if abs(diff_rho_0 - diff_rho)/max(diff_rho,eps) < eps, break; end % stagnation
if i>= 10 && mod(i,10)==0 % update every 10 iterations to avoid early termination.
    diff_rho_0 = diff_rho;
end

% Update rho_0 and v_min_0 to test convergence.
    rho_0(j) = rho(j); v_min_0 = v_min(:,j);
end

if rconv(j) == 0, rconv = 0; return; end

end

% After K iteration finish check to see if *all* K iterative refined Ritz values
% have all converged. Reset rconv to return an integer value, rconv = k all
% converged and rconv = 0, not all converged.
if all(rconv == 1), rconv = k; else rconv = 0; end

end

%-------------------------------------%
% END: COMPUTE REFINED RITZ ITERATION %
%-------------------------------------%