Flexible And Robust Iterative Methods For The Partial Singular Value Decomposition

Steven Goldenberg
William & Mary - Arts & Sciences, sgoldenberg@wm.edu

Flexible and Robust Iterative Methods for the Partial Singular Value Decomposition

Steven Goldenberg

Fairfax, Virginia

Master of Science, William & Mary, 2017
Bachelor of Arts, Johns Hopkins University, 2014
Bachelor of Music, Peabody Institute, 2014

A Dissertation presented to the Graduate Faculty of
The College of William & Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

College of William & Mary
August 2022

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

[illegible]

Steven Goldenberg

Approved by the Committee, July 2022

[illegible]

Committee Chair
Andreas Stathopoulos, Professor, Computer Science
College of William & Mary

Weizhen Mao

Weizhen Mao, Professor, Computer Science
College of William & Mary

[illegible]

Robert Michael Lewis, Associate Professor, Computer Science
College of William & Mary

Zhenming Liu

Zhenming Liu, Research Scholar, Computer Science
College of William & Mary

Ron Morgan

Ronald Morgan, Professor, Mathematics
Baylor University

ABSTRACT

The Singular Value Decomposition (SVD) is one of the most fundamental matrix factorizations in linear algebra. As a generalization of the eigenvalue decomposition, the SVD is essential for a wide variety of fields including statistics, signal and image processing, chemistry, quantum physics and even weather prediction. The methods for numerically computing the SVD mostly fall under three main categories: full, partial, and streaming. Full SVD methods focus on solving the problem in its entirety, making them suitable for smaller dense matrices where the computation cost is tractable. On the other end of the spectrum, streaming methods provide an "on-line" algorithm that computes an approximate SVD on large datasets by analyzing small chunks of the data at a time, which limits their overall accuracy. Partial SVD solvers fill in the large gap between these two extremes by providing accurate solutions for a subset of singular values on large (often sparse) matrices.

In this dissertation, we focus on the development of fast, flexible, and robust partial SVD solvers. We first introduce a novel solver, GKD, based on the Golub-Kahan and Davidson methods and demonstrate its performance and ability to produce accurate solutions on difficult problems through comparisons with the PRIMME software package. Then, we investigate the use of flexible stopping criteria for GKD and other SVD solvers that are tailored to specific applications. Finally, we analyze the effect of SVD stopping criteria on matrix completion algorithms.

In total, this work has enhanced the landscape of large scale SVD solvers by providing a novel, efficient SVD algorithm, as well as the implementation of a flexible framework for stopping criteria that can be adapted to study and solve a diverse set of problems.

TABLE OF CONTENTS

Acknowledgments iv

Dedication v

List of Tables vi

List of Figures viii

1 Introduction 2

1.1 Overview 2

1.1.1 Motivation 4

2 The Golub-Kahan Davidson Method 7

2.1 Introduction 7

2.1.1 Related Work 10

2.2 Main Contribution 13

2.2.1 Algorithm 13

2.2.2 Restarting and Locking 16

2.2.3 Resetting 18

2.2.4 Inner Solver 21

2.3 Benefits over PHSVDS 26

2.3.1 Avoiding the Augmented Problem 26

2.3.2 Switching Problems 27

2.3.3 Space and Time Comparisons 27

2.4 Numerical Results 28
2.4.1 Unpreconditioned Results 30
2.4.2 Single Precision Results 32
2.4.3 Preconditioned Results 34
2.5 Chapter Summary 36

3 Low-Rank Stopping Criteria for the SVD 37
3.1 Introduction 38
3.2 Background and Motivation 39
3.2.1 Common LRA Criteria 41
3.2.2 Issues with Current SVD Methods 42
3.3 Proposed Solution 44
3.3.1 Guidelines for Standard LBD 47
3.3.2 Guidelines for Subspace Iteration 48
3.3.3 Specific Solutions Required for GKD 49
3.3.4 GKD MATLAB Implementation 52
3.4 Stopping Criteria 52
3.4.1 Frobenius Norm Criteria 56
3.4.2 Spectral Norm Criteria 60
3.4.3 Singular Gap Finding 65
3.5 Matrix Completion 67
3.6 Hard-Impute Testing 70
3.7 Soft-Impute Testing 71
3.8 Chapter Summary 74

4 GKD Software Implementation 75
4.1 Targeting and Stopping Functions 78

5 Conclusion 80

5.1 Future Research Directions 81

Bibliography 83

ACKNOWLEDGMENTS

My journey in pursuit of a Ph.D. has been a long and winding road full of new experiences, fruitful work, and dozens of wonderful people and memories. I would like to take this opportunity to thank all those that have given me guidance, encouragement and joy during my time here at William and Mary.

First, I would like to thank my doctoral advisor, Dr. Andreas Stathopoulos. He has known me from the beginning of this journey and has left an indelible mark on my life and work. This thesis is the culmination of the countless emails and meetings we shared. Without him, I would not be the researcher I am today.

A special thanks goes to my dissertation defense committee: Dr. Weizhen Mao, Dr. Robert Michael Lewis, Dr. Zhenming Liu, and Dr. Ronald Morgan. Thank you for giving your time, support, and informed critique.

I am incredibly grateful to William and Mary for providing a welcoming environment in which I could grow as a student and a person. In particular, I would like to thank the administrators of the Computer Science Department: Vanessa Godwin, Dales Hayes, and Jacqulyn Johnson. They have always been happy to talk with me about anything and provide help whenever I needed it.

Of course, I cannot forget the friends and mentors I have met along the way, especially Eloy Romero, Xiaodan Zhu, and Heather Switzer. Thank you for all the time you have spent with me talking about research or eating our favorite foods. I cannot express how grateful I am to know you all.

Last but certainly not least, I would like to thank my family for always supporting me and my studies. Their patience and love gave me the strength to persevere when I needed it most.

To my parents, Michael Goldenberg and Dr. Susan Hasselquist

LIST OF TABLES

2.1 Basic Properties of Square Matrices 29

2.2 Basic Properties of Rectangular Matrices 29

2.3 Basic Properties of Large Scale Matrices 29

3.1 Set of Symbols 40

3.2 Example subset of `solverdata` variables in GKD and their descriptions. Variable names come directly from GKD and may not match their use in this paper. 46

3.3 Matrices chosen from SuiteSparse 51

3.4 Searching for all singular values above the threshold $\delta_2 = \max(\frac{\boldsymbol{\sigma}_{100}+\boldsymbol{\sigma}_{101}}{2\boldsymbol{\sigma}_1}, 1e-4)$ for 12 SuiteSparse matrices. For each matrix, we note the optimal $\boldsymbol{k}$, the threshold, the first iteration where k_{max} is an upper bound, the first iteration where k_{min} is optimal, the maximum ρ (relative to $\|A\|$) that will return at least $k_{max} > \boldsymbol{k}$, and the first iteration that the least-squares regression has a slope of 0. 64

3.5 Searching for an additive gap larger than δ_G, which is chosen to be the geometric mean of the two largest gaps. The columns $k_{max} = \boldsymbol{k}$ and Criteria Stop give iteration counts. The criteria stops when all $r_i/\sigma_{k_{max}} < 10^{-6}$ with $i \in 1 : k_{max}+1$. We testednopss_11k, however GKD stops early before finding all 4 multiplicities, so no values are given for $k_{max} = \boldsymbol{k}$ or $\max(r/\sigma_{k_{max}})$. 68

4.1 Output variables in GKD 76

4.2 Optional Name/Value pairs for GKD 77

LIST OF FIGURES

2.1 Demonstrating the need for resetting on lshp3025 ($\|A\| = 7$) with GKD ($q = 35$, $s = 15$, δ =1E-14, and $k = 1$). 21

2.2 Convergence of GKD and GKJD when there are more SVs below $\sqrt{\epsilon_{mach}}$ than the MaxBasisSize ($q = 35$, $s = 15$). 24

2.3 Convergence of GKJD on a problem with 20 SVs below $\sqrt{\epsilon_{mach}}$ in single precision with varying minimum restart sizes. (Maximum Matvecs = 75,000, $q = 50$) 24

2.4 Convergence of PHSVDS on a poorly conditioned problem ($\kappa(A)$ = 1E+13) 26

2.5 Stagnations caused by a failure to fully converge in the first stage of PHSVDS (κ =1.1E+4) 26

2.6 Unpreconditioned Results on 12 problems from the SuiteSparse Matrix Collection with a relative user tolerance of $\delta = 1e - 14$. 30

2.7 Large-Scale Unpreconditioned Results. Required matvecs for GKD, GD+k, GKJD and JDQMR are shown in the table. Note that for sls, GKJD finds 3 of the singular values with multiplicity 14 while JDQMR finds only 2. 31

2.8 Similar performance can be achieved with a relatively small basis size even when searching for 100 values. 32

2.9 IRLBA wastes matrix vector multiplications building a full basis without checking convergence. 32

2.10 Maximum accuracy achievable with GKD in single precision for the 10 largest SVs on Rucci1. We set $\delta = 0$ and rotate the target at each iteration to allow all triplets to converge. 34

2.11 Preconditioned Results with an ILU Preconditioner for finding the smallest and 10 smallest singular triplets. 35

2.12 Large-Scale Results with Block Jacobi Preconditioner (block size=600 on A^TA) for the 5 smallest singular triplets. Required matvecs for GKD,GD+k, GKJD and JDQMR are shown in the table. 36

3.1 Empirical Frobenius norm convergence compared to theoretical bounds from [49, Algorithm 1/2]. The constants $c = [0.0096, 0.001]$ for SI and BGKD respectively are chosen to be as small as possible. 45

3.2 Empirical spectral norm convergence compared to theoretical bounds from [49, Algorithm 1/2]. The constants $c = [0.253, 0.0241]$ for SI and BGKD respectively are chosen to be as small as possible. 45

3.3 Ratio of matrix vector products required by GKD using standard targeting (R) and largest residual targeting (L) to achieve a residual tolerance of 1e-6 on 100 singular triplets. The ratio ($R/L - 1$) for GKJD on c-42 is 2.32. 49

3.4 Geometric mean of the ratio between the real and approximated residual norms ($\|\boldsymbol{r}_i\|/\|r_i\|$) over the first min($d$, 100) singular triplets and all iterations using GKD and GKJD to find 100 singular values with standard (suffix -R) and largest residual targeting (suffix -L). 49

3.5 Maximum multiplicative error of ϵ using RSVD on SNAP/Amazon0302. 54

3.6 Maximum multiplicative error of ϵ using RSVD on SNAP/email-ENRON. 54

3.7 Convergence of GKD error estimates compared to $\boldsymbol{\epsilon}$ for the 1st singular value on the matrix A = diag(1000:-1:1). 56
3.8 Convergence of GKD error estimates compared to $\boldsymbol{\epsilon}$ for the 10th singular value on the matrix CurlCurl_0. 56
3.9 Comparing the optimal rank k (100) and optimal $\|A_{100}\|_F$ with the returned rank and corresponding maximum achievable $\|A_k\|_F$ when stopping GKD with the standard Frobenius norm criteria. The maximum achievable $\|A_k\|_F$ is derived from the true singular values of A. 59
3.10 Comparing estimations of k_{min} with the optimal $\boldsymbol{k}$ using GKD for the Frobenius norm criterion on a Gaussian Kernel matrix. ϵ for (3.21) is calculated by (3.17) with either exact residuals or approximate residuals from (3.12). 59
3.11 Convergence of k_{max} and k_{min} on CurlCurl_0 using GKD where the threshold is given in Table 3.4. k_{max} is calculated using accurate residual norms and (3.17). 62
3.12 Convergence of the first $k_{max} + 1$ residual norms using GKD on CurlCurl_0. δ_2 matches the threshold used for Figure 3.11. 62
3.13 Using Least-Squares Regression to track the rate of increase in k_{min}. 63
3.14 Comparing various SVD tolerances (surface plot) with our specialized stopping criteria (line plot). (Left) Results with a very flat spectrum $\Sigma = \text{linspace}(0.99, 1, 30)$ for the low rank space. (Right) Results with a slightly decaying spectrum $\Sigma = \text{linspace}(1, 50, 30)$ for the low rank space. 70
3.15 Comparing the number of singular values needed at each iteration for varying thresholds 72

3.16 Comparing the distance between two successive Soft Impute steps with varying thresholds 72

3.17 Comparing the Frobenius norm distance to the full (dense) low rank matrix A after 1000 iterations for 20 different thresholds 73

3.18 Comparing the number of singular values needed at each iteration for varying thresholds with heuristic 3.31 73

Flexible and Robust Iterative Methods for the Partial Singular Value Decomposition

Chapter 1

Introduction

1.1 Overview

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

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

where $\boldsymbol{U} \in \Re^{m,n}$ and $\boldsymbol{V} \in \Re^{n,n}$ are orthonormal bases and $\Sigma = \text{diag}(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n) \in \Re^{n,n}$ with $\boldsymbol{\sigma}_1 \leq \boldsymbol{\sigma}_2 \leq \cdots \leq \boldsymbol{\sigma}_n$ is a diagonal matrix containing the singular values of A. The singular triplets of A are defined as $(\boldsymbol{u}_i, \boldsymbol{\sigma}_i, \boldsymbol{v}_i)$, where bold face differentiates from approximate vectors and values in this thesis. Given the approximate singular triplet (u_i, σ_i, v_i), we have the left and right singular value residuals, defined as $r_u = A^T u_i - \sigma_i v_i$ and $r_v = Av_i - \sigma_i u_i$.

The SVD is crucial to a variety of fields including statistics for principal component analysis [36], computer science for image compression [57] and web search clustering [53], and genomics for expression data processing [3]. More specifically, finding the smallest singular triplets is useful for total least squares problems, the determination of the effective rank of a matrix [21], and variance reduction of inverse operators [18]. When searching for the largest singular values, the SVD is frequently used to provide a low-rank approximation such that $\|A - A_k\|$ is minimized for some norm and rank k. In fact, without additional constraints, the SVD is known to provide the best such rank k approximation for all Schatten p-norms including the 2-norm $\|\cdot\|_2$ (spectral norm), Frobenius norm $\|\cdot\|_F$ and trace norm $\|\cdot\|_*$.

$$\|A\|_2 = \sigma_1, \qquad \|A\|_F = \left(\sum_{i=1}^{min(m,n)} \sigma_i^2\right)^{1/2}, \qquad \|A\|_* = \sum_{i=1}^{min(m,n)} \sigma_i. \tag{1.2}$$

In this thesis unless otherwise specified, the 2-norm can be assumed.

Frequently, the SVD is framed within the context of eigenvalue decomposition on the symmetric matrices A^TA, AA^T (normal equations) and on the augmented matrix

$$B = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}. \tag{1.3}$$

This is due to the following relations given $A = U\Sigma V^T$: $A^TA = V\Sigma^2V^T$, $AA^T = U\Sigma^2U^T$, and $Q^TBQ = \text{diag}(\sigma_n, \ldots, \sigma_1, -\sigma_n, \ldots, -\sigma_1, 0, \ldots, 0)$ with

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} V & V & 0 \\ U_1 & -U_1 & \sqrt{2}U_2 \end{bmatrix} \text{ and } U = \begin{bmatrix} U_1 & U_2 \end{bmatrix}. \tag{1.4}$$

These observations have lead to a number of algorithms derived from previous work on the well-studied symmetric eigenvalue problem [55]. However, the computation strategies for SVD vary significantly based on the problem size and structure of the matrix A.

For smaller dense problems, a direct solver is typically used. Direct solvers generally utilize an implicit version of the symmetric QR algorithm for eigenvalues on A^TA. This involves an initial bidiagonalization step where the matrix A is reduced to an upper bidiagonal matrix through Householder reflections (or possibly Givens' rotations). Without considering the remaining reduction to diagonal form, this process requires O(mn^2) time, making it prohibitively expensive for large matrices.

When the matrix A is large enough, it can be inefficient to compute the SVD with direct methods. Furthermore, the matrix often can be relatively inexpensive to apply as a matrix-vector multiplication (MatVec), even when the dimensions are large. This can occur when the matrix is highly sparse, or when the matrix action on a vector can be represented as an inexpensive function. In some applications, it is possible to highly sparsify the matrix to reduce the MatVec cost without perturbing the required spectrum significantly [1].

These considerations have led to the use of iterative algorithms like Golub-Kahan-Lanczos (GKL) also known as Lanczos bidiagonalization [20]. While these methods generally only provide a portion of the singular triplets, many applications often require only a few of the largest or smallest singular values and vectors.

When the solution requires many iterations, it may be infeasible to store all the GKL vectors necessary to maintain stability through orthogonalization. To solve this, restarted versions of GKL that limit the maximum basis size, such as IRLBA [6], have been developed. Additionally, other methods have emerged, such as Jacobi-Davidson (JDSVD) [29], the Preconditioned Hybrid SVD method (PHSVDS) [73], and the Preconditioned Locally Minimal Residual method (PLMR_SVD) [70]. These methods can use the more advanced +k (also known as locally optimal) restarting and can take advantage of preconditioning, which can provide significant speedups for difficult problems.

In general without preconditioning or +k restarting, these methods build Krylov spaces on the normal equations matrix $C = A^T A$ or on the augmented matrix B in (1.3). We denote a k-dimensional Krylov space on a square matrix A with initial vector v_1 by $K_k(A, v_1) = span\{v_1, Av_1, \ldots, A^{k-1}v_1\}$.

1.1.1 Motivation

Methods on $A^T A$ generally converge more quickly to the smallest singular triplets compared to methods on the augmented matrix B. This can be attributed to the highly interior nature of the smallest eigenvalues of B. However, methods on $A^T A$ generally exhibit poor conditioning as their condition number is κ^2, i.e., the square of the condition number of A,

$\kappa = \sigma_n/\sigma_1$. In this dissertation, we present a new algorithm, the Golub-Kahan Davidson method (GKD), which removes this trade-off. We show that GKD converges with the speed of methods on $A^T A$ without their accuracy limitations.

We achieve this novel improvement through two key steps. First, we enforce the relation $AV = UR$, where the left space, U, is built through the QR decomposition of AV. Second, we update the right search space V with left residuals $A^T U - \Sigma V$. These residuals can be preconditioned like the Davidson method for eigenvalue problems, which is crucial for fast convergence on poorly conditioned matrices. We show in comparisons with the PRIMME [64] software package that our novel GKD algorithm consistently requires fewer iterations when searching for the smallest singular values, while it is equally efficient when searching for the largest.

While computing the smallest singular values presents significant numerical difficulties, finding the largest values is critical for a large number of applications. In particular, the low rank approximation of a matrix A requires the computation of many of its largest singular values. In our second work, we expand GKD with a focus on novel stopping criteria that target the needs of low rank approximations. GKD and most other iterative methods for SVD use the standard residual stopping criterion

$$\begin{aligned} \|r_v\| &= \|Av - \sigma u\| < \delta \\ \|r_u\| &= \|A^T u - \sigma v\| < \delta \end{aligned} \tag{1.5}$$

with a user specified δ. While the residual criterion is useful in many situations, it may require significantly more computation than necessary for applications that only require a subspace that delivers a low rank matrix close to A in some norm. We analyze multiple alternative stopping criteria including novel criteria for matrix completion algorithms.

During our investigation of matrix completion algorithms, we found a few interesting methods that were iterative in nature and relied on one SVD computation per iteration. This includes the Singular Value Thresholding algorithm [9], the Soft Impute algorithm

[44], and the Accelerated Proximal Gradient algorithm (APGL) [67]. All three of these algorithms use a heuristic to determine the number of singular values to calculate at each iteration, but the authors claim this heuristic is only necessary because PROPACK [39] does not provide an appropriate stopping criteria.

We implemented these suggested stopping criteria in GKD and developed also an alternative stopping criterion based on the actual functional that the matrix completion algorithms seek to minimize. We have made two key observations. One, we show that the thresholding criteria proposed by other authors performs signficantly worse than the heuristic implementations they implement. This leads to the conclusion that these relatively unstudied heuristics play a far larger role in convergence than the authors suggest. Secondly, we found that our stopping criterion based on checking the distance to the true matrix (instead of residuals) could significantly improve performance. Surprisingly, this criterion frequently chose to return low accuracy answers to the SVD problem, but produced good answers for the matrix completion problem. We intend on investigating these phenomena further to provide additional theory or insights that can lead to algorithmic improvements.

Chapter 2

The Golub-Kahan Davidson Method

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

2.1 Introduction

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

$$A = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{2.1}$$

where $\boldsymbol{U} \in \Re^{m,n}$ and $\boldsymbol{V} \in \Re^{n,n}$ are orthonormal bases and $\Sigma = diag(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n) \in \Re^{n,n}$ with $\boldsymbol{\sigma}_1 \leq \boldsymbol{\sigma}_2 \leq \cdots \leq \boldsymbol{\sigma}_n$ is a diagonal matrix containing the singular values of A. The singular triplets of A are defined as $(\boldsymbol{u}_i, \boldsymbol{\sigma}_i, \boldsymbol{v}_i)$, where bold face differentiates from search space vectors and values in this paper. Given the approximate singular triplet (u_i, σ_i, v_i), we have the left and right singular value residuals, defined as $r_u = A^T u_i - \sigma_i v_i$ and $r_v = Av_i - \sigma_i u_i$.

This decomposition has become increasingly important and is frequently used in fields like statistics for principal component analysis [36], computer science for image compression [57] and web search clustering [53], and genomics for expression data processing [3]. More specifically, finding the smallest singular triplets is useful for total least squares problems, the determination of the effective rank of a matrix [21], and variance reduction of inverse operators [18].

Additionally, finding high accuracy solutions is crucial when running in a single or low precision environment. In single precision, matrix multiplication can only provide 1.2E-7$\|A\|$ of accuracy, and in practice this bound is optimistic for iterative solvers due to accumulated error. Despite this limitation, single-precision calculations have become increasingly important for deep learning applications [26] which are often resistant to errors and therefore require less than full double precision. Reducing the precision of matrix vector multiplications can provide speed ups on CPUs due to increased vectorization, and GPUs can obtain speedups of 2x-4x [76]. In addition, using single precision cuts the storage requirements in half. Specifically, the use of single precision calculations is encouraged by Advanced Micro Devices (AMD) for OpenCL applications [2], and half precision, which can only provide 1E-3$\|A\|$ digits of accuracy, has been growing in popularity on NVIDIA's GPUs [43].

When the matrix A is large enough, it can be inefficient to compute the SVD with dense methods. Furthermore, applications often require only a few of the largest or smallest singular values and vectors. These considerations have led to the use of iterative algorithms like Golub-Kahan-Lanczos (GKL) also known as Lanczos bidiagonalization [20]. However, when the solution requires many iterations, it may be infeasible to store all the GKL vectors necessary for full or partial reorthogonalization. To solve this, restarted versions of GKL that limit the maximum basis size, such as IRLBA [6], have been developed. Additionally, other methods have emerged, such as Jacobi-Davidson (JDSVD) [29], the Preconditioned Hybrid SVD method (PHSVDS) [73], and the Preconditioned Locally Minimal Residual method (PLMR_SVD) [70]. These methods can use the more advanced +k (also known as locally optimal) restarting and can take advantage of preconditioning, which can provide significant speedups for difficult problems.

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

$$B = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}. \tag{2.2}$$

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

Frequently, methods that build their search space with B, like JDSVD and PLMR_SVD, are able to achieve accuracy of $\|r_B\| < O(\|A\|\epsilon_{mach})$ when searching for the smallest singular triplets, where ϵ_{mach} is the working machine precision and $r_B = [r_u; r_v]$ is the eigenvalue residual on B. However, B has singular values $\pm\boldsymbol{\sigma}_i$ [55], so searching for the smallest singular triplets is a highly interior eigenvalue problem that can converge slowly. Worse, when A is rectangular, the spectrum of B contains $m - n$ zero eigenvalues that are not in the spectrum of A. Therefore, methods on B are unable to distinguish real zero singular values of A within the spectrum when $m \neq n$.

Alternatively, methods that build $K_k(C, v_1)$ explicitly are only able to achieve accuracy $O(\|C\|\epsilon_{mach}) = O(\|A\|^2\epsilon_{mach})$ for the eigenvalue residual on C, r_C. If $u = Av/\sigma$ or $r_v = 0$, then r_C is equivalent in exact arithmetic to a scaling of r_u,

$$r_C = A^T Av - \sigma^2 v = \sigma(A^T u - \sigma v) = \sigma r_u. \tag{2.3}$$

When these methods compute the smallest singular value, and if $\sigma \neq 0$, it is easy to see that (2.3) limits the accuracy with respect to the r_u residual to $O(\|A\|\kappa(A)\epsilon_{mach})$, where $\kappa(A) = \frac{\sigma_n}{\sigma_1}$ is the condition number of A.

Despite the squaring of the spectrum, these methods usually converge faster than methods on B, both in theory and in practice, due to the extremal problem they solve. Furthermore, these methods are often able to find real zero singular values of A, as the corresponding eigenproblem on C does not introduce extraneous zero eigenvalues.

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

2.1.1 Related Work

GKL [39] builds two vector bases, one for the right space $K_k(A^TA, v_1)$ and one for the left space $K_k(AA^T, Av_1)$. It builds the second basis while computing the first one without additional matrix vector multiplications (matvecs). More importantly, it avoids directly multiplying vectors with A^TA and thus avoids the numerical problems associated with working on C. This is done by keeping two orthogonal spaces, U and V, where the last vector of V, v_k, is used to expand U as $u_k = Av_k$ and the last vector of U, u_k, is used to expand V as $v_{k+1} = A^Tu_k$. These new vectors are orthonormalized to the previous ones in their corresponding bases and the coefficients from this process are used to create the bidiagonal projection matrix U^TAV. GKL solves the smaller singular value problem on this projection matrix to approximate the singular triplets.

While GKL is considered to be one of the most accurate and effective algorithms for finding small singular triplets, the standard version is unrestarted and cannot be preconditioned. Therefore, GKL tends to be computationally slow for poorly separated triplets of large matrices. Many restarted versions have been developed [7, 6, 35] but use primarily implicit or thick restarting [71] and thus are unable to maintain the convergence of the unrestarted method. Locally optimal (also known as +k) restarting uses vectors from successive iterations in a way similar to a non-linear conjugate gradient and has been shown to converge similarly to an unrestarted method for both eigenvalue [38, 61, 63] and singular value problems [73].

SVDIFP [42] implements an inner-outer method where the inner one builds a preconditioned Krylov space $K_k(M(C-\rho_i I), x_i)$, where M is a preconditioner for C and (x_i, ρ_i) is the approximate right singular vector and value at the i-th step of the outer iteration. SVDIFP is able to avoid numerical problems, at least for the right singular vectors, by using a two sided projection similarly to GKL. SVDIFP's structure, however, does not allow for many of the optimization techniques of Davidson-type methods which can significantly improve convergence [73].

JDSVD [29] works on B by using two independent subspaces rather than one. It is an inner outer method that expands both spaces by solving a Jacobi-Davidson type correction equation on B. Without preconditioning, restarting, or solving the correction equation, the JDSVD outer method builds subspaces that span the following Krylov spaces:

$$U_k = K_{\frac{k}{2}}(AA^T, u_1) \oplus K_{\frac{k}{2}}(AA^T, Av_1), \quad V_k = K_{\frac{k}{2}}(A^TA, v_1) \oplus K_{\frac{k}{2}}(A^TA, A^Tu_1). \tag{2.4}$$

These spaces are similar to the ones used in GKL, but crucially, each space is the sum of two different spaces of half dimension. This allows JDSVD to take advantage of initial guesses for both the left and right singular vectors. However, it also means that the outer solver in JDSVD requires twice as many matvecs to build a space of equal Krylov dimension. Furthermore, if we choose initial vectors that satisfy $v_1 = A^Tu_1$, the outer iteration of JDSVD becomes wasteful as it builds the same space as a GKL with half the dimension (in this case the spaces $K_{\frac{k}{2}}(A^TA, v_1)$ and $K_{\frac{k}{2}}(A^TA, A^Tu_1)$ in (2.4) differ only by one vector). This is also true of eigensolvers on B as seen below,

$$B^2 \begin{bmatrix} v \\ Av \end{bmatrix} = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}^2 \begin{bmatrix} v \\ Av \end{bmatrix} = \begin{bmatrix} A^T Av \\ AA^T(Av) \end{bmatrix}. \tag{2.5}$$

Despite a slower outer iteration, the inner correction equation used in JDSVD is essential to its performance as it often allows for faster convergence than eigenvalue methods on B while maintaining the ability to converge to full accuracy. However, it can still suffer from the same issues as other eigenmethods on B.

PHSVDS [73] exploits the different advantages of eigenmethods on B and C by utilizing each in a two-stage method. The first stage can use any state-of-the-art eigensolver on C, which gives it fast convergence until either the user tolerance is met or until switching to a second stage using an eigensolver on B is necessary to reach the remaining user tolerance. Switching to an eigensolver on B after a fully converged first stage can effectively utilize good initial guesses from the first stage on C, and thus PHSVDS can avoid resolving the entire accuracy on an indefinite problem. Its implementation in PRIMME can use any of the two near-optimal eigensolvers, GD+k or JDQMR. This two-stage approach has been shown to be faster than eigensolvers on B alone, and typically has better performance than other SVD methods.

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

PLMR_SVD [70] is a recent method based on a stationary iteration that uses two separate four-term recurrences to build the following spaces,

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

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

2.2 Main Contribution

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

2.2.1 Algorithm

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

and the expansion of the spaces avoid a direct multiplication with C, we avoid the squaring of the norm and condition number that occurs with eigensolvers on C.

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

$$\begin{aligned} Av - \sigma u &\perp \mathcal{Q}, \\ A^T u - \sigma v &\perp \mathcal{V}. \end{aligned} \tag{2.6}$$

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

$$\begin{aligned} Q^T AVy = \sigma Q^T Qx &\Rightarrow Ry = \sigma x \\ V^T A^T Qx = \sigma V^T Vy &\Rightarrow R^T x = \sigma y. \end{aligned} \tag{2.7}$$

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

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

We expand the left space Q with Av_{i+1} instead of a preconditioned right residual. This differentiates the method from JDSVD with the goal of producing a faster converging outer method. Specifically, from (2.3) the left residual r_u is colinear with the residual r_C of the Generalized Davidson (GD) method [46] on the matrix C, which is also colinear with the new GKL direction for V. In addition, the Rayleigh-Ritz on C used by GD gives the same answer as (2.7),

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

so, in exact arithmetic, GKD is equivalent to GD solving the eigenproblem on $A^T A$. Without preconditioning or restarting, it is also equivalent to GKL and thus it is twice as fast as JDSVD if the latter is used only as an outer method. By construction, GKD has similar numerical properties as GKL, whereas the accuracy of GD is limited by working directly on $A^T A$. GKD can also be used with thick and +k restarting, which in exact arithmetic makes it equivalent to GD+k on C, the first stage method of PHSVDS, but without the numerical limitations. Algorithm 1 shows the restarted and preconditioned version of GKD when seeking one singular triplet. Although the orthogonalization of step 13 can be avoided without preconditioning [59], it is needed for high accuracy and allows our more general method to use flexible preconditioning. Furthermore, the algorithm can be extended to find more than one singular triplet by using soft or hard locking. A block version is similarly possible.

Algorithm 1 GKD Iteration

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

2.2.2 Restarting and Locking

Our restart procedure takes the current best approximations to the s singular triplets closest to the user specified target, τ, and uses them together with those from the +k restarting to compress V, Q and R down to dimension $s + k$. The steps for building the restarted V follow closely the description in [63] and are shown in lines 1-7 of Algorithm 2.

Algorithm 2 Restart Procedure

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

The simplest method to restart Q and R, without recomputing the QR factorization of the restarted AVt, is to set them as $Q\tilde{Q}$ and $\tilde{R}$ respectively, where $\tilde{Q}\tilde{R} = Rt$ is the QR factorization of Rt with $t = [Y_1, v_{new}]$ from line 6 of Algorithm 2. This can introduce numerical error of magnitude $O(\|R\|\epsilon_{mach})$, which can be as large as $O(\|A\|\epsilon_{mach})$. Although this error is acceptable for a single QR factorization, the error accumulates over many restarts causing the factorization not to correspond to the actual AV and eventually causing loss of convergence. It is possible to intelligently compute Q and R to avoid direct multiplications with R through the already available SVD of R as seen below,

$$\begin{aligned} AVt = QRt &= Q \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} \Sigma_r^{(1)} & 0 \\ 0 & \Sigma_r^{(2)} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & Y_2^T v_{old} \end{bmatrix} \\ &= Q \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_r^{(2)} Y_2^T v_{old} \end{bmatrix}. \end{aligned} \tag{2.8}$$

From (2.8), the new Q and R can be obtained with minimal effort by performing a QR factorization on $\Sigma_r^{(2)} Y_2^T v_{old} = \tilde{Q}\tilde{R}$. The restarted Q and R are given in Line 12 of Algorithm 2. This strategy has better numerical behavior because we separate the space of small singular values that are kept in thick restarting (X_1) from the +k restarting space which has correction directions over the entire singular space (including those of large magnitude). By explicitly decoupling $\Sigma_r^{(1)}$ and $\tilde{R}$ in R, any errors in $\tilde{R}$ do not affect the ability of the algorithm to compute the smallest eigenvectors and they only affect the correction directions. Moreover, as the +k algorithm typically uses only $k = 1$ previous vectors, no errors are expected.

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

It is known that hard locking can cause stagnation in some rare cases or when the number of locked vectors is large. This is caused by the error still present in the locked vectors, which may contain critical directions for other singular triplets [62]. We have not seen any matrices in this paper that exhibit this behavior. However, soft-locking can provide left and right singular vectors that are orthogonal to machine precision, while hard-locking only obtains left singular vectors orthogonal up to $O(\|A\|\delta)$. Therefore, we present only soft-locking results in this paper. We intend to address the issues with hard-locking more thoroughly in the future.

2.2.3 Resetting

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

$$r_v = Av - \sigma u = AVy - Q(\sigma x) = AVy - QRy = (AV - QR)y = 0. \tag{2.9}$$

Generally, this means we can avoid the extra matrix-vector multiplication (or storage for AV) necessary to compute r_v. In practice though, $\|r_v\|$ cannot be better than $O(\|A\|\epsilon_{mach})$ due to the multiplication of AV when computing the left space. Worse, $\|r_v\|$ grows as $O(\sqrt{\text{numRestarts}}\|A\|\epsilon_{mach})$, which has also been noticed in [72]. Therefore, our method must calculate $\|r_v\|$ explicitly when $\|r_u\| < \|A\|\delta$, where δ is the user selected tolerance. This ensures we meet the convergence criteria of Algorithm 1.

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

Additionally, we have noticed in our experiments that resetting can cause an increase in the residual norm up to $\|A\|\kappa(A)\epsilon_{mach}$, which may require a few iterations to reduce back to its previous level. This can be seen by analyzing a reset of the method when $V = \boldsymbol{v_1}$, the exact singular vector. In this situation, after resetting the new $\tilde{q} = (A\boldsymbol{v_1} + \mathsf{e})/\|A\boldsymbol{v_1} + \mathsf{e}\|$ contains an error due to the matrix vector multiplication, with $\|\mathsf{e}\| \leq \|A\|\epsilon_{mach}$. When $\kappa(A)$ is not too large, the error term e in the nominator will be amplified by a factor of $1/(\boldsymbol{\sigma_1} - \|\mathsf{e}\|) \approx 1/\boldsymbol{\sigma_1}$ at most. Thus, we expect the error in $\tilde{q}$ to be at most $\kappa(A)\epsilon_{mach}$. This means that the norm of the left residual norm, which we compute explicitly, can increase up to $\|A\|\kappa(A)\epsilon_{mach}$. In order to track the errors mentioned above, we have devised two inexpensive criteria that help to avoid unnecessary resets.

Since the error in the orthogonality of V may cause convergence issues, we must estimate how large $\|E\| = \|V^TV - I\|$ can be before it needs correction. To do so, we analyze the Galerkin condition on the equivalent eigenproblem on C, i.e. $A^TA\boldsymbol{v} = \boldsymbol{\sigma^2 v}$. Applying the Galerkin condition with V, the projected eigenproblem should have been $V^TA^TAV\tilde{y} = \tilde{\sigma}^2V^TV\tilde{y}$. However, our algorithm solves $V^TA^TAVy = \sigma^2 y$ regardless of the orthonormality of V. Therefore, we obtain a Ritz vector $v = Vy$ and Ritz value σ^2 that will not converge to a zero residual. The Ritz pair produced by our inexact Galerkin can be considered as a Ritz pair of an exact Galerkin condition applied to the nearby generalized eigenproblem $A^TA\boldsymbol{\tilde{v}} = \boldsymbol{\lambda}M\boldsymbol{\tilde{v}}$ where $M = V(V^TV)^{-2}V^T$. This can be seen by using the Galerkin condition with V, which yields the same Ritz vector Vy and Ritz value $\lambda = \sigma^2$,

$$V^TA^TAVy = \lambda V^TMVy = \lambda V^TV(V^TV)^{-2}V^TVy = \lambda y. \qquad (2.10)$$

In order to correctly monitor and maintain convergence, the residual $r_C = \sigma r_u = A^TAv/\|v\| - \sigma^2 v/\|v\|$ should not drift too far from the exact residual of the generalized eigenproblem, $r_E = A^TAv/\|v\| - \sigma^2V(V^TV)^{-2}V^Tv/\|v\|$, where $\|v\| = \|Vy\|$ since Vy may not be unit length. However, from [65, 30], $\|Vy\| \geq \sigma_{min}(V) \geq \sqrt{1 - \|E\|}$ and $\|I - (V^TV)^{-1}\| \leq \|E\|/\|(1 - \|E\|)\|$. Therefore, assuming $\|E\| < 1$, we have

$$
\begin{aligned}
\|r_E - r_C\| &= \sigma^2 \left\| \frac{Vy}{\|Vy\|} - \frac{V(V^TV)^{-1}y}{\|Vy\|} \right\| \\
&\leq \frac{\sigma^2 \|V\| \|I - (V^TV)^{-1}\|}{\sqrt{1 - \|E\|}} \\
&\leq \sigma^2 \|V\| \frac{\|E\|}{(1 - \|E\|)^{3/2}} \\
&\leq \sigma^2 (1 + \|E\|) \left(\sum_{i=0}^{\infty} \|E\|^{i+1} (-1)^i \begin{pmatrix} -\frac{3}{2} \\ i \end{pmatrix} \right) \\
&\leq \sigma^2 (\|E\| + O(\|E\|^2)).
\end{aligned} \tag{2.11}
$$

Since we want $r_u = r_C/\sigma$ to converge to tolerance $\|A\|\delta$, we want the distance $\|r_E - r_C\| < \|A\|\delta\sigma$. Thus, from (2.11), we should perform a reset when $\|E\| \geq \|A\|\delta/\sigma$. In practice, the second criterion described below will be satisfied earlier than (2.11), and therefore $\|E\|$ does not need to be computed explicitly.

The accuracy of the QR factorization also directly impacts the convergence of r_u. From (2.9), we can estimate errors in the QR factorization directly from the norm of the right residual. We choose to reset when $\|r_u\| < 1.25\|r_v\|$, which includes a small 25% buffer that we have found is necessary to detect potential stagnation in a few experimental cases. Since $\|r_v\|$ grows at a rate of approximately $O(\sqrt{\text{numRestarts}}\|A\|\epsilon_{mach})$, explicit computation of r_v is unnecessary.

To demonstrate this problem, we ran lshp3025, a problem from the SuiteSparse Matrix Collection [13], which requires thousands of restarts before convergence. Properties of this problem can be found in Table 2.1. The criteria outlined in the previous paragraphs combine to avoid the stagnation seen in Figure 2.1. Due to the very low tolerance of 1E-14 $= 50 * \epsilon_{mach}$, approximately 2,500 restarts or 35,000 matvecs may cause the reset criteria to be met. It is clear our criteria is somewhat conservative, as resets occur approximately every 40,000 matvecs, even when the method is able to converge without it. However,

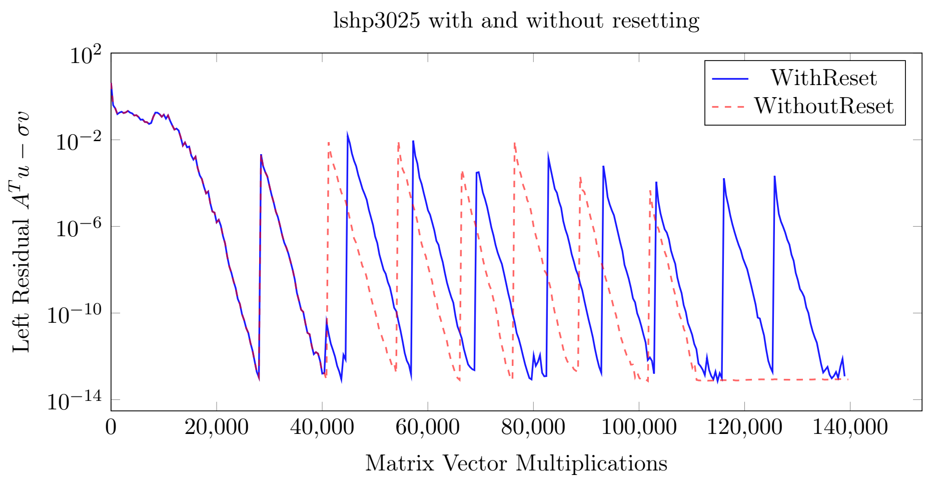


Figure 2.1: Demonstrating the need for resetting on lshp3025 ($\|A\| = 7$) with GKD ($q = 35$, $s = 15$, δ =1E-14, and $k = 1$).

without resetting, the method completely stagnates at around 110,000 matvecs. Moreover, with or without resets, we observe convergence to the first 8 smallest singular values in a similar number of matvecs (110,000), even though adding resets should increase the overall number of matvecs. This indicates the increased stability of the method also can improve performance slightly.

2.2.4 Inner Solver

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

GKD can be extended to a Jacobi-Davidson variant, GKJD, that expands the subspace

V by the approximate solution of the correction equation

$$(I - vv^T)(A^TA - \sigma^2 I)(I - vv^T)t = -r_u \tag{2.12}$$

instead of applying a preconditioner at line 10 of Algorithm 1. Here, and for the remainder of this section, σ without a subscript denotes the shift used for the inner solver, which may be different from the user specified target τ or the current approximate singular value. As before, σ_i will denote the ith singular value.

The inner equation can also utilize a preconditioner, improving convergence further. In particular, our inner solver is based on the symmetric Quasi-Minimal Residual method (QMRs) used in PRIMME's JDQMR. QMRs benefits from the ability to utilize indefinite preconditioners and solve indefinite systems which may occur when σ lies in the interior of the spectrum.

In order to avoid overutilizing the inner method when convergence is poor or the correction equation does not match the desired singular values, or underutilizing the inner method when convergence is good, extra steps must be taken. There is a significant volume of research on stopping criteria for inner iterations, including results for Jacobi-Davidson type methods for eigenvalue and SVD problems [31, 34, 33, 63, 52]. In this paper, we adopt the dynamic criteria used in PRIMME's JDQMR [63] which take advantage of the smooth convergence of QMRs to estimate the relevant eigenvalue residuals and stop the linear solve in a near-optimal way. Of course, other stopping criteria can also be implemented or an entirely different inner solver may be used.

The inner solver for (2.12) works directly on $A^TA - \sigma^2 I$ so its numerical stability needs to be justified. As with an outer iteration on A^TA, no numerical issues are expected when σ is in the largest part of the spectrum, but when seeking the smallest part, singular values below $O(\|A\|\sqrt{\epsilon_{mach}})$ will become indistinguishable when squared. However, the solution of the inner correction equation still provides useful directions even when a few singular values of A are below $O(\|A\|\sqrt{\epsilon_{mach}})$. The reason is well understood numerically and it is why inverse iteration works well despite a nearly singular linear system [55, sec. 4.3].

Assume there are k singular values below the noise level, i.e., $\sigma_k \leq \|A\|\sqrt{\epsilon_{mach}} < \sigma_{k+1}$, and a shift $\sigma \leq \|A\|\sqrt{\epsilon_{mach}}$. If we ignore the projectors for simplicity, the numerically computed solution of (2.12), $\tilde{t}$, satisfies

$$\tilde{t} = t + \boldsymbol{V}(\Sigma^2 - \sigma^2)^{-1}\boldsymbol{V}^T E\tilde{t}, \tag{2.13}$$

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

$$|c_i| = \frac{|\boldsymbol{v}_i^T E\tilde{t}|}{|\sigma_i^2 - \sigma^2|\|\tilde{t}\|} \leq \frac{\|A\|^2\epsilon_{mach}}{|\sigma_i^2 - \sigma^2|}. \tag{2.14}$$

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

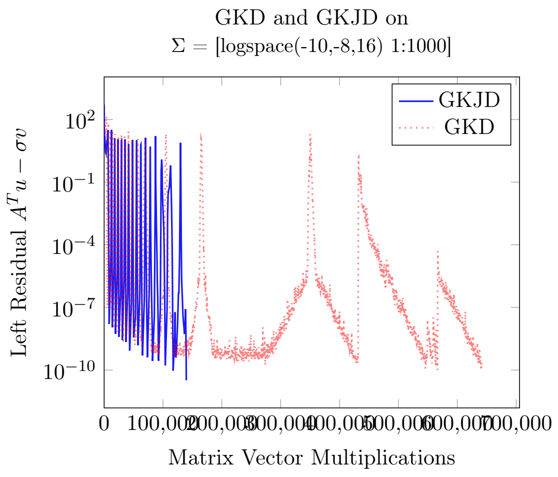
The behavior is similar when the backward error is at the level of the residual norm at which we solve (2.12), i.e., $\|E\| \leq \|A\|^2\theta$, for some tolerance θ. Typically we ask for a residual norm reduction relative to $\|r_u\|$ but this can be translated to a θ. Then, the $|c_i|$ in (2.14) have the same bounds as above only multiplied by θ/ϵ_{mach}. Since the approximate solution has $\|t\| = O(\theta)$, the effect of the noise error is larger.

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

k-dimensional space is sufficiently represented in the outer search space.

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

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



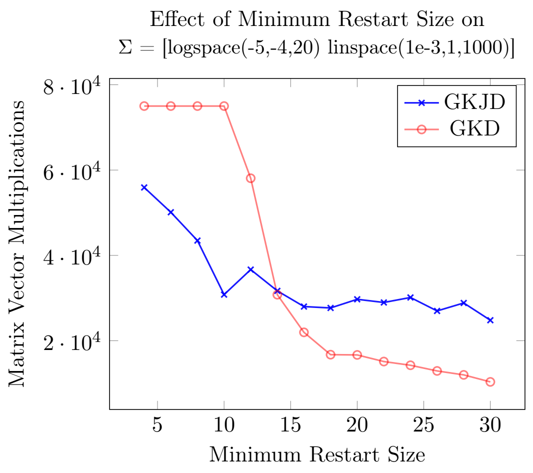


Figure 2.2: Convergence of GKD and GKJD when there are more SVs below $\sqrt{\epsilon_{mach}}$ than the MaxBasisSize ($q = 35$, $s = 15$).

Figure 2.3: Convergence of GKJD on a problem with 20 SVs below $\sqrt{\epsilon_{mach}}$ in single precision with varying minimum restart sizes. (Maximum Matvecs = 75,000, $q = 50$)

Figures 2.2 and 2.3 show examples of how GKJD with dynamic stopping conditions for the inner iteration can converge even when several singular values are below $\|A\|\sqrt{\epsilon_{mach}}$. They also show that GKJD is competitive and sometimes faster than GKD in terms of matrix-vector products, in addition to the benefit of a less expensive iteration. The matrices have a specified spectrum Σ and random left and right singular vectors.

In Figure 2.2 the matrix has 16 singular values below $\|A\|\sqrt{\epsilon_{mach}}$ but we limit GKD and GKJD to a restart size of only 15. Even with this limitation, GKJD is able to converge to the smallest singular triplet with a relative accuracy of 1E-14, and it does so three times faster than GKD. Additionally, with only a few extra outer iterations, GKJD can find 14 of the smallest singular values.

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

In Figure 2.3 we consider an example where the matrix has 20 singular values below the $\|A\|\sqrt{\epsilon_{mach}}$ threshold. We use single precision arithmetic, which allows for relatively larger spectral gaps that make convergence tractable. We search for the smallest singular value with a maximum basis size of 50, the dynamic inner stopping criteria, and a tolerance of 1E-5 for all tests while varying the restart size used by the GKD and GKJD. We see that smaller restart sizes do not impede convergence of GKJD and only slow it down by less than a factor of two. However, the effects of a small restart size are much more severe on GKD, which is unable to converge to the desired tolerance within 75,000 matvecs for restart sizes less than 10. This shows that GKJD is able to rebuild the space lost during restarting much more quickly than GKD, as the inner equation can sufficiently filter out directions corresponding to the unwanted portions of the spectrum.

2.3 Benefits over PHSVDS

2.3.1 Avoiding the Augmented Problem

As mentioned earlier, methods on B often exhibit problems due to the interior nature of the spectrum that they work on. In order to demonstrate these issues, Figure 2.4 shows convergence on the problem A = diag([1e-10, 2e-10, 5e-10, 1e-9, 3e-9, 1e-8, 1e-6, 1e-4, 1:1000]). First, this problem is very poorly conditioned ($\kappa(A)$ = 1E13) and since the 6 smallest singular values are below 1E-8, the first stage of PHSVDS is unable to distinguish them from zero. Second, because the spectrum is reflected across 0 for the augmented problem, it is very difficult to converge only to the positive part of the spectrum.

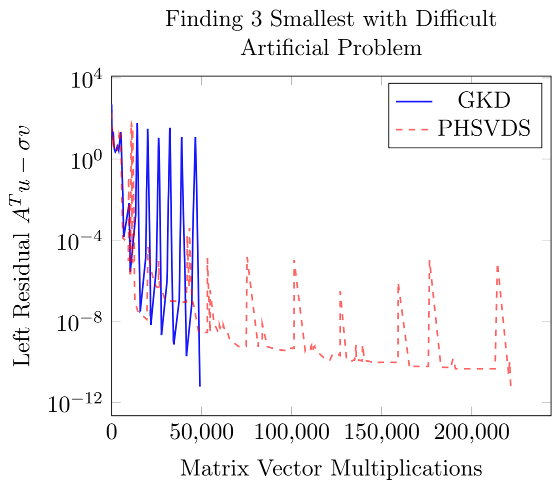


Figure 2.4: Convergence of PHSVDS on a poorly conditioned problem ($\kappa(A)$ = 1E+13)

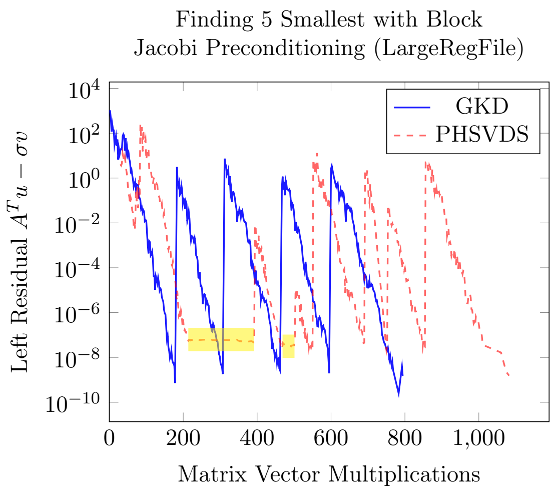


Figure 2.5: Stagnations caused by a failure to fully converge in the first stage of PHSVDS (κ =1.1E+4)

In searching for 3 singular values to a user tolerance of 1E-14, PHSVDS took more than 4 times more matvecs, but more importantly, it missed 5 smaller singular values as the third converged value was 1e-4. Even worse, the vectors that were returned for left and right spaces were not orthogonal, as $\|Q^TQ - I\| \approx \|V^TV - I\| \approx$ 6E-5. Therefore, the true residuals after orthogonalization did not meet the full user tolerance. Comparatively, GKD converges to all 6 of the smallest singular values and did so with fully orthogonal left and right vectors. As we can see from the figure, the convergence for GKD is fairly smooth, converging to each of the six singular values below 1E-8 before finishing. This is a vast improvement over the second stage of PHSVDS, which exhibits irregular convergence with large spikes in the left residual and long stagnations.

2.3.2 Switching Problems

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

Specifically, we found these switching issues to occur when testing PHSVDS on LargeRegFile (another matrix from the SuiteSparse Collection [13]) with Block Jacobi preconditioning and δ =1E-12. It is clear from the highlighted portions of Figure 2.5 that PHSVDS is unable to meet the convergence criteria for the first stage. In fact, while the case shown in Figure 2.5 is able to reach the criteria eventually, most cases like this stagnate completely. For example, the same problem (LargeRegFile) when solved with an inner solver (JDQMR) is never able to meet the first stage convergence criteria. Since GKD never requires switching methods, we can avoid these problems entirely and provide more reliable convergence.

2.3.3 Space and Time Comparisons

For computations on large matrices, it is important to consider the convergence rate, the space requirements, and the total work that the algorithm requires. Therefore, we provide a short comparison of the latter between our method and PHSVDS before presenting numerical results in Section 2.4.

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

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

2.4 Numerical Results

To verify our algorithm's performance, we utilized the same matrices given in the original PHSVDS publication [73] as well as three matrices with dimension larger than one million

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

Table 2.1: Basic Properties of Square Matrices

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

Table 2.2: Basic Properties of Rectangular Matrices

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

Table 2.3: Basic Properties of Large Scale Matrices

from [72]. These matrices are publicly available through the SuiteSparse Matrix Collection [13] and represent real world applications. These problems are quite difficult for iterative solvers and are used to stress test the capabilities of GKD and PHSVDS. Since these matrices are sparse, we provide their dimensions and the number of non-zero entries of A, $nnz(A)$, as well as the norm of A, $\|A\|$, the condition number of A, $\kappa(A)$, and the gap ratio for σ_1, $\gamma_1 = (\sigma_2 - \sigma_1)/(\sigma_n - \sigma_2)$.

The matrices listed in Table 2.1 and Table 2.2 are listed from least to most difficult (left to right) as generally their condition numbers increase, and the gap ratios for their smallest singular values decrease. It should be noted that none of these matrices are particularly poorly conditioned, and do not require the second stage in PHSVDS to improve the singular vector estimates more than a few orders of magnitude. Therefore, the benefits we would expect to gain on very poorly conditioned problems are significantly larger.

We restrict GKD and PRIMME's PHSVDS Matlab interface, primme_svds, to a maximum basis size of 35 vectors, a minimum restart size of 15 vectors and a user tolerance of δ = 1E-14 for the smaller matrices and δ = 1E-12 for the larger ones. We also enforce one retained vector from the previous iteration (for +1 restarting) except for the three large cases, where we enforce +2 restarting. Additionally, we choose to soft lock converged triplets, but due to the interior nature of the augmented method in primme_svds, we are unable to set soft-locking for the second stage while searching for the smallest singular triplets. It should be noted that hard-locking generally improves performance for our method when searching for more than one singular value, but does not provide the same orthogonality guarantees and is subject to the numerical issues mentioned earlier.

2.4.1 Unpreconditioned Results

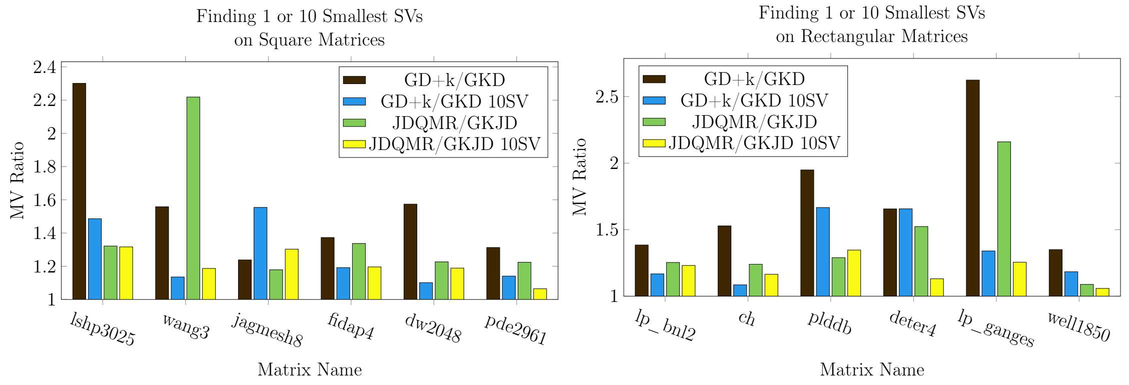
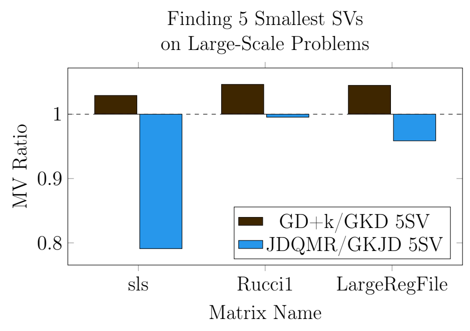


Figure 2.6: Unpreconditioned Results on 12 problems from the SuiteSparse Matrix Collection with a relative user tolerance of $\delta = 1e - 14$.

We compare GD+k (implemented as the default MIN_MATVECS method in primme_svds) against GKD, and the JDQMR method (MIN_TIME in primme_svds) against GKJD. As shown in Figure 2.6, GKD and GKJD require fewer matrix-vector multiplications than their primme_svds counterparts for all matrices. Also, the matrices that show the largest benefits are lshp3025, wang3, jagmesh8, and lp_ganges. As expected, these correspond to the matrices that required more significant use of the second stage in primme_svds, due to their larger $\kappa(A)$.



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

Figure 2.7: Large-Scale Unpreconditioned Results. Required matvecs for GKD, GD+k, GKJD and JDQMR are shown in the table. Note that for sls, GKJD finds 3 of the singular values with multiplicity 14 while JDQMR finds only 2.

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

For rectangular matrices, we also tested whether our method could find a true zero singular value by appending one extra column to the matrix equal to the first column. GKD is able to find the real zero in all cases. primme_svds will not return this numerically zero value, as outlined in its documentation, since its second stage has no way to distinguish real zeros from the null space created by the augmented matrix.

For the large scale matrices, Figure 2.7 shows fairly similar performance between primme_svds and GKD/GKJD. This is expected as the tolerance is higher (tol = 1E-12) than the small cases, and therefore primme_svds only uses the second stage sparingly. The biggest difference is seen for sls and for the inner-outer methods (JDQMR/GKJD), where the high multiplicity (14) at the second smallest singular value causes issues with convergence. Specifically, JDQMR only converges to two of these numerically equal singular values before finding five converged triplets, while GKJD is able to recognize the higher multiplicity and spends extra iterations finding a third. We also note that the number of matvecs for GKD/GKJD is significantly smaller than the numbers for SLEPc's implementation of LBD reported in [72].

In general, iterative methods may have trouble finding multiplicities or may converge out of order causing the methods to miss directions [45]. This is especially true for Krylov solvers which, in exact arithmetic, are unable to find more than one eigenvector corresponding to a multiplicity. In order to solve this problem, many algorithms, including PHSVDS, can utilize a block solver where the block size approximates the degree of the multiplicity [7, 5, 23]. Additionally, multiple initial guesses can be used to reduce the likelihood of initial vectors being deficient in the invariant space of the multiplicity. Both of these ideas would be simple extensions that could be added to GKD to improve robustness.

2.4.2 Single Precision Results

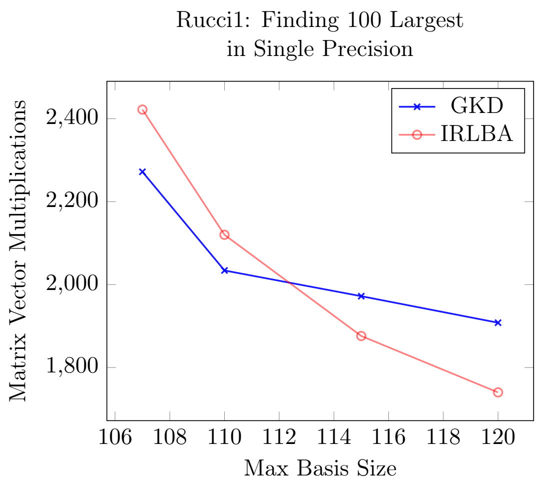


Figure 2.8: Similar performance can be achieved with a relatively small basis size even when searching for 100 values.

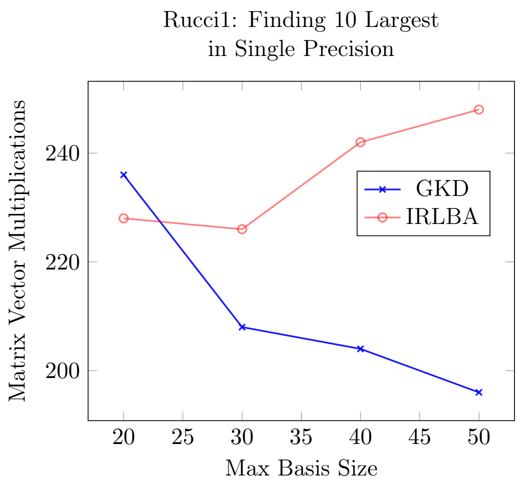


Figure 2.9: IRLBA wastes matrix vector multiplications building a full basis without checking convergence.

In order to demonstrate the versatility of our method, we ran tests in single precision looking for the largest 10 or 100 singular values of matrices to tolerance $\delta = 1\text{E-}4$. Additionally, our initial basis is built with a simple GKL iteration. Although much less taxing on the solver, these kinds of requirements are common in many SVD applications. We compare our results to IRLBA, which is the default method in MATLAB's `svds` for seeking the largest singular values. Since we are looking for low accuracy, we omit results from PRIMME since it would use only the first stage which is equivalent to GKD.

Figures 2.8 and 2.9 report results on Rucci1. We also ran these tests on sls and LargeRegFile, but convergence was achieved in too few iterations (requiring only one restart) so all methods were similar. We vary the maximum basis size to understand how GKD compares when the user has more or less space than IRLBA uses as a default. When searching for 100 singular triplets, we choose basis sizes close to 100 to mimic the situation where space is at a premium and only a small number of extra vectors can be stored. For 10 singular triplets, we show how IRLBA compares to GKD when the basis size is much larger than the number for desired triplets.

Figure 2.8 shows that both IRLBA and GKD provide fairly similar results for 100 singular values. GKD performs better in the most extreme memory limitation as it can selectively target the desired values when building its space. However, when there is more room to build a Krylov space, this targeting is no longer required.

Figure 2.9 shows increased advantages of GKD when fewer singular values are needed. For 10 singular values, the standard version of IRLBA defaults to a maximum basis size of 30. In some cases, the system may have additional space for a larger basis size which can improve convergence. However, since IRLBA generally only checks convergence after a full basis is built, a larger basis size can limit how often IRLBA performs these checks. This allows GKD to outperform IRLBA, even though they obtain nearly identical performance for smaller basis sizes.

To demonstrate the accuracy limit of our method, we run GKD searching for the 10 largest singular values of Rucci1 with a user tolerance of 0. Since this tolerance is not reachable, the original algorithm will stagnate on the largest singular triplet, before it has the chance to target more singular values. To allow all 10 singular triplets to improve, we rotate the target index from 1 to 10 at every iteration, and stop when all of the values begin to stagnate at their accuracy limit. Figure 2.10 shows the convergence of each residual to a stagnation at $\approx$ 2.5E-5. This is impressive given that $\|A\| \approx 7$, $\epsilon_{mach} = 1.2\text{E-}7$, and the matrix dimensions are 2 million by 110K, which affects the dot product accuracy. The associated error bound for a single iteration is $\|A\|\epsilon_{mach}\sqrt{m+n} \approx 1.2\text{E-}3$. Our algorithm goes well below this bound.

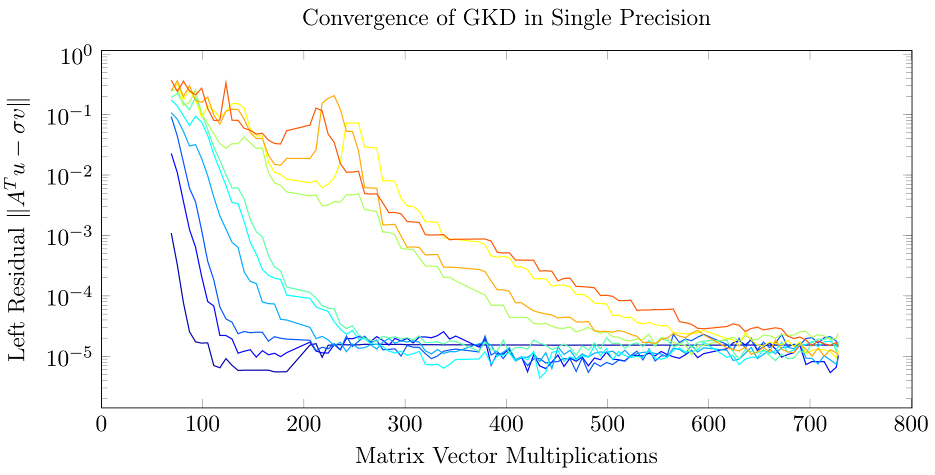
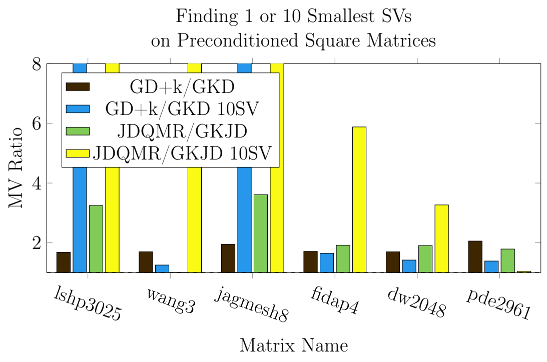


Figure 2.10: Maximum accuracy achievable with GKD in single precision for the 10 largest SVs on Rucci1. We set $\delta = 0$ and rotate the target at each iteration to allow all triplets to converge.

2.4.3 Preconditioned Results

In order to test the efficacy of preconditioning GKD, we ran tests on the six smaller square matrices using a preconditioner built from Matlab's ILU with the ilutp factorization, a drop-tolerance of 1E-3, and a pivot threshold of 1.0. Our results in Figure 2.11 show the significant benefit of an effective preconditioner, as all of the small problems required less than 150 matvecs when searching for one singular value with GKD. However, these preconditioners sometimes caused significant issues for primme_svds, as it was unable to converge for lshp3025 when searching for the 10 smallest singular values, and exhibited significant difficulty converging to 10 singular values for wang3, jagmesh8 and fidap4. Specifically, when searching for 10 singular values, wang3 requires 12x more matvecs for JDQMR, and jagmesh8 requires 56x and 14x more matvecs for GD+k and JDQMR respectively. These issues are caused by primme_svds' switching issues mentioned earlier.



MVs to Find 1 Smallest SV

	lshp3025	wang3	jagmesh8	fidap4	dw2048	pde2961
GKD	56	132	40	48	46	36
GD+k	94	224	78	82	78	74
GKJD	146	268	82	96	84	66
JDQMR	474	268	296	184	160	118

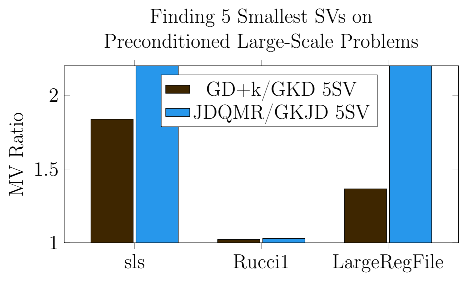
MVs to Find 10 Smallest SVs

	lshp3025	wang3	jagmesh8	fidap4	dw2048	pde2961
GKD	389	687	285	399	389	303
GD+k	DNF	860	16038	656	552	420
GKJD	1211	1393	721	1063	1027	779
JDQMR	DNF	17132	10656	6250	3354	808

Figure 2.11: Preconditioned Results with an ILU Preconditioner for finding the smallest and 10 smallest singular triplets.

For the three large rectangular matrices, we use a block-Jacobi preconditioner, inverting exactly diagonal blocks of $A^T A$ each of size 600. This is relatively inexpensive to compute and it is also parallelizable. Again, we see a significant decrease in matvecs as all three problems required less than 15% of the matvecs needed for the unpreconditioned cases. For Rucci1 the convergence differences between our methods and primme_svds are negligible, but for sls and LargeRegFile, GKD and GKJD provide significant improvements in speed and robustness. Again, as seen earlier in Figure 2.5, primme_svds' switching criteria are too stringent for preconditioned cases, which causes slowdowns for GD+k on LargeRegFile. Worse, primme_svds' JDQMR suffers stagnations that cause failures to converge when preconditioned on sls and LargeRegFile.

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



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

Figure 2.12: Large-Scale Results with Block Jacobi Preconditioner (block size=600 on A^TA) for the 5 smallest singular triplets. Required matvecs for GKD,GD+k, GKJD and JDQMR are shown in the table.

2.5 Chapter Summary

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

Chapter 3

Low-Rank Stopping Criteria for the SVD

The singular value decomposition (SVD) is one of the most commonly used low rank approximation techniques due to its optimality for all Schatten p-norms. To solve large scale SVD problems, robust, efficient and flexible iterative methods have been implemented in a variety of software packages. However, these typically require practitioners to determine an appropriate target rank and accuracy, often without prior spectral information. If practitioners desire SVD solutions with attributes that cannot be described with these two parameters, they must overestimate these parameters, leading to potentially significant inefficiencies.

We have developed a novel interface to iterative methods that allows users to directly implement new stopping criteria to meet their needs and it is simple enough to implement in a variety of solvers. We analyze a few examples of its use with newly developed robust criteria for four cases: low rank approximations with certain magnitude and accuracy in the Frobenius norm, in the 2-norm which is equivalent to singular value thresholding, the computation of a well separated space in terms of spectral gap, as well as a unique and purpose-built criterion for a real-world application. These model criteria are currently implemented in GKD.

3.1 Introduction

Low rank approximations have become increasingly relevant over the past few years as high dimensional data sets can be analysed in a variety of fields as low dimensional subspaces. This kind of analysis can be far more efficient than using the full data and more accurate than sketching or Monte-Carlo type methods. While integral to algorithms like principle component analysis (PCA), low rank approximations have also been used for a wide range of applications including facial recognition [68], image denoising [17], text information retrieval using latent semantic indexing (LSI) [15], and signal processing [58] including adaptive beamforming modeling [54]. Additionally, SVD is a critical component for applications in scientific computing including preconditioning [8], variance reduction [18] and model reduction [74], among others.

Given a $m \times n$ matrix A, the goal of the low rank approximation (LRA) problem is to find a rank constrained version of A, Z, that satisfies some additional constraint. This secondary constraint is frequently posed as the minimization of $\|A - Z\|$ for a particular norm and rank k. More generally, the second constraint can be viewed as a minimization over any cost function. Many of these functions, including the standard $\|A - Z\|$ minimization for a variety of norms, have optimal solutions that can be directly described with the truncated singular value decomposition (SVD) of A.

When the matrix is large and $k \ll \min(m, n)$, large scale iterative SVD solvers are far more efficient than their full SVD counterparts which require $O(mn^2)$ time. Highly optimized software for most state of the art such iterative solvers is readily available in [64, 39, 41, 6, 28, 32] These methods must implement criteria in order to determine when a given solution has converged sufficiently. However, the criteria implemented in these high quality software packages are based on the accuracy of the SVD solution, rather than the best LRA solution. Often it is not clear how the two types of accuracy relate. Additionally, these implementations require a priori knowledge of an appropriate k, which may be unknown.

If the rank or required SVD accuracy is unknown, we are currently unaware of any software solution that can guarantee an appropriate or optimal solution to the LRA problem without solving the SVD problem repeatedly and varying these parameters. Our aim in this paper is to show that these types of solutions are neither robust nor efficient. Therefore, we developed a simple but powerful interface to stop SVD methods directly based on LRA criteria which we describe in Section 3.3. Additionally, in Section 3.4 we describe the algorithms that are required to ensure good quality, robust solutions for a few common LRA criteria. These algorithms must be able to measure the LRA error with minimal overhead, which can be especially difficult in the early steps of an iterative method when the SVD solutions are inaccurate. Lastly, to show that this solution could be implemented in place of the standard criteria in current available software, we implement it within the Golub-Kahan-Davidson (GKD) solver [19] and test its efficacy on a variety of problems.

3.2 Background and Motivation

The singular value decomposition (SVD) of A is given by:

$$A = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}, \tag{3.1}$$

where $\boldsymbol{U} \in \mathbb{R}^{m,n}$ and $\boldsymbol{V} \in \mathbb{R}^{n,n}$ are orthonormal bases and $\boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n) \in \mathbb{R}^{n,n}$ with $\boldsymbol{\sigma}_1 \geq \boldsymbol{\sigma}_2 \geq \cdots \geq \boldsymbol{\sigma}_n \geq 0$ is a diagonal matrix containing the singular values of A. The singular triplets of A are defined as $(\boldsymbol{u}_i, \boldsymbol{\sigma}_i, \boldsymbol{v}_i)$. In this paper, the use of bold font denotes exact values. Given this definition and an approximate singular triplet (u, σ, v), the left and right residuals are defined as $r^u = A^T u - \sigma v$ and $r^v = Av - \sigma u$ respectively. For brevity, r without the superscript refers to the residual (r^u or r^v) with maximum 2-norm. Subscripts are used for indexing based on the values corresponding singular or Ritz value (e.g. r_i and σ_i) unless otherwise defined. For indexing matrices however, we subscript using standard MATLAB notation (e.g. $A_{(1:i,j)}$). As a reference, we provide **??**

for notation.

Name	Symbol	Example/Definition
Size of A	$m \times n$	
Rank of a solution	k	—
Block size	b	—
Current Basis Size	d	—
Iteration	(j)	$\sigma_i^{(j)}$
Index	i	—
Left Residual	r	$A^T u - \sigma v$
SVal Error	ϵ	$\boldsymbol{\sigma} = \sigma + \epsilon$
Spectral Threshold	δ_2	$\|A - Z\| < \delta_2 \|A\|$, (3.6)
Frobenius Threshold	δ_F	$\|A - Z\|_F < \delta_F \|A\|_F$, (3.7)
Residual Tolerance	ρ	$\|r_i\| < \rho \|A\| \quad \forall i$ in a given range (3.13)
Rank Bounds	k_{min}, k_{max}	$\sigma_{k_{min}} \geq \delta_2 \|A\| \geq \sigma_{k_{max}}$
Max Rank Overestimate	o	$k_{max} - k_{min} \leq o$
Gap Window Size	w	$\sigma_{i+w}/\sigma_i < \delta_G$, (3.26)
SVal Add Gap	γ_i	$\gamma_i = \min(\sigma_{i-1} - \sigma_i, \sigma_i - \sigma_{i+1})$
SVal Mult Gap	τ_i	$\sigma_{i+1}/\sigma_i = \tau$
Iteration Distance	ϕ	$\phi_i^{(j)} = \sigma_i^{(j)} - \sigma_i^{(j-1)}$
Maximum Allowed Rank	q	Interface Requirement (Subsection 3.4.3)

Table 3.1: Set of Symbols

Additionally, we define the singular value error at iteration $\boldsymbol{\epsilon}_i = \boldsymbol{\sigma}_i - \sigma_i$. Since the set of estimated singular values is often smaller than n, the index i for both $\boldsymbol{\sigma}$ and σ refers merely to the ith largest value of each set independently. Therefore, $\boldsymbol{\epsilon}$ is only defined over the set of estimated singular values, Σ. As a consequence of this definition, an accurate singular triplet with $\|r_i\| = 0$ can have $\boldsymbol{\epsilon}_i \neq 0$ when there exists a singular triplet in $\boldsymbol{\Sigma}$ with a larger singular value that is not included in Σ. These missing singular values are a common problem for iterative SVD solvers. We discuss choices for approximations to this error, ϵ_i, in Section 3.4.

A rank k approximation can be derived from the first k columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ corresponding to the k largest singular values, which we denote with $\boldsymbol{U}_k$, $\boldsymbol{V}_k$. The truncated SVD solution can be formulated in a number of ways as seen below:

$$\boldsymbol{A}_k = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^T = \boldsymbol{U}_k \boldsymbol{U}_k^T A = A \boldsymbol{V}_k \boldsymbol{V}_k^T. \tag{3.2}$$

The equality in (3.2) is only true for exact singular triplets. In Section 3.4, we show how these formulations can behave differently for inexact singular triplets. The importance of the SVD stems from its optimality property,

$$\min_{rank(Z)\leq k} \|A - Z\| = \|A - \boldsymbol{A}_k\| \tag{3.3}$$

under a variety of norms including all unitarily invariant norms [16, 22, 48]. This property allows us to define the minimum approximation errors for the spectral 2-norm and Frobenius norm with the singular values as follows:

$$\min_{rank(Z)\leq k} \|A - Z\|_2 = \boldsymbol{\sigma}_{k+1}, \tag{3.4}$$

$$\min_{rank(Z)\leq k} \|A - Z\|_F = \sqrt{\sum_{i=k+1}^{n} \boldsymbol{\sigma}_i^2}. \tag{3.5}$$

3.2.1 Common LRA Criteria

Two of the most common LRA criteria are based on the above two norms:

$$\|A - Z\| < \delta_2 \|A\|, \tag{3.6}$$

$$\|A - Z\|_F < \delta_F \|A\|_F. \tag{3.7}$$

with both $0 \leq \delta_{2/F} \leq 1$. Generally, we seek a Z with minimum rank. We focus on these relative norm criteria, however absolute versions are also frequently used.

Examples of these two criteria are easily found in the literature. For instance when denoising images, it may be desirable to have $\|A - Z\|_F = O(\mu)$ where μ is the standard deviation of the noise matrix [25]. A similar Frobenius norm criterion can be found when performing shot boundary detection [75] as well as denoising of magnetic resonance images [40]. On the other hand, the spectral norm criterion (3.6) may be preferred over (3.7) for big data computations where the sum of squared singular values may be dominated by a heavy tail of many small singular values [66].

Due to its minimum approximation error (3.4), the spectral norm criterion with exact SVD solutions is equivalent to a thresholding criterion on the singular values of A where all singular values above a given threshold are returned. This thresholding criterion is particularly useful for the nuclear norm regularization needed in applications like matrix completion [9]. It is important to note that when the SVD solution is inexact, the spectral norm criterion and thresholding may differ significantly as the thresholding problem is in essence a maximization problem on k, while the spectral criterion attempts to minimize k. In Section 3.4, we analyse this difference as well as others in more detail.

In some cases, practitioners may expect their problems to exhibit a specific low rank structure that is not captured purely through $\|A - Z\|$. For instance, in [74], the matrix is believed to have large additive gaps up to the noise threshold, and therefore the appropriate rank is defined by:

$$k = \max_i(\sigma_i - \sigma_{i+1} > \delta_G). \tag{3.8}$$

Another gap criteria can be seen in [67] where a large multiplicative gap is expected over a window of singular values with size w. The optimal rank for this problem is given by:

$$k = \min_i \left(\frac{\sum_{j=i+1}^{i+w+1} \sigma_j}{\sum_{j=i-w}^{i} \sigma_j} < \delta_G \right). \tag{3.9}$$

Often these criteria are paired with other restrictions as they do not impose any accuracy constraint and there is no guarantee that an appropriate rank k can be found within the entire spectrum of A.

3.2.2 Issues with Current SVD Methods

When the matrix is small, the SVD can be computed directly with any standard linear algebra package like LAPACK [4]. This will find all singular values of A, which can be truncated to the appropriate rank k as a post-processing step. To deal with larger matrices, many iterative algorithms have been developed to compute only the largest k singular values. When the number of values needed, $k \ll n$, these iterative SVD solvers are far more efficient.

There are a wide range of iterative methods designed for the truncated SVD, as well as iterative eigensolvers which can be adapted to produce accurate singular triplets. The number of these methods and the differences between them are far too many to provide an exhaustive list, however some of the most popular include Randomized SVD (RSVD) [27], Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) [38], Lanzcos Bidiagonalization (LBD) [46], Thick Restarted Lanczos [71], Generalized Davidson (GD) [64], Jacobi-Davidson (JDSVD) [29], and the Golub-Kahan Davidson method (GKD) [19].

In general, these methods require at least two parameters to stop the iteration: the number of desired singular values and an accuracy parameter. Frequently, accuracy is measured either directly through the SVD residuals (r^u or r^v) or indirectly through total iterations completed. The latter of these choices is particularly popular with block Krylov and subspace iteration methods like RSVD as there has been extensive work done to provide convergence bounds for both per vector and global LRA error [27, 49, 24].

For LRA, both of these parameters present significant challenges. To start, the rank required to satisfy any of the criteria previously described may be unknown. If this is the case, the optimal rank $\boldsymbol{k}$ may be approximated as shown in [47, 69, 27]. However, these estimates require a non-negligible investment in computation when performed separately from an SVD method. Rank estimators based on Krylov type methods would need to replicate the effort of the SVD method, and randomized estimators would need to perform extra matvecs in addition to being significantly less accurate. Therefore, it seems obvious to try and integrate dynamic rank estimation within the SVD methods themselves. Our solution for this is presented in Section 3.3.

The second parameter required by iterative SVD solvers to determine stopping accuracy also presents a significant challenge, even if $\boldsymbol{k}$ is known. Most common SVD accuracy parameters like per vector residual norms or iterations do not directly measure convergence to the desired criteria, and therefore may not provide optimal performance for common LRA criteria like (3.6) or (3.7). For instance, if the SVD method stops based on iterations, we must rely on theoretical bounds in order to guarantee our criteria has been met. However, we cannot use any bound that includes singular value gaps or any spectral information, since this information is rarely known a priori. This leaves bounds like the gap-free bounds presented in [49] which are limited to the simultaneous iteration and block Krylov methods and are known to be loose compared to empirical results. Figure 3.1 and Figure 3.2 show this "loose" behavior, especially for block Krylov methods. Additionally, the graphs show a best case scenario for these bounds as the smallest possible constant is chosen for the given norm, even though this constant would not be known in practice.

On the other hand, we know of almost no work that directly relates a per vector residual stopping criterion to LRA criteria like (3.7) or (3.6), even though this is the predominant way to stop an iterative SVD solver. Due to the issues present in these common ways of stopping by iterations or residual norms, we propose stopping the SVD method directly based on LRA criteria. We note that a few criteria of this nature have been developed for particular methods and LRA criteria, such as solving (3.7) using LBD [59], however there has been little work done to provide generic solutions that cover a variety of methods and criteria. We present our solutions to this issue in Section 3.4.

3.3 Proposed Solution

First, we address creating a solver that can handle an unknown rank constraint. The main programmatic issue is memory management. In general, it is not practical to allocate enough memory to handle rank n solutions, nor should we expect a low-rank problem to require a solution this large. Also, we should not expect the user to know the correct rank within some small multiplicative factor. Therefore, we have two options; we can either implement a dynamic memory management scheme to allow for slowly increasing rank estimates, or we can insist on a user specified overestimate. The latter of these solutions

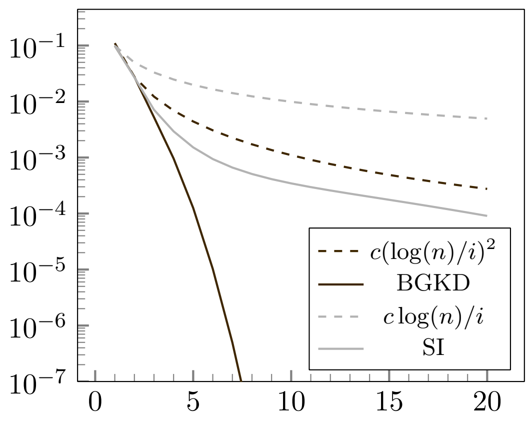


Figure 3.1: Empirical Frobenius norm convergence compared to theoretical bounds from [49, Algorithm 1/2]. The constants $c = [0.0096, 0.001]$ for SI and BGKD respectively are chosen to be as small as possible.

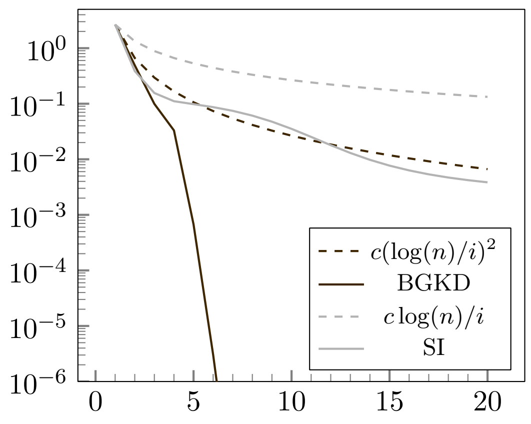


Figure 3.2: Empirical spectral norm convergence compared to theoretical bounds from [49, Algorithm 1/2]. The constants $c = [0.253, 0.0241]$ for SI and BGKD respectively are chosen to be as small as possible.

may sound limiting, but there are multiple reasons for why it is our preferred solution. First, some methods, like subspace iteration, do not have well studied ways of increasing the current basis size. Second, it is reasonable to expect that some maximum amount of memory is available or practical, especially for large scale problems. Last, for methods that can restart, a dynamic memory solution would require additional heuristics to determine when to restart and how many vectors to maintain. For these reasons, our solution requires an a priori overestimate of the rank, q, but crucially does not require this estimate to be close to $\boldsymbol{k}$. Even if the estimate is very inaccurate, our solution should still provide similar results to when $\boldsymbol{k}$ is known a priori. This is because we integrate the stopping criterion into the SVD method itself and can reduce the maximum basis size during runtime as information becomes available.

Next, we discuss how to handle various LRA criteria by utilizing an SVD interface. This interface should not only meet the requirements of the criteria we investigate in this paper, but should also be sufficient for criteria that we have not analysed. Additionally, the interface must be relatively lightweight, as it will generally be called once per iteration in order to check for convergence. For some methods, software developers may choose to reduce the frequency of convergence checks to balance iteration costs with optimal stopping. We present the basic framework below.

Interface 3

```
1: function [$flag, numVals$] = StopFunc($numVals, solverdata, userdata$)
2:     Determine if additional work is needed; set flag
3:     If a new upper bound on $\boldsymbol{k}$ is known; set numVals
4: end function
```

Broadly, the inputs required for any interface to stop an SVD method fall into two categories: external user data and solver generated data. The external user data might include parameters that control tolerances, thresholds, and bounds on computation. For more intricate criteria, `userdata` can also include function pointers to routines that are required to verify the accuracy of the solution at the current iteration. Importantly, these routines may not actually measure the accuracy of the SVD itself, but rather the accuracy of an outside optimization problem. In this situation, `userdata` may also include variables that store the best solution seen up to the current iteration, or other data generated within the stopping criteria function. On the other hand, solver generated data may include a variety of information that is generally updated at each iteration of the SVD solver. As an example, Table 3.2 provides descriptions of the `solverdata` structure passed to our GKD interface.

Variable	Description	Size
`mvs`	Number of matrix vector multiplies	1
`iters`	Solver iterations	1
`time`	Total elapsed time	1
`normA`	Current estimate of $\|A\|_2$	1
`k`	Current basis size	1
`s`	Singular values	`k`
`b`	Block size	1
`resid_est`	Approximate residual norms	`k`-1

Table 3.2: Example subset of `solverdata` variables in GKD and their descriptions. Variable names come directly from GKD and may not match their use in this paper.

In terms of output, the interface should only require a completion flag and the number of values desired, k. When the completion flag is set, the solver should stop at the current iteration and return the number of singular triplets desired as indicated by the second output argument. Since $\boldsymbol{k}$ is unknown initially, *numVals* can be set using the matching input variable, but can be changed if a user determines a new upper bound to the number of desired singular triplets. This can be done even if the singular triplets have not reached a sufficient accuracy to set the completion flag. Some solvers may be able to take advantage of this decreasing estimate of $\boldsymbol{k}$ by changing internal parameters like block size, the number of vectors kept after restart, or the maximum number of vectors allowed in the basis.

In the following subsections, we investigate a few additional guidelines for specific methods. Mainly, we discuss when it makes sense to call the interface and how to provide useful `solverdata` fields without excessive overheads. These are general guidelines and may need to be adapted to fit the specific method, programming language, and architecture being used.

3.3.1 Guidelines for Standard LBD

In order to provide useful `solverdata` information, Lanczos-type methods will need to perform a small SVD on the $d \times d$ bidiagonal matrix to produce Ritz values and vectors, where d is the column dimension of the current basis. If performing $O(d^3)$ additional work at each iteration significantly degrades performance, the decomposition and interface can be called after a "super-step" consisting of many iterations. For restarted methods, the SVD computation can be postponed until restart.

In order to gauge the accuracy of solutions, Lanczos methods can provide residual norm estimates using the well known relations, $AV = UB$ and $A^T U = VB^T + \beta_d v_{d+1} e_d^T$ where $\beta_d = \|A^T u_d - \alpha_d v_d\|, \alpha_d = \|Av_d - \beta_{d-1} u_{d-1}\|$, and B is the bidiagonal coefficient matrix (see [46]). The first relation gives $\|r_i^v\| = 0$ for all $i = 1, \ldots, d$, while the second combined with the SVD of $B = X\Sigma Y^T$ gives the left residual norms,

$$\|r_i^u\| = \|A^T U X_{(:,i)} - V Y_{(:,i)} \sigma_i\| = |\beta_d X_{(d,i)}|, \tag{3.10}$$

where $X(d,i)$ is the ith entry in the last row of X. For block versions of Lanczos, the absolute value is replaced with a norm over the last b rows of X. These cheap and accurate approximations to the residual norms make LBD a prime candidate for our interface.

3.3.2 Guidelines for Subspace Iteration

For methods like RSVD, it is possible to calculate residuals without additional memory overhead, although it is computationally expensive and therefore should be limited to situations where residuals are necessary. If there is enough memory to hold an extra $(n \times d)$ matrix, the associated code can be simplified but still incurs the same computational cost. Since there are many different ways to implement this functionality with varying trade-offs, we leave this decision up to the developer. However, the option to compute them on request would allow users to trade some performance for verifying accuracy when necessary.

On the other hand, the QR of AV is a byproduct of the RSVD method and thus computing singular value approximations from the SVD of the $d \times d$ upper triangular matrix R is significantly less costly than the computation of residuals. Instead of residual norms, we can use the iteration distance between iteration j and $j-1$,

$$\phi_i^{(j)} = \sigma_i^{(j)} - \sigma_i^{(j-1)}, \tag{3.11}$$

and the well understood convergence properties of SI to obtain information about the singular value error ϵ. We discuss this further in Section 3.4.

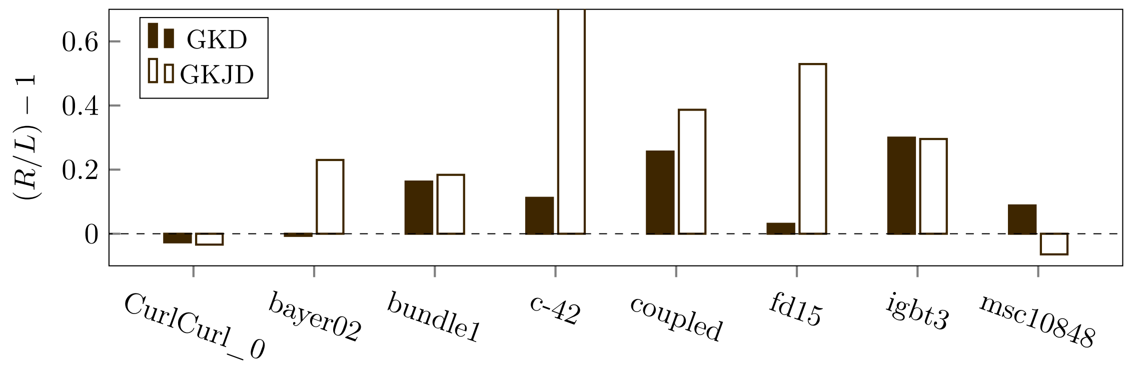


Figure 3.3: Ratio of matrix vector products required by GKD using standard targeting (R) and largest residual targeting (L) to achieve a residual tolerance of 1e-6 on 100 singular triplets. The ratio $(R/L-1)$ for GKJD on c-42 is 2.32.

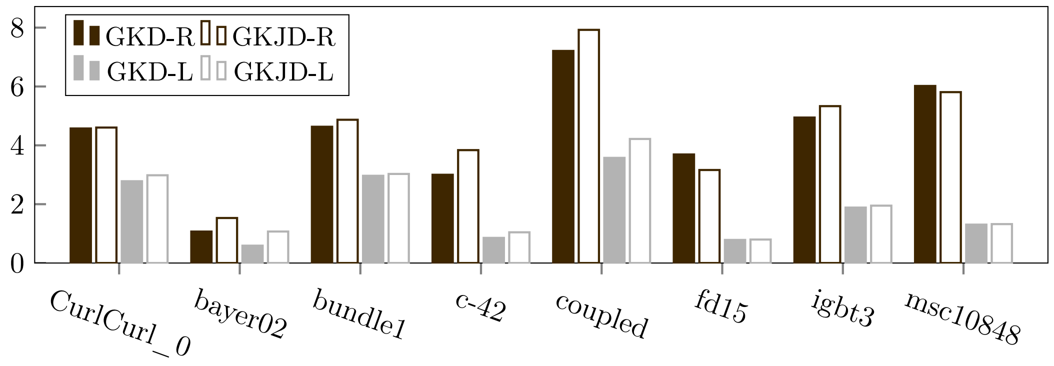


Figure 3.4: Geometric mean of the ratio between the real and approximated residual norms ($\|\boldsymbol{r}_i\|/\|r_i\|$) over the first $\min(d,100)$ singular triplets and all iterations using GKD and GKJD to find 100 singular values with standard (suffix -R) and largest residual targeting (suffix -L).

3.3.3 Specific Solutions Required for GKD

For Davidson methods like GKD, a lot of information needed for the interface is directly available. For example, Davidson must compute a residual (or more in case of a block method) in order to expand the basis at each step. This requires performing an SVD on the $(d \times d)$ projected problem, yielding d Ritz values and a block size number of residuals at each step. Therefore, we call the stopping interface after this small SVD computation.

The expansion of the basis depends on which particular value and its residual is targeted. Although, in theory, all residuals of a Krylov method are co-linear, floating point arithmetic makes convergence faster toward the targeted value while convergence to others is delayed. In Davidson type methods, locally optimal restarting and preconditioning further accentuate the effects of targeting. This means that if a particular singular value is targeted, there is little information about the rest of the spectrum until it has converged. Moreover, it imposes a stringent, per vector residual criterion which is unnecessary for the criteria discussed in this paper.

There are changes to GKD that can help improve global convergence instead of per vector convergence. A simple scheme targets all sought singular values (or at least the ones the method restarts with) one by one in a round-robin fashion. However, this often degrades convergence. We have found that we can simply target the vectors corresponding to the largest b known residual norms at each iteration without any performance degradation. This targeting scheme is presented in Algorithm 4. In fact, as seen in Figure 3.3, this new targeting scheme often outperforms the standard one for a variety of problems when computing 100 singular triplets to a 1e-6 residual tolerance. Additionally, targeting in this way computes accurate residual norms for the entire space over a smaller number of steps which can serve as a reasonable upper bound on the true residuals if no other approximation is available. This is particularly important for preconditioned GKD or GKJD. Our testing utilizes matrices from the SuiteSparse Matrix Collection [14], and general information can be found in Table 3.3. This table also includes data about other matrices from SuiteSparse used in this chapter.

Algorithm 4

```
1: Initialize all_resid(1:q) = $\infty$.
2: while not Converged do
3:     Stable sort all_resid and obtain the sorting indices. [~,i] = sort(all_resid,'descend')
4:     Calculate $r_{i(1:b)}$ and their norms.
5:     Update all_resid(i(1:b)).
6:     Expand basis with targeted residuals
7: end while
```

Since GKD only expands with a block of $b < d$ residuals at a time, it may be beneficial to have residual norm approximations for the other $d - b$ residual vectors at each step.

Name	SS_ID	Dimensions	$\|A\|$
CurlCurl_0	2569	11083×11083	4.3e+10
bayer02	453	13935×13935	1.3e+05
bundle1	1347	10581×10581	6.4e+12
c-42	1561	10471×10471	4e+04
coupled	1183	11341×11341	2e+02
fd15	918	11532×11532	2.9e+08
igbt3	969	10938×10938	1.2e+12
msc10848	361	10848×10848	6.3e+11
nopss_11k	2348	11685×11685	1e+20
pkustk02	848	10800×10800	1.1e+02
psse1	1871	14318×11028	2e+05
t2dah_a	1204	11445×11445	2.7e+02
SNAP/amazon0302	2304	262111×262111	2.1e+01
SNAP/email-Enron	2290	36692×36692	1.2e+02

Table 3.3: Matrices chosen from SuiteSparse

For unpreconditioned GKD and GKJD, we have found that it is possible to approximate the exact residual norms cheaply, since the space produced is near Krylov and we can use a formula similar to (3.10). Specifically, we use the SVD of $B = U^T AV = X\Sigma Y^T$ and assume that the angle between the residuals and the next block of expansion vectors, $V_{(:,d+1:d+b)}$, is small (for exact LBD it would be exactly zero). This yields

$$\|r_i^u\| \approx \|V_{(:,d+1:d+b)}^T(A^T U x_i - V y_i \sigma_i)\| = \|B_{(1:d,d+1:d+b)}^T x_i\|, \tag{3.12}$$

where x_i and y_i are the ith columns of X and Y respectively. Since our stopping interface is called prior to expansion, these estimates will be one iteration behind the true residuals. Additionally, since GKD does not match the Krylov expansion exactly, (3.12) will be approximate instead of exact. However, we see in Figure 3.4 that it generally performs quite well with an average underestimate of less than 4 when using our largest residual targeting scheme. We expect this solution will be more inaccurate for preconditioned systems.

3.3.4 GKD MATLAB Implementation

Our MATLAB implementation of GKD requires a number of additional changes to optimize performance. Specifically, we must adapt our interface due to MATLAB's copy-on-write behavior for function variables. This modified interface is given in Interface 5. For input, we directly pass large data objects like the matrix A and bases U and V as variables to the interface to avoid a significant performance penalty caused by the way MATLAB creates structures with duplicated references. For output, `userdata` is included to allow the interface to save or modify information from one iteration to another. These changes would not be needed in a language like C where structures like `solverdata` and `userdata` can be passed by reference.

Interface 5

```
1: function [$flag, numVals, userdata$] = GKDStop($numVals, solverdata, userdata, A, U, V$)
2:     Determine if additional work is needed; set flag
3:     If a new upper bound on $\boldsymbol{k}$ is known; set numVals
4: end function
```

3.4 Stopping Criteria

The stopping criterion that is implemented in almost all iterative SVD solvers is to check the standard residual stopping criterion

$$\begin{aligned} \|r^v\|_2 &= \|Av - \sigma u\|_2 < \rho\|A\| \\ \|r^u\|_2 &= \|A^T u - \sigma v\|_2 < \rho\|A\|, \end{aligned} \tag{3.13}$$

where (u, σ, v) are the approximate singular triplets of A and ρ is a user specified tolerance. Even with our new interface, Equation (3.13) continues to be the default stopping criterion in GKD. Additionally, we use this criterion with a default value of $\rho = 1e-6$ as a stop-gap measure for all other stopping criteria in GKD.

The residual norm provides strong guarantees, which makes it ideal for determining high accuracy solutions. From the Davis-Kahan $\sin\theta$ theorem [12, 55], for any approximate singular triplet (u, σ, v), there is an exact singular vector $\boldsymbol{u}$ and singular value $\boldsymbol{\sigma}$ such that if gap $= |\boldsymbol{\sigma} - \sigma|$, then

$$\sin \angle(\boldsymbol{u}, u) \leq \frac{\|r^u\|}{\text{gap}}. \tag{3.14}$$

The main drawback of this bound is that $\boldsymbol{\sigma}$ may not correspond to the singular value the user wants. We examine this issue further later. If the SVD method utilizes Ritz values from a Rayleigh-Ritz procedure, recent work has tightened this bound significantly providing useful information even when $\frac{\|r^u\|}{\text{gap}} \geq 1$ [50].

While stopping by (3.13) provides strong guarantees for the singular vectors, the criterion may be far too strict for certain applications. For instance, practitioners may only desire solutions where the returned vectors span the correct space. As a trivial example, take A to be a diagonal matrix with entries sorted from largest to smallest. If we are interested in the Frobenius norm criterion, $\|(I - ZZ^T)A\|_F < \delta_F \|A\|_F$, any $n \times k$ matrix $Z = [X; 0]$ where X is a $k \times k$ orthonormal matrix would give an optimal rank k solution.

The following subsections discuss a few common alternative criteria that are useful in many applications with low-rank approximation requirements. However, the standard formulation of these criteria is given in terms of accurate singular triplets which does not translate optimally to approximate triplets, which is the case at each iteration of an SVD solver. In this case, we require additional constraints for the SVD stopping criteria in order to ensure solutions that satisfy the LRA requirement. We provide suggestions for these new constraints that are more robust in the face of inaccurate singular triplet approximations. Additionally, when used with our interface, each criterion can stop the SVD solver nearly optimally while avoiding wasted compute time and eliminating any need for post-processing checks on the solution.

For these new criteria, an estimation of the true error in the singular value, $\boldsymbol{\epsilon}$, is required in order to estimate the distance between the current solution and the optimal low rank solution. Importantly, we desire an estimate that converges during the iterative process and one that provides an upper bound to $\boldsymbol{\epsilon}$ to ensure our criteria provide accurate

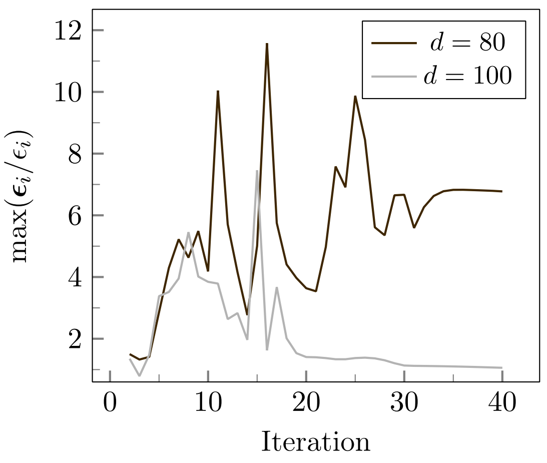


Figure 3.5: Maximum multiplicative error of ϵ using RSVD on SNAP/Amazon0302.

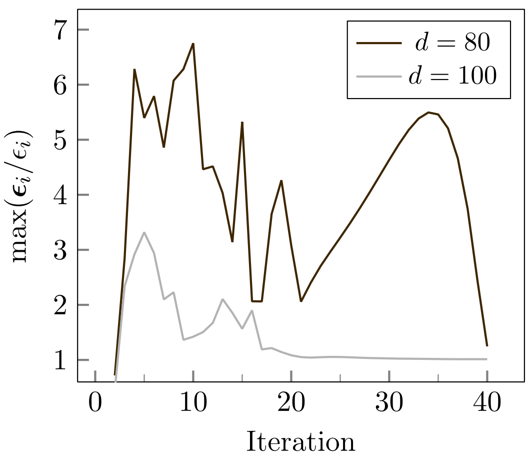


Figure 3.6: Maximum multiplicative error of ϵ using RSVD on SNAP/email-ENRON.

answers. One choice is the iteration distance given in (3.11). It is convenient as it incurs minimal cost and can work well near convergence. For Davidson and Lanczos methods, however, convergence is monotonic which implies that the true error after j iterations is $\boldsymbol{\epsilon}_i^{(j)} = \sum_{l=j+1}^{\infty} \phi_i^{(l)}$, with $\phi_i^{(l)} \geq 0$, so iteration distance is always an underestimate of the error and unsuitable for our criteria on its own.

On the other hand, for RSVD, (3.11) can be used to develop a usable approximation as $\boldsymbol{\epsilon}_i^{(j)} = \boldsymbol{\sigma}_i - \sigma_i^{(j)} \approx c_i \boldsymbol{\nu}_i^j$ for some constant c_i that depends on the initial guess and the asymptotic convergence rate of the singular value $\boldsymbol{\nu}_i = \boldsymbol{\sigma}_i/\boldsymbol{\sigma}_{k+1}$. Even though exact values for $\boldsymbol{\sigma}_{k+1}$ and $\boldsymbol{\sigma}_i$ are not known, we can approximate $\boldsymbol{\nu_i} \approx \nu_i = \frac{\phi_i^{(j)}}{\phi_i^{(j-1)}}$ based on the following relation:

$$\phi_i^{(j)} = \sigma_i^{(j)} - \sigma_i^{(j-1)} = c_i \nu_i^{j-1} - c_i \nu_i^j = c_i \nu_i^{j-1}(1 - \nu_i). \tag{3.15}$$

Additionally, solving the above equation for c_i yields an estimate for $\boldsymbol{\epsilon}_i$,

$$\epsilon_i = \phi_i^{(j)} \frac{\nu_i}{1 - \nu_i}. \tag{3.16}$$

The accuracy of this estimate is shown in Figure 3.5 and Figure 3.6 by calculating

$\max(\boldsymbol{\epsilon}_i/\epsilon_i\,, i = 1, \ldots, 80)$ at each iteration of RSVD. For these graphs, any values with ϵ less than the square root of machine precision are ignored in calculating the maximum as they are considered "converged". Additionally, when both ϵ_i and $\boldsymbol{\epsilon}_i$ are near machine precision, $|\epsilon_i - \boldsymbol{\epsilon}_i|$ is small enough to ignore even if $\boldsymbol{\epsilon}_i/\epsilon_i$ is large. We see that multiplicative errors larger than a factor of 10 are rare and that oversampling ($d = 100$) can drastically improve the behavior of our estimates. The minimum multiplicative errors are not analyzed since overestimates of the singular value error are actually preferred.

For non-SI methods, other options for providing ϵ may be preferable. One such option is the residual norm $\|r_i\|$ which works well since in general, $\|r_i\| \geq \boldsymbol{\epsilon_i}$. However, when the singular triplet has converged such that $\|r_i\| < \gamma_i = \min(\sigma_{i-1} - \sigma_i, \sigma_i - \sigma_{i+1})$, the following tighter heuristic can be used:

$$\epsilon_i = \begin{cases} \|r_i\|^2/\gamma_i & \text{if } \|r_i\| < \gamma_i \\ \|r_i\| & \text{otherwise.} \end{cases} \tag{3.17}$$

If σ_i approximates some other $\boldsymbol{\sigma_j}$ with $j < i$, both of these bounds will generally underestimate the error.

To illustrate these options for estimating $\boldsymbol{\epsilon}$ with non-SI methods, we present Figures 3.7 and 3.8. The left figure shows the expected behavior for well separated values as $\boldsymbol{\epsilon}$ is bounded between the iteration distance and the residual, along with the increased accuracy obtained from (3.17) when the residual norm is smaller than the singular value gap. The right figure shows the behavior for values within a cluster, where all methods underestimate the true error due to missing directions in the largest part of the spectrum in early iterations. When singular values are too close together or residuals too large, residuals will only bound the error to the nearest true singular value, which may or may not be $\boldsymbol{\sigma}_i$. In this case, all of our estimations will underestimate $\boldsymbol{\epsilon}_i$. Due to differences between criteria, we propose solutions to this issue in the following subsections. For the general case, these errors can be mitigated with larger block sizes or by enforcing a maximum

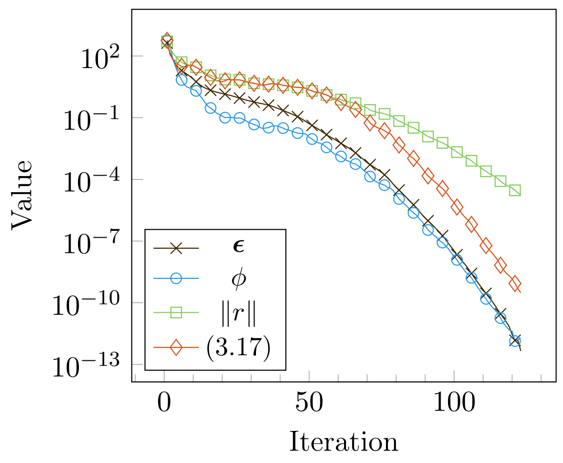


Figure 3.7: Convergence of GKD error estimates compared to $\boldsymbol{\epsilon}$ for the 1st singular value on the matrix A = diag(1000:-1:1).

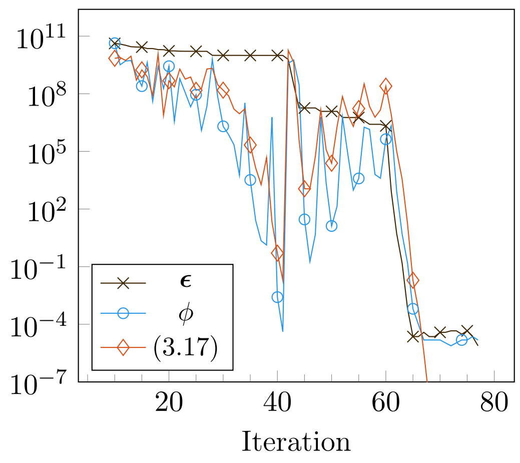


Figure 3.8: Convergence of GKD error estimates compared to $\boldsymbol{\epsilon}$ for the 10th singular value on the matrix CurlCurl_0.

acceptable residual for returned solutions similar to (3.13).

3.4.1 Frobenius Norm Criteria

One commonly referenced criterion is based on the percentage of the Frobenius norm maintained by the rank-k matrix approximation as in Equation (3.7). Many iterative methods create solutions from the Rayleigh-Ritz (RR) procedure, which ensures $U_k^T A V_k = \Sigma_k$. If the low rank solution is chosen such that $Z = U_k \Sigma_k V_k^T$, then

$$\begin{aligned} \|A - Z\|_F^2 &= \|A - U_k \Sigma_k V_k^T\|_F^2 \\ &= \operatorname{tr}((A - U_k \Sigma_k V_k^T)^T (A - U_k \Sigma_k V_k^T)) \\ &= \operatorname{tr}(A^T A) + tr(\Sigma_k^2) - 2 \operatorname{tr}(V_k \Sigma_k U_k^T A) \\ &= \|A\|_F^2 + tr(\Sigma_k^2) - 2 \operatorname{tr}(\Sigma_k^2) \\ &= \|A\|_F^2 - \|\Sigma_k\|_F^2. \end{aligned} \tag{3.18}$$

If Z is chosen to be a projection such that $Z = U_k U_k^T A$ or $Z = A V_k V_k^T$, where U or V is still chosen from the RR procedure, the corresponding matrix of residuals must also be included in the decomposition of $\|A - Z\|_F^2$. We show this for the left projected problem,

$\|(I - U_k U_k^T)A\|_F^2$, with $E_k = A^T U_k - V_k \Sigma_k$, while the right projected problem is derived similarly.

$$\begin{aligned}
\|A - Z\|_F^2 &= \|(I - U_k U_k^T)A\|_F^2 \\
&= \|A\|_F^2 - \|A^T U_k\|_F^2 \\
&= \|A\|_F^2 - \text{tr}((V_k\Sigma_k + E_k)^T(V_k\Sigma_k + E_k)) \\
&= \|A\|_F^2 - \text{tr}(\Sigma_k^2) - \text{tr}(E_k^T E_k) - 2tr(E_k^T V_k \Sigma_k) \\
&= \|A\|_F^2 - \|\Sigma_k\|_F^2 - \|E_k\|_F^2
\end{aligned} \tag{3.19}$$

with $\text{tr}(E_k^T V_k \Sigma_k) = 0$ since $E_k^T V_k \Sigma_k = (U_k^T A V_k - \Sigma_k)\Sigma_k = 0$.

If the residuals are not known, we can always safely stop based on (3.18) since its norm is larger. Equation (3.19) raises the possibility that a large random basis of vectors may have large enough residuals to cause $\|(I - ZZ^T)A\| < \delta_F \|A\|$ even when $\|A - U\Sigma V^T\|$ is very far from $\delta_F\|A\|$. For small ranks k, this is not a concern because Krylov methods with block size of 1 quickly achieve enough accuracy to ensure $\|\Sigma_k\|_F^2 \gg \|E_k\|_F^2$. With subspace iteration methods, the difference between these two norms is negligible after the first 1-2 iterations.

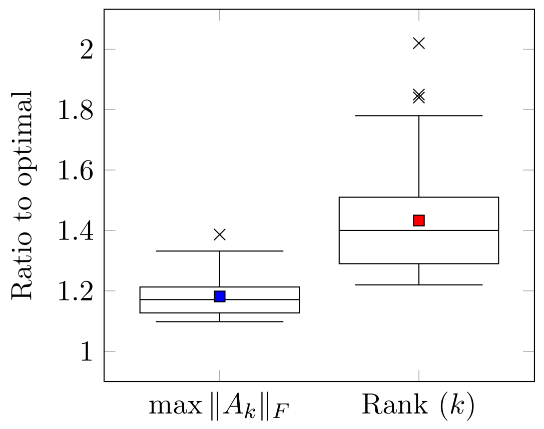
These equations allow iterative methods to avoid direct computation of $\|A - Z\|_F$. Instead, we need to know the quantities $\|A\|_F^2$, $\|\Sigma_k\|_F^2 = \sum_{i=1}^k \sigma_i^2$, and potentially $\|E_k\|_F^2 = \sum_{i=1}^k \|r_i\|_2^2$ if the projected norm is desired. On the other hand, if $\|A\|_F$ is not easily determined, these bounds will not be useful, and instead, the remainder of the spectrum would need to be approximated. An obvious upper bound would be $\|A\|_F^2 < \|\Sigma_k\|_F^2 + (n-k)\sigma_{k+1}^2$, which together with (3.18) give $\|A - Z\|_F^2 < (n-k)\sigma_{k+1}^2$. For large matrices, this is only practical when the spectrum has significant decay. Alternatively, a stochastic estimate could be derived as $\|A\|_F = \sqrt{mn\bar{x}}$ where $\bar{x}$ is the average squared norm of the entries of A. Of course, other options have been developed as in [56]. For this paper, we assume that an accurate version of $\|A\|_F$ is known or sufficient singular value decay is present to allow for accurate estimations.

We turn now to a common problem with the Frobenius norm criterion; since the optimal k is usually unknown, many possible solutions exist with varying degrees of residual accuracy and rank. As an example of the rank issue, we take a random Gaussian matrix of size 5,000 and set a threshold δ_F that requires exactly 100 values to achieve with full residual accuracy. Then we can implement the $\|A - Z\|_F < \delta_F$ criterion without any restriction on accuracy. Figure 3.9 shows the number of returned values from GKD if we vary the block size used from 1 to 75. We see that GKD returns on average 43% more values than required, and at worst returns 102% more. In terms of Frobenius norm, the rank chosen in the worst case could produce a 38.65% larger Frobenius norm at full residual accuracy. This clearly shows that this stopping criterion alone offers very little guarantee of good answers with iterative methods.

There are many options to alleviate these issues and provide stronger guarantees, which generally fall into two categories: combining weaker criteria with known strong criteria, or creating new focused criteria. For an example of the former, the Frobenius norm criteria could easily be strengthened by requiring a residual criteria for all returned singular triplets. However, this and other criteria may only be tangentially related to the Frobenius norm criterion and may impose requirements that are ultimately too strict.

Therefore, we propose a new criterion which attempts to limit the distance to the optimal rank so that $k - \boldsymbol{k} < o$ where $o \in \mathbb{N}$ is small. Using (3.18), we can calculate the maximum rank required based on the current value of $\|A - Z\|_F$. Importantly, we know that this rank will be an upper bound on $\boldsymbol{k}$. For a lower bound on $\boldsymbol{k}$, we utilize our singular value error estimates from (3.17). The difference between these ranks can be used to obtain our criteria

$$k_{max} - k_{min} \leq o, \tag{3.20}$$



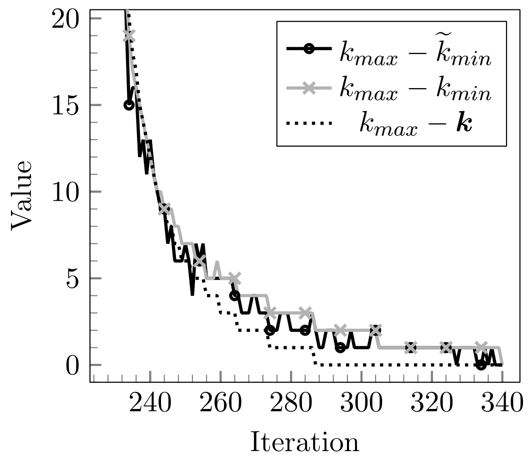


Figure 3.9: Comparing the optimal rank k (100) and optimal $\|A_{100}\|_F$ with the returned rank and corresponding maximum achievable $\|A_k\|_F$ when stopping GKD with the standard Frobenius norm criteria. The maximum achievable $\|A_k\|_F$ is derived from the true singular values of A.

Figure 3.10: Comparing estimations of k_{min} with the optimal $\boldsymbol{k}$ using GKD for the Frobenius norm criterion on a Gaussian Kernel matrix. ϵ for (3.21) is calculated by (3.17) with either exact residuals or approximate residuals from (3.12).

where o is the user given accuracy parameter and k_{min} and k_{max} are defined below:

$$\begin{aligned} k_{max} &= \min_k \left(\sqrt{\|A\|_F^2 - \|\Sigma_k\|_F^2} < \delta_F \|A\|_F \right) \\ k_{min} &= \min_k \left(\sqrt{\|A\|_F^2 - \sum_{i=1}^{k} (\sigma_i + \epsilon_i)^2} < \delta_F \|A\|_F \right). \end{aligned} \tag{3.21}$$

Additionally, if the solver implements `numVals` as a return parameter in the interface, k_{max} can be safely used to provide updated overestimates of the required rank.

This criterion provides the user control of the final rank and an accuracy parameter directly related to the LRA. If $Z = (I - UU^T)A$ is desired, our method may stop late (since the residuals decrease the norm $\|A - Z\|$). However, if a solution with nearly optimal rank $\boldsymbol{k}$ is desired (small o), our requirement that $k_{max} - k_{min} < o$ ensures that the residuals must be small as well. Therefore, the Ritz values will dominate the norm calculation and the two norms can be used interchangeably. Additionally, we have found this criterion performs nearly optimally as seen in Figure 3.10. This figure compares our estimations of

k_{min} and the optimal rank $\boldsymbol{k}$ using both accurate and approximate residuals with (3.17) on a Gaussian Kernel matrix made with 5 random vectors (of length 5000) and $\gamma = 1$. We see that both approximations do a very good job at tracking the true distance from k_{max} on this problem.

If (3.17) does not bound the true error due to out of order convergence of singular values, k_{min} may overestimate $\boldsymbol{k}$ significantly. This may cause our criterion to return too many values. However, we will still provide a valid solution to (3.7), just with a larger rank relative to the optimal one requested by the user. To help avoid this situation, we can determine whether our estimation of the optimal $\|\boldsymbol{A}_k\|_F \approx \sqrt{\sum_{i=1}^{k}(\sigma_i + \epsilon_i)^2}$ is increasing or decreasing at each iteration. When converging to $\|\boldsymbol{A}_k\|_F$, we should expect an overestimate to decrease towards this value. If we see an increase, that means at least one $\sigma_i + \epsilon_i$ increased, which suggests that it was not previously an upper bound. Therefore, we can require that this value is decreasing in addition to (3.20). Alternatively, we can increase the block size used by the solver (if the solver supports this feature), as this increases the effective gap between values, and makes out of order convergence less likely.

3.4.2 Spectral Norm Criteria

Another common criterion minimizes the spectral norm $\|A - Z\|$ instead of the Frobenius as shown in (3.6) and restated here for convenience: $\|A - Z\| < \delta_2\|A\|$. Like the Frobenius norm, an absolute δ_2 can be defined instead of the relative version used here. However, unlike the Frobenius norm, the 2-norm does not have a convenient expansion that allows direct computation. Instead of directly calculating this error, Halko, Martinsson and Tropp [27] bound $\|(I - QQ^T)A\|$ by calculating $\|(I - QQ^T)A\Omega\|$ where the columns of Ω are a small number of random normal Gaussian vectors. While this method can derive some information, according to [27, Remark 4.1] this bound is “somewhat crude" and we have found it to be too inaccurate for our purposes.

Instead, we look to the thresholding problem since, if $Z = \boldsymbol{A_k}$ is chosen, (3.6) can be restated as finding all i such that $\sigma_i \geq \delta_2\|A\|$. Since this problem only deals with the singular values themselves, we can use our approximations of σ_i and ϵ_i to develop a reasonable criterion that takes advantage of our SVD method during runtime. In general, this criterion can provide sufficient results for the more general spectral norm problem even though we do not directly measure it. For any iteration of a solver we define the following terms,

$$k_{min} = \{i|\sigma_i > \delta_2\|A\| > \sigma_{i+1}\} \tag{3.22}$$

$$k_{max} = \min\{i|\sigma_i + \epsilon_i > \delta_2\|A\| > \sigma_{i+1} + \epsilon_{i+1}\}. \tag{3.23}$$

We know k_{min} is uniquely defined since the σ_i are sorted. Due to the Cauchy Interlacing theorem, which shows that the singular values converge towards matrix extrema [55], we know k_{min} should always provide a lower bound on the rank required. On the other hand, k_{max} may not be unique without the minimum constraint, since in general, we cannot expect $\sigma_i + \epsilon_i$ to be sorted. Additionally, since ϵ_i often grows as i increases, there can be large ϵ_i which may cause $\sigma_i + \epsilon_i > \delta_2\|A\|$ even when the first $\boldsymbol{k}$ singular triplets are already fully converged. An alternative solution is to choose the maximum index, but limit k_{max} to less than $d - b$. This can ensure that large residuals that occur when a vector is first added to the basis do not effect the k_{max} calculation.

While $k_{min} \leq \boldsymbol{k}$, k_{max} is an upper bound only when $\epsilon_i > \boldsymbol{\epsilon}_i$ for all i, i.e., there are no singular values missing in the Rayleigh Ritz extraction. We can see from Figure 3.11 that this is true only once k_{min} is near $\boldsymbol{k}$. Additionally, the distance $k_{max} - k_{min}$ may be 0 well before $k_{max} \geq \boldsymbol{k}$. Therefore, we need some indication of when we can trust k_{max} to be an upper bound. One option is to enforce a residual criterion with

$$\rho < \delta_2 \tag{3.24}$$

on the first $k_{max} + 1$ singular triplets. Intuitively, this seems reasonable since we should expect the user would like at least some accuracy on the values near δ_2. If reasonably high

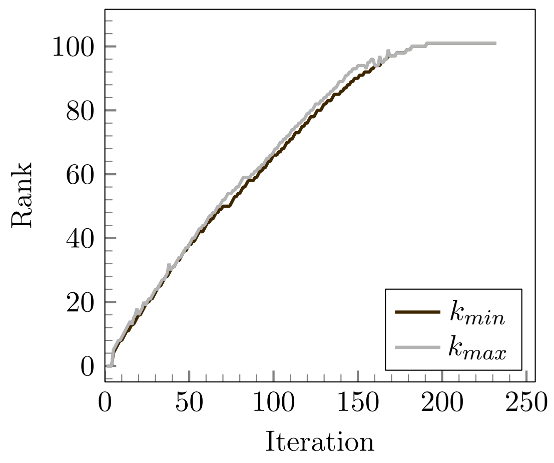


Figure 3.11: Convergence of k_{max} and k_{min} on CurlCurl_0 using GKD where the threshold is given in Table 3.4. k_{max} is calculated using accurate residual norms and (3.17).

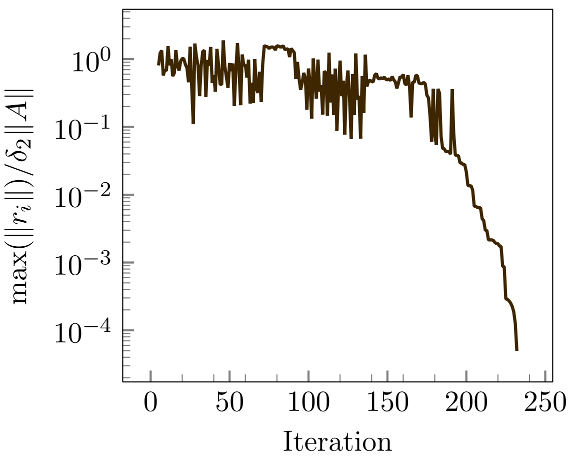


Figure 3.12: Convergence of the first $k_{max}+1$ residual norms using GKD on CurlCurl_0. δ_2 matches the threshold used for Figure 3.11.

accuracy is required compared to δ_2, we can expect that the chances of underestimating $\boldsymbol{\epsilon}$ are lower. We can see this in Figure 3.12, where the first $k_{max}+1$ residuals do not converge beyond 1e-2 until k_{max} is an upper bound. We expect that this additional criterion will be sufficient for the majority of use cases.

However, if less residual accuracy is required, we see in Figure 3.12 that the convergence is erratic and could cause the algorithm to stop early. In these cases, we have developed a heuristic that tracks the rate of increase in k_{min}. Logically, we know k_{min} will grow until it reaches $\boldsymbol{k}$. Additionally, we can expect this rate to slow as the distance to $\boldsymbol{k}$ decreases. We measure this growth rate using a least-squares regression over some small number of steps, which can be seen in Figure 3.13. In particular, for a given block size b, and current iteration j, we find the slope of the least-squares regression of k_{min} from iteration $\max(1, \min(\lceil 0.95j \rceil, j - \lceil 20/b \rceil))$ to j. This ensures that the slope is determined over at least 40 matrix vector products of information, while averaging over more steps for slowly converging problems. We also know that this slope will always become zero within $\max(\lceil 20/b \rceil, \lfloor 0.05j \rfloor)$ iterations of the first iteration where $k_{min} = \boldsymbol{k}$. The user can choose a threshold near 0 for the slope of this regression heuristic to balance accuracy of

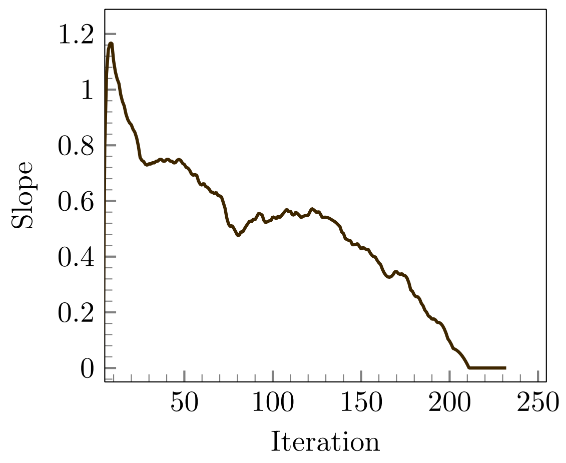


Figure 3.13: Using Least-Squares Regression to track the rate of increase in k_{min}.

the estimated rank and the work required to improve that accuracy. For our experiments, we choose this threshold to be exactly 0 to avoid issues with early stopping as much as possible. We note that this heuristic may still fail in pathological problems as many clusters of multiplicities will always exhibit poor behavior with single vector methods. In these cases, using our heuristic with an increased block size can greatly improve robustness.

Even if k_{max} is an upper bound, the $k_{max} - k_{min}$ criterion we used for the Frobenius norm may cause issues. For example, $\delta_2\|A\|$ may lie within a cluster of singular values, and therefore $\boldsymbol{\sigma}_{k_{min}+o+1}$ may be nearly equal to $\delta_2\|A\|$. If this occurs, the $k_{max} - k_{min}$ criterion will require a significant amount of computation to reduce the estimated singular value error to nearly 0. In other words, the amount of computation required may be highly dependent on the choice of δ_2 and unknown spectral information.

Instead, we can use the least-squares regression to prevent early stopping and the additional residual criterion in (3.24). This combination ensures we will not stop too early when only a modest residual threshold is required, while also requiring returned singular values to be larger than $(\delta_2 - \rho)\|A\|$.

We show results for a variety of problems in Table 3.4. We can see that the maximum ρ required to ensure k_{max} is an upper bound varies significantly, including one problem

Matrix	$\boldsymbol{k}$	δ_2	$k_{max} > \boldsymbol{k}$	$k_{min} = \boldsymbol{k}$	$\max(\rho)$	LSR $= 0$
CurlCurl_0	100	2.98e-01	191	191	2.69e-02	211
bayer02	17	1.00e-04	48	48	3.77e-07	68
bundle1	100	3.05e-02	109	149	1.19e-01	169
c-42	100	1.65e-03	148	149	4.62e-04	169
coupled	100	6.32e-02	242	432	4.93e-03	308
fd15	62	1.00e-04	75	76	1.18e-02	96
igbt3	100	1.22e-01	227	227	1.90e-03	247
msc10848	100	1.69e-04	175	214	2.56e-05	234
nopss_11k	4	1.00e-04	6	7	1	–
pkustk02	100	4.18e-01	187	236	4.90e-02	256
psse1	100	1.00e-01	421	499	1.96e-04	483
t2dah_a	100	6.58e-02	141	245	1.15e-02	196

Table 3.4: Searching for all singular values above the threshold $\delta_2 = \max(\frac{\boldsymbol{\sigma}_{100}+\boldsymbol{\sigma}_{101}}{2\boldsymbol{\sigma}_1}, 1e{-}4)$ for 12 SuiteSparse matrices. For each matrix, we note the optimal $\boldsymbol{k}$, the threshold, the first iteration where k_{max} is an upper bound, the first iteration where k_{min} is optimal, the maximum ρ (relative to $\|A\|$) that will return at least $k_{max} > \boldsymbol{k}$, and the first iteration that the least-squares regression has a slope of 0.

where any $\rho < 1$ would have sufficiently stopped the method. Additionally, we see that the least-squares regression (LSR) always has a non-zero slope until k_{max} is an upper bound (for all problems other than nopss_11k), and is never more than 20 iterations larger than the first iteration where $k_{min} = \boldsymbol{k}$. While we cannot guarantee that the least-squares regression will always be an accurate indicator of $k_{max} > \boldsymbol{k}$, our results indicate that it performs well for a variety of problems.

We want to specifically address the anomalous results shown for nopss_11k. The low rank structure of nopss_11k has a multiplicity of 4 at σ_1 and all other singular values are numerically zero. This means that the problem takes fewer than 20 iterations to converge, so the least-squares regression gives a non-zero slope for all iterations. Additionally, this problem does not require any accuracy requirement to guarantee $k_{max} > \boldsymbol{k}$ because the first time k_{max} was defined, k_{max} was already an upper bound. Our code includes a safeguard for problems like this one which stops the method when $\|r_i\|$ is approximately machine precision for each of the first $k_{max} + 1$ singular triplets.

3.4.3 Singular Gap Finding

Unlike the criteria mentioned in earlier sections, gap criteria focus directly on the structure of the singular values themselves, rather than an achieved $\|A - Z\|$ norm. This can be described as an additive gap, as in [74], where the user is searching for

$$\max_{i=1:q} \boldsymbol{\sigma}_i - \boldsymbol{\sigma}_{i+1} > \delta_G, \tag{3.25}$$

where q is the user chosen maximum rank required by our interface. Alternatively, multiplicative gaps could be desired, as in [67], where we want to find a gap such that,

$$\min_{i=1:q} \frac{\sum_{j=i+1}^{i+w+1} x_j}{\sum_{j=i-w}^{i} x_j} < \delta_G \tag{3.26}$$

with $x_j = \max(0, \boldsymbol{\sigma}_j - \theta)$, a user given θ, and window size w. For example, when $\theta = 0$ and $w = 0$ this is a simple ratio of two successive singular values. When $w > 0$ the criterion involves the ratio of the additive weights of two neighboring non-overlapping windows. Unlike the Frobenius norm and thresholding criteria, these gap criteria require the additional minimum or maximum rank restriction as there could be multiple gaps that satisfy the inequality. However, it is also possible that no significant spectral gap exists at the level desired by the user. In order to have more robust criteria, these gap criteria are often paired with additional criteria similar to (3.7), (3.6), or (3.13) with a given k.

To bound the maximum number of values required, we can view (3.25) as both a gap criterion and a thresholding criterion where $\delta_2\|A\| = \delta_G$, since all singular values are non-negative and therefore any solution requires

$$\boldsymbol{\sigma}_i \geq \boldsymbol{\sigma}_i - \boldsymbol{\sigma}_{i+1} > \delta_G.$$

Viewing the multiplicative gap criterion in this manner is not as useful without additional information, since the relative distance $\boldsymbol{\sigma}_{i+1}/\boldsymbol{\sigma}_i$ can be arbitrarily small. However, if $\boldsymbol{\sigma}_n$ or the condition number is known, we can choose $\delta_2\|A\| = \boldsymbol{\sigma}_n/\delta_G$ because any $\boldsymbol{\sigma}_i < \delta_2\|A\| = \boldsymbol{\sigma}_n/\delta_G$ implies also that

$$\boldsymbol{\sigma}_{i+1}/\boldsymbol{\sigma}_i > \delta_G(\boldsymbol{\sigma}_{i+1}/\boldsymbol{\sigma}_n) \geq \delta_G.$$

Using these thresholding criteria in combination with their associated gap criteria allows an online refinement of q which can improve performance and allow the solver to stop earlier.

The rest of this section will focus on the minimum rank restriction for (3.25) without additional restrictions, since finding the maximum rank restricted solution can be satisfied similarly by searching backwards from the largest acceptable rank, q. In fact, the maximum rank restriction may be easier to solve optimally for single vector methods because many vectors need to be computed before evaluating the gap criterion. In order to develop a robust version of (3.25), like in the previous sections, we desire lower and upper bounds on the optimal $\boldsymbol{k}$. For the additive criteria we have:

$$\begin{aligned} k_{min} &= \min_{i=1:q} \sigma_i + \epsilon_i - \sigma_{i+1} \geq \delta_G \\ k_{max} &= \min_{i=1:q} \sigma_i - (\sigma_{i+1} + \epsilon_{i+1}) \geq \delta_G. \end{aligned} \tag{3.27}$$

Given these definitions and upper bounds to the true error $\boldsymbol{\epsilon}$, we have the following.

Lemma 1. *Let* $\forall i\ \epsilon_i \geq \boldsymbol{\epsilon_i}$ *and* $\sigma_i \leq \boldsymbol{\sigma_i}$. *Then* $k_{min} \leq \boldsymbol{k} \leq k_{max}$.

Proof. Because $\sigma_{\boldsymbol{k}} + \epsilon_{\boldsymbol{k}} - \sigma_{\boldsymbol{k}+1} \geq \boldsymbol{\sigma_k} - \sigma_{\boldsymbol{k}+1} \geq \boldsymbol{\sigma_k} - \boldsymbol{\sigma}_{\boldsymbol{k}+1} \geq \delta_G$ and because k_{min} is the minimum index that achieves that bound, then $k_{min} \leq \boldsymbol{k}$.

From (3.27), $\sigma_{k_{max}} - (\sigma_{k_{max}+1} + \epsilon_{k_{max}+1}) \geq \delta_G$. However, $\boldsymbol{\sigma}_{k_{max}} - \boldsymbol{\sigma}_{k_{max}+1} \geq \sigma_{k_{max}} - \boldsymbol{\sigma}_{k_{max}+1} \geq \sigma_{k_{max}} - (\sigma_{k_{max}+1} + \epsilon_{k_{max}+1}) \geq \delta_G$. This means that k_{max} is a solution to the gap criterion, and since $\boldsymbol{k}$ is the minimum index that satisfies this solution, $\boldsymbol{k} \leq k_{max}$.

Given Lemma 1, we know the requirement $k_{min} = k_{max}$ is sufficient to find $\boldsymbol{k}$ for methods that typically overestimate $\boldsymbol{\epsilon}_i$ like RSVD. For GKD and Lanczos-type methods ϵ_{i+1} can underestimate the true $\boldsymbol{\epsilon}_{i+1}$ and therefore k_{max} can underestimate $\boldsymbol{k}$ like we've seen with the thresholding criterion. Additionally, this criterion may be very sensitive to gaps $\sigma_i - \sigma_{i+1} \approx \delta_G$. For example, let's assume $\boldsymbol{\sigma}_i - \boldsymbol{\sigma}_{i+1} + \tau = \delta_G$ for some small τ and $i < \boldsymbol{k}$. In order to have $k_{min} = \boldsymbol{k} = k_{max}$, we must have $k_{min} > i$, so from (3.27) we want $\sigma_i + \epsilon_i - \sigma_{i+1} < \delta_G = \boldsymbol{\sigma}_i - \boldsymbol{\sigma}_{i+1} + \tau$, which implies $(\sigma_i + \epsilon_i - \boldsymbol{\sigma}_i) + (\boldsymbol{\sigma}_{i+1} - \sigma_{i+1}) < \tau$, or equivalently $\epsilon_i - \boldsymbol{\epsilon}_i + \boldsymbol{\epsilon}_{i+1} < \tau$. Thus, k_{min} will not skip over i until both the i and the $i+1$ errors become smaller than τ. A similar problem with k_{max} and $\epsilon_{\boldsymbol{k}+1}$ exists if the optimal solution is only slightly larger than δ_G: $\sigma_{\boldsymbol{k}} - \sigma_{\boldsymbol{k}+1} - \tau = \delta_G$. Both situations would require significant computation for very small τ to ensure $k_{min} = \boldsymbol{k} = k_{max}$ even though a solution could be found easily for a slightly different choice of δ_G.

Therefore, it is preferred to accept k_{max} as $\boldsymbol{k}$ if $\epsilon_{k_{max}}$ and $\epsilon_{k_{max}+1}$ are small enough. Specifically, the returned solution would have rank k_{max} with $\epsilon_{k_{max}+1} < \rho\|A\|$ which acts as a residual criterion like (3.13). This should guarantee the rank of the solution is the optimal solution for the problem where the gap threshold is $\delta_G - \rho\|A\|$.

Results for this criterion can be seen in Table 3.5. Specifically, we can see that most problems need only an order or two of accuracy on the first k_{max} values relative to $\sigma_{k_{max}}$, with only 3 problems that require more (CurlCurl_0, nopss_11k, and psse1). These 3 problems contain multiplicities or near multiplicities for the largest singular values, and we've found that the accuracy requirements can be lessened or completely removed with a larger block size. Additionally, even with a residual tolerance relative to $\sigma_{k_{max}}$ of 10^{-6}, the criteria stops within only a few iterations of the optimal stopping point (when $k_{max} = \boldsymbol{k}$). This should not be particularly surprising since the largest additive gap separates the values we desire and the rest of the spectrum.

3.5 Matrix Completion

Over the last decade matrix completion algorithms have seen increased importance due to the growth of large, but incomplete, datasets. These datasets include user ratings for a variety of platforms including Amazon, Microsoft and Netflix. Additionally, the matrix

Matrix	$\delta_G/\|A\|$	$\boldsymbol{k}$	$k_{max} = \boldsymbol{k}$	Criteria Stop	$\max(r/\sigma_{k_{max}})$
CurlCurl_0	1.83e-01	14	59	65	2.76e-04
bayer02	4.03e-01	1	2	4	1
bundle1	1.39e-01	14	29	34	3.04e-02
c-42	3.36e-01	1	6	8	8.21e-02
coupled	2.16e-01	1	2	13	1
fd15	2.40e-01	1	5	9	2.57e-01
igbt3	1.89e-01	3	8	19	2.02e-01
msc10848	1.82e-01	15	20	23	2.86e-01
nopss_11k	1.28e-08	4	–	3	–
pkustk02	2.81e-02	45	86	101	4.42e-02
psse1	4.05e-01	2	12	7	4.10e-12
t2dah_a	9.67e-02	1	8	15	1

Table 3.5: Searching for an additive gap larger than δ_G, which is chosen to be the geometric mean of the two largest gaps. The columns $k_{max} = \boldsymbol{k}$ and Criteria Stop give iteration counts. The criteria stops when all $r_i/\sigma_{k_{max}} < 10^{-6}$ with $i \in 1 : k_{max} + 1$. We testednopss_11k, however GKD stops early before finding all 4 multiplicities, so no values are given for $k_{max} = \boldsymbol{k}$ or $\max(r/\sigma_{k_{max}})$.

completion problem can be useful for localization of internet of things networks where only partial distance information is available [51]. In all of these cases, we have a large matrix A with an unknown but (hopefully) small rank, where only a small portion of its elements is known. We denote the indices of these observed elements as the set Ω. Matrix completion algorithms attempt to "fill in" these missing elements.

Often, these problems are considered to be low rank due to the categorization of human preferences. A few examples include Amazon grouping items into product segments, or Netflix grouping movies into genres. Predictions can then be made based on an individual's affinity for any given grouping. In general, if A is not low rank, this problem is ill-posed and may not result in "correct" values for the entries outside of Ω. However given a low rank structure, bounds on error as well as the number of observed entries required to converge have been studied [10, 11, 37, 60]. Additionally, many algorithms have been created based on a convex nuclear norm relaxation of the rank constrained matrix completion problem

$$\begin{aligned} &\text{minimize } \|Z\|_* \\ &\text{subject to } \sum_{(i,j)\in\Omega} (A_{ij} - Z_{ij})^2 < \delta, \end{aligned} \tag{3.28}$$

given a low rank approximation Z. While it might be preferable to minimize the rank of Z instead of the nuclear norm, rank minimization in this context is a non-convex optimization problem. A few of the algorithms created to solve this problem include the work of Cai, Candes, and Shen in their work on singular value thresholding (SVT) [9], the Soft-Impute algorithm by Mazumder et al. [44] and the APGL algorithm by Toh and Yun [67]. Each of these works was published shortly after the conclusion of the Netflix competition in 2009.

Noticeably, SVT, Soft-Impute and APGL all rely on computing a set of singular values above a given threshold at each step. Specifically, a θ parameter is chosen (potentially at each step as seen in APGL) and the resulting low rank approximation S is created from the SVD of the approximate $Z = U\Sigma V^T$ by

$$S = U(\Sigma - \theta I)_+ V^T, \tag{3.29}$$

where I is the identity matrix, and for any matrix X, $(X)_+ = max(X, 0)$. Hard-Impute [44] removes the subtraction from Σ, but relies on a threshold θ as in (3.6) to choose the appropriate low rank-k parameter. Since they chose PROPACK as the SVD software when implementing these algorithms, they each developed a heuristic to determine a priori the number of singular values to compute in each PROPACK call. With other software, such as the SVD solver in the PRIMME software package [64], calculating all values above a threshold is relatively trivial. This leads to a natural question: Is there a performance benefit to their heuristics and why?

Hard Impute with Varying SVD Tolerance

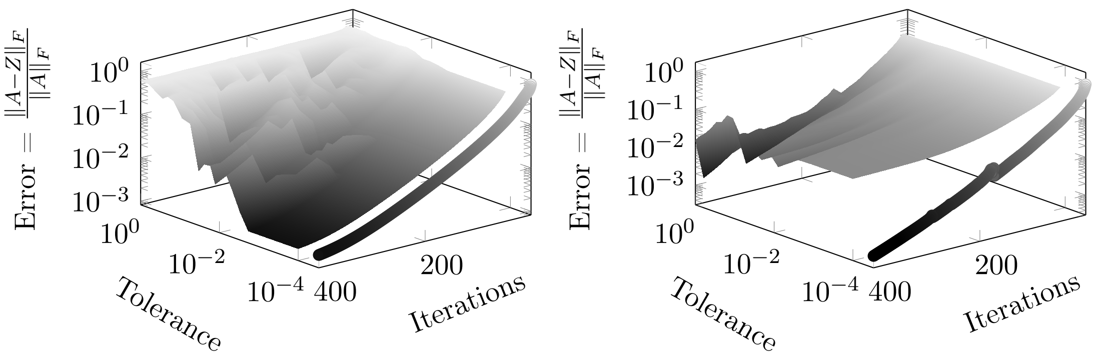


Figure 3.14: Comparing various SVD tolerances (surface plot) with our specialized stopping criteria (line plot). (Left) Results with a very flat spectrum $\Sigma = \text{linspace}(0.99, 1, 30)$ for the low rank space. (Right) Results with a slightly decaying spectrum $\Sigma = \text{linspace}(1, 50, 30)$ for the low rank space.

3.6 Hard-Impute Testing

In order to better understand the role of SVD tolerance, we began our analysis using a Hard-Impute type method, where θ is chosen at each step to obtain the same rank k. This allows us to choose k to equal the true rank of our test matrices. In Figure 3.14, we show results for two rank 30 matrices with varying SVD residual tolerances.

Interestingly, problems with more decay within the low rank space converged much more quickly with very high tolerances. Since problems with very flat spectra did not exhibit this behavior, we developed a specialized criterion that could handle either situation (seen as lines in Figure 3.14). To do this, we monitored the convergence of the sum in Equation (3.28) inside of our GKD implementation. If the value increased, we stopped GKD and returned the best rank 30 approximation currently available.

This criterion may be expensive as it results in $O(gk)$ additional operations for each SVD step where g is the number of entries of A used for comparison. In comparison, each matvec is made up of a sparse matvec, $O(|\Omega|)$, and a vector multiplication with two low rank bases $O(nk+mk)$. Two matvecs are performed for each outer iteration, so the overall cost of our criterion may not exceed the matvec cost for small $|\Omega|$. While we utilized the full sample size available for $O(|\Omega|k)$ operations per step, potentially a subsample could suffice, further increasing performance. Also, the convergence check could be performed less frequently, at each restart of the SVD. Considering the large cost of orthogonalization for large matrices, it should be possible to include criteria like this with minimal overhead.

While the results shown in Figure 3.14 are promising, we found that comparing to the sampled values of A sometimes was not sufficient to provide optimal performance beyond 1e-3. For the slightly decaying spectra $\Sigma = \text{linspace}(1, 50, 30)$, a tolerance of 5e-1 provided superlinear convergence to 1e-12 error, while our criterion converged significantly slower. If our criterion tests against the entire matrix A (instead of just the sampled entries), we are able to improve results beyond those of the standard 5e-1 tolerance. Obviously, this could not be done in practice, but these results suggest that stopping the SVD appropriately can produce high quality and fast results for the matrix completion problem.

3.7 Soft-Impute Testing

In order to determine the benefits (or drawbacks) of their heuristics, we first analyze the performance of the Soft-Impute algorithm with varying thresholds on a small artificial test case. This matrix is a 500×500 rank 20 matrix, with singular values $\text{diag}(\Sigma) = \text{linspace}(1, 50, 20)$. We uniformly choose a random sample comprising 30% of A to use as A_Ω. For a rank 20 problem, 30% of the entries should easily provide enough information to converge to the correct low rank space. At each step, we calculate all singular values using Matlab's `svd` function, then update our approximation using all values above the threshold provided. We run the same test with 20 thresholds from $\text{linspace}(0.999, 0.1, 20)$. These thresholds are chosen to include all non-zero singular values, while not including the entire spectrum of A_Ω at the first step.

Figure 3.15 shows the convergence of Soft-Impute towards the low rank approximation size of 20. It is important to note that only the smallest two thresholds complete 1000 iterations with the correct number of singular values. While larger thresholds converge

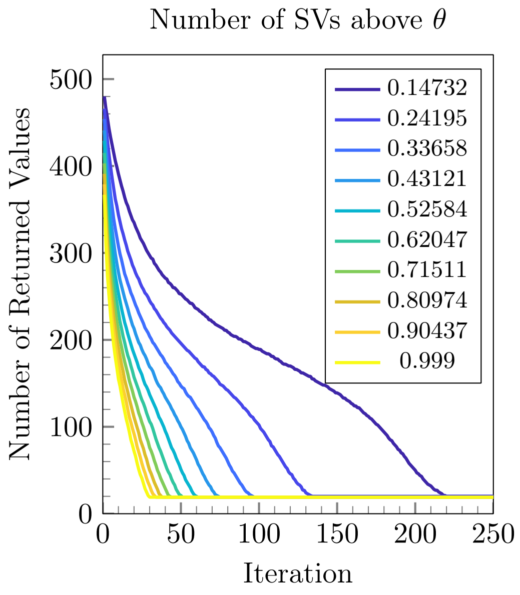


Figure 3.15: Comparing the number of singular values needed at each iteration for varying thresholds

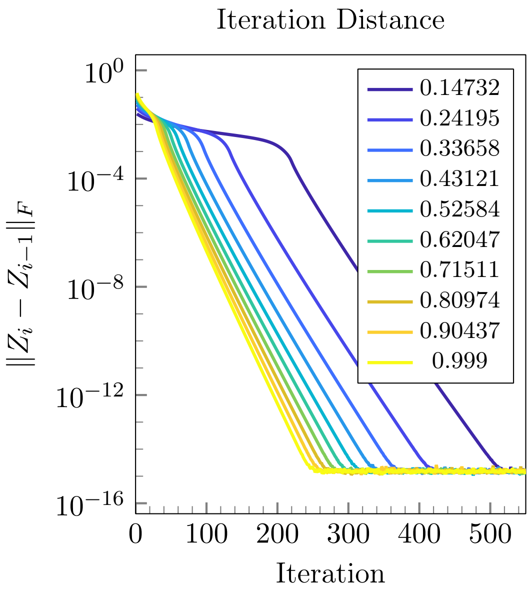


Figure 3.16: Comparing the distance between two successive Soft Impute steps with varying thresholds

more quickly, they tend to overshoot, leading to an underestimate of the rank. Obviously, this can limit their ability to converge to the correct solution as the Frobenius norm of a solution missing the smallest value is off by at least

$$1 - \frac{\sqrt{\sum_{i=1}^{19} \sigma_i^2}}{\sqrt{\sum_{i=1}^{20} \sigma_i^2}} = 2.87 \times 10^{-5} \tag{3.30}$$

relative to the true solution. If there is also error in the largest portion of the spectrum, even more error may accumulate.

Next, we focus on the standard stopping criteria for Soft-Impute, which is based on the distance between two successive iterates. Again, we see that higher thresholds begin their convergence more quickly, however the asymptotic rate of convergence seems similar for all thresholds. Perhaps not surprisingly, the larger threshold values remove more singular values and therefore have larger iterate distances for the first 30 iterations prior to achieving asymptotic convergence. Interestingly, we see that the asymptotic convergence for each

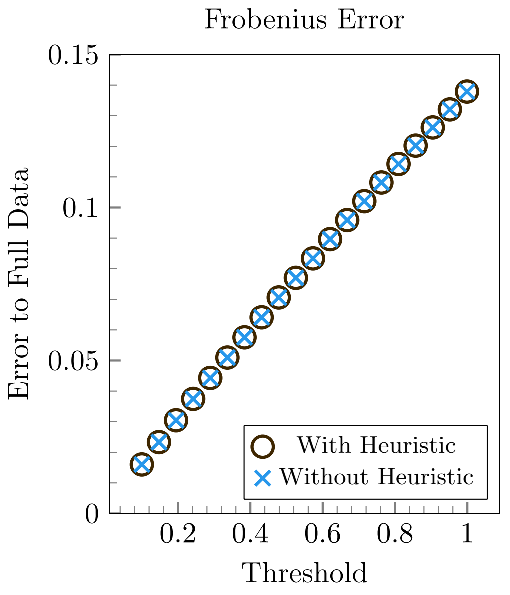


Figure 3.17: Comparing the Frobenius norm distance to the full (dense) low rank matrix A after 1000 iterations for 20 different thresholds

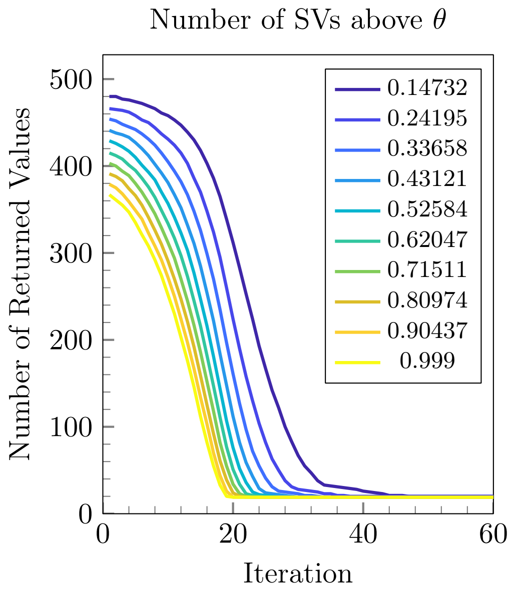


Figure 3.18: Comparing the number of singular values needed at each iteration for varying thresholds with heuristic 3.31

threshold seems to begin when the method has converged to the correct (or nearly correct) number of singular values (from Figure 3.15).

Figure 3.17 shows the significant trade-off that exists between the speed of convergence of larger tolerances and the distance to the true solution. While it should be theoretically possible to converge to the correct values of A, we find larger thresholds tend to miss crucial values and can even underestimate the largest singular values, causing significant error in the final result. Seeing this, we tried choosing a threshold of 1e-2, but the slowdown in finding the correct number of values was significant. Within 1,000 iterations, a threshold of 1e-2 still returned more than 240 singular values and the error larger than 0.45.

It is clear that small thresholds generally perform better in terms of error, but the significantly slower convergence speed must be addressed. The first option to try is utilizing the heuristics given in [44]. Essentially, instead of updating with all the singular values above the threshold, we use the following criteria:

$$\textbf{Return: } \sigma_i^{(j)} - \theta > 0 \text{ with } i \leq j \tag{3.31}$$

where (j) denotes the iteration number and θ is the chosen threshold. This limits the amount of work needed at the beginning of the algorithm significantly, but may introduce errors as we leave out significant portions of the initial space.

Using this heuristic we see nearly identical performance in error to the full dataset (Figure 3.17). However, as seen in Figure 3.18, the convergence to the correct number of singular values is nearly 5 times faster in iterations with the heuristic. If this problem were larger and required an iterative solver, this heuristic would reduce time by even larger margins, as each iteration only needs to compute a small number of singular values.

3.8 Chapter Summary

In this chapter, we presented our novel interface to the GKD solver that provides tailored solutions to a variety of low rank approximation problems. On top of providing additional flexibility for SVD stopping criteria, we have added new targeting strategies that can improve runtime of GKD in general as well as the accuracy of a new efficient residual approximation strategy. Additionally, we have analysed a few common low rank problems that focus on Frobenius and spectral norm optimization as well as gap finding. To ensure these problems are well-posed when fully accurate SVD solutions are not available or feasible to compute, we have provided additional conditions that ensure acceptable solutions for both single vector and block SVD solvers. We have demonstrated through testing on a variety of matrices that these novel criteria provide nearly optimal stopping conditions. Lastly, we have shown the flexibility of our interface for solving matrix completion problems using both standard and custom stopping criteria.

Chapter 4

GKD Software Implementation

As mentioned in previous chapters, we have developed an implementation of GKD using the MATLAB programming language for our testing as well as for practical applications. In this chapter, we focus on the specifics of the code including its user interface and runtime options.

GKD can be invoked using the following function call in MATLAB.

```
[U,S,V,H,D] = GKD(A,nv,...)
```

Descriptions of the output variables and their size can be found in Table 4.1. MATLAB syntax allows for fewer output parameters and defaults to returning values from left-to-right. As an example, if only singular values are desired, the following function call can be used.

```
[~,S] = GKD(A,nv,...)
```

The input variables include the matrix `A`, the number of values desired `nv`, which can optionally be followed by a variety of Name/Value pairs. The matrix `A` can be given as a standard matrix or as a function handle object. If using the latter, the Name/Value pairs `m` and `n` must be provided to describe the dimensions of the matrix function.

As a general purpose partial SVD solver, the code has numerous default values for the optional Name/Value pairs to provide a simplified user experience for the vast majority

Variable	Description	Size
`U`	Left singular vectors	$m \times$ `nv`
`S`	Singular Values	`nv` $\times$ `nv`
`V`	Right singular vectors	$n \times$ `nv`
`H`	Convergence History	$outerits \times 6$
`D`	User data	N/A

Table 4.1: Output variables in GKD

```
%%Code to compute the largest 20 SVs of a matrix function using
%%   a pseudo three term recurrence to a residual tolerance of
%%   1e-4. A random seed guarantees consistent results.
A = @(x,transp) mat_fun(x,transp);
[U,S,V] = GKD(A,20,'tol',1e-4,'m',1500,'n',1000,'b',20,'minRestart',20, ...
    'maxBasis',60,'numOld',0,'seed',1);
```

Algorithm 4.1: GKD usage example

of users. These additional options along with their default values are listed in Table 4.2. We provide an example using a number of these parameters in Algorithm 4.1. Some additional discussion is required for a few of these options. `SIGMA` determines whether the largest singular values (‘L’) or the smallest singular values (‘S’) are desired. This affects restarting as well as the targeting order for the majority of targeting methods.

The options `maxMV` and `maxTime` are available to stop the method prior to convergence to all `nv` values. By default, these options are not used. When these options are set, the method stop when either `maxMV` matrix vector products have been computed or the total elapsed time is greater than `maxTime`. In this scenario, the method will return the entire bases U, V as well as all current Ritz values. This may be more than `nv` values.

If available, the user can provide an estimate of $\|A\|$ using `normA`. By default this value is set to match the current largest Ritz value found. This is generally less accurate when searching for the smallest singular values (`SIGMA` = ‘S’), but is often good enough if no estimate is known a priori.

If initial guesses are available for the right singular vectors, they can be included using `v0`. Importantly, the dimension of these vectors should match the smaller dimension of A. Currently, there is no support for supplying left singular vectors. The default uses a

Variable	Description	Default
`tol`	Residual norm tolerance	1E-6
`SIGMA`	Sets which values are desired (‘L’ or ‘S’)	‘L’
`m`	Number of rows in A (function handle)	max(size(A))
`n`	Number of cols in A (function handle)	min(size(A))
`maxMV`	Maximum number of MVs	inf
`maxTime`	Maximum solver time	inf
`normA`	$\|A\|_2$ estimate	Current $\max(\sigma_i)$
`display`	Prints partial history to console if set	0 (off)
`v0`	Initial guess	randn(n,b)
`seed`	Random seed	’shuffle’
`b`	Block size	1
`minRestart`	Number of vectors to maintain after restart	Equation (4.1)
`maxBasis`	Max number of basis vectors in V,U	Equation (4.2)
`numOld`	Number of +k vectors to keep during restart	-1
`maxQMR`	Max number of QMR iterations	0
`P`	Preconditioner for A^TA	[ ]
`target_fn`	Function used for expanding the basis	‘resid’
`stop_fn`	Function used for stopping the solver	‘resid’
`userdata`	External user data	[ ]

Table 4.2: Optional Name/Value pairs for GKD

block of random standard Gaussian vectors, which are orthonormalized prior to starting the main algorithm. In order to obtain consistent results from one run to the next, the user can provide a consistent `v0` or use the default value and use `seed`, which sets the random seed for MATLAB’s random number generator. By default, `seed` is set to ‘shuffle’ which sets the random number generator based on the current time. This will produce different results for each run of GKD.

The Name/Value pairs for `minRestart`, `maxBasis` and `numOld` control the restarting parameters for GKD. `minRestart` and `maxBasis` default to the following equations:

$$\texttt{minRestart} = \max(7, \texttt{nv} + 5), \tag{4.1}$$

$$\texttt{maxBasis} = \max(15, \texttt{minRestart} + 4\texttt{b}, \lfloor 1.3\,\texttt{minRestart} \rfloor). \tag{4.2}$$

`numOld` controls the number of Ritz vectors from the previous iteration to include for $+k$

restarting which defaults to the current block size (-1). This value should be non-negative unless the default behavior is desired.

Preconditioners can be given using the `P` Name/Value pair in a number of different ways. If given as a matrix, the input should be an approximation to A^TA which will be backsolved using MATLAB's backslash operator (\). If `P` is a function_handle, the input function should approximate the action of $(A^TA)^{-1}$ on a set of vectors. Similarly, if `P` is given as a cell array, the first two arguments are treated as function handles that act on the vectors X sequentially (e.g. `P{1}(P{2}(X))`). No preconditioning is applied when `P = []` which is the default behavior.

4.1 Targeting and Stopping Functions

Advanced users may wish to change the way GKD targets vectors for basis expansion or include their on stopping criterion as presented in Chapter 3. We include the `target_fn` and `stop_fn` Name/Value pairs for this purpose. The `userdata` structure can be used in conjunction with a custom `target_fn` or `stop_fn` to pass in important external information to these functions.

The available defaults for `target_fn` include 'prog_tol' (a progressive targeting scheme), 'resid' (targets vectors sequentially to a residual of `tol`), and 'large' (targets vectors with the largest residuals). Most users should use 'resid' unless they know that their stopping criterion requires a non-standard targeting scheme. It is also possible for users to write their own targeting function, however this is not generally recommended. If a custom targeting scheme is desired, the function handle should follow the form

```
[index,UD] = fn(SD,UD,A,U,V)
```

where `index` should return an ordered list of the integers from one to `SD.k`. The solver will target vectors in the order that their index appears in `index`, unless these vectors have residuals less than $\|A\|$`SD.tol`. `UD` is included to allow the user to maintain any user

```
function [index,UD] = resid_target(SD,UD,A,U,V)
    r = A(U*SD.ur,SD.transp) - V*SD.vr*diag(SD.s);  %Compute residuals
    r_norm = vecnorm(r);                            %Compute norms
    index = find(r_norm > SD.normA*SD.tol);         %Find unconverged triplets
end
```

Algorithm 4.2: Targeting function example

```
function [done,nv,UD] = resid_stop(nv,SD,UD,A,U,V)
    done = 0;
    if SD.k > nv
        r_norm = SD.rn;
        if all(r < SD.normA*SD.tol)
            done = 1;
        end
    end
end
```

Algorithm 4.3: Stopping function example

data generated by the targeting function if desired. The input parameters are described in Section 3.3. We provide an example `target_fn` in Algorithm 4.2.

Similarly, `stop_fn` is available to advanced users who need the functionality presented in Chapter 3. Currently, the only default available is 'resid' which stops by the standard residual criterion (1.5). If a custom function is desired, the function handle should follow the form

```
[done,nv,UD] = fn(nv,SD,UD,A,U,V)
```

where `done` is a completion flag and `nv` should return to the solver the number of desired singular triplets as explained in Section 3.3. An example `stop_fn` is given in Algorithm 4.3.

Chapter 5

Conclusion

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

Second, the lack of good stopping criteria within traditional SVD solvers like PROPACK, severely limit the ability of practitioners to test their theoretical results properly. We have demonstrated a number of stopping criteria and useful heuristics for the singular value decomposition, including those based on the residual norm, Frobenius norm, and thresholding. This work updates a few of the most common low rank criteria to ensure the problem is well-posed when working with iterative SVD solvers. To verify our criteria and heuristics, we have built a flexible interface into our MATLAB implementation of GKD. Through this, we have shown that well crafted stopping functions can provide appropriate low rank solutions with only a minor computational overhead.

Additionally, purpose built criteria such as the ones proposed for matrix completion may significantly improve convergence when applied appropriately as shown in Section 3.5. We have also demonstrated the significant performance gap between the heuristics and theory of previous works like [44] and [9]. This shows that our work on flexible SVD software can provide a significant benefit to both the speed of these applications as well as our theoretical understanding of them.

5.1 Future Research Directions

In the future, further research should be done to incorporate specialized SVD criteria in a variety of different fields including for the kernel ridge regression problem. Due to the large sizes of many low rank approximation problems, it is often crucial to avoid additional computation when possible. Initial testing on low signal to noise ratio problems with our colleagues has shown promising results for stopping SVD calculations based on small singular value gaps as mentioned in Chapter 3.

Similarly, a much deeper study into optimally stopping matrix completion algorithms could prove fruitful, specifically for the SVD heavy APGL algorithm. This includes searching for SVD specific criteria that could provide accurate and fast results, as well as devising additional theory for matrix completion that can explain a few of the results seen in this dissertation. Potentially, this research could be further extended to unconstrained non-smooth convex optimization problems like the ones discussed in [67].

Of course, a high performance implementation of GKD as a standalone package or within an existing package like PRIMME would be an obvious improvement in addition to the work seen in this dissertation. This implementation could also include high performance versions of other popular SVD methods. By having many methods implemented within the same basic framework, research could be done to produce mixed methods where the basis is first built using one method like subspace iteration and then improved (if necessary) using GKD. Alternatively, work could be done to determine the appropriate method and/or block size for a given problem based on a small set of input parameters (e.g. number of desired singular triplets, desired residual tolerance). This would lead to a “black-box” solution for less experienced users.

Lastly, Krylov solvers are known to have difficulty converging to values of the same magnitude (multiplicities) for both the singular value decomposition (nonsymmetric) and the eigenvalue decomposition. When working on problems that do not require high accuracy like low rank approximations, this difficulty is accentuated as solutions with sufficient accuracy are found before finding the multiplicities as seen in Chapter 3. Frequently, the solution to this problem is to utilize a block method where the size of the block is larger than the largest multiplicity in the spectrum of interest. However, if there are many clustered values (also called a near multiplicity), the block size required may be prohibitively large. This can occur at the largest part of the spectrum in low rank approximation, or the lowest part of the spectrum if the matrix is highly singular. It is possible that the use of randomization within general eigensolvers by injecting noise into their recurrences may be able to avoid missing multiple eigenvalues without relying on block methods. This could be done by adding small random perturbations to the expansion vectors at each step. Initial testing has shown that smaller block sizes can reliably find near multiplicities of high degree when adding this noise, but additional work is still needed.

Bibliography

[1] Dimitris Achlioptas and Frank McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM (JACM)*, 54(2):9, 2007.

[2] Advanced Micro Devices Inc. *AMD APP SDK OpenCL Optimization Guide*, August 2015. `https://usermanual.wiki/Pdf/AMDOpenCLProgrammingOptimizationGuide.1224332988` Accessed 2018-02-14.

[3] Orly Alter, Patrick O Brown, and David Botstein. Singular value decomposition for genome-wide expression data processing and modeling. *Proceedings of the National Academy of Sciences*, 97(18):10101–10106, 2000.

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[5] James Baglama, Daniela Calvetti, and Lothar Reichel. IRBL: An implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems. *SIAM Journal on Scientific Computing*, 24(5):1650–1677, 2003.

[6] James Baglama and Lothar Reichel. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM J. Sci. Comput.*, 27(1):19–42, 2005.

[7] James Baglama and Lothar Reichel. Restarted block Lanczos bidiagonalization methods. *Numerical Algorithms*, 43(3):251–272, 2006.

[8] MR Bai and SJ Elliott. Preconditioning multichannel adaptive filtering algorithms using evd-and svd-based signal prewhitening and system decoupling. *Journal of sound and vibration*, 270(4-5):639–655, 2004.

[9] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on optimization*, 20(4):1956–1982, 2010.

[10] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[11] Emmanuel J Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[12] Chandler Davis and William Morton Kahan. The rotation of eigenvectors by a perturbation. iii. *SIAM Journal on Numerical Analysis*, 7(1):1–46, 1970.

[13] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[14] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[15] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[16] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[17] Linwei Fan, Ran Meng, Qiang Guo, Miaowen Shi, and Caiming Zhang. Image denoising by low-rank approximation with estimation of noise energy distribution in svd domain. *IET Image Processing*, 13(4):680–691, 2019.

[18] Arjun Singh Gambhir, Andreas Stathopoulos, and Kostas Orginos. Deflation as a Method of Variance Reduction for Estimating the Trace of a Matrix Inverse. *SIAM Journal on Scientific Computing*, 39(2):A532–A558, 2017.

[19] Steven Goldenberg, Andreas Stathopoulos, and Eloy Romero. A golub–kahan davidson method for accurately computing a few singular triplets of large sparse matrices. *SIAM Journal on Scientific Computing*, 41(4):A2172–A2192, 2019.

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

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

[22] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 4th edition, 2013.

[23] Roger G Grimes, John G Lewis, and Horst D Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 15(1):228–272, 1994.

[24] Ming Gu. Subspace iteration randomization and singular value problems. *SIAM Journal on Scientific Computing*, 37(3):A1139–A1173, 2015.

[25] Qiang Guo, Caiming Zhang, Yunfeng Zhang, and Hui Liu. An efficient svd-based method for image denoising. *IEEE transactions on Circuits and Systems for Video Technology*, 26(5):868–880, 2015.

[26] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[27] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[28] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[29] Michiel E. Hochstenbach. A Jacobi–Davidson type SVD method. *SIAM Journal on Scientific Computing*, 23(2):606–628, 2001.

[30] Walter Hoffmann. Iterative algorithms for gram-schmidt orthogonalization. *Computing*, 41(4):335–348, 1989.

[31] Jinzhi Huang and Zhongxiao Jia. On inner iterations of jacobi-davidson type methods for large svd computations. *to appear in SISC*, 11 2018.

[32] *hypre*: High performance preconditioners. `https://llnl.gov/casc/hypre`, `https://github.com/hypre-space/hypre`.

[33] ZhongXiao Jia and Cen Li. Inner iterations in the shift-invert residual arnoldi method and the jacobi-davidson method. *Science China Mathematics*, 57(8):1733–1752, 2014.

[34] Zhongxiao Jia and Cen Li. Harmonic and refined harmonic shift-invert residual arnoldi and jacobi–davidson methods for interior eigenvalue problems. *Journal of Computational and Applied Mathematics*, 282:83–97, 01 2015.

[35] Zhongxiao Jia and Datian Niu. An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 25(1):246–265, 2003.

[36] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[37] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. *IEEE transactions on information theory*, 56(6):2980–2998, 2010.

[38] Andrew V Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM journal on scientific computing*, 23(2):517–541, 2001.

[39] Rasmus Munk Larsen. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 27(537), 1998.

[40] Nallig Leal, Eduardo Zurek, and Esmeide Leal. Non-local svd denoising of mri based on sparse representations. *Sensors*, 20(5), 2020.

[41] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, volume 6. Siam, 1998.

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

[43] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S. Vetter. NVIDIA tensor core programmability, performance & precision. *CoRR*, abs/1803.04014, 2018.

[44] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research*, 11(Aug):2287–2322, 2010.

[45] James R McCombs and Andreas Stathopoulos. Iterative validation of eigensolvers: a scheme for improving the reliability of Hermitian eigenvalue solvers. *SIAM Journal on Scientific Computing*, 28(6):2337–2358, 2006.

[46] K. Meerbergen and R. Morgan. Inexact methods. In *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst, editors. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[47] Maike Meier and Yuji Nakatsukasa. Fast randomized numerical rank estimation. *arXiv preprint arXiv:2105.07388*, 2021.

[48] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11(1):50–59, 1960.

[49] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems*, pages 1396–1404, 2015.

[50] Yuji Nakatsukasa. Sharp error bounds for ritz vectors and approximate singular vectors. *Mathematics of Computation*, 89(324):1843–1866, 2020.

[51] Luong Trung Nguyen, Junhan Kim, Sangtae Kim, and Byonghyo Shim. Localization of iot networks via low-rank matrix completion. *IEEE Transactions on Communications*, 67(8):5833–5847, 2019.

[52] Y. Notay. Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. *Numer. Lin. Alg. Appl.*, 9:21–44, 2002.

[53] Stanisław Osiński, Jerzy Stefanowski, and Dawid Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent information processing and web mining*, pages 359–368. Springer, 2004.

[54] Peter Parker, Patrick J Wolfe, and Vahid Tarokh. A signal processing application of randomized low-rank approximations. In *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, pages 345–350. IEEE, 2005.

[55] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, 1980.

[56] David Persson, Alice Cortinovis, and Daniel Kressner. Improved variants of the hutch++ algorithm for trace estimation. *arXiv preprint arXiv:2109.10659*, 2021.

[57] HS Prasantha, HL Shashidhara, and KN Balasubramanya Murthy. Image compression using svd. In *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on*, volume 3, pages 143–145. IEEE, 2007.

[58] Louis L Scharf. The svd and reduced rank signal processing. *Signal processing*, 25(2):113–133, 1991.

[59] Horst D Simon and Hongyuan Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM Journal on Scientific Computing*, 21(6):2257–2274, 2000.

[60] Nathan Srebro, Noga Alon, and Tommi S Jaakkola. Generalization error bounds for collaborative prediction with low-rank matrices. In *Advances In Neural Information Processing Systems*, pages 1321–1328, 2005.

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

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

[63] Andreas Stathopoulos. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. part i: Seeking one eigenvalue. *SIAM J. Sci. Comput.*, 29(2):481–514, 2007.

[64] Andreas Stathopoulos and James R. McCombs. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Transactions on Mathematical Software*, 37(2):21:1–21:30, 2010.

[65] Andreas Stathopoulos and Kesheng Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM Journal on Scientific Computing*, 23(6):2165–2182, 2002.

[66] Arthur Szlam, Yuval Kluger, and Mark Tygert. An implementation of a randomized algorithm for principal component analysis. *arXiv preprint arXiv:1412.3510*, 2014.

[67] Kim-Chuan Toh and Sangwoon Yun. An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems. *Pacific Journal of optimization*, 6(615-640):15, 2010.

[68] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991. PMID: 23964806.

[69] Shashanka Ubaru and Yousef Saad. Fast methods for estimating the numerical rank of large matrices. In *International Conference on Machine Learning*, pages 468–477. PMLR, 2016.

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

[71] Kesheng Wu and Horst Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.

[72] Lingfei Wu, Eloy Romero, and Andreas Stathopoulos. PRIMME_SVDS: A high-performance preconditioned SVD solver for accurate large-scale computations. *arXiv preprint arXiv:1607.01404*, 2016.

[73] Lingfei Wu and Andreas Stathopoulos. A preconditioned hybrid SVD method for accurately computing singular triplets of large matrices. *SIAM Journal on Scientific Computing*, 37(5):S365–S388, 2015.

[74] Qiong Wu, Felix Ming Fai Wong, Zhenming Liu, Yanhua Li, and Varun Kanade. Adaptive reduced rank regression. *arXiv preprint arXiv:1905.11566*, 2019.

[75] Bendraou Youssef, Essannouni Fedwa, Aboutajdine Driss, and Salam Ahmed. Shot boundary detection via adaptive low rank and svd-updating. *Computer Vision and Image Understanding*, 161:20–28, 2017.

[76] Peng Zhang and Yuxiang Gao. Matrix multiplication on high-density multi-GPU architectures: theoretical and experimental investigations. In *International Conference on High Performance Computing*, pages 17–30. Springer, 2015.