Recycling Techniques for Sequences of Linear Systems and Eigenproblems

Arielle Katherine Grim Carr

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematics

Eric de Sturler, Chair
Julianne Chung
Mark Embree
Serkan Gugercin

June 23, 2021
Blacksburg, Virginia

Keywords: sequences of linear systems, sequences of eigenvalue problems, recycling preconditioners, Krylov subspace recycling methods, Krylov-Schur

Recycling Techniques for Sequences of Linear Systems and Eigenproblems

Arielle Katherine Grim Carr

(ABSTRACT)

Sequences of matrices arise in many applications in science and engineering. In this thesis we consider matrices that are closely related (or closely related in groups), and we take advantage of the small differences between them to efficiently solve sequences of linear systems and eigenproblems. Recycling techniques, such as recycling preconditioners or subspaces, are popular approaches for reducing computational cost. In this thesis, we introduce two novel approaches for recycling previously computed information for a subsequent system or eigenproblem, and demonstrate good results for sequences arising in several applications.

Preconditioners are often essential for fast convergence of iterative methods. However, computing a good preconditioner can be very expensive, and when solving a sequence of linear systems, we want to avoid computing a new preconditioner too often. Instead, we can recycle a previously computed preconditioner, for which we have good convergence behavior of the preconditioned system. We propose an update technique we call the sparse approximate map, or SAM update, that approximately maps one matrix to another matrix in our sequence. SAM updates are very cheap to compute and apply, preserve good convergence properties of a previously computed preconditioner, and help to amortize the cost of that preconditioner over many linear solves.

When solving a sequence of eigenproblems, we can reduce the computational cost of constructing the Krylov space starting with a single vector by warm-starting the eigensolver with a subspace instead. We propose an algorithm to warm-start the Krylov-Schur method using a previously computed approximate invariant subspace. We first compute the approximate Krylov decomposition for a matrix with minimal residual, and use this space to warm-start the eigensolver. We account for the residual matrix when expanding, truncating, and deflating the decomposition and show that the norm of the residual monotonically decreases. This method is effective in reducing the total number of matrix-vector products, and computes an approximate invariant subspace that is as accurate as the one computed with standard Krylov-Schur. In applications where the matrix-vector products require an implicit linear solve, we incorporate Krylov subspace recycling.

Finally, in many applications, sequences of matrices take the special form of the sum of the identity matrix, a very low-rank matrix, and a small-in-norm matrix. We consider convergence rates for GMRES applied to these matrices by identifying the sources of sensitivity.

Recycling Techniques for Sequences of Linear Systems and Eigenproblems

Arielle Katherine Grim Carr

(GENERAL AUDIENCE ABSTRACT)

Problems in science and engineering often require the solution to many linear systems, or a sequence of systems, that model the behavior of physical phenomena. In order to construct highly accurate mathematical models to describe this behavior, the resulting matrices can be very large, and therefore the linear system can be very expensive to solve. To efficiently solve a sequence of large linear systems, we often use iterative methods, which can require preconditioning techniques to achieve fast convergence. The preconditioners themselves can be very expensive to compute. So, we propose a cheap update technique that approximately maps one matrix to another in the sequence for which we already have a good preconditioner. We then combine the preconditioner and the map and use the updated preconditioner for the current system.

Sequences of eigenvalue problems also arise in many scientific applications, such as those modeling disk brake squeal in a motor vehicle. To accurately represent this physical system, large eigenvalue problems must be solved. The behavior of certain eigenvalues can reveal instability in the physical system but to identify these eigenvalues, we must solve a sequence of very large eigenproblems. The eigensolvers used to solve eigenproblems generally begin with a single vector, and instead, we propose starting the method with several vectors, or a subspace. This allows us to reduce the total number of iterations required by the eigensolver while still producing an accurate solution.

We demonstrate good results for both of these approaches using sequences of linear systems and eigenvalue problems arising in several real-world applications.

Finally, in many applications, sequences of matrices take the special form of the sum of the identity matrix, a very low-rank matrix, and a small-in-norm matrix. We examine the convergence behavior of the iterative method GMRES when solving such a sequence of matrices.

For Mom and Dad

Acknowledgments

I would first like to thank my advisor and chair, Eric de Sturler, most importantly for introducing me to the awes of research nearly ten years ago as an undergraduate student, and for his continued mentorship since then. I am humbled by his knowledge and high level of mathematical rigor and am appreciative to have been able to study under him.

I would also like to thank my committee members Julianne Chung, Mark Embree, and Serkan Gugercin. Their passion for research and enthusiastic teaching styles are infectious, and had a huge impact on my own desire to pursue a career in academia.

I am also deeply grateful for the support of my family and friends and am profoundly lucky enough to say there are too many to name, but a few deserve specific acknowledgement. To my mom, Adrienne: In every facet of my life, including through my PhD, you never allowed me to give up. Thank you for always encouraging me to be my best. To my dad, Michael: My love of math grew from seeds you planted, and you were always the first person I wanted to tell when I got an A. I believe this thesis serves as proof that I have, in fact, overdelivered. To my husband, Trevor: There are very few words to express how appreciative I am of your love and unconditional support, especially over the last year as I worked to finish my research and this thesis. It would have been impossible without you. To my brother and sister, Tucker and Sydney: From office hours to Friday dinners, having you at Tech made Blacksburg feel even more like home, and provided comfort during some of the more arduous times in my program.

Finally, to Thorn: You were with me for every major life event over the past 15 years, and this is the first one I will go through without you by my heel. It is a bittersweet moment, but I know I will see you at the rainbow bridge, dude.

This work was supported by the National Science Foundation under grant numbers NSF-DMS 172305, NSF-DMS 10-25327, and NSF-DMS 12-17156; and by the Air Force Office of Scientific Research under grant number AFOSR-BRI FA9550-17-1-0205.
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the Air Force Office of Scientific Research.

Contents

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Overview of Contributions . . . 2

1.2 Applications . . . 3

1.3 Background . . . 4

1.3.1 Krylov Subspace Methods . . . 5

1.3.2 Krylov Subspace Recycling Methods . . . 7

1.3.3 Numerical Experiments . . . 10

1.3.4 Summary . . . 13

2 Preconditioning Parametrized Linear Systems 15

2.1 Introduction . . . 16

2.2 Recycling Preconditioners . . . 18

2.3 Experimental Analysis of Sparsity Patterns for SAMs . . . 23

2.4 Implementation . . . 28

2.5 Numerical Experiments . . . 31

2.5.1 Topology Optimization . . . 31

2.5.2 Interpolatory Model Reduction . . . 35

2.5.3 Indefinite Matrices . . . 40

2.6 Conclusions and Future Work . . . 41
2.7 Acknowledgements . . . 43

3 Subspace Recycling for a Sequence of Eigenproblems 44
3.1 Introduction . . . 44
3.2 Background . . . 46
3.3 Warm-Start Krylov-Schur for a Sequence of Eigenproblems . . . 50
3.3.1 Computing the Best Approximate Krylov Decomposition . . . 50
3.3.2 Expanding, Truncating, and Deflating an Approximate Krylov Decomposition . . . 52
3.3.3 Computing Approximate Eigenpairs . . . 55
3.3.4 Properties of Warm-Start Krylov-Schur . . . 59
3.4 Numerical Simulation of Brake Squeal . . . 60
3.4.1 Krylov Subspace Recycling When Solving Generalized Eigenvalue Problems . . . 62
3.5 Numerical Experiments . . . 62
3.6 Conclusions . . . 70
3.7 Acknowledgements . . . 71

4 An Analysis of Low-Rank and Small-in-Norm Perturbations of the Identity Matrix 72
4.1 Introduction . . . 72
4.2 GMRES Convergence for Perturbed Identity Plus Low-Rank Coefficient Matrices . . . 73
4.3 Sequences of Matrices of the Form $\mathbf{I} + \mathbf{L} + \mathbf{K}$ . . . 81
4.3.1 Slater Matrices for Insulators . . . 81
4.3.2 Reusing Preconditioners for Slater Matrices . . . 83
4.4 Conclusions . . . 84

5 Conclusions 85
5.1 Attributions . . . 86

Bibliography **87**

Appendices **99**

Appendix A ILUT(P) Implementation **100**

A.1 ILUT Pseudocode 100

A.2 ILUTP Modifications to ILUT Pseudocode 104

Appendix B Nonlinear Convection-Diffusion Discretization **105**

B.1 Matlab Implementation 105

List of Figures

1.1 Harmonic Ritz values and and exact interior eigenvalues of a matrix from a sequence of linear systems arising in a transient hydraulic tomography application [14]. 8

1.2 Iteration counts for each Newton step when recomputing, reusing, and recycling the preconditioner using GMRES and GCRO-DR when solving the nonlinear convection-diffusion equation. 12

1.3 Iteration counts when recomputing, reusing, and recycling the preconditioner using GMRES and GCRO-DR for the discretized Euler equations from SENSEI [59, 94]. 13

2.1 Relative SAM residual, $\frac{\|\mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0\|_F}{\|\mathbf{A}_0\|_F}$, of the flow matrices using a priori sparsity patterns for the SAM updates, $\mathbf{N}_k$, for batch/shift 1/3 through 2/6. An ILUTP preconditioner is computed for batch/shift 1/2. Details of these linear systems and the notation are provided in Sections 2.3 and 2.5.2. 24

2.2 Descriptions of the finite element mesh and sparsity patterns for the topology optimization matrices. 28

2.3 Eigenvalues of preconditioned system 1/2, $\mathbf{A}_1\mathbf{P}_1$, with the preconditioner recomputed. 37

2.4 Eigenvalues of preconditioned system 1/5 with (a) $\mathbf{P}_4$ recomputed, (b) $\mathbf{P}_4 = \mathbf{N}_4\mathbf{P}_1$ (recycled), and (c) $\mathbf{P}_1$ reused. *Note the different scale along the real axis compared with Figure 2.3 and 2.5.* 37

2.5 Eigenvalues of preconditioned system 1/6 with (a) $\mathbf{P}_5$ recomputed, (b) $\mathbf{P}_5 = \mathbf{N}_5\mathbf{P}_1$ (recycled), and (c) $\mathbf{P}_1$ reused. *Note the different scale along the real axis compared with Figure 2.3 and 2.4.* 40

2.6 GMRES convergence and selected eigenvalues for a discretized Helmholtz problem. 42

3.1 The three unstable eigenvalues for $\mathcal{A}$ corresponding to $\Omega = 4\pi$. 65

3.2 Comparing the quality of the invariant subspace computed for $numev = 50$. 67

3.3 Comparing the quality of the invariant subspace computed for $numev = 10$. 67

3.4 The 100 smallest (in magnitude) eigenvalues of $\mathbf{K}_{\tau_i}$ for the industrial model. 69

3.5 Cosines of the canonical angles between the approximate invariant subspace computed when using the current method and when using warm-start Krylov-Schur with the large and small GCRO-DR convergence tolerances ("WSKS large" and "WSKS small", respectively) for $\mathcal{A}_{\tau_1}$ and when using Krylov-Schur for $\mathcal{A}$. Figure 3.5(b) enlarges the plot for the cosines of the largest five canonical angles for $\mathcal{A}$ and $\mathcal{A}_{\tau_1}$ when using the smaller GCRO-DR convergence tolerance. 71

4.1 Pseudospectrum of a rank-two matrix with dimension $n = 25$. The color-bar on the right gives the δ value for the corresponding $\sigma_\delta(\mathbf{I} + \mathbf{L})$ shown on the left. The black dots indicate the eigenvalues of $\mathbf{I} + \mathbf{L}$. 76

4.2 $\|\mathbf{r}_m\|_2$, $\|\boldsymbol{\rho}_m\|_2$, and the upper bound (4.7) on $\|\mathbf{r}_m\|_2$ for $\delta = 10^{-1}, 10^{-2}, 10^{-3}$, corresponding to $\sigma_\delta(\mathbf{I} + \mathbf{L})$ in Figure 4.1. 76

4.3 $\mathbf{I}+\mathbf{L}$ with dimension $n = 25$, $\text{rank}(\mathbf{L}) = 5$, and two ill-conditioned eigenvalues of $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$. 79

4.4 $\mathbf{I} + \mathbf{L}$ with dimension $n = 25$, $\text{rank}(\mathbf{L}) = 5$, and two small angles between $\mathbf{U}_1$ and $\mathbf{V}_2$. 80

4.5 Sparsity pattern of a typical Slater matrix for a system with 1024 particles [4]. 82

4.6 GMRES iterations for Slater matrices when reusing an initial ILUTP preconditioner with drop tolerances 10^{-6}, 10^{-4}, and 10^{-2}. $\|\mathbf{S}\|_2$ indicates the small-in-norm perturbations ($\|\mathbf{S}\|_2 = 10^{-3}$, 10^{1}, 10^{2}, respectively). 84

List of Tables

1.1 Total iterations when recomputing, reusing, and recycling the preconditioner using GMRES and GCRO-DR with Newton's method for solving the nonlinear convection-diffusion equation. 11

1.2 Total iterations when recomputing and recycling the preconditioner using GMRES and GCRO-DR for the discretized Euler equations from SENSEI [59, 94]. 13

2.1 GMRES iterations and total run times for selected shifts of the flow matrices using a priori sparsity patterns for the SAM update. An ILUTP preconditioner is computed for the system 1/2, with SAM updates computed for all remaining shifts 1/3-2/6. "Patt" indicates ""Pattern of". "SAM time" is the total amount of time spent computing all ten maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 0.8 s to compute. Totals are given for the entire sequence, as well as for the sequence omitting system 2/1. (5001) indicates GMRES did not converge in the maximum allowed iterations. 25

2.2 GMRES iterations and total run times for selected shifts of the flow matrices using a priori *sparsified* sparsity patterns. A global threshold of 10^{-2} is used to sparsify. An ILUTP preconditioner is computed for system 1/2, with SAM updates computed for all remaining shifts 1/3-2/6 "Sp Patt" indicates "Sparsified Pattern". "SAM time" is the total amount of time spent computing all maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 0.8 s to compute. Totals are given for the entire sequence, as well as the sequence omitting system 2/1. (5001) indicates GMRES did not converge in the maximum allowed iterations. 26

2.3 GMRES iterations and total run times for matrices from selected steps of the topology optimization application using a priori mesh-based sparsity patterns. "SAM time" is the total amount of time spent computing all maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 122.41s to compute. 29

2.4 Run times and iterations for the large topology optimization systems, when reusing the AMG preconditioner for each system after the first. When max iterations (400) is reached, we recompute the AMG preconditioner for the next system and reuse this for subsequent systems. 33

2.5 Run times and iterations for the large topology optimization systems, when computing the AMG preconditioner for the first system and a SAM update for each system after the first. SAMs use the sparsity pattern Patt-1. 33

2.6 Run times and iterations for the small topology optimization systems with the initial ILUTP reused for all systems. Results are given in groups of ten except for (150-166). **For only one system convergence was reached within max iterations. *Convergence was not reached for any system within max iterations. 34

2.7 Run times and iterations for the small topology optimization systems with the initial ILUTP recycled with a SAM update for every tenth system, using Patt-1. Results are given in groups of ten except for (150-166). "Gain" indicates the decrease in GMRES time and iterations compared with reusing the initial ILUTP. There are two SAM updates in (150-160). *For one matrix convergence was not reached within max iterations. 34

2.8 Run times and iterations for the small topology optimization matrices with the dynamic strategy for ILUTP only. Each group of systems starts with a new ILUTP, as indicated in the last column. 35

2.9 Run times and iterations for the small topology optimization matrices with the dynamic strategy for both ILUTP and SAM updates with Patt-1. Each group of systems starts with a new ILUTP, and includes one SAM update, as indicated in the last two columns. 35

2.10 Shifts for the flow Matrices. For subsequent tables, B/S = Batch/Shift Number. 38

2.11 Run times and iterations for the flow matrices with ILUTP recomputed for each shift. 38

2.12 Run times and iterations for the flow matrices with ILUTP computed for the first shift and reused for each remaining shift. 38

2.13 Run times and iterations for the flow matrices with ILUTP computed for the first system and SAM updates computed for all other shifts. 38

2.14 Run times and iterations for the flow matrices with ILUTP computed for the first two systems and SAM updates computed for all other shifts. 39

2.15 Run times and iterations for the flow matrices with ILUTP computed for the first two systems, and $\mathbf{P}_1$ reused for all remaining shifts. 39

2.16 Timings for flow matrices with ILUTP computed for the first two systems and SAM updates computed for selected shifts. 39

3.1 Condition number of $\mathcal{A}_1$ and $\mathcal{A}_2$ from the full sequence, along with the corresponding shifted matrices, $\mathcal{A}_{j_{\tau_i}}$ for $j = 1, 2$ and $i = 1, 2, 3$. 64

3.2 Total GCRO-DR iterations for each matrix-vector product in Krylov-Schur, with the first five matvecs highlighted. "Total Iters" denotes the total number of GCRO-DR iterations for all calls to the iterative solver for all cycles of Krylov-Schur; "Total Matvecs" denotes the total number of matrix-vector products computed for all cycles of the eigensolver; and "Avg Iter" denotes the average number of iterations per matrix-vector product. Note that the first five matrix-vector products highlighted are computed during the expansion phase of the Krylov-Schur method as in (3.6). 66

3.3 Total number of GCRO-DR iterations for each matrix-vector product in Krylov-Schur for $\mathcal{A}$ and warm-start Krylov-Schur for $\mathcal{A}_{\tau_i}$, with the first five matvecs highlighted. "Total Iters" denotes the total number of GCRO-DR iterations for all calls to the iterative solver for all cycles of Krylov-Schur; "Total Matvecs" denotes the total number of matrix-vector products computed for all cycles of the eigensolver; and "Avg Iter" denotes the average number of iterations per matrix-vector product. Note that the first five matrix-vector products highlighted are computed as in (3.15) when we build the approximate Krylov-Schur decomposition with minimum norm residual. 66

3.4 Summary of the total iterative solver iterations and eigensolver iterations when applying (from left to right) the strategies (1)-(5) for the full sequence. 68

3.5 Comparison of iterations for seven select matrices when using strategies (3)-(5). 68

3.6 Summary of the total iterations when using (from left to right) a larger convergence tolerance for GCRO-DR (with the smaller convergence tolerance for WSKS) and a larger convergence tolerance for WSKS (with the smaller convergence tolerance for GCRO-DR). 69

3.7 Total GCRO-DR iterations for each matrix-vector product in the eigensolver, with the first five matvecs highlighted. "Total GCRO-DR Iterations" denotes the total number of GCRO-DR iterations for all calls to the iterative solver for all cycles of the eigensolver; "Total Matvecs in Eigensolver" denotes the total number of matrix-vector products required; and "Average Iterations" denotes the average number of iterations per matrix-vector product. We use the smaller convergence tolerance, 10^{-10}, for GCRO-DR. 70

4.1 Values of C_3 for different δ for $\mathbf{I} + \mathbf{L}$ defined below. 75

Chapter 1

Introduction

Sequences of linear systems and eigenvalue problems are ubiquitous in scientific applications and many involve large, sparse matrices. In this thesis, we are interested in sequences of matrices that change slowly, and so we aim to attain fast solutions by exploiting small changes between matrices using two approaches: (1) recycling preconditioners and (2) recycling search spaces.

For (1), we consider recycling preconditioners for both the general case of sequences of linear systems

$$\mathbf{A}(\mathbf{p}_k)\mathbf{x}_k = \mathbf{b}_k \tag{1.1}$$

as well as the important special case of the type $(s_k\mathbf{E} + \mathbf{A})\mathbf{x}_k = \mathbf{b}_k$. Preconditioners are generally essential for fast convergence in the iterative solution of linear systems of equations when using Krylov subspace methods. However, computing a good preconditioner can be expensive and when solving a sequence of many systems, we want to avoid the potentially high cost of recomputing a new preconditioner too often. In this case, it can be advantageous to recycle (or update and reuse) a previously computed preconditioner for subsequent matrices in the sequence. This allows us to maintain the good convergence properties of this preconditioner while amortizing its cost over many linear solves.

Recycling Krylov subspaces from previous systems is a complementary method to recycling preconditioners for reducing computational cost when solving a sequence of linear systems. For (2), we extend the idea of subspace recycling to a sequence of eigenproblems. When computing invariant subspaces for sequences of eigenvalue problems, the invariant subspace for one is likely a good approximation to that of another in the sequence. Rather than discarding the approximate invariant space generated for one system, we can recycle a subspace of this previously computed approximate invariant subspace to warm-start the eigensolver for the next, closely related system.

For each of the objectives, (1) and (2) above, we introduce a novel recycling approach, using respectively:

1. A sparse approximate mapping (SAM) technique for updating preconditioners for sequences of closely related systems [49]; and

2. A warm-started Krylov-Schur algorithm with convergence testing for computing the invariant subspaces for a sequence of eigenvalue problems.

Throughout this thesis, we will use boldface capital letters to denote matrices (e.g., $\mathbf{A}$) and boldface lowercase letters to denote vectors (e.g., $\mathbf{x}$).

1.1 Overview of Contributions

When solving a **sequence of linear systems**, preconditioners are often essential for fast convergence of the linear solver, but can be expensive to compute. To reduce this cost, we can reuse a previously computed preconditioner; however, as the matrices become less closely related, convergence of the linear solver may slow. Instead, when solving a long sequence of linear systems, it can be lucrative to recycle preconditioners. In Chapter 2, we introduce a preconditioner update technique, the SAM update, which is a simple and effective strategy for amortizing the potentially high cost of computing a preconditioner over many linear solves. In particular, we compute a map from a new matrix to another matrix for which a good preconditioner has already been computed; in other words, the linear solver converges quickly. We combine the map with this previously computed preconditioner to define the recycled preconditioner for the current system. Our update technique is independent from the initial preconditioner, and is cheap to compute and apply for judiciously chosen a priori sparsity patterns. So, the SAM update allows the user flexibility in choosing the type and quality of the preconditioner.

In our results in Chapter 2, we show that for several applications, SAM updates applied to every matrix, or to select matrices, can reduce total computational cost compared with recomputing and reusing the preconditioner. We also show that for nondiagonally dominant and indefinite matrices, the SAM update can be used to construct an effective preconditioner, where computing a preconditioner from scratch may be very expensive, or even impossible. Since we use timings to compare the different strategies of recomputing, reusing, and recycling preconditioners, we compare runtimes for interpreted m-files for SAMs, for ILUTP, and for an algebraic multigrid (AMG) preconditioner. In Section 2.4, we give the algorithms to preprocess the sparsity pattern and to compute the SAM update. In Appendix A, we provide the algorithm for efficient MATLAB® implementations of Saad's ILUT and ILUTP algorithms [118, 119]. Finally, the AMG preconditioner implementation used in Chapter 2 is from [87].

Chapter 2 appears in [A. Carr, E. de Sturler, and S. Gugercin, *Preconditioning parametrized linear systems*, SIAM Journal on Scientific Computing, 43 (2021), pp. A2242-A2267].

In Chapter 3, we introduce the warm-start Krylov-Schur algorithm for solving a **sequence of eigenvalue problems**. This recycling strategy makes solving very large-scale eigenproblems tractable by reducing the total number of matrix-vector products computed in the eigensolver. We modify the Krylov-Schur algorithm to begin with a subspace rather than a single vector, and while any subspace can be used, we propose using a previously computed approximate invariant subspace for a nearby system in the sequence. Our warm-start Krylov-Schur algorithm begins by computing the approximate Krylov decomposition for a matrix with minimum norm residual based on the analysis in [132], which introduces a residual matrix to this (quasi) Krylov-Schur decomposition. We show that we achieve a monotonic decrease in the norm of the residual when expanding, truncating, and deflating the decomposition.

Additionally, for those eigenproblems whose matrix-vector products require an implicit linear solve, we incorporate a Krylov subspace recycling method to our warm-start Krylov-Schur algorithm. Iterative methods allow us to avoid the potentially expensive cost of a direct solve, and we show that Krylov subspace recycling can improve convergence. In the numerical results in Chapter 3, we highlight an application arising in the numerical simulation of brake squeal [76], and demonstrate good results for two sequences arising in this application. We provide background on Krylov methods and Krylov subspace recycling methods of GMRES-type at the end of the current chapter (see Section 1.3).

Finally, in many applications, sequences of linear systems arise where the coefficient matrix takes the form $\mathbf{I} + \mathbf{L} + \mathbf{S}$ (where $\mathbf{I}$ is the identity, $\text{rank}(\mathbf{L}) = p \ll n$, $\|\mathbf{S}\|_2$ is small, but $\|\mathbf{L}\|_2$ may be large). In Chapter 4, we discuss the GMRES convergence properties for sequences of matrices of the form $\mathbf{I} + \mathbf{L} + \mathbf{S}$. In particular, we identify those eigenvalues of $\mathbf{I} + \mathbf{L}$ that are sensitive to the introduction of a small-in-norm perturbation (e.g., $\mathbf{S}$).

1.2 Applications

Our approaches make solving sequences of large-scale linear systems and eigenvalue problems tractable, where direct methods are otherwise too computationally expensive. They are also generalizable to *any* sequence of linear systems or standard eigenproblems. Throughout this thesis, we highlight the benefits of the proposed recycling techniques using several applications, and provide a list of those applications considered with forward reference to the sections in which they appear:

- Discretized 2D nonlinear convection-diffusion equations [101, 102], discussed in Section 1.3.3;
- Discretized Euler equations [59, 94], discussed in Section 1.3.3;

- Topology optimization [22, 24, 114, 145], discussed in Sections 2.3 and 2.5.1;
- Model reduction, in particular in the Iterative Rational Krylov Algorithm (IRKA) [7, 8, 81], discussed in Sections 2.3 and 2.5.2;
- Discretized 2D Helmholtz equations [41, 67, 68], discussed in Section 2.5.3;
- Numerical simulation of disk brake squeal [76], discussed in Section 3.5; and
- Slater matrices arising in a Quantum Monte Carlo method [4], discussed in Section 4.3.1.

1.3 Background

In this section, we summarize properties of iterative methods of GMRES-type that are used in this thesis. In Chapter 2, we compare computational cost using GMRES when comparing our preconditioning update strategy with reusing and recomputing the preconditioner. In Chapter 3, we incorporate the Krylov subspace recycling technique GCRO-DR in our proposed algorithm for solving a sequence of eigenproblems. Finally, in Chapter 4, we consider the effect of sensitive eigenvalues on GMRES convergence. So, the discussion and results given in this chapter serve to introduce the methods we use in this thesis, as well as to highlight the effectiveness of subspace recycling and preconditioner recycling. We consider linear systems of the form

$$\mathbf{A}^{(\ell)}\mathbf{x}^{(\ell)} = \mathbf{b}^{(\ell)}, \tag{1.2}$$

for $\ell = 1, 2, \ldots, N$ and $\mathbf{A}^{(\ell)} \in \mathbb{C}^{n\times n}$. In (1.2), the coefficient matrix may be parametrized, and so we can write our sequence as

$$\mathbf{A}(p^{(\ell)})\mathbf{x}^{(\ell)} = \mathbf{b}^{(\ell)}. \tag{1.3}$$

In Chapter 2, we introduce the SAM update [49], a preconditioner recycling strategy that reduces total computational cost when preconditioning a sequence of linear systems. The SAM update is cheap to compute and approximately maps one system to another, closely related system in the sequence for which we have already computed a good preconditioner. This allows us to amortize the potentially high cost of an initial or previous preconditioner over the sequence of linear systems.

Recycling Krylov subspaces from previous systems is another effective recycling strategy that can help to reduce computational cost when solving a sequence of systems. Rather than discarding the spaces generated for one system, we can select a subspace (or the recycle space) to use for another nearby system in order to speed up convergence. In this chapter, we

show that Krylov subspace recycling alone, and in combination with preconditioner recycling, can improve convergence of the iterative method.

In Section 1.3.1, we provide background on GMRES-type Krylov subspace methods, specifically highlighting two minimum residual norm Krylov methods: the generalized minimum residual (GMRES) [122] and generalized conjugate residual (GCR) [63] methods. In Section 1.3.2, we discuss related Krylov subspace recycling methods, and specifically the GMRES-DR [108] and GCRO-DR [110] methods. We refer the reader to [77, 120, 141] for more details on Krylov methods, and to [128] (and references therein) for a survey of Krylov subspace recycling techniques.

In Section 1.3.3, we give results using both Krylov subspace recycling and preconditioner recycling applied to a sequence of matrices arising in the numerical solution of a discretized nonlinear convection-diffusion equation, and another sequence of systems arising in the numerical solution of the Euler equations for flow over an airfoil (i.e., the wing of a plane). In both applications, we show that the combination of preconditioner recycling using SAMs and Krylov subspace recycling using GCRO-DR reduces total iterations compared with reusing the initial preconditioner and using GMRES for the iterative solver.

Throughout this section, we will use the notation $\mathbf{A}^{(\ell)}$ to denote the ℓ^{th} matrix in a sequence, with corresponding solution and right hand side vector, $\mathbf{x}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$, respectively. We use $\mathbf{x}_j$ to represent the approximate solution at the j^{th} step of the iterative solver.

1.3.1 Krylov Subspace Methods

A Krylov subspace method iteratively computes an approximate solution $\mathbf{x}_j$ at iteration j for the linear system $\mathbf{Ax} = \mathbf{b}$ by updating the initial solution $\mathbf{x}_0$ as $\mathbf{x}_j = \mathbf{x}_0 + \mathbf{z}_j$, with $\mathbf{z}_j \in \mathcal{K}^j(\mathbf{A}, \mathbf{v})$. Here,

$$\mathcal{K}^j(\mathbf{A}, \mathbf{v}) = \text{Span}(\{\mathbf{v}, \mathbf{Av}, \mathbf{A}^2\mathbf{v}, \ldots, \mathbf{A}^{j-1}\mathbf{v}\}) \tag{1.4}$$

is the Krylov space of dimension j for $\mathbf{A} \in \mathbb{C}^{n\times n}$ and $\mathbf{v} \in \mathbb{C}^n$. The update $\mathbf{z}_j$ comes from a projection onto the Krylov space, and its computation is specific to the method. Next, we briefly review two Krylov subspace methods that use the *minimum norm residual approach*: GMRES [122] and GCR [63]. In [141], three other classes of Krylov subspace methods besides minimal residual approaches are considered, including *Ritz-Galerkin approaches*, *Petrov-Galerkin approaches*, and *minimum norm error approaches*.

Minimum Norm Residual Approaches

For linear solvers, in (1.4) $\mathbf{v} = \mathbf{r}_0$, where $\mathbf{r}_0$ is the initial residual and is defined as

$$\mathbf{r}_0 = \mathbf{Ax}_0 - \mathbf{b}. \tag{1.5}$$

Minimum norm residual methods minimize

$$\|\mathbf{r}_j\|_2 = \|\mathbf{b} - \mathbf{A}\mathbf{x}_j\|_2, \tag{1.6}$$

which is equivalent to choosing $\mathbf{z}_j \in \mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$ such that $\mathbf{r}_j \perp \mathcal{K}^j(\mathbf{A}, \mathbf{A}\mathbf{r}_0)$. We highlight two minimum norm residual methods and, specifically, how each constructs a basis for the Krylov space and the subsequent updated approximate solution.

GMRES

GMRES [122] builds an orthonormal basis for $\mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$, spanned by the columns of $\mathbf{V}_j$ constructing the Arnoldi recurrence

$$\mathbf{A}\mathbf{V}_{j-1} = \mathbf{V}_j\mathbf{H}_{j,j-1} = \mathbf{V}_{j-1}\mathbf{H}_{j-1,j-1} + \mathbf{v}_j\mathbf{h}_j^*, \tag{1.7}$$

where $\mathbf{H}_{j,j-1}$ is an upper Hessenberg matrix and $\mathbf{h}_j^* = h_{j,j-1}\mathbf{e}_j^*$. Equation (1.7) is generated using the Arnoldi method [11], and is referred to as the *Arnoldi decomposition of order* $j-1$. GMRES then minimizes the (small) least squares problem of the form

$$\widetilde{\mathbf{y}} = \min_{\mathbf{y}} \left\| \|\mathbf{r}_0\|_2\mathbf{e}_1 - \mathbf{H}_{j,j-1}\mathbf{y} \right\|_2, \tag{1.8}$$

which can be efficiently solved by computing the QR factorization of $\mathbf{H}_{j,j-1} = \mathbf{Q}_{j,j-1}\mathbf{R}_{j-1,j-1}$ using Givens or Householder transformations [74, 142]. In particular, GMRES computes the updated solution $\mathbf{x}_j$ as

$$\mathbf{x}_j = \mathbf{x}_0 + \mathbf{V}_{j-1}\widetilde{\mathbf{y}} = \mathbf{x}_0 + \mathbf{V}_{j-1}\mathbf{R}_{j-1,j-1}^{-1}\mathbf{Q}_{j,j-1}^*\|\mathbf{r}_0\|_2\mathbf{e}_1,$$

with the corresponding updated minimum residual

$$\mathbf{r}_j = \mathbf{r}_0 - \mathbf{V}_j\mathbf{Q}_{j,j-1}\mathbf{Q}_{j,j-1}^*\|\mathbf{r}_0\|_2\mathbf{e}_1.$$

GCR

GCR [63] constructs matrices $\mathbf{U}_j, \mathbf{C}_j \in \mathbb{C}^{n\times j}$ such that the columns of $\mathbf{U}_j$ span $\mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$ and

$$\mathbf{C}_j = \mathbf{A}\mathbf{U}_j, \tag{1.9}$$

where

$$\mathbf{C}_j^*\mathbf{C}_j = \mathbf{I}, \tag{1.10}$$

with $\mathbf{I}$ the identity matrix of dimension j. The updated residual is computed as

$$\mathbf{r}_j = \left(\mathbf{I} - \mathbf{C}_j\mathbf{C}_j^*\right)\mathbf{r}_0,$$

and the updated solution is

$$\mathbf{x}_j = \mathbf{x}_0 + \mathbf{U}_j \mathbf{C}_j^* \mathbf{r}_0.$$

GCR and GMRES are algebraically equivalent where $\mathbf{C}_j = \mathbf{V}_j \mathbf{Q}_{j,j-1}$ and $\mathbf{U}_j = \mathbf{V}_{j-1} \mathbf{R}_{j-1,j-1}^{-1}$ [57, 122]. However, GMRES requires less storage since it stores only one matrix $\mathbf{V}_{1:j}$, and is more robust than GCR [57, 122]. However, as the Krylov subspace grows larger, both GCR and GMRES can become very expensive. We can limit this cost by restarting the method; or, in the case of GCR, by only orthogonalizing against the most recent search directions. Truncated GMRES-like methods are also discussed in [120]. However, these strategies require discarding information from previous iterations and this loss of information can be avoided using Krylov subspace recycling.

1.3.2 Krylov Subspace Recycling Methods

Fast convergence of Krylov subspace methods generally depends on the clustering of the eigenvalues of the coefficient matrix away from the origin [122] and so approximately deflating the smallest (in magnitude) eigenvalues of $\mathbf{A}$ can improve convergence of the Krylov method [108]. Restarted Krylov methods can be used to reduce computational cost and storage, but any approximate eigenvectors in the Krylov space are discarded upon restarting. By maintaining a subspace of the Krylov space that corresponds to the smallest eigenvalues, we can improve convergence of the iterative method. How we update this space upon restarting depends on whether we are restarting the method for a single linear system or recycling the space for another closely related system in a sequence.

In addition to building an orthonormal basis for the Krylov space, the Arnoldi method can also be used to compute the approximate eigenvalues and eigenvectors of a system. For $(\theta, \mathbf{y})$ an eigenpair of $\mathbf{H}_{j-1,j-1}$ as in (1.7), the Ritz pair $(\theta, \mathbf{V}_j \mathbf{y})$ is often a good approximation to an eigenvalue of $\mathbf{A}$ [130]. The harmonic Ritz values of $\mathbf{A}$ are defined as the reciprocals of the Ritz values of $\mathbf{A}^{-1}$ with respect to the Krylov space $\mathbf{A}\mathcal{K}^j(\mathbf{A}, \mathbf{r}_0)$ [141]. In practice, the harmonic Ritz values are often a good approximation to the eigenvalues closest to the origin, and by deflating the harmonic Ritz vectors, we can improve convergence of the iterative method. As an example, we show in Figure 1.1 the harmonic Ritz values and exact eigenvalues of a matrix from a sequence of linear systems arising in a transient hydraulic tomography application [14]. A comprehensive analysis of Ritz values and vectors can be found in [121, 130], and for a discussion of harmonic Ritz values and vectors, we refer the reader to [109, 127].

GMRES-DR

GMRES with deflated restarting (GMRES-DR) [108] is a variation of the restarted GMRES method that retains several vectors when restarting, rather than discarding and starting

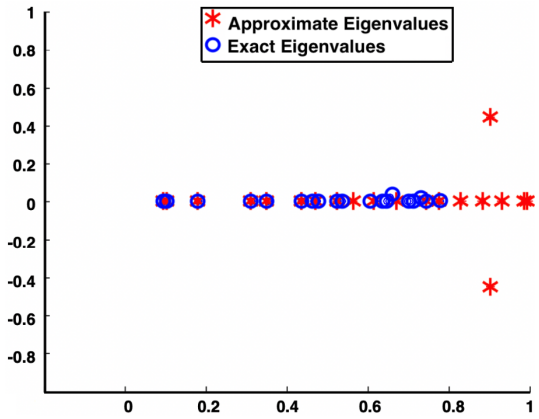


Figure 1.1: Harmonic Ritz values and and exact interior eigenvalues of a matrix from a sequence of linear systems arising in a transient hydraulic tomography application [14].

from scratch with a single vector. The first cycle of GMRES-DR is identical to restarted GMRES [108]. For subsequent cycles, GMRES-DR adapts the idea of thick restarting for eigenproblems given in [147] to solving a system of equations [108]. In particular, the method computes k harmonic Ritz vectors $\mathbf{X}_k \in \mathbb{C}^{n\times k}$ from $\mathbf{H}_{j-1,j-1}$ in the Arnoldi recurrence (1.7), where we note that $\mathbf{X}_k$ is a Krylov space of $\mathbf{A}$ [107]. Upon retaining these vectors for the next cycle of the method, GMRES-DR orthogonalizes $\mathbf{X}_k$ and builds the Krylov space of dimension m by taking $m-k$ Arnoldi steps to give the decomposition [108]

$$\mathbf{A}\left[\mathbf{X}_k \;\middle|\; \mathbf{V}_{m-k}\right] = \left[\mathbf{X}_k \;\middle|\; \mathbf{V}_{m-k+1}\right]\mathbf{H}_{m+1,m}.$$

The method then proceeds just as in GMRES for the remainder of the cycle [108, 110]. By deflating the harmonic Ritz values, GMRES-DR has been shown to be more robust than restarted GMRES, and reduces iterations [108].

While GMRES-DR is effective for a single system, it cannot be used for a sequence of systems. Though $\left[\mathbf{X}_k \;\middle|\; \mathbf{V}_{m-k+1}\right]$ remains a Krylov space for a single matrix [108], for a general sequence of systems, this is no longer the case since the harmonic Ritz vectors computed for one matrix do not form a Krylov space for another matrix [110]. If the sequence is defined by a shift, GMRES-DR can be used [55]; however, if we use preconditioning, as is often the case, $\left[\mathbf{X}_k \;\middle|\; \mathbf{V}_{m-k+1}\right]$ is again no longer a Krylov space for the new system.

GCRO-DR

When solving a sequence of linear systems of the form (1.2), we can instead use the Krylov subspace recycling method GCRO-DR [110], which uses GCRO [56] for recycling. This method retains a subspace that approximates an invariant subspace of interest, and does not require that the recycle space be a Krylov space for the system. In fact it allows for

any subspace to be recycled [110]. For the implementation used in this thesis, GCRO-DR deflates the approximate invariant subspace corresponding to k harmonic Ritz values; see [110] for the pseudocode.

Given the recycle space spanned by the columns of $\mathbf{U}_k^{(\ell-1)} \in \mathbb{C}^{n\times k}$ retained from a previous application of the method to $\mathbf{A}^{(\ell-1)}$, GCRO-DR first computes the QR-decomposition $\mathbf{A}^{(\ell)}\mathbf{U}_k^{(\ell-1)} = \mathbf{W}_k\mathbf{S}_{k,k}$ [110]. The method continues by updating $\mathbf{C}_k = \mathbf{W}_k$ and $\mathbf{U}_k = \mathbf{U}_k^{(\ell-1)}\mathbf{S}_{k,k}^{-1}$, so that (1.9) and (1.10) hold. For efficiency, $\mathbf{U}_k$ is generally not explicitly computed [110]. Let $\mathbf{x}_{-1}$ be the initial solution, and $\mathbf{r}_{-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{-1}$. The method sets

$$\mathbf{r}_0 = (\mathbf{I}_k - \mathbf{C}_k\mathbf{C}_k^*)\,\mathbf{r}_{-1}, \tag{1.11}$$

and

$$\mathbf{x}_0 = \mathbf{x}_{-1} + \mathbf{U}_k\mathbf{C}_k^*\mathbf{r}_{-1} \tag{1.12}$$

noting that (1.12) is the optimal solution over the range of $\mathbf{U}_k$ [110]. The Krylov space $\mathbf{V}_j \in \mathbb{C}^{n\times j}$ is generated with the operator $(\mathbf{I} - \mathbf{C}_k\mathbf{C}_k^*)\mathbf{A}^{(\ell)}$ giving the recurrence [110]

$$(\mathbf{I}_k - \mathbf{C}_k\mathbf{C}_k^*)\mathbf{A}^{(\ell)}\mathbf{V}_j = \mathbf{V}_{j+1}\mathbf{H}_{j+1,j}.$$

Then, for $\mathbf{B}_j = \mathbf{C}_k^*\mathbf{A}^{(\ell)}\mathbf{V}_j$, the method defines the relation

$$\mathbf{A}^{(\ell)}\left[\begin{array}{c|c}\mathbf{U}_k & \mathbf{V}_j\end{array}\right] = \left[\begin{array}{c|c}\mathbf{C}_k & \mathbf{V}_{j+1}\end{array}\right]\begin{pmatrix}\mathbf{I}_k & \mathbf{B}_j \\ \mathbf{0} & \mathbf{H}_{j+1,j}\end{pmatrix}. \tag{1.13}$$

Letting $m = k+j$, we denote (1.13) as $\mathbf{A}^{(\ell)}\widetilde{\mathbf{V}}_m = \widehat{\mathbf{V}}_{m+1}\widehat{\mathbf{H}}_{m+1,m}$ [110]. The updated solution, $\mathbf{x}_j$, can be computed by first solving

$$(\boldsymbol{\zeta}_j, \boldsymbol{\eta}_j) = \min_{\substack{\boldsymbol{\zeta}\in\mathbb{C}^k \\ \boldsymbol{\eta}\in\mathbb{C}^j}} \left\| \begin{pmatrix}\mathbf{I}_k & \mathbf{B}_j \\ \mathbf{0} & \mathbf{H}_{j+1,j}\end{pmatrix}\begin{pmatrix}\boldsymbol{\zeta} \\ \boldsymbol{\eta}\end{pmatrix} - \begin{pmatrix}\mathbf{C}_k^*\mathbf{r}_0 \\ \mathbf{e}_1\beta\end{pmatrix}\right\|, \tag{1.14}$$

where $\beta = \|(\mathbf{I} - \mathbf{C}_k\mathbf{C}_k^*)\mathbf{r}_0\|$, and $\mathbf{e}_1$ is the first Euclidean basis vector [128]. In particular, (1.14) can be solved blockwise for $\boldsymbol{\eta}_j = \min_{\boldsymbol{\eta}\in\mathbb{C}^j}\|\mathbf{H}_{j+1,j}\boldsymbol{\eta} - \mathbf{e}_1\beta\|$ (as for GMRES), and then $\boldsymbol{\zeta}_j = \mathbf{C}_k^*\mathbf{r}_0 - \mathbf{B}_j\boldsymbol{\eta}_j$ [56, 128]. Then, for $\widetilde{\mathbf{x}}_j = \mathbf{U}_k\boldsymbol{\zeta}_j$ and $\widetilde{\mathbf{y}}_j = \mathbf{V}_j\boldsymbol{\eta}_j$, the updated solution is $\mathbf{x}_j = \mathbf{x}_0 + \widetilde{\mathbf{x}}_j + \widetilde{\mathbf{y}}_j$ [56, 128].

In order to retain the harmonic Ritz vectors to recycle for a subsequent system in the sequence, GCRO-DR computes the k eigenvectors $\boldsymbol{\xi}_i$ of the generalized eigenvalue problem $\widehat{\mathbf{H}}_{m+1,m}^*\widehat{\mathbf{H}}_{m+1,m}\boldsymbol{\xi}_i = \gamma_i\widehat{\mathbf{H}}_{m+1,m}^*\widehat{\mathbf{V}}_{m+1}^*\widetilde{\mathbf{V}}_m\boldsymbol{\xi}_i$ associated with the smallest (in magnitude) eigenvalues γ_i [110]. For $\mathbf{Y}_k = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_k] \in \mathbb{C}^{n\times k}$, the harmonic Ritz vectors are updated at the restart for the current system by taking the QR decomposition of $\widetilde{\mathbf{V}}_m\mathbf{Y}_k = \mathbf{Q}\mathbf{R}$ and setting $\mathbf{C}_k = \widetilde{\mathbf{V}}_{m+1}\mathbf{Q}$ and $\mathbf{U}_k = \widetilde{\mathbf{V}}_m\mathbf{Y}_k\mathbf{R}^{-1}$ [110].

GCRO-DR is effective in reducing the cost of solving a sequence of linear systems [110] and, in one of the applications presented next, we show that we can further improve convergence of the iterative method when recycling the preconditioner using SAM updates.

1.3.3 Numerical Experiments

We analyze the effectiveness of reusing, recomputing, and recycling preconditioners using GMRES-type iterative methods with and without subspace recycling. We consider sequences of matrices arising in two applications involving the discretization of a PDE. We compare the number of iterations required to solve these sequences when recomputing, reusing, and recycling the preconditioner with and without Krylov subspace recycling using GCRO-DR and GMRES, respectively. We show that in both applications, recycling preconditioners with GCRO-DR reduces iterations compared with reusing the preconditioner with GMRES. In one of our applications, recycling preconditioners with GCRO-DR results in the fewest iterations overall, even compared with recomputing the preconditioner with GCRO-DR. Given the small size of the matrices in both applications, making a time-wise comparison when using the different recycling strategies as we do in Chapter 2 is not as meaningful here. So, we omit timings, and focus our attention on the number of iterations.

Discretized Nonlinear Convection-Diffusion Equation

The sequence of linear systems considered in this section comes from solving the nonlinear convection-diffusion equation

$$-\nabla \cdot ((\eta + \gamma \mathbf{u}^2)\nabla \mathbf{u}) + r\mathbf{u}_x + s\mathbf{u}_y + t\mathbf{u} = f. \tag{1.15}$$

We let $\eta = 0.1$, $\gamma = 1$, $r = s = 1$, and $t = f = 0$. On the domain $[0, 1] \times [0, 1]$, we use the boundary conditions

$$\mathbf{u}(x, 1) = 0, \quad \mathbf{u}(x, 0) = 0,$$

$$\mathbf{u}(1, y) = 0, \quad \mathbf{u}(0, y) = 0.2 + y(1 - y^2).$$

We use second order central finite differences to discretize (1.15), and since the PDE is nonlinear in the diffusion coefficient, we use Newton's method with line search to compute

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} - \alpha_{k+1}\mathbf{J}(\mathbf{u}^{(k)})^{-1}\mathbf{F}(\mathbf{u}^{(k)}). \tag{1.16}$$

Here, $\mathbf{u}^{(k+1)}$ denotes the approximate solution in the $(k+1)^{st}$ step of Newton's method; $\mathbf{J}(\mathbf{u}^{(k)})$ is the Jacobian evaluated at the previous solution $\mathbf{u}^{(k)}$; and $\mathbf{F}(\mathbf{u}^{(k)}) = \nabla f(\mathbf{u}^{(k)})$.

The parameters for this application are chosen by following those given in [101, 102], where cumulative preconditioner updates are truncated in order to reduce the cost of applying the preconditioner. In [101, 102], Broyden-type rank-one updates are used to update the initial preconditioner such that $\mathbf{P}_m = (\mathbf{I} + \mathbf{T}_m)\mathbf{P}_0$, where $\mathbf{T}_m$ is a rank-m matrix [37, 101, 102]. By computing the SVD of $\mathbf{T}_m$, the rank-p approximation $\widetilde{\mathbf{T}}_p \approx \mathbf{T}_m$ $(p < m)$, is constructed to give the (truncated) preconditioner $\mathbf{P}_m \approx (\mathbf{I} + \widetilde{\mathbf{T}}_p)\mathbf{P}_0$ [101, 102].

The MATLAB® code for the discretization scheme in Appendix B.1 is our own implementation, and in this section we are highlighting an application where the combination of preconditioner updates and subspace recycling also lead to a further reduction in iterations. We do not compare our recycling techniques with those proposed in [101, 102].

We let $n_x = n_y = 250$ be the number of grid points in the x and y directions, respectively, and so each matrix in the sequence has dimension $n = 62500$. Newton's method converges in 64 steps, giving as many linear systems of the form (1.16) to solve. We use preconditioned GMRES and GCRO-DR. For both iterative solvers, we set the convergence tolerance to 10^{-10}, the maximum number of iterations to n, and the restart to 50. We use the zero vector for our initial guess, and for GCRO-DR, we use a recycle space dimension of 20. We compute an ILUTP preconditioner using the implementation provided in Appendix A with a drop tolerance of 10^{-3} and fill-in of 10. We update our preconditioner with SAM updates, using the sparsity pattern of $\mathbf{J}(\mathbf{u}^{(0)})$.

For this application, we compare GMRES and GCRO-DR iterations when recomputing the preconditioner, reusing the initial preconditioner for all systems, and recycling the preconditioner at each step after the first using SAM updates. For this sequence of linear systems, preconditioned GCRO-DR using SAM updates gives the lowest total iterations. Updating the preconditioner using SAM updates results in approximately the same number of GMRES iterations compared with recomputing the initial preconditioner. However, there are significantly fewer total GCRO-DR iterations when recycling with SAMs compared with recomputing the initial preconditioner. So, while recycling the preconditioner approximately maintains the total number of GMRES iterations compared with recomputing the initial preconditioner, coupling SAM updates with Krylov subspace recycling improves convergence substantially. The results are given in Table 1.1 and Figure 1.2.

	Recompute ILUTP, GMRES	**Reuse ILUTP, GMRES**	**Recycle ILUTP, GMRES**	**Recompute ILUTP, GCRO-DR**	**Reuse ILUTP, GCRO-DR**	**Recycle ILUTP, GCRO-DR**
Total Iterations	2149	3521	2153	1816	1341	604

Table 1.1: Total iterations when recomputing, reusing, and recycling the preconditioner using GMRES and GCRO-DR with Newton's method for solving the nonlinear convection-diffusion equation.

Discretized Euler Equations

Next, we consider a sequence of matrices arising in the discretization of the weak form of the Euler equations from the Structured, Euler/Navier-Stokes Explicit and Implicit (SENSEI) solver, a computational fluid dynamics code that uses a finite volume discretization [59, 94].[1]

[1]We thank Christopher Roy, Andrew McCall, and Will Tyson for their guidance in running SENSEI.

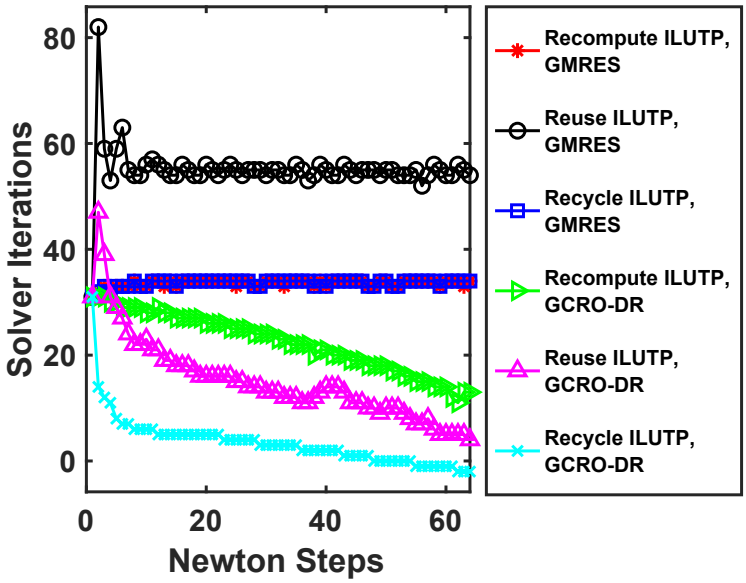


Figure 1.2: Iteration counts for each Newton step when recomputing, reusing, and recycling the preconditioner using GMRES and GCRO-DR when solving the nonlinear convection-diffusion equation.

Details of the equations and the discretization techniques can be found in [59, Section II]. In particular, we use SENSEI's inviscid Euler solver with mesh size $h = 16$ to generate a sequence of 200 matrices with size $n = 17920$ [59, Section III]. We show results for the first 48 matrices in the sequence using GMRES and GCRO-DR when recomputing and recycling the preconditioner. The matrices are very ill-conditioned, and so we use row and column scaling. For both iterative solvers, we set the convergence tolerance to 10^{-10}, the maximum number of iterations to 1000, and the restart to 200. We use the zero vector for our initial guess and for GCRO-DR, we use a recycle space dimension 15. We use an ILUTP preconditioner again using the implementation provided in Appendix A with a drop tolerance of 10^{-3} and fill-in of 25. We update our preconditioner with SAM updates, using the sparsity pattern of the initial matrix.

In our tests, neither GMRES nor GCRO-DR converge in the maximum number of iterations for any of the matrices in the sequence when reusing the preconditioner, and so we omit these tests from our results. Recomputing the preconditioner when using GCRO-DR gives the fewest total iterations. Even for the slowest test (reusing the initial preconditioner with GMRES), the average number of iterations per linear system is approximately 55, and so while recycling subspaces combined with recycling the preconditioner results in the fewest iterations, it is likely not a practical choice (timewise). Results are given in Table 1.2 and Figure 1.3.

	Recompute ILUTP, GMRES	Recycle ILUTP, GMRES	Recompute ILUTP, GCRO-DR	Recycle ILUTP, GCRO-DR
Total Iterations	650	1056	593	1044

Table 1.2: Total iterations when recomputing and recycling the preconditioner using GMRES and GCRO-DR for the discretized Euler equations from SENSEI [59, 94].

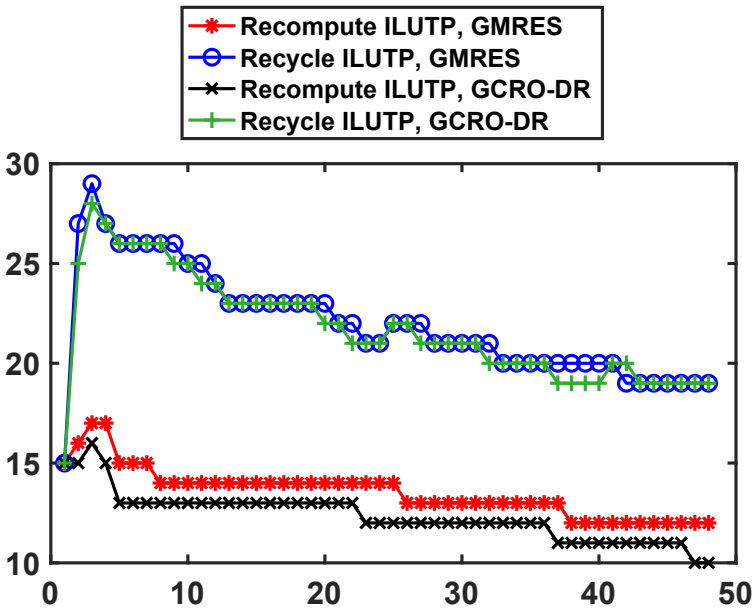


Figure 1.3: Iteration counts when recomputing, reusing, and recycling the preconditioner using GMRES and GCRO-DR for the discretized Euler equations from SENSEI [59, 94].

1.3.4 Summary

Krylov subspace recycling can be used to improve the convergence of Krylov methods when solving a sequence of linear systems. By recycling the space corresponding to the harmonic Ritz values, we can improve convergence when solving a sequence of linear systems. In this section, we show that for two applications, subspace recycling combined with preconditioner recycling reduces iterations and competitive runtimes compared with reusing the initial preconditioner. While recomputing the preconditioner often results in the fewest iterations for many applications, for one of our applications, we show that recycling the initial preconditioner along with subspace recycling results in fewer iterations than recomputing the initial preconditioner.

Next, we describe our preconditioner update technique and how it can be used to improve the convergence of GMRES when solving a sequence of linear systems. In Chapter 3, we show how Krylov subspace recycling can be incorporated into the eigensolver when matrix-vector

products require an implicit linear solve.

Chapter 2

Preconditioning Parametrized Linear Systems

Preconditioners are generally essential for fast convergence in the iterative solution of linear systems of equations. However, the computation of a good preconditioner can be expensive. So, while solving a sequence of many linear systems, it is advantageous to recycle preconditioners, that is, update a previous preconditioner and reuse the updated version. In this chapter, we introduce a simple and effective method for doing this. We consider recycling preconditioners for both the general case of sequences of linear systems $\mathbf{A}(\mathbf{p}_k)\mathbf{x}_k = \mathbf{b}_k$ as well as the important special case of the type $(s_k\mathbf{E} + \mathbf{A})\mathbf{x}_k = \mathbf{b}_k$. The right hand sides may or may not change.

We update preconditioners by defining a map from a new matrix to a previous matrix, for example, the first matrix in the sequence. We then combine the preconditioner for this previous matrix with the map to define the new preconditioner. This approach has several advantages. *The update is independent from the original preconditioner, so it can be applied to any preconditioner.* The possibly high cost of an initial preconditioner can be amortized over many linear solves. The cost of updating the preconditioner is more or less constant and there is flexibility in balancing the quality of the map with the computational cost.

In the numerical experiments section, we demonstrate good results for several applications, in particular when using an algebraic multigrid preconditioner.

2.1 Introduction

We discuss the efficient computation of preconditioners for sequences of systems that change slowly. We consider both the general case

$$\mathbf{A}(\mathbf{p}_k)\mathbf{x}_k = \mathbf{b}_k, \tag{2.1}$$

as well as the important special case

$$(s_k\mathbf{E} + \mathbf{A})\mathbf{x}_k = \mathbf{b}_k, \tag{2.2}$$

where the right hand side(s) may or may not change. The first class of matrices in (2.1) arises, for example, in topology optimization, discussed later in this chapter, where the parameter vector $\mathbf{p}_k$ represents the changing densities in each element (during the optimization). $\mathbf{A}(\mathbf{p}_k)$ represents the finite element discretization of a three-dimensional elasticity problem given a density distribution $\mathbf{p}_k$ [22, 24, 114, 145]. In addition, we consider two sequences of linear systems of the form (2.2). One arises in model reduction, in particular, in the Iterative Rational Krylov Algorithm (IRKA) [7, 8, 81] for finding the optimal shifts s_k; the other arises in a sequence of discretized 2D Helmholtz equations [41, 67, 68]. Other applications where sequences of the form (2.2) arise include oscillatory and transient hydraulic tomography (OHT/THT) [46] and diffuse optical tomography (DOT) [1, 58, 97, 123], though we do not consider these applications in the current chapter. For the second class of matrices, s_k is a shift (often related to a frequency), and the matrices $\mathbf{A}$ and $\mathbf{E}$ ($\mathbf{E} \neq \mathbf{I}$) represent discretizations of partial differential operators, or more generally arise in the simulation of a dynamical system. In these applications, the matrices $\mathbf{A}$ and $\mathbf{E}$ may also depend on a parameter vector $\mathbf{p}$, but this is not considered here.

For the special case of shifted systems, where $\mathbf{E} = \mathbf{I}$, other approaches for iterative solvers have been considered. Flexible preconditioning is used for problems of this form in [15, 80]. In [2] and [97], the authors take advantage of the shift invariance of Krylov subspaces.

Preconditioners are often essential for fast iterative solutions of linear systems of equations, but the computation of a good preconditioner can be expensive. Therefore, we consider *recycling preconditioners*, that is, updating a previous preconditioner and using the updated version for solving a new linear system. For a sequence of linear systems, this may provide a substantial reduction in cost compared with *recomputing* a new preconditioner for each system. We can also periodically compute a new preconditioner from scratch, which includes the important case of solving all systems with a single preconditioner. We refer to this as *reusing* the initial preconditioner.

The main idea for our approach comes from [4]. Given a sequence of matrices, $\mathbf{A}_k$, for $k = 0, 1, 2, \ldots$, and a good preconditioner $\mathbf{P}_0$ for $\mathbf{A}_0$, such that $\mathbf{A}_0\mathbf{P}_0$ (or $\mathbf{P}_0\mathbf{A}_0$) yields fast convergence, we could compute for each system the ideal map $\widehat{\mathbf{N}}_k$ such that

$$\mathbf{A}_k\widehat{\mathbf{N}}_k = \mathbf{A}_0 \tag{2.3}$$

and define the updated preconditioner as

$$\mathbf{P}_k = \widehat{\mathbf{N}}_k \mathbf{P}_0. \tag{2.4}$$

Then, $\mathbf{A}_0\mathbf{P}_0 = \mathbf{A}_1\mathbf{P}_1 = \cdots = \mathbf{A}_k\mathbf{P}_k$, and $\mathbf{A}_k\widehat{\mathbf{N}}_k\mathbf{P}_0 = \mathbf{A}_0\mathbf{P}_0$ will yield the same fast convergence for each k as the original preconditioned system. In general, the matrix $\widehat{\mathbf{N}}_k\mathbf{P}_0$ is never computed; in an iterative method, we can multiply vectors successively by these two matrices (which does lead to some overhead). If computing these maps can be made cheap and the initial preconditioner is very good, we obtain fast convergence for all systems at low cost. In some cases, as with the flow matrices discussed in Section 2.5.2, the initial preconditioner may not result in fast convergence, for example, because the initial matrix $\mathbf{A}_0$ is very ill-conditioned and/or far from diagonally dominant. In such a case, the preconditioner for another, more appropriate, system matrix $\mathbf{A}_j$, $j > 0$, may be chosen, and we recycle $\mathbf{P}_j$.

In this chapter, we present a more general update scheme for recycling preconditioners by mapping one matrix to another for which we have a good preconditioner. This generalizes the approach in [4] (see next section) to any set of closely related matrices. We do not seek an exact map, but rather compute an approximate map $\mathbf{N}_k$ such that

$$\mathbf{A}_k\mathbf{N}_k \approx \mathbf{A}_0. \tag{2.5}$$

In Section 2.2, we review previous work on updating preconditioners and Sparse Approximate Inverses (SAI), the technique motivating our proposed update scheme. We then introduce our update scheme, the Sparse Approximate Map (SAM) update.

In Section 2.3, we analyze sparsity patterns for SAMs. Denser patterns can give more accurate maps, but they also increase the cost to compute the map and to apply it (every iteration), which needs to be compensated with a further reduction in iterations. On the other hand, if effective maps can be found that are significantly sparser than the matrix, recycling preconditioners will be highly favorable. We demonstrate this for 3D elasticity problems arising in topology optimization.

In Section 2.4, we discuss efficient implementations of SAMs, as well as an efficient MATLAB® m-file implementation of the ILUTP factorization [118, 119]. For our numerical experiments, we also use an algebraic multigrid (AMG) preconditioner implemented in MATLAB® [87].[1] The computation of SAMs as well as multiplying by SAMs is easily parallelized, though we do not address this in the current chapter.

SAM updates are particularly effective for sequences of hard problems where expensive preconditioners are needed for fast convergence and reusing a preconditioner is not effective. This is the case for many KKT systems and problems where an AMG preconditioner is needed and the set-up phase is expensive. Another example are matrix-free methods where, cost-wise, we can compute a matrix only once to compute a preconditioner. In this chapter, we demonstrate the effectiveness of SAMs for ILUTP-type preconditioners [119, 120]

[1] We thank Xiaozhe Hu for sharing his MATLAB® m-file for the AMG preconditioner with us.

and AMG preconditioners [116, 117, 133, 135] [140, Appendix A: An Introduction to Algebraic Multigrid, K. Stüben]. ILUTP-type preconditioners are widely used, and AMG preconditioners make a good case as they are very effective for hard problems but expensive to compute. While we use ILUTP and AMG preconditioners here to make the case for recycling preconditioners, SAMs can be used with any other preconditioner.

We note that the purpose of this chapter is not a time-wise comparison between SAMs and any preconditioner; rather, they play complementary roles. Nevertheless, recycling a preconditioner does not make sense if computing the SAM takes more time than computing a new preconditioner. So, time-wise comparisons between computing the two are needed. To make these comparisons fair, we compare run times for (interpreted) m-files: our m-file for SAMs, our m-file implementation for ILUTP [47], and an m-file implementation for the AMG preconditioner from [87]. Comparing with run times from MATLAB®'s (compiled) `ilu` (type 'ilutp') has two important drawbacks. First, compiled code runs much faster than interpreted code, which would seriously skew the comparisons. Second, MATLAB®'s more recent implementation of Saad's ILUTP determines the amount of fill automatically, sometimes allowing large amounts of fill. This makes comparisons difficult and potentially makes computing MATLAB®'s `ilu` more expensive than necessary.

Recycling preconditioners by periodically updating an initial or previous preconditioner with a SAM update can significantly reduce total runtime compared with (1) computing a new preconditioner for every system or periodically and (2) reusing a fixed preconditioner for all systems. If computing (and using) the SAM is cheaper than computing a new preconditioner and yields faster convergence, recycling clearly wins in comparison (1). This is not usually the case, but it is possible; see Section 2.5.3, where we demonstrate this for indefinite matrices arising from the Helmholtz equation. With respect to (1), generally the issue is whether the (typically) lower cost of computing the SAMs outweighs the cost of additional iterations (due to a less effective preconditioner) and the additional matrix-vector product per iteration. With respect to (2), the issue is whether the reduction in iterations due to an improved preconditioner outweighs the cost of computing the SAM update plus the extra cost per iteration.

In Section 2.5, we demonstrate the effectiveness of SAM updates for applications from topology optimization and model reduction, as well as for indefinite matrices arising from Helmholtz equations, along with providing some details of these applications.

2.2 Recycling Preconditioners

To avoid the potentially high cost of computing a new preconditioner, we propose recycling an existing one using maps between matrices. In [4], this idea was exploited for a Markov chain Monte Carlo (MCMC) process that resulted in a long sequence of matrices changing by one row at a time. So, $\mathbf{A}_{k+1} = \mathbf{A}_k + \mathbf{e}_{i_k}\mathbf{u}_k^T$, where i_k indicates which row changes.

The ideal map for this case, $\mathbf{I} - (1 + \mathbf{u}_k^T \mathbf{A}_k^{-1} \mathbf{e}_{i_k})^{-1} \mathbf{A}_k^{-1} \mathbf{e}_{i_k} \mathbf{u}_k^T$, comes for free, as one already needs to compute $\mathbf{u}_k^T \mathbf{A}_k^{-1} \mathbf{e}_{i_k}$ for the transition probability in the MCMC process. While this update is specific to the change in the matrix, the approach proposed in the present chapter generalizes the idea of recycling preconditioners to *any* set of closely related matrices.

Our preconditioner update is advantageous in several ways. (1) To compute the map (ideal or approximate), knowledge of the original preconditioner, $\mathbf{P}_0$, is not required. *Therefore, the map is independent of* $\mathbf{P}_0$ *and can be applied to any type of preconditioner.* (2) The cost of updating $\mathbf{P}_0$ in this fashion is more or less constant, and the potentially high cost of computing a good $\mathbf{P}_0$ can be amortized over many linear solves. (3) In practice, we do not need the ideal map (2.3), but rather an approximation, $\mathbf{N}_k$, satisfying (2.5). We can balance the accuracy of $\mathbf{N}_k$ with the cost of computing it.

Our update scheme is motivated by the Sparse Approximate Inverse (SAI). So, we refer to it as a Sparse Approximate Map (SAM) update. The SAI was proposed in [27] and further developed in [28, 54, 79, 85, 88] and references therein. To define SAIs and SAMs we need the following definitions.

Definition 2.1. A sparsity pattern for $\mathbb{C}^{n\times n}$ is any subset of $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$.

Definition 2.2. Let S be a sparsity pattern for $\mathbb{C}^{n\times n}$. We define the subspace $\mathcal{S} \subseteq \mathbb{C}^{n\times n}$ as $\mathcal{S} = \{\mathbf{X} \in \mathbb{C}^{n\times n} \mid X_{ij} = 0 \text{ if } (i, j) \notin S\}$.

SAIs can be defined in several ways, but for the current discussion we use the following.

Definition 2.3. For $\mathbf{P}, \mathbf{A} \in \mathbb{C}^{n\times n}$, $\mathbf{I}$ the identity matrix in $\mathbb{C}^{n\times n}$, and a given sparsity pattern S, the Sparse Approximate Inverse, $\mathbf{P}$, for a matrix, $\mathbf{A}$, is defined as the minimizer of

$$\min_{\mathbf{P}\in\mathcal{S}} \|\mathbf{I} - \mathbf{AP}\|_F. \tag{2.6}$$

The computation of a SAI (and variations, such as MSAI [91]) is easily parallelized as the computation of n independent, and very small, least squares problems [10, 78, 79, 91, 92, 95]. For example, for the flow application discussed in Section 2.5.2, the maximum size of the least squares problems to compute the SAM updates is 20×7, independent of the matrix dimension $n = 9\,669$. SAM updates can analogously be computed in parallel, which can be a substantial advantage on modern architectures. However, we do not consider a parallel implementation of SAMs in this chapter.

Rather than considering the identity matrix in (2.6), other work has focused on replacing it with another matrix, sometimes referred to as a target matrix [85]. The problem then becomes

$$\min_{\mathbf{P}\in\mathcal{S}} \|\mathbf{B} - \mathbf{AP}\|_F. \tag{2.7}$$

In [54, 85], (2.7) is solved to improve a preconditioner, $\mathbf{B}^{-1}$, aiming to make $\mathbf{APB}^{-1}$ closer to the identity matrix than $\mathbf{AB}^{-1}$. As a preconditioner, $\mathbf{B}^{-1}$ is assumed to be available, for example through an approximate factorization of $\mathbf{A}$ (or $\mathbf{A}^{-1}$). However, the columns of $\mathbf{B}$ must be computed in order to solve (2.7), and the cost of constructing these columns can be relatively high [54]. Indeed, for a preconditioner like AMG, as the components (or factors) of an AMG preconditioner are not individually invertible, an *iterative solve for each column of* $\mathbf{B}$ is required, which is typically quite expensive (although one can use $\mathbf{A}$ as a preconditioner). In order to reduce the cost of explicitly constructing $\mathbf{B}$, iterative methods with numerical dropping are used to approximate the columns of $\mathbf{B}$ in [54]. In special cases, the structure or type of the matrix can be exploited. In [85], using the advection-diffusion equation and targeting the Laplacian, Holland, et al. are able to use a fast solver for the action of $\mathbf{B}^{-1}$ with good results.

Our update scheme involves solving

$$\mathbf{N}_k \quad = \quad \arg\min_{\mathbf{N}\in\mathcal{S}} \|\mathbf{A}_k\mathbf{N} - \mathbf{A}_0\|_F, \tag{2.8}$$

and defining the updated preconditioner as

$$\mathbf{P}_k = \mathbf{N}_k\mathbf{P}_0. \tag{2.9}$$

Here $\mathcal{S}$ is the subspace defined by a chosen sparsity pattern S as in Definition 2.2, and $\mathbf{A}_0$ and $\mathbf{A}_k$ are matrices from a given sequence and are relatively close to one another. From (2.8), we obtain the approximation (2.5).

While the minimization in (2.8) has a form similar to (2.7), there are fundamental differences in the approach to preconditioning. First, computing (2.7) involves improving an existing preconditioner, $\mathbf{B}^{-1}$, for a fixed matrix, $\mathbf{A}$, whereas our approach aims to compute a sequence of maps between nearby matrices to update a given preconditioner to an earlier matrix to compute preconditioners for the new matrices; the maps depend only on the sequence of nearby matrices, not on the existing preconditioner. Second, for most preconditioners, $\|\mathbf{B}-\mathbf{A}\|_F$ is quite large [54]. So, typically an accurate solution cannot be expected, unless the sparsity pattern of $\mathbf{P}$ contains relatively many nonzeros. Of course, if an accurate solution is obtained, the benefit is faster convergence rather than maintaining the same convergence. On the other hand, our approach seeks to map one matrix to another *closely related one* in a sequence of linear systems. So, $\|\mathbf{A}_k - \mathbf{A}_0\|_F$ is likely to be relatively small, and therefore, typically, we expect a relatively accurate solution. Third, computing the columns of $\mathbf{B}$ can be quite expensive. For the preconditioners used in this chapter, this is certainly the case. When using an ILUTP preconditioner, computing $\mathbf{B}$ as in (2.7) requires the relatively expensive product $\mathbf{LU}$. Computing the inverse of an AMG preconditioner requires an iterative solve per column. Our approach requires only the solution of the small least squares problems, as the columns of $\mathbf{A}_0$, a previous matrix in the sequence of linear systems, are readily available.

Next, we consider some theoretical properties of the preconditioned systems using the map. These properties provide some insight into how well the approach may work, as well as how

to choose parameters. However, for practical use, the latter also requires an assessment of the cost of the map, and we leave this for future work.

First, we show that if $\mathbf{A}_0\mathbf{P}_0$ is well-conditioned then a sufficiently good map, i.e., a sufficiently small residual $\mathbf{R}_k = \mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0$, implies that $\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0$ is well-conditioned as well. Note that ill-conditioning generally leads to poor convergence of iterative methods, and hence is important to avoid. For nonsymmetric systems the converse is unfortunately not true. For that reason, we also provide a discussion of the field of values for the preconditioned system using the map.

Suppose we compute a map, $\mathbf{N}_k$, such that (for some submultiplicative norm $\|.\|$)

$$\|\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0\| < \gamma\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|^{-1}, \tag{2.10}$$

for a chosen $0 < \gamma < 1$. Then

$$\|(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0)^{-1}\| \leq \frac{\|\mathbf{I}\|}{1-\gamma}\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|, \tag{2.11}$$

and as $\|\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0\| = \|\mathbf{A}_0\mathbf{P}_0 + (\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0)\| \leq \|\mathbf{A}_0\mathbf{P}_0\| + \gamma\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|^{-1}$,

$$\kappa(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0) \leq \frac{\|\mathbf{I}\|}{1-\gamma}(\kappa(\mathbf{A}_0\mathbf{P}_0) + \gamma). \tag{2.12}$$

The proof of (2.11) is a minor variation of [61, Corollary 7.19] or [74, section 2.3.4]. The parameter γ need not be small; for instance, taking $\gamma = \frac{1}{2}$ and using the matrix 2-norm, gives

$$\kappa_2(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0) \leq 2\kappa_2(\mathbf{A}_0\mathbf{P}_0) + 1. \tag{2.13}$$

We do not control $\|\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0\| = \|\mathbf{R}_k\mathbf{P}_0\|$ directly, but could choose instead of (2.10)

$$\|\mathbf{R}_k\| \leq \frac{\gamma}{\|\mathbf{P}_0\|}\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|^{-1}, \tag{2.14}$$

and so the bound in (2.13) depends on controllable features: how well $\mathbf{P}_0$ approximates $\mathbf{A}_0^{-1}$, and the accuracy of $\mathbf{N}_k$. An estimate of $\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|$ can be computed during the iterative solve with $\mathbf{A}_0\mathbf{P}_0$. One obvious way to improve the update is to use a denser sparsity pattern when computing $\mathbf{N}_k$. However, the sparsity pattern of the SAM update must be chosen carefully in order to minimize runtime. We examine choices in sparsity patterns in more detail in Section 2.3.

So, our approach can keep the condition number from increasing substantially. While this is important, it does not guarantee fast convergence for nonsymmetric systems. For this reason, we also consider a field of values related argument. The field of values of a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ is defined as [86]

$$\mathcal{F}(\mathbf{A}) = \left\{ \frac{\mathbf{y}^*\mathbf{A}\mathbf{y}}{\mathbf{y}^*\mathbf{y}} \;\middle|\; \mathbf{y} \in \mathbb{C}^n, \mathbf{y} \neq 0 \right\}.$$

If the field of values of a matrix is contained in a disk or an ellipse not containing the origin, convergence bounds for GMRES are available; see, e.g., [77, pp. 56-57]. If we take $\|\mathbf{R}_k\|_2 \leq \gamma/\|\mathbf{P}_0\|_2$ (note the use of the matrix 2-norm), we get for any unit vector $\mathbf{y}$,

$$\begin{aligned} \mathbf{y}^*(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0)\mathbf{y} &= \mathbf{y}^*(\mathbf{A}_0\mathbf{P}_0)\mathbf{y} + \mathbf{y}^*(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0)\mathbf{y} \quad \Longrightarrow & (2.15)\\ \mathbf{y}^*(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0)\mathbf{y} &\in \mathcal{F}(\mathbf{A}_0\mathbf{P}_0) + \mathcal{D}(\mathbf{0},\gamma), & (2.16)\end{aligned}$$

where $\mathcal{F}$ denotes the field of values and $\mathcal{D}(\mathbf{0},\gamma)$ is the closed disk centered at the origin with radius γ. So, $\mathcal{F}(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0) \subseteq \mathcal{F}(\mathbf{A}_0\mathbf{P}_0) + \mathcal{D}(\mathbf{0},\gamma)$. In this case γ could be chosen based on an estimate of $\min|\mathcal{F}(\mathbf{A}_0\mathbf{P}_0)|$ (obtained during the iterative solve with $\mathbf{A}_0\mathbf{P}_0$).

This chapter focuses on solving (2.8) for each k or selected k, but we can also apply such a map incrementally. In that case, for the k^{th} matrix we solve

$$\mathbf{N}_{k,j_m} = \arg\min_{\mathbf{N}\in\mathcal{S}} \|\mathbf{A}_k\mathbf{N} - \mathbf{A}_{j_m}\|_F, \tag{2.17}$$

and define $\mathbf{P}_k = \mathbf{N}_{k,j_m}\mathbf{P}_{j_m}$ with $\mathbf{P}_{j_m} = \mathbf{N}_{j_m,j_l}\mathbf{P}_{j_l}$, for $0 \leq j_l < j_m < k$ (and so on). This includes the special case $j_m = k-1$, $j_l = j_m - 1$ (and so on).

When preconditioning from the left, we can take advantage of row-wise changes made to $\mathbf{A}_k$, as is the case with the MCMC matrices described above. We can define

$$\mathbf{N}_k = \arg\min_{\mathbf{N}\in\mathcal{S}} \|\mathbf{N}\mathbf{A}_k - \mathbf{A}_0\|_F, \tag{2.18}$$

with $\mathbf{P}_k = \mathbf{P}_0\mathbf{N}_k$. In this case, the computation of the map can be made significantly cheaper by considering only those rows of $\mathbf{A}_k$ that differ from $\mathbf{A}_0$ when computing the least squares minimization. The same applies when computing the map as in (2.8) if only a few columns of the matrix change. Using the maps as in (2.17) and (2.18) is future research.

Other preconditioner update schemes for sequences of linear systems have been proposed. For these schemes, though, the initial preconditioner typically must be of a specific type. A cheap update to the factorized approximate inverse (AINV) preconditioner is discussed in [30]. Algorithms computing AINV preconditioners and AINV updates can be found in [19, 30, 31, 32, 33, 34, 112]. Several incremental, or iterative, update techniques to an ILU factorization are described in [44]. Two efficient update techniques for an ILUT decomposition in a matrix-free environment are described in [137]. The balanced incomplete factorization [42], a modification of ILU, can be updated using the scheme proposed in [50]. For symmetric positive definite (SPD) matrices, we may consider an incomplete Choleksy (IC) factorization for our preconditioner. An overview of efficient techniques for updating an IC preconditioner using low rank updates is provided in [36]. For SPD matrices with a diagonal shift, we could use the update scheme to an $\mathbf{LDL}^T$ factorization presented in [20]. For a sequence of complex symmetric systems defined by a diagonal shift, we can also consider the update to an $\mathbf{LDL}^H$ factorization as presented in [39].

A cheap update for incomplete factorizations of sequences of linear systems is presented in [9], which uses the iterative algorithm proposed in [53]. In [53], the authors introduce a

method for computing an ILU factorization in parallel and provide insight into the costs of computing an ILU factorization on modern architectures. The update scheme in [9] uses a factorization for a previous matrix in the sequence as an initial guess to the iterative algorithm from [53]. This results in an update to the previous factorization. Good results are provided for a sequence of linear systems coming from model reduction [138], however this update scheme requires that the initial preconditioner be an ILU factorization.

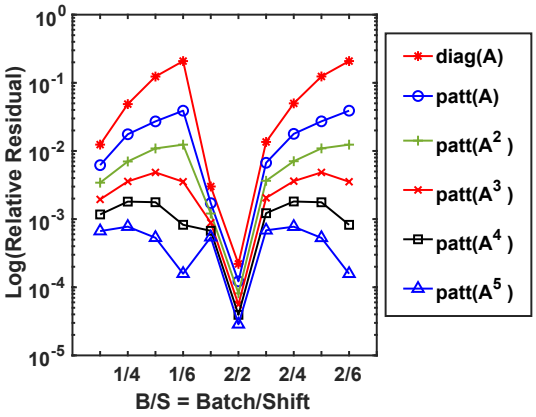
A scheme for recomputing an AMG preconditioner at a reduced cost for sequences of stochastic collocation systems is proposed in [75]. Prior to its use as either a linear solver or a preconditioner, a (generally) expensive setup phase is necessary for AMG. In [75], the authors show that by reusing some of this setup information for a sequence of systems, the setup phase for the next system in the sequence can be performed faster than doing so from scratch. Again, this update scheme is specific to the AMG preconditioner.

2.3 Experimental Analysis of Sparsity Patterns for SAMs

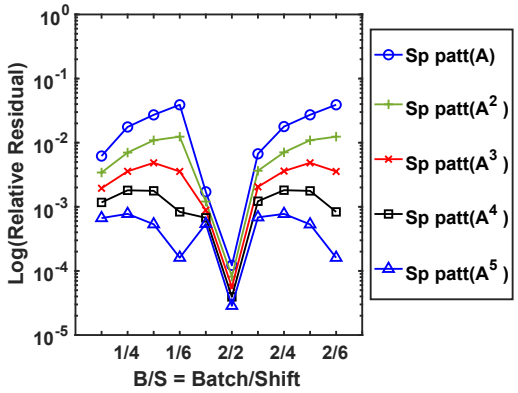
As the cost and effectiveness of a SAM depends on the sparsity pattern, we analyze some choices here. In choosing sparsity patterns for SAMs, we aim to balance the cost of computing and applying the map with the number of GMRES iterations to reduce total runtime. For SAIs, both adaptive and fixed sparsity patterns have been considered. Computing the pattern and preconditioner adaptively tends to be expensive [29, 35, 51, 52, 89]. Therefore, we focus on fixed, a priori, sparsity patterns for SAMs, although some previous work on SAIs has focused on making adaptive strategies more efficient [54, 79, 90]. We examine choices in sparsity patterns using matrices from two applications, flow and topology optimization (see Sections 2.5.2 and 2.5.1, respectively, for more detail on these applications).

Sparsity patterns derived from powers of the matrix, $\mathbf{A}$, are a standard choice. This choice is based on the decay of the elements in the matrix representing the discrete Green's function [51, 89, 136] associated with the Laplace operator. While the denser sparsity patterns of higher powers of $\mathbf{A}$ may result in better maps, solving (2.8) becomes more costly. We can alleviate this cost by sparsifying the powers of $\mathbf{A}$. One possibility is to discard elements of $\mathbf{A}^k$ that are smaller in magnitude than a chosen threshold [51]. An alternative are sparsity patterns derived from the mesh on which the matrix is based. Using the underlying mesh for the topology optimization application, we experiment with sparsity patterns that are subsets of the sparsity pattern of the matrix, leading to maps that are much sparser than the system matrix.

We first analyze sparsity patterns defined by powers of the matrix, from $\mathbf{A}$ to $\mathbf{A}^5$, sparsifications of those, and the diagonal pattern. The sparsifications are obtained using a global threshold of 10^{-2}. To further minimize cost, we first sparsify $\mathbf{A}$ using this global threshold, and then take powers of the logical matrix representing the nonzero pattern of this sparsified



(a) SAM relative residual norm vs shift for flow matrices, for all patterns without sparsification.



(b) SAM relative residual norm vs shift for flow matrices, for all patterns with sparsification. "Sp Patt" indicates "Sparsified Pattern".

Figure 2.1: Relative SAM residual, $\frac{\|\mathbf{A}_k\mathbf{N}_k-\mathbf{A}_0\|_F}{\|\mathbf{A}_0\|_F}$, of the flow matrices using a priori sparsity patterns for the SAM updates, $\mathbf{N}_k$, for batch/shift 1/3 through 2/6. An ILUTP preconditioner is computed for batch/shift 1/2. Details of these linear systems and the notation are provided in Sections 2.3 and 2.5.2.

A. We test these patterns for the flow matrices [106, 138]. We study the relative residual norms of the resulting SAMs, the time to compute the map, the number of GMRES iterations and GMRES runtime, and the total solution time. GMRES(200) is used for this application. As the sparsity pattern of the map becomes denser, we expect the approximation in (2.8) to become more accurate, resulting in fewer GMRES iterations.

We consider eleven flow matrices, $\mathbf{A}_k = s_k\mathbf{E} - \mathbf{A}$, and corresponding linear systems for a fixed right hand side $\mathbf{b}$ (see Section 2.5.2). The Iterative Rational Krylov Algorithm (IRKA) [81] computes optimal interpolation points, or shifts. Each iteration of IRKA generates a batch of new shifts until IRKA converges. We will refer to these matrices using the notation *batch/shift* (e.g., 1/2 is batch 1, shift 2). We consider shifts 2 through 6 and 1 through 6 of batches 1 and 2, respectively (see Table 2.10 in Section 2.5.2 for a list of all shifts).

We compute an ILUTP preconditioner, $\mathbf{P}_0$, for the first system in the sequence, $\mathbf{A}_0$, and we consider the powers of $\mathbf{A}_0$ to derive the denser patterns. For the flow application, we compute and recycle the preconditioner for 1/2, not 1/1. For this first shift the matrix is quite ill-conditioned and not diagonally dominant, and this leads to a poor ILUTP preconditioner. So, we compute an ILUTP for this second matrix and recycle it for subsequent systems. We define the residual of the map and its relative residual norm as

$$\mathbf{R}_k = \mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0 \quad \text{and} \quad \frac{\|\mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0\|_F}{\|\mathbf{A}_0\|_F}.$$

Figure 2.1 gives the relative residual norm for all shifts and all patterns, and Tables 2.1 and

Batch/Shift	Diag	Patt **A**	Patt $\mathbf{A}^2$	Patt $\mathbf{A}^3$	Patt $\mathbf{A}^4$	Patt $\mathbf{A}^5$
1/2	13	13	13	13	13	13
1/3	20	19	18	17	16	15
1/4	33	30	27	27	25	23
1/5	108	53	49	44	38	31
1/6	202	79	54	34	20	13
2/1	496	488	1620	(5001)	(5001)	(5001)
2/2-2/6	376	194	160	134	112	95
********** **Totals** **********						
Total Iter	1248	876	1941	5270	5225	5191
SAM Time (s)	1.04	2.04	3.57	9.43	19.33	41.42
GMRES Time (s)	3.16	1.47	3.49	11.70	14.41	16.36
Total Time (s)	5.01	4.32	7.87	21.93	34.53	58.58
nnz($\mathbf{N}_k$)/n	1	6.97	18.94	37.02	61.39	92.36
********** **Totals without 2/1** **********						
Total Iter	752	388	321	269	224	190
SAM Time (s)	0.95	1.85	3.22	8.51	17.41	37.30
GMRES Time (s)	2.44	0.75	0.60	0.56	0.54	0.53
Total Time (s)	4.19	3.39	4.63	9.87	18.75	38.62

Table 2.1: GMRES iterations and total run times for selected shifts of the flow matrices using a priori sparsity patterns for the SAM update. An ILUTP preconditioner is computed for the system 1/2, with SAM updates computed for all remaining shifts 1/3-2/6. "Patt" indicates ""Pattern of". "SAM time" is the total amount of time spent computing all ten maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 0.8 s to compute. Totals are given for the entire sequence, as well as for the sequence omitting system 2/1. (5001) indicates GMRES did not converge in the maximum allowed iterations.

2.2 give, for all patterns, GMRES iterations for selected shifts and (total) run times.

The data show that for subsequent shifts of increasing magnitude[2], the relative residual norm grows and the recycled preconditioner $\mathbf{N}_k\mathbf{P}_0$ becomes less effective, resulting in more GMRES iterations as expected. Nevertheless, recycling preconditioners using SAMs keeps the iteration counts relatively low for most shifts; see Section 2.5.2 for more details and for comparison with the results listed here. The data also show that, as the relative residual norm decreases for denser patterns, the number of iterations decreases as well, with the exception of batch/shift 2/1, which corresponds to a very hard system. In Tables 2.1 and 2.2, we also show the cumulative results for both the entire sequence, as well as the sequence omitting batch/shift 2/1. In these tables, we specifically show iterations for batch/shift 1/2 through 2/1, and then aggregate numbers for batch/shift 2/2-2/6. For 2/2-2/6, the iterations for each shift (and the rate of growth of those iterations) are similar to those of 1/2-1/6.

However, for the sparsity patterns derived from higher powers of $\mathbf{A}_0$, the decrease in runtime from reduced GMRES iterations does not outweigh the increased costs of computing more expensive SAMs. For the flow matrices, using the pattern of $\mathbf{A}_0$ comes out as the most

[2]Note that the relative residual norm does drop at matrix 2/2, and begins to increase again for 2/3-2/6. As IRKA iterates, the corresponding shifts from one batch to the next do not drastically change (see Table 2.10).

Batch/Shift	Sp Patt $\mathbf{A}$	Sp Patt $\mathbf{A}^2$	Sp Patt $\mathbf{A}^3$	Sp Patt $\mathbf{A}^4$	Sp Patt $\mathbf{A}^5$
1/2	13	13	13	13	13
1/3	19	18	17	16	15
1/4	30	27	27	25	23
1/5	53	50	45	40	34
1/6	81	62	48	39	35
2/1	(5001)	1437	(5001)	(5001)	(5001)
2/2-2/6	196	170	149	133	120
********** **Totals** **********					
Total Iter	5393	1777	5300	5267	5241
SAM Time (s)	1.95	3.13	6.46	12.37	25.64
GMRES Time (s)	8.57	3.14	10.79	12.48	14.60
Total Time (s)	11.32	7.07	18.05	25.66	41.04
nnz($\mathbf{N}_k$)/n	5.30	13.64	26.26	43.36	65.39
********** **Totals without 2/1** **********					
Total Iter	392	340	299	266	240
SAM Time (s)	1.75	2.83	5.82	11.14	23.11
GMRES Time (s)	0.80	0.69	0.65	0.61	0.58
Total Time (s)	3.35	4.32	7.27	12.55	24.49

Table 2.2: GMRES iterations and total run times for selected shifts of the flow matrices using a priori *sparsified* sparsity patterns. A global threshold of 10^{-2} is used to sparsify. An ILUTP preconditioner is computed for system 1/2, with SAM updates computed for all remaining shifts 1/3-2/6 "Sp Patt" indicates "Sparsified Pattern". "SAM time" is the total amount of time spent computing all maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 0.8 s to compute. Totals are given for the entire sequence, as well as the sequence omitting system 2/1. (5001) indicates GMRES did not converge in the maximum allowed iterations.

efficient for total runtime (both with and without batch/shift 2/1), and we use this pattern for the experiments in Section 2.5.2. Though, if we omit batch/shift 2/1, the sparsified pattern of $\mathbf{A}_0$ leads to a slightly lower overall total runtime.

Next, we analyze sparsity patterns derived from the finite element mesh from which the matrices are derived, and we focus on subsets of the sparsity pattern of the system matrix that are much sparser than the matrix itself. This leads to maps that are cheap to compute compared with ILU-type preconditioners, which is particularly relevant for matrices that have many nonzeros per column. For this reason, we examine a sequence of matrices arising in topology optimization [143, 145] that result from discretization of the 3D linear elasticity equations on a $100 \times 20 \times 20$ trilinear (B8) element mesh, with three unknowns per node, u, v, and w, giving the displacements in the x-, y-, and z-direction. We order nodes and variables per node lexicographically. The size of these matrices is $n = 132\,300$, the maximum number of nonzeros in a column is 81, and the average number of nonzeros per column varies but is approximately 70 (after the first few iterations). In the numerical results given in Section 2.5.1, we also consider a second, larger mesh for topology optimization.

To describe the stiffness matrix derived from the mesh, we consider a typical node (i, j, k) that is not on the boundary. Let s, $s + 1$, and $s + 2$ be the column indices corresponding

to the u, v, and w variables, respectively, for node (i, j, k). Then columns $s+3$, $s+4$, and $s+5$ correspond to the u, v, and w variables, respectively, for node $(i+1, j, k)$, and columns $s-3$, $s-2$, and $s-1$ correspond to the u, v, and w variables for node $(i-1, j, k)$. Column $s+300$ corresponds to the u variable for node $(i, j+1, k)$, and $s+6300$ corresponds to the u variable for node $(i, j, k+1)$. For the remainder of the stiffness matrix, we refer to Figure 2.2(a), which shows the column indices, $s+m$, corresponding to the u variables of nodes in elements that contain node (i, j, k). The column indices for the v and w variables at those nodes are given by $s+m+1$ and $s+m+2$, respectively, for each m.

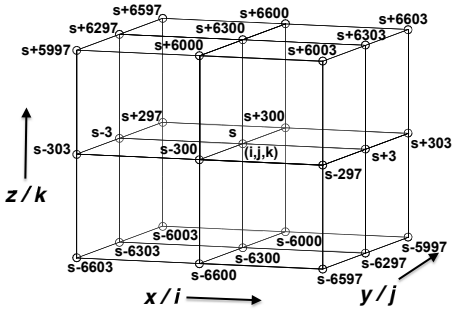
With 27 nodes, each with three displacements, there are numerous choices in sub-patterns of the matrix sparsity pattern that can be made depending on user preference. A user may include certain displacements and omit others, dictating how many and which nonzeros will be included in the sparsity pattern. We evaluate five of such possible patterns, Patt-0 to Patt-4. For each, we choose a selection of mesh nodes relative to (i, j, k) and displacements associated with those nodes. We stress that each of these patterns (and any of the other resulting patterns from choosing a different subset of displacements) are much sparser than the stiffness matrix itself.

To define Patt-0 for the u column of the node (i, j, k), corresponding to column s, we combine its index with the index for the u variable at the nodes marked by '$\times$' in Figure 2.2(b). The resulting sparsity pattern contains, for column s, the ordered pairs (s, s), $(s \pm 3, s)$, $(s \pm 300, s)$, and $(s \pm 6300, s)$. For the v and w columns, we use the same pattern, but with s indicating the v and w columns for node (i, j, k). On boundaries, this pattern is adjusted to take the boundary and boundary conditions into account. This will also be done for Patt-1 discussed below.

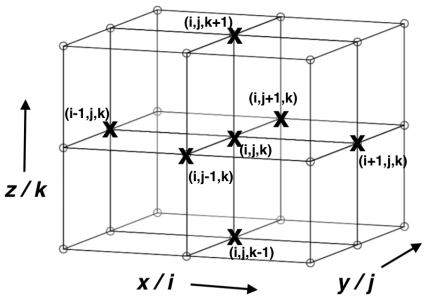
Patt-1 was obtained by experimenting with minor variations of Patt-0. This sparsity pattern contains, for column s, the ordered pairs (s, s), $(s \pm 1, s)$, $(s \pm 300, s)$, and $(s \pm 6000, s)$. For the v and w columns, we use the same pattern, but with s indicating the v, respectively, w column for node (i, j, k). Patterns Patt-2, Patt-3, and Patt-4 include more column indices from the original sparsity pattern of the matrix based on the finite element mesh described above.[3] All patterns, Patt-0–Patt-4, contain *substantially fewer* nonzeros than the stiffness matrix. Table 2.3 shows the numbers of nonzeros in the sparsity patterns with timing and iterations results.

To evaluate the effectiveness of these patterns, we compute an ILUTP preconditioner for the matrix at optimization step 100, and compute SAMs for the matrices at steps 105, 110, 115, 120, 125, and 130. We solve the preconditioned systems using full GMRES. The results are shown in Table 2.3. While these maps have substantially fewer nonzeros than the matrices themselves, recycling the initial preconditioner using these SAMs keeps the GMRES iterations low. When computing the SAM with either Patt-0 or Patt-1, each map takes less than three seconds to compute. Including more nonzeros from the sparsity pattern

[3]We omit specific details for Patt-2–Patt-4 here for brevity. We refer the interested reader to [48] for discussions of these patterns.



(a) The 8 elements containing node (i, j, k) and the column indices, $s + m$, in the stiffness matrix that correspond to the u variables of their nodes, where s is the column index corresponding to the u variable for node (i, j, k).



(b) Mesh nodes involved in the sparsity pattern Patt-0. The details are given in the text.

Figure 2.2: Descriptions of the finite element mesh and sparsity patterns for the topology optimization matrices.

of the stiffness matrix (Patt-2–Patt-4), decreases the total number of GMRES iterations a bit further; however, there is a substantial increase in the time to compute the maps. Both times and iteration counts are comparable between Patt-0 and Patt-1 as shown in Table 2.3, though the latter is slightly faster across the longer sequence of topology optimization matrices considered in Section 2.5.1. So, we show results using Patt-1 for those experiments. In particular, we demonstrate the effectiveness of recycling preconditioners using SAMs in detail in Section 2.5, providing comparisons with recomputing preconditioners and reusing preconditioners for several applications.

2.4 Implementation

We have developed efficient implementations for computing SAMs and ILUTP preconditioners as MATLAB® m-files to make useful runtime comparisons between recycling a preconditioner and computing a new one. We use a MATLAB® implementation of the AMG preconditioner developed as part of the work done in [87]. We also refer the reader to [116, 117, 133, 134, 135] [140, Appendix A: An Introduction to Algebraic Multigrid, K. Stüben].

First, we describe an efficient implementation for computing SAMs. To efficiently compute the SAM updates, the solution of (2.8) should be implemented in sparse-sparse fashion. For most problems, the nonzero pattern of the matrices does not change, and we preprocess the sparsity pattern for the maps to set up data structures for the small least squares problems just once; see Algorithm 1. Since we store $\mathbf{A}_k$ as a MATLAB® sparse matrix, access to

Optimization Step	Patt-0 Iter	Patt-1 Iter	Patt-2 Iter	Patt-3 Iter	Patt-4 Iter
100	166	166	166	166	166
105	173	173	173	170	169
110	184	184	183	177	176
115	195	196	194	184	183
120	208	208	205	190	189
125	216	220	213	194	193
130	228	226	222	197	197
Total Iter	1370	1373	1356	1278	1273
SAM Time (s)	14.96	15.10	45.47	91.52	97.23
GMRES Time (s)	97.21	98.49	98.19	95.03	94.96
Total Time (s)	234.57	236.00	266.06	308.96	314.61
nnz($\mathbf{N}_k$)/n	5.47	5.33	9.46	29.07	33.67

Table 2.3: GMRES iterations and total run times for matrices from selected steps of the topology optimization application using a priori mesh-based sparsity patterns. "SAM time" is the total amount of time spent computing all maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 122.41s to compute.

columns is cheap.

Given a pattern, S, define $s_\ell = \{i \,|\, (i,\ell) \in S\}$, the set of indices of potential nonzeros in $\mathbf{n}_\ell$, column ℓ of $\mathbf{N}_k$. To compute $\mathbf{n}_\ell$, we only need the m columns $\mathbf{a}_j$ of $\mathbf{A}_k$ such that $j \in s_\ell$, and as these columns are sparse we need only consider rows i such that $a_{i,j} \neq 0$ for some $j \in s_\ell$. Let r_ℓ be the set of relevant row indices. Then the least squares problem for $\mathbf{n}_\ell$ is defined as

$$\mathbf{n}_\ell(s_\ell) \quad = \quad \arg\min_{\widetilde{\mathbf{n}} \in \mathbb{C}^m} \|\mathbf{A}_k(r_\ell, s_\ell)\widetilde{\mathbf{n}} - \mathbf{A}_0(r_\ell, \ell)\|_2,$$

where $\mathbf{A}_k(r_\ell, s_\ell)$ is the submatrix of $\mathbf{A}_k$ indexed by $r_\ell \times s_\ell$, and $\mathbf{A}_0(r_\ell, \ell)$ is the corresponding part of the ℓth column of $\mathbf{A}_0$. Note that if $(a_0)_{i,\ell} \neq 0$ but $a_{i,j} = 0$ for all $j \in s_\ell$, row i is irrelevant for computing $\mathbf{n}_\ell$ since $\mathbf{e}_i \perp \text{Span}(\{\mathbf{a}_j \,|\, j \in s_\ell\})$. However, if we wish to compute, in addition to $\mathbf{N}_k$, the residual $\mathbf{R}_k = \mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0$ or its norm on Line 7 of Algorithm 2, we need to include such rows as well. If the matrices $\mathbf{A}_k$ and $\mathbf{A}_0$ have the same sparsity pattern and the pattern of $\mathbf{N}_k$ includes at least the diagonal, this is not an issue. So, the size of each least squares problem depends only on the chosen sparsity pattern of $\mathbf{N}$ and the sparsity pattern of $\mathbf{A}_k$, not on the matrix size. So the least squares problems are small, independent of n, and most are about the same size. For example, for the flow application discussed in Section 2.5.2, the maximum size of the least squares problems to compute the SAM updates is 20×7, independent of $n = 9\,669$.

Finally, to ensure the map is computed and then stored as efficiently as possible, in Lines 3-11 of Algorithm 2, we compute $\mathbf{N}$ in coordinate format (COO). After the entire map has been computed, we convert this temporary data structure into a MATLAB® sparse matrix using the command `sparse` in Line 12.

Our implementation of ILUTP closely follows [118, 119]. To make the implementation efficient in MATLAB®, we made the following main changes. (1) We transpose $\mathbf{A}$, in

Algorithm 1 Preprocessing for Computing Sparse Approximate Maps

Given sparsity pattern S and matrix $\mathbf{A}$
$maxSk = 0$; $maxRk = 0$; { initialize max num of columns, max num of rows }
for $k = 1 : n$ **do** { for each column do }
    $s_k = \{i \,|\, (i,k) \in S\}$ { get indices; typically defined in advance }
    $r_k = \emptyset$ { Initialize set of rows for kth LS problem }
    for all $j \in s_k$ **do**
        $t = \texttt{find}(\mathbf{a}_j)$ { find indices of nonzeros in column $\mathbf{a}_j$ }
        $r_k = r_k \cup t$
    end for
    $nnz_k = \#(s_k)$ { #() gives number of elements in a set }
    if $nnz_k > maxSk$ **then** $maxSk = nnz_k$ **end if**
    if $\#(r_k) > maxRk$ **then** $maxRk = \#(r_k)$ **end if**
end for
Allocate $maxRk \times maxSk$ array for storing the LS matrices, $maxRk$ vector for storing the right hand side, and $maxSk$ vector for storing the solution.

Algorithm 2 Computing $\mathbf{N} = \arg\min_{\widetilde{\mathbf{N}} \in \mathcal{S}} \|\mathbf{A}\widetilde{\mathbf{N}} - \widehat{\mathbf{A}}\|_F$

$cnt = 0$ { counts number of nonzeros in preconditioner }
(Preallocate space for $\mathbf{A}_{\text{tmp}}$)
for $k = 1 : n$ **do**
    $\mathbf{A}_{\text{tmp}} = \mathbf{A}(r_k, s_k)$ { get submatrix indexed by r_k and s_k for LS problem }
    $\mathbf{f} = \widehat{\mathbf{A}}(r_k, k)$ { get rhs for LS problem }
    Solve LS $\mathbf{A}_{\text{tmp}}\mathbf{z} = \mathbf{f}$
    (possibly save residual, norm of residual, etc.)
    $rowN[cnt + 1 : cnt + nnz_k] = s_k$ { assign indices in order of row ind. in s_k }
    $colN[cnt + 1 : cnt + nnz_k] = k$
    $valN[cnt + 1 : cnt + nnz_k] = \mathbf{z}$
end for
$\mathbf{N} = \texttt{sparse}(rowN, colN, valN)$ { convert into sparse matrix }

MATLAB® sparse matrix storage, to access the rows efficiently. (2) Where possible, we use MATLAB® routines, such as `find`, `sort`, `sparse`, and `min`. (3) Where possible, we have vectorized loops. (4) We use `sparse` to build **L** and **U** efficiently, and we use `tril` and `triu` to ensure MATLAB® recognizes and uses the **L** and **U** factors as triangular matrices. The m-file is available from [47].

2.5 Numerical Experiments

We analyze the effectiveness of reusing, recomputing, and recycling preconditioners for several applications. All systems are solved using preconditioned GMRES. We compare the results of computing a new ILUTP or AMG preconditioner for each matrix or selected matrices, reusing the initial $\mathbf{P}_0$ for all systems, updating $\mathbf{P}_0$ with a new SAM update for all systems, and updating $\mathbf{P}_0$ with a SAM update only at selected systems in the sequence, combined with computing a new $\mathbf{P}_0$ at selected systems. Computing a new preconditioner for every matrix is always the most expensive in runtime, but it provides a useful benchmark in terms of the number of iterations. Computing a SAM update only at selected systems (for a long sequence, combined with recomputing the preconditioner at selected systems) is usually the winner in runtime. We report run times for computing preconditioners and SAMs, the number of iterations and runtime to solve each system, and total runtime and number of iterations for the whole sequence. Our focus is total runtime.

We tested several indicators for computing a new SAM update or new preconditioner. A simple and effective strategy is to compute a new SAM or preconditioner based on the estimated time for this computation and the (relative) increase in the solution time for a system or the number of iterations. For the smaller topology optimization problem, we give results for recomputing based on a percent increase in iterations.

2.5.1 Topology Optimization

This test problem leads to a long sequence of linear systems, where the matrix has a relatively large number of nonzeros per row. As a result, this problem is particularly useful to demonstrate effective SAMs that are much sparser than the matrix itself.

Topology optimization is a structural optimization method that optimizes the material distribution inside a given domain [24, 104, 113, 126, 145]. The method computes a design by determining the locations where material should be located (and where it should not), combining finite element approximation, linear solvers (for linear partial differential equations), and optimization. In this case, we minimize the compliance (see below) subject to a volume

constraint. We can define the minimization problem as

$$\begin{aligned} \min_{\rho,\mathbf{u}} \quad & c(\rho,\mathbf{u}) = \mathbf{u}^T\mathbf{A}(\rho)\mathbf{u} \\ \text{s.t.} \quad & \mathbf{A}(\rho)\mathbf{u} = \mathbf{f} \\ & 0 < \rho_0 \le \rho_e \le 1 \qquad e = 1, 2, \cdots, n_e \\ & \int_\Omega \rho \, \mathrm{d}\Omega \le V, \end{aligned}$$

with c, the compliance; $\mathbf{A}(\rho)$, the stiffness matrix; $\mathbf{u}$ and $\mathbf{f}$, respectively the displacement and load vectors; and V, the total volume. Here, $\rho_0 > 0$ represents the lower bound for the density displacement, which avoids potential singularity of the stiffness matrix. The Solid Isotropic Material with Penalization method [21, 23] is used to model the materials by using a single design variable for the density in an element. We also refer the reader to [143, 144] for more details on this problem.

The structure of the matrices is detailed in Section 2.3. We consider matrices that result from discretizing the 3D linear elasticity equations for variable density on a trilinear (B8) finite element mesh. We examine two such meshes with (1) $100 \times 20 \times 20$ elements, and (2) $150 \times 30 \times 30$ elements, with three unknowns per node, u, v, and w, giving the displacements in the x-, y-, and z-directions resulting in matrices of sizes (1) $n = 132\,300$, and (2) $n = 432\,450$. (see [143, 145] for details). We refer to these respectively as the "small" and "large" topology optimization matrices. For both, we take a representative sequence from an optimization that typically takes hundreds, but possibly up to 1000, optimization steps before reaching the optimal design. We demonstrate the effects of several strategies for reusing, recycling, and recomputing the preconditioner. We use the ILUTP preconditioner for the small matrices and the AMG preconditioner for the large matrices. Results are provided in Tables 2.4-2.9.

For the large matrices, we show that in the case of an expensive preconditioner resulting in fast GMRES convergence, we can preserve this good convergence with SAM updates. We use a path cover adaptive algebraic multigrid [87] for the preconditioner. The grid and operator complexities are 1.166 and 1.863, respectively.[4] In particular, both are less than 2.0. The computation of this preconditioner takes about 16.5 minutes, whereas the SAM updates take less than five seconds, with five additional seconds for preprocessing at the first update. We use full GMRES with maximum iterations set to 400 and the zero initial guess.

Table 2.4 shows the results when we reuse the initial AMG preconditioner. In this case, the GMRES iterations grow quickly and reach the maximum allowed twice. At that point, we compute a new preconditioner and reuse it for subsequent systems. So, we recompute the preconditioner halfway[5] through the sequence and would do this again at optimization

[4]The grid complexity gives the ratio of the number of all grid points to that on the finest level, and the operator complexity gives the ratio of the number of nonzero entries in all operators to that on the finest level [135][140, p. 487, Appendix A: An Introduction to Algebraic Multigrid, K. Stüben].

[5]If we let GMRES continue beyond the maximum number of iterations for system 65, convergence is reached at 435 iterations.

Mats	Prec (s)	GMRES (s)	Iter
60	994.52	80.34	35
61	0	127.03	58
62	0	222.88	99
63	0	416.09	171
64	0	741.18	299
65	0	1026.75	400
66	1002.24	83.57	37
67	0	168.36	77
68	0	623.67	259
69	0	1026.96	400
Total	6513.59		1835

Table 2.4: Run times and iterations for the large topology optimization systems, when reusing the AMG preconditioner for each system after the first. When max iterations (400) is reached, we recompute the AMG preconditioner for the next system and reuse this for subsequent systems.

Mats	Prec (s)	GMRES (s)	Iter
60	994.52	80.34	35
61	10.52	81.92	38
62	4.97	88.08	41
63	4.89	94.31	44
64	4.75	101.35	47
65	4.80	112.10	52
66	4.89	126.10	58
67	4.92	147.39	68
68	4.95	177.88	81
69	4.93	221.85	101
Total	2275.47		565

Table 2.5: Run times and iterations for the large topology optimization systems, when computing the AMG preconditioner for the first system and a SAM update for each system after the first. SAMs use the sparsity pattern Patt-1.

step 70. Total computation time is nearly two hours. Apparently, modest changes in material distribution may already make the coarse grid correction in the AMG preconditioner substantially less effective.

Table 2.5 shows the results when we update the initial AMG preconditioner using SAMs at every step. We use Patt-1, described in Section 2.3, which has seven nonzeros in a typical column compared with up to 81 nonzeros in a typical column of the stiffness matrix. As a single GMRES iteration takes about two seconds (in both tests) due to the ILU-smoothing in the AMG preconditioner, preserving a low number of iterations is important. Recycling the initial preconditioner using SAM updates achieves this goal, with the number of iterations growing slowly from 35 to 101, avoiding recomputations of the preconditioner. The total computation time when recycling the AMG preconditioner using SAMs is reduced to less than 38 minutes.

For the small topology optimization matrices, computing the ILUTP preconditioner takes about two minutes.[6] Fill-in for ILUTP is set to 250 and the drop tolerance is 10^{-3}. We use Patt-1 for the SAMs, as it gives slightly faster run times then Patt-0 for the sequence of systems considered here. Computing a SAM update takes a bit over two seconds. We use full GMRES with the maximum number of iterations set to 1000. When computing an ILUTP preconditioner for topology optimization matrices, diagonal scaling is essential

[6]Using `ilu` (with type 'ilutp') in MATLAB® also takes about two minutes, with approximately the same number of nonzeros in the **L** and **U** factors, and results in a similar number of GMRES iterations. The matrices are SPD but far from diagonally dominant. MATLAB®'s IC(0) results in a poor preconditioner, while its incomplete Cholesky with threshold fails with negative pivots.

Mats	Prec (s)	GMRES (s)	Iter
40-49	120.85	134.98	1909
50-59	0	202.42	2667
60-69	0	272.05	3353
70-79	0	331.72	3880
80-89	0	396.47	4405
90-99	0	892.64	7405
100-109	0	1375.30	(9779)**
110-119	0	1430.16	(10010)*
120-129	0	1432.25	(10010)*
130-139	0	1436.56	(10010)*
140-149	0	1429.65	(10010)*
150-166	0	2454.73	(17017)*
Totals	11909.79		90455

Table 2.6: Run times and iterations for the small topology optimization systems with the initial ILUTP reused for all systems. Results are given in groups of ten except for (150-166). **For only one system convergence was reached within max iterations. *Convergence was not reached for any system within max iterations.

Mats	Prec (s)	GMRES (s)	Iter	Gain GMRES (s)	Gain Iter
40-49	120.85	134.98	1909	0	0
50-59	2.41	189.72	2495	–12.71	–172
60-69	2.19	241.57	3012	–30.48	–341
70-79	2.15	283.75	3396	–47.97	–484
80-89	2.13	333.22	3823	–63.25	–582
90-99	2.14	361.81	4067	–530.83	–3338
100-109	2.14	404.84	4414	–970.46	–5365
110-119	2.14	476.00	4799	–954.16	–5211
210-129	2.29	446.45	4689	–985.80	–5321
130-139	2.15	465.68	4844	–970.88	–5166
140-149	2.18	535.47	5324	–894.18	–4686
150-166	4.30	1689.75	(13187)*	–903.72	–3830
Totals	5710.32		55959	–6225.71	–34496

Table 2.7: Run times and iterations for the small topology optimization systems with the initial ILUTP recycled with a SAM update for every tenth system, using Patt-1. Results are given in groups of ten except for (150-166). "Gain" indicates the decrease in GMRES time and iterations compared with reusing the initial ILUTP. There are two SAM updates in (150-160). *For one matrix convergence was not reached within max iterations.

to mitigate the huge ratio in local stiffness [144]. This scaling varies with each matrix and combined with pivoting complicates using the previous solution as an initial guess, so we use the zero initial guess. Table 2.6 provides the results for reusing the initial preconditioner for all systems. Results for computing a new preconditioner for every system are not shown, given the high cost of computing the ILUTP preconditioner. Computing the SAM update for every matrix after the first reduces iterations substantially, but is still too expensive in runtime. Therefore, we compute a SAM update for each tenth system and reuse the recycled preconditioner until the next update. Results are provided in Table 2.7.

We can improve Run times by recomputing the ILUTP preconditioner and the SAMs based on increases in iterations. We refer to this strategy as the "dynamic strategy". Considering only the ILUTP preconditioner, we recompute the preconditioner when the number of iterations at an optimization step increases by 50% over those from the latest ILUTP computation. Table 2.8 shows the results. In addition, we can compute a SAM update when the number of iterations at an optimization step increases by 20% over those from the latest ILUTP computation. We only compute one SAM update between ILUTP computations. The results are given in Table 2.9.

The dynamic strategy for both ILUTP computation and SAM update reduces the total number of GMRES iterations (by 578 iterations), the overall runtime (by 129 seconds), and

Mats	Prec (s)	GMRES (s)	Iter	ILUTP at Step
40-53	120.85	210.85	2891	40
54-74	120.92	320.93	4402	54
75-98	123.24	358.48	4921	75
99-132	120.26	524.04	7181	99
133-161	121.42	433.29	5993	133
162-166	121.21	46.49	667	162
Totals	2621.90		26055	(×6)

Table 2.8: Run times and iterations for the small topology optimization matrices with the dynamic strategy for ILUTP only. Each group of systems starts with a new ILUTP, as indicated in the last column.

Mats	Prec (s)	GMRES (s)	Iter	ILUTP at Step	SAM at Step
40-54	123.29	223.513	3057	40	47
55-77	122.52	358.62	4735	55	66
78-103	122.65	391.04	5297	78	90
104-144	122.31	610.46	8305	104	122
145-166	122.95	295.21	4083	145	160
Totals	2492.50		25477	(×5)	(×5)

Table 2.9: Run times and iterations for the small topology optimization matrices with the dynamic strategy for both ILUTP and SAM updates with Patt-1. Each group of systems starts with a new ILUTP, and includes one SAM update, as indicated in the last two columns.

the number of ILUTP computations (by 1) compared with the dynamic strategy for the ILUTP preconditioner only. Over the full sequence of optimization steps, these numbers increase proportionally.

2.5.2 Interpolatory Model Reduction

Our next set of linear systems arises in the Iterative Rational Krylov Algorithm (IRKA) [8, 81] for computing H_2-optimal interpolatory reduced order models [8]. Model reduction aims to replace a high-dimensional linear dynamical system (here single-input/single-output)

$$\mathbf{E}\dot{\mathbf{x}}(t) + \mathbf{A}\mathbf{x}(t) = \mathbf{b}u(t), \quad y(t) = \mathbf{c}^T\mathbf{x}(t), \tag{2.19}$$

where $\mathbf{E}, \mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b}, \mathbf{c} \in \mathbb{R}^n$, input and output $u(t), y(t) \in \mathbb{R}$, and state vector $\mathbf{x}(t) \in \mathbb{R}^n$, and n is very large, by a much lower dimensional dynamical system

$$\mathbf{E}_r\dot{\mathbf{x}}_r(t) + \mathbf{A}_r\mathbf{x}_r(t) = \mathbf{b}_r u(t), \quad y_r(t) = \mathbf{c}_r^T\mathbf{x}_r(t), \tag{2.20}$$

where $\mathbf{E}_r, \mathbf{A}_r \in \mathbb{R}^{r\times r}$, $\mathbf{b}_r, \mathbf{c}_r \in \mathbb{R}^r$, and $r \ll n$, such that $y_r(t) \approx y(t)$ in an appropriate norm for a wide range of input selections $u(t)$. Here, for brevity, we consider single-input/single-output dynamical systems, i.e., $u(t)$ and $y(t)$ are scalar valued functions. Discussion similarly extends to the multi-input/multi-output case. Dynamical systems with large state-space dimension n appear in many applications, ranging from nonlinear parameter inversion to optimal control to circuit design. The repeated simulation of these large systems may be infeasible, but model reduction allows us to perform sufficiently accurate simulations with a much smaller system.

The reduced model quantities in (2.20) are obtained by construction of the matrices $\mathbf{V}_r, \mathbf{W}_r \in \mathbb{R}^{n\times r}$ (the model reduction bases) and a Petrov-Galerkin projection

$$\mathbf{A}_r = \mathbf{W}_r^T\mathbf{A}\mathbf{V}_r, \quad \mathbf{E}_r = \mathbf{W}_r^T\mathbf{E}\mathbf{V}_r, \quad \mathbf{b}_r = \mathbf{W}_r^T\mathbf{b}, \quad \mathbf{c}_r = \mathbf{V}_r^T\mathbf{c}. \tag{2.21}$$

Model reduction approaches differ in their choices of $\mathbf{V}_r$ and $\mathbf{W}_r$ [6, 8, 16, 25, 26, 82, 111]. In interpolatory model reduction, $\mathbf{V}_r$ and $\mathbf{W}_r$ are constructed so that the reduced model transfer function $H_r(s) = \mathbf{c}_r^T(s\mathbf{E}_r - \mathbf{A}_r)^{-1}\mathbf{b}_r$ is a rational interpolant of the full model transfer function $H(s) = \mathbf{c}^T(s\mathbf{E} - \mathbf{A})^{-1}\mathbf{b}$. In this case, we focus on the case that $H_r(s)$ is a Hermite interpolant to $H(s)$, as this is required for optimality [8, 81]. Therefore, the goal is to construct $\mathbf{V}_r$ and $\mathbf{W}_r$ such that $H_r(s_j) = H(s_j)$ and $H_r'(s_j) = H'(s_j)$ for some given set of points $s_1, \ldots, s_r$. Constructing $\mathbf{V}_r$ and $\mathbf{W}_r$ as

$$\mathbf{V}_r = [(s_1\mathbf{E} - \mathbf{A})^{-1}\mathbf{b}, \ldots, (s_r\mathbf{E} - \mathbf{A})^{-1}\mathbf{b}], \tag{2.22}$$

$$\mathbf{W}_r = [(s_1\mathbf{E}^T - \mathbf{A}^T)^{-1}\mathbf{c}, \ldots, (s_r\mathbf{E}^T - \mathbf{A}^T)^{-1}\mathbf{c}], \tag{2.23}$$

achieves this; see [7, 8] for more details. IRKA [81] finds the optimal interpolation points by alternatingly computing (2.22)–(2.23) for a given set of interpolation points $\{s_j\}$ and computing a new set $\{s_j\}$ given $\mathbf{V}_r$ and $\mathbf{W}_r$; for details and some variants of IRKA, see [8, 17, 43, 81, 84, 148, 149]. Since IRKA may take many iterations to converge to the final set of interpolation points $\{s_j\}$, many shifted systems must be solved in computing (2.22) and (2.23). Each set of shifts for an IRKA iteration is called a *batch*.

The efficient solution of (2.22)–(2.23) is an important research topic. In [18], inexact solves within a Petrov-Galerkin framework are used. In [5], the recycling BiCG algorithm is proposed for model reduction, which is extended to recycling BiCGSTAB for parametric model reduction in [3]. Further discussion of recycling Krylov subspace methods for model reduction applications can be found in [70, 71].

We give results for one set of matrices, flow, from [138]. These matrices arise in a simulation of the heat exchange between a solid body and a fluid flow, and background on this benchmark problem can be found in [138]. Rather than using computational fluid dynamics, which is quite expensive, a flow region with a given velocity profile is used [106]. However, this requires a much larger number of elements, and model reduction is used to make the simulation efficient [106]. For more information see [98, 106, 115, 138]. The matrices $\mathbf{A}, \mathbf{E} \in \mathbb{R}^{n\times n}$ are sparse with $n = 9\ 669$. Although $\mathbf{A}$ is not symmetric, it turns out that the shifts remain real for the three steps of IRKA used here. We use three batches of six shifts, which are real and range from $O(1)$ to $O(10^4)$. The shifts for the flow matrices are provided in Table 2.10.

We use GMRES(200) for this application and set the maximum number of iterations to 5000. Fill-in for ILUTP is 56 and the drop tolerance is 10^{-3}. The pattern of $\mathbf{A}_0$ is used for the SAM updates. Results are given in Tables 2.11–2.16.

An interesting case arises here. The number of iterations for the first preconditioned system, $\mathbf{A}_0\mathbf{P}_0$, is very high. For this first shift, the matrix is ill-conditioned and not diagonally dominant, and this leads to a poor ILUTP preconditioner. It makes no sense to reuse or recycle this preconditioner. As an alternative, we compute a new preconditioner, $\mathbf{P}_1$, for the second system. As this preconditioned system results in fast convergence, we recycle $\mathbf{P}_1$ with SAMs for each subsequent system (see Table 2.14) and at select systems (see Table 2.16). This leads to much better iteration counts and lower run times. When computing

SAM updates for select shifts, as shown in Table 2.16, for the other shifts we reuse $\mathbf{P}_1$, *not* the SAM updated preconditioner from a previous step. As IRKA converges, the shifts from one batch to the next do not change very much. If the initial preconditioner works well for certain shifts in one batch, we can reuse it for the corresponding shifts in subsequent batches (see Table 2.15). For comparison we also provide results with $\mathbf{P}_1$ reused for all subsequent systems, leading to sightly longer run times than using the SAMs (see Table 2.15).

As poor clustering of eigenvalues tends to lead to slow GMRES convergence, we compare the spectra of the preconditioned systems for two shifts. Figure 2.3 shows the spectrum of the initial preconditioned system, while Figures 2.4 and 2.5 show the spectra of preconditioned systems 1/5 and 1/6, respectively, when recomputing, recycling, and reusing the preconditioner computed for system 1/2. The figures show that the SAMs improve the eigenvalue clustering substantially, in particular along the real axis. At these shifts, SAMs result in much lower number of iterations compared with reusing the preconditioner (see Tables 2.14 and 2.15).

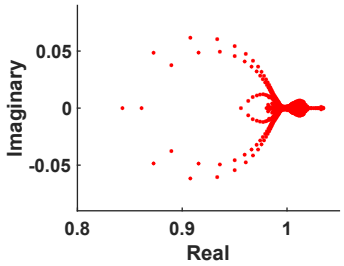
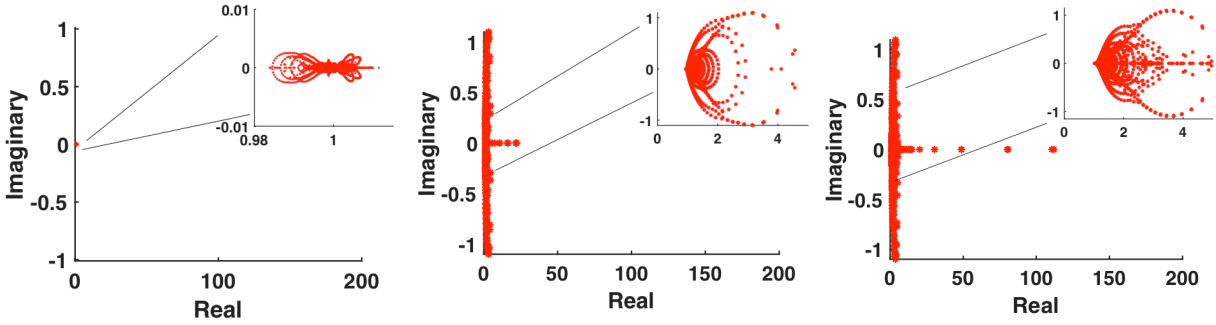


Figure 2.3: Eigenvalues of preconditioned system 1/2, $\mathbf{A}_1\mathbf{P}_1$, with the preconditioner recomputed.



(a) (b) (c)

Figure 2.4: Eigenvalues of preconditioned system 1/5 with (a) $\mathbf{P}_4$ recomputed, (b) $\mathbf{P}_4 = \mathbf{N}_4\mathbf{P}_1$ (recycled), and (c) $\mathbf{P}_1$ reused. *Note the different scale along the real axis compared with Figure 2.3 and 2.5.*

Shift	Batch 1	Batch 2	Batch 3
1	1.4091	1.4115	1.4116
2	28.123	30.121	30.146
3	150.70	163.43	163.58
4	669.26	691.84	692.12
5	3536.7	3565.9	3566.2
6	17329	17353	17353

Table 2.10: Shifts for the flow Matrices.
For subsequent tables, B/S = Batch/Shift Number.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0.70	0.02	11
1/4	0.67	0.02	10
1/5	0.62	0.02	7
1/6	0.55	0.02	7
2/1	0.71	0.60	455
2/2	0.70	0.03	13
2/3	0.70	0.03	11
2/4	0.70	0.03	10
2/5	0.62	0.02	7
2/6	0.55	0.02	7
3/1	0.71	5.66	4231
3/2	0.69	0.03	13
3/3	0.73	0.02	11
3/4	0.68	0.02	10
3/5	0.62	0.02	7
3/6	0.56	0.02	7
Total	21.28		6771

Table 2.11: Run times and iterations for the flow matrices with ILUTP recomputed for each shift.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0	0.02	18
1/3	0	0.03	26
1/4	0	0.59	206
1/5	0	0.59	211
1/6	0	0.62	232
2/1	0	0.02	15
2/2	0	0.02	18
2/3	0	0.05	41
2/4	0	0.60	207
2/5	0	0.62	210
2/6	0	0.78	231
3/1	0	4.00	3007
3/2	0	0.03	18
3/3	0	0.05	40
3/4	0	0.61	206
3/5	0	0.63	209
3/6	0	0.71	227
Total	13.41		7063

Table 2.12: Run times and iterations for the flow matrices with ILUTP computed for the first shift and reused for each remaining shift.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.23	0.03	18
1/3	0.20	0.04	24
1/4	0.20	0.51	189
1/5	0.19	0.19	99
1/6	0.19	0.14	83
2/1	0.19	0.37	275
2/2	0.20	0.03	18
2/3	0.19	0.03	24
2/4	0.19	0.25	124
2/5	0.19	0.62	207
2/6	0.20	0.36	155
3/1	0.19	6.85	5001
3/2	0.19	0.03	18
3/3	0.19	0.03	24
3/4	0.19	0.25	124
3/5	0.19	0.19	99
3/6	0.19	0.57	196
Total	17.23		8619

Table 2.13: Run times and iterations for the flow matrices with ILUTP computed for the first system and SAM updates computed for all other shifts.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0.20	0.03	19
1/4	0.20	0.05	30
1/5	0.20	0.10	53
1/6	0.20	0.18	79
2/1	0.20	0.72	488
2/2	0.20	0.02	12
2/3	0.20	0.03	20
2/4	0.21	0.05	30
2/5	0.21	0.10	53
2/6	0.19	0.18	79
3/1	0.20	0.56	382
3/2	0.21	0.02	12
3/3	0.20	0.03	20
3/4	0.20	0.05	30
3/5	0.20	0.10	53
3/6	0.20	0.18	79
Total	9.83		3393

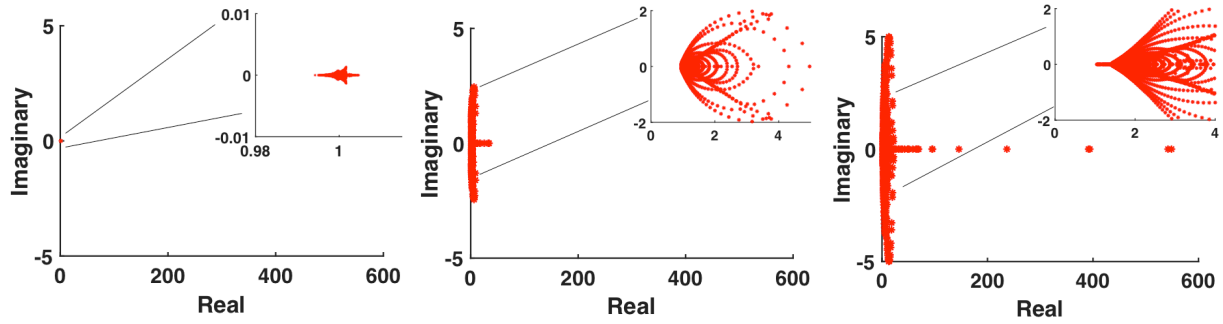
Table 2.14: Run times and iterations for the flow matrices with ILUTP computed for the first two systems and SAM updates computed for all other shifts.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0	0.03	21
1/4	0	0.11	60
1/5	0	0.84	202
1/6	0	0.89	215
2/1	0	0.12	86
2/2	0	0.02	12
2/3	0	0.03	21
2/4	0	0.06	37
2/5	0	0.97	202
2/6	0	0.89	215
3/1	0	0.69	486
3/2	0	0.02	12
3/3	0	0.03	22
3/4	0	0.06	36
3/5	0	0.96	202
3/6	0	1.01	215
Total	10.90		3998

Table 2.15: Run times and iterations for the flow matrices with ILUTP computed for the first two systems, and $\mathbf{P}_1$ reused for all remaining shifts.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0	0.03	21
1/4	0	0.11	60
1/5	0.20	0.10	53
1/6	0.20	0.19	79
2/1	0	0.12	86
2/2	0	0.02	12
2/3	0	0.03	21
2/4	0	0.06	37
2/5	0.21	0.10	53
2/6	0.20	0.18	79
3/1	0	0.68	486
3/2	0	0.02	12
3/3	0	0.03	22
3/4	0	0.06	36
3/5	0.21	0.10	53
3/6	0.19	0.18	79
Total	7.44		3143

Table 2.16: Timings for flow matrices with ILUTP computed for the first two systems and SAM updates computed for selected shifts.



(a) (b) (c)

Figure 2.5: Eigenvalues of preconditioned system 1/6 with (a) $\mathbf{P}_5$ recomputed, (b) $\mathbf{P}_5 = \mathbf{N}_5\mathbf{P}_1$ (recycled), and (c) $\mathbf{P}_1$ reused. *Note the different scale along the real axis compared with Figure 2.3 and 2.4.*

2.5.3 Indefinite Matrices

In the previous tests, computing a new ILUTP for each system gives the lowest number of GMRES iterations, but it is too expensive in time. Here, we consider linear systems where the computation of the ILUTP preconditioner for the system of interest may fail or be unstable, resulting in poor preconditioners. This is the case, for example, for indefinite systems [120, Chapter 10]. We consider discretized 2D Helmholtz equations $-\Delta u - k^2 u = f$, which arise in wave propagation problems [67, 68] and in flow control for unstable systems, giving eigenvalues in both the right- and left-half planes [41]. In such cases, we can select from the set (or more generally) a reference matrix for which the ILUTP algorithm computes an effective preconditioner and recycle this preconditioner using SAMs to (approximately) map matrices for which ILUTP may fail to the reference matrix.

Using a modified Helmholtz equation to compute a preconditioner has also been applied for other preconditioning approaches [68]. Previous work has successfully used operator-based preconditioners to achieve fast convergence for Krylov methods. The shifted Laplace preconditioner [68] is used along with multilevel Krylov methods in [67, 69, 124], while a sweeping preconditioner is constructed layer-by-layer in [66]. Preconditioning by replacing a subset of the Sommerfeld-type boundary conditions of the Helmholtz equation with Dirichlet or Neumann boundary conditions is examined in [64, 65]. We use this test problem just to demonstrate another possible use of SAMs; we do not consider whether this approach is competitive with the methods above.

We compute the matrix $\mathbf{K}_0$ and right hand side $\mathbf{b}$ by discretizing the 2D Laplacian on the unit square with Dirichlet boundary conditions, $u(x, 0) = 1$, $u(0, y) = 1$, $u(x, 1) = 0$, and $u(1, y) = 0$, using a vertex-centered finite volume discretization. $\mathbf{K}_0$ is symmetric, positive definite and has size 100×100. We compute an ILUTP preconditioner, $\mathbf{P}_0$, for $\mathbf{K}_0$. Next,

we solve the systems

$$\mathbf{K}_i = \mathbf{K}_0 - s_i\mathbf{I}, \tag{2.24}$$

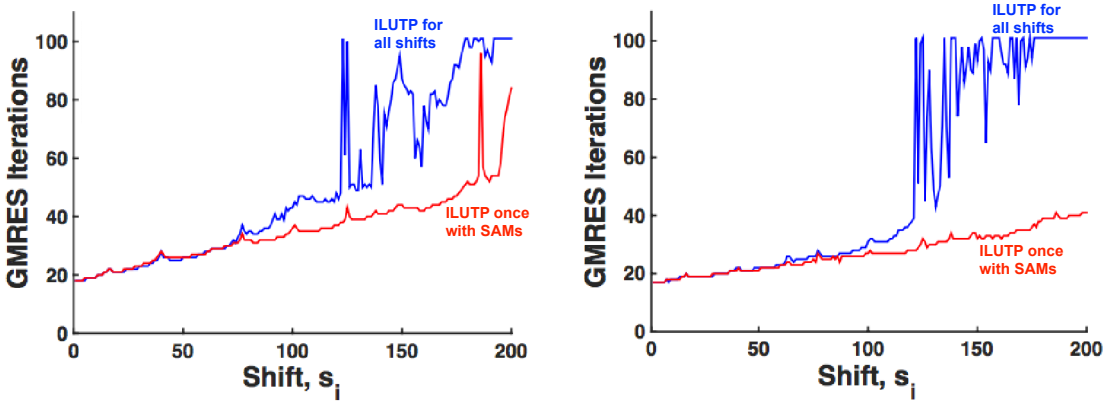
where $\mathbf{I}$ is the identity matrix, and $s_i = i\Delta s$ with $\Delta s = 0.01$, for $i = 1, 2, \ldots, 200$. We solve these systems with preconditioned full GMRES, comparing a new ILUTP preconditioner for every shift with recycling $\mathbf{P}_0$ using a SAM with the pattern of $\mathbf{K}_0$ for each system. The relative convergence tolerance is 10^{-10} and we use a zero initial guess for each system. For our ILUTP implementation, fill-in is 20 and the drop tolerance is 10^{-3}. For MATLAB®'s `ilu` with type 'ilutp', the drop tolerance is 10^{-3}.

The results are presented in Figure 2.6(a). Figure 2.6(b) shows the eigenvalues for selected $\mathbf{K}_i$. While $\mathbf{K}_i$ becomes indefinite at the twentieth shift, ILUTP produces good preconditioners until about shift s_{125}. After this shift, both our ILUTP implementation and MATLAB®'s `ilu` with type 'ilutp' fail to produce a good preconditioner, and the number of GMRES iterations increases substantially (or GMRES fails to converge). However, using SAM updates to recycle $\mathbf{P}_0$ keeps the GMRES iterations low for almost all shifts. For these small problems, we are not concerned with runtime and just demonstrate the superior convergence behavior obtained with the recycled preconditioners using SAMs compared with ILUTP.

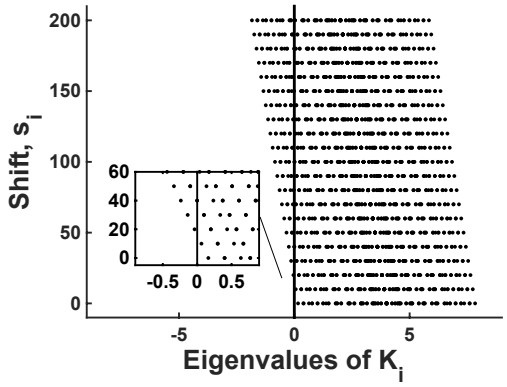
2.6 Conclusions and Future Work

In applications that involve many linear systems, recycling a preconditioner can be advantageous, especially when computing a preconditioner from scratch is expensive. We develop a flexible update to arbitrary preconditioners that we call the Sparse Approximate Map, or SAM, update, which can be computed for any set of closely related matrices. The SAM is motivated by the Sparse Approximate Inverse; however, rather than approximately inverting a matrix, a SAM update approximately maps a matrix to a nearby matrix for which a good preconditioner is available. Using SAMs, the cost of computing a very good preconditioner can be amortized over many systems in a sequence, since computing SAMs is cheap. Further, a SAM is independent of preconditioner type and quality. The sparsity patterns for SAMs can be based on powers of $\mathbf{A}_0$, mesh-based patterns, or any other salient feature of a specific problem.

In future work, we plan to consider incremental SAM updates as in (2.17), applying maps from the left as in (2.18), and maps that allow CG and MINRES to be used. Another important future topic are approaches that update only a few columns or rows of the map to match localized changes in the matrix $\mathbf{A}_k$. More generally, we plan to consider maps that are tuned to various structural changes in a sequence of matrices that are known a priori. We also plan to develop more indicators for computing a new map. Finally, we plan to consider other types of maps, including different, potentially adaptive, choices in sparsity patterns.



(a) Number of GMRES iterations to converge for the discretized Helmholtz equation, comparing a new ILUTP for each $\mathbf{K}_i$ (blue line) with recycling $\mathbf{P}_0$ using SAM updates for the $\mathbf{K}_i$ (red line). The results on the left are based on our ILUTP implementation. Those on the right are based on the MATLAB® ilu preconditioner with type 'ilutp'.



(b) Eigenvalues of every tenth matrix, $\mathbf{K}_i$. At the twentieth shift, the matrices become indefinite.

Figure 2.6: GMRES convergence and selected eigenvalues for a discretized Helmholtz problem.

2.7 Acknowledgements

We thank Xiaozhe Hu for sharing his MATLAB® implementation of the AMG preconditioner with us and for his advice how to incorporate it into our solvers. We also thank Tania Bakhos, Arvind Saibaba, and Peter Kitanidis for providing us with matrices from a THT application [14]. The application is not represented in the final draft of this chapter, but it was very helpful in the development of our ideas. We further thank the anonymous reviewers for their suggestions, which led to several valuable improvements in the publication of this chapter.

Chapter 3

Subspace Recycling for a Sequence of Eigenproblems

In this chapter, we introduce a warm-started Krylov-Schur algorithm for solving a sequence of eigenvalue problems. When computing invariant subspaces for a sequence of matrices, we can recycle the approximate invariant subspace computed for one matrix to warm-start the eigensolver for another matrix in the sequence, rather than starting the eigensolver from scratch with a single vector. This recycling technique makes computing invariant subspaces of very large-scale matrices tractable by reducing the total number of matrix-vector products required in the eigensolver, while still producing invariant subspaces that are as accurate as for the standard Krylov-Schur method.

An important application we consider in this context arises in the numerical simulation of brake squeal [76], which we introduce in Section 3.4. An additional, interesting complication with this problem is that the matrix-vector products in the eigensolver require an implicit linear solve. If the matrix is very large, and fill-in is substantial, then solving these matrix-vector products using direct methods is intractable. Krylov subspace methods are an efficient alternative, and Krylov subspace recycling methods can improve convergence.

In the numerical experiments section, we demonstrate good results for two sequences of eigenproblems arising in this brake squeal application.

3.1 Introduction

We propose a technique to warm-start the Krylov-Schur method [129, 131] using a subspace, rather than a single vector, when solving a sequence of standard eigenvalue problems (SEP) of the form

$$\mathcal{A}^{(k)}\mathbf{y} = \mu\mathbf{y}, \tag{3.1}$$

for $k = 1, 2, \ldots, N$. Our approach consists of recycling a previously computed approximate invariant subspace for $\mathcal{A}^{(k)}$ to warm-start Krylov-Schur for some $\mathcal{A}^{(k+\ell)}$, $\ell \geq 1$. We note that while we do not require that the matrices in the sequence all be near to one another, we will assume that $\mathcal{A}^{(k)}$ is close to $\mathcal{A}^{(k+\ell)}$. In other words, $\mathcal{A}^{(k)}$ may be nearby the next (or previous) matrix, or a group of matrices may be close to one another.

Our eigensolver extends the idea introduced in [40] for improving an approximate invariant subspace corresponding to the smallest eigenvalues of a matrix when recycling Krylov subspaces for evolving structures. In [40], a few cycles of Krylov-Schur are run to improve the recycle space for recycling MINRES (rMINRES) [145]. We propose a complete eigensolver that allows for a starting subspace, rather than a starting vector.

Since we start the eigensolver with a space that is not an exact Krylov space for the given matrix, we make several modifications to our eigensolver to account for this. (1) Given the start space, we first compute the Krylov decomposition with minimum residual norm based on the analysis in [132]. (2) After we expand this approximate Krylov space, we must account for the residual matrix when computing the Rayleigh quotient. (3) The standard convergence test for Krylov-Schur can no longer be used, and so we propose a convergence test based on the residual of the eigenvalue problem. Details of these, as well as the derivation of the algorithm, are given in Section 3.3.

This warm-start Krylov-Schur algorithm makes the solution of standard eigenvalue problems (SEP) feasible for a sequence of very large-scale problems. An additional complication emerges when solving the sequence of eigenvalue problems that arise in the numerical simulation of brake squeal [76]. Here, a sequence of N generalized eigenvalue problems (GEP) of the form

$$\mathbf{A}^{(k)}\mathbf{y} = \mu\mathbf{B}^{(k)}\mathbf{y}, \tag{3.2}$$

are solved as a SEPs with the coefficient matrix of the form

$$\mathcal{A}^{(k)} = (\mathbf{B}^{(k)})^{-1}\mathbf{A}^{(k)}. \tag{3.3}$$

In this case, because the matrix-vector product in (3.3) requires a linear solve, an iterative linear solver, such as GMRES [122], can be used to efficiently compute these matrix-vector products. We can improve convergence of the iterative linear solver by using a Krylov subspace recycling method, such as GCRO-DR [110]. While recycling Krylov subspaces in the iterative solution of linear systems is not a new idea, we show that when the matrices in the sequence of eigenvalue problems take the form (3.3), using GCRO-DR reduces the total number of iterations compared with GMRES. We also refer the reader to a survey on subspace recycling methods in [128]. We note that the use of Krylov subspace recycling in our proposed eigensolver is specific to applications where the GEPs are rewritten as SEPs, and is not required as part of our warm-start Krylov-Schur eigensolver.

This chapter is organized as follows. In Section 3.2, we give relevant background information on the Krylov-Schur algorithm, including the standard convergence test for this eigensolver.

In Section 3.3, we describe our warm-start Krylov-Schur method for solving a sequence of eigenvalue problems. In Section 3.4, we give background information on a sequence of eigenvalue problems arising in the numerical simulation of brake squeal [76], and we provide results for an academic and industrial model from this application in Section 3.5. Finally, in Section 3.6 we give conclusions and discuss future work.

Throughout this chapter, we refer to the subspaces used to warm-start the eigensolver as the *warm-start spaces*, and those recycled for the linear solver as the *recycle spaces*. We use the terms *Krylov-Schur* and *warm-start Krylov-Schur* to refer to the standard and warm-start Krylov-Schur methods, respectively. We use subscripts to define the size of a matrix, and superscripts to define the matrix in the sequence. For instance, $\mathbf{A}_{1:j}$ indicates the leading j columns of a matrix $\mathbf{A} \in \mathbb{C}^{n\times p}$ for $j \leq p$, whereas $\mathbf{A}^{(k)}$ denotes the k^{th} matrix in a sequence. Where necessary, we will also indicate when we are taking a subset of the rows and columns by using a range of indices, e.g., for $\mathbf{A}_{1:h,1:j}$ indicates the leading $h \times j$ submatrix of $\mathbf{A} \in \mathbb{C}^{n\times n}$. We use the notation $\mathbf{a}_j$ and $\mathbf{a}_k^*$ to indicate respectively the j^{th} column and k^{th} row of a matrix $\mathbf{A} \in \mathbb{C}^{n\times p}$, $j \leq p$ and $k \leq n$.

3.2 Background

We provide relevant background on the Krylov-Schur method for computing the approximate eigenpairs of a matrix, $\mathcal{A}$ [129, 130]. As is done in [129, 130], and to keep the descriptions of the background information and our algorithm simple, we assume our matrices are complex. This allows us to avoid the complication of complex eigenvalues in the real Schur form.

The main advantages of using this eigensolver over another commonly used eigensolver, Implicitly Restarted Arnoldi (IRA) [100], are that Krylov-Schur does not restrict the type of Krylov decomposition required, and is more robust than IRA [129]. Krylov-Schur also allows us to more easily deflate converged Ritz values than IRA [100].

The Krylov-Schur algorithm begins with a *Krylov-Schur decomposition* of the form

$$\mathcal{A}\mathbf{U}_{1:k} = \mathbf{U}_{1:k+1}\mathbf{S}_{1:k+1,1:k} = \mathbf{U}_{1:k}\mathbf{S}_{1:k,1:k} + \mathbf{u}_{k+1}\mathbf{b}_{k+1}^*, \tag{3.4}$$

where $\mathbf{S}_{1:k,1:k}$ is upper triangular and $\mathbf{b}_{k+1}^*$ is (generally) a dense row. $\mathbf{U}_{1:k+1}$ has orthonormal columns, and so we also refer to (3.4) as an *orthonormal Krylov decomposition* [130]. In (3.4), $\mathbf{S}_{1:k,1:k}$ is the *Rayleigh quotient of the decomposition* [129]; since $\mathbf{U}_{1:k+1}^* = (\mathbf{U}_{1:k}\ \mathbf{u}_{k+1})^*$, we can write

$$\mathbf{S}_{1:k+1,1:k} = \begin{pmatrix} \mathbf{S}_{1:k,1:k} \\ \mathbf{b}_{k+1}^* \end{pmatrix} = \begin{pmatrix} \mathbf{U}_{1:k}^*\mathcal{A}\mathbf{U}_{1:k} \\ \mathbf{u}_{k+1}^*\mathcal{A}\mathbf{U}_{1:k} \end{pmatrix}. \tag{3.5}$$

The Krylov-Schur algorithm begins each cycle by expanding the decomposition (3.4) using $m-k$ steps of the Arnoldi algorithm [11] to give

$$\mathcal{A}\mathbf{U}_{1:m} = \mathbf{U}_{1:m+1}\mathbf{S}_{1:m+1,1:m} = \mathbf{U}_{1:m}\mathbf{S}_{1:m,1:m} + \mathbf{u}_{m+1}\mathbf{b}_{m+1}^*, \tag{3.6}$$

where m and k are chosen parameters such that m is the maximum dimension of the expansion space and k is the dimension of the truncation space. We summarize this expansion in Algorithm 3, letting $\mathbf{U} = \mathbf{U}_{1:k+1}$ and $\mathbf{S} = \mathbf{S}_{1:k+1,1:k}$ as in (3.4). Note that in practice, we normally reorthogonalize to ensure accuracy [129].

Algorithm 3 Arnoldi Algorithm

Given: $\mathbf{U}$, $\mathbf{S}$
for $j = k+1 : m$ **do**
  $\mathbf{t} = \mathcal{A}\mathbf{u}_j$
  $\mathbf{w} = \mathbf{U}_{1:j}^{*}\mathbf{t}$
  $\mathbf{t} = \mathbf{t} - \mathbf{U}_{1:j}\mathbf{w}$
  $\mathbf{S}_{1:j,j} = \mathbf{w}$
  $\mathbf{S}_{j+1,j} = \|\mathbf{t}\|_2$
  $\mathbf{u}_{j+1} = \mathbf{t} \;/\; \mathbf{S}_{j+1,j}$
end for

By computing the Schur decomposition of $\mathbf{S}_{1:m,1:m}$, we can rearrange the eigenvalues of $\mathbf{S}_{1:m,1:m}$ in the desired order (e.g., in ascending or descending order). Specifically, we compute an orthogonal matrix $\mathbf{Q}$ such that

$$\widehat{\mathbf{S}}_{1:m,1:m} = \mathbf{Q}^{*}\mathbf{S}_{1:m,1:m}\mathbf{Q} \tag{3.7}$$

is again upper triangular, but with those eigenvalues we wish to keep in $\mathbf{S}_{11}$, the upper left block of $\widehat{\mathbf{S}}_{1:m,1:m}$, and those we wish to discard in $\mathbf{S}_{22}$, the lower right block of $\widehat{\mathbf{S}}_{1:m,1:m}$. We transform

$$\mathbf{U}_{1:m}\mathbf{Q} = \left[\mathbf{U}^{(keep)} \mid \mathbf{U}^{(trunc)}\right] \tag{3.8}$$

and the transformed decomposition can be written as

$$\mathcal{A}\left[\mathbf{U}^{(keep)} \mid \mathbf{U}^{(trunc)}\right] = \left[\mathbf{U}^{(keep)} \mid \mathbf{U}^{(trunc)}\right] \begin{pmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{0} & \mathbf{S}_{22} \end{pmatrix}$$

$$+\mathbf{u}_{m+1}\left[\left(\mathbf{b}_{m+1}^{(keep)}\right)^{*} \;\middle|\; \left(\mathbf{b}_{m+1}^{(trunc)}\right)^{*}\right]. \tag{3.9}$$

Here, the superscripts $(keep)$ and $(trunc)$ refer to those eigenvalues of $\widehat{\mathbf{S}}_{1:m,1:m}$ and corresponding vectors of $\mathbf{U}_{1:m}\mathbf{Q}$ that we wish to maintain and discard, respectively, when truncating (3.9). Following this truncation, we have

$$\mathcal{A}\mathbf{U}^{(keep)} = \mathbf{U}^{(keep)}\mathbf{S}_{11} + \mathbf{u}_{m+1}\left(\mathbf{b}_{m+1}^{(keep)}\right)^{*}, \tag{3.10}$$

where (3.10) is still a Krylov-Schur decomposition [131]. If necessary, the algorithm then repeats the expansion (and subsequent truncation) using (3.10) until the desired (approximate) eigenpairs of the matrix $\mathcal{A}$ are computed. Note that in general, we are interested in computing only a very small subset of the total number of eigenpairs of $\mathcal{A}$.

We check the accuracy of the approximate eigenpairs of $\mathcal{A}$ computed by the Krylov-Schur method by directly considering the residual of the approximate eigenpair $(\theta, \mathbf{z})$,

$$\mathbf{r} = \mathcal{A}\mathbf{z} - \theta\mathbf{z}. \tag{3.11}$$

In particular, we know there exists $\mathbf{E} \in \mathbb{C}^{n\times n}$, with $\|\mathbf{E}\|_F = \|r\|_2$, such that $(\theta, \mathbf{z})$ is an exact eigenpair of $\mathcal{A}+\mathbf{E}$ [130, Theorem 2.8, Chapter 4]. If $\|\mathbf{r}\|_2 \leq tol\|\mathcal{A}\|_F$, we consider the eigenpair $(\theta, \mathbf{z})$ converged for our system $\mathcal{A}$ [130, Chapter 5.2]. To compute $\|\mathbf{r}\|_2$, we can take advantage of the form of our decomposition and definition of a Ritz pair.

Given (3.10), if $(\theta, \mathbf{y})$ is an eigenpair of $\mathbf{S}_{11}$, we refer to θ and $\mathbf{U}^{(keep)}\mathbf{y}$ respectively as a *Ritz value* and *Ritz vector* of $\mathcal{A}$, and we call $(\theta, \mathbf{U}^{(keep)}\mathbf{y})$ a *Ritz pair* of $\mathcal{A}$. If the search space is sufficiently good, Ritz pairs are generally good approximations of the eigenvalues of a matrix [121, 130]. If $(\theta, \mathbf{z}) = (\theta, \mathbf{U}^{(keep)}\mathbf{y})$ is a Ritz pair of $\mathcal{A}$, then

$$\|\mathbf{r}\|_2^2 = \|\mathcal{A}\mathbf{z} - \theta\mathbf{z}\|_2^2 = \|\mathbf{S}_{11}\mathbf{y} - \theta\mathbf{y}\|_2^2 + \left|\left(\mathbf{b}_{m+1}^{(keep)}\right)^* \mathbf{y}\right|^2 = \left|\left(\mathbf{b}_{m+1}^{(keep)}\right)^* \mathbf{y}\right|^2. \tag{3.12}$$

This gives a convenient test for convergence of the Ritz pair. Since we compute the approximate exterior eigenvalues (or those largest in magnitude), we can use the approximation

$$\left|\left(\mathbf{b}_{m+1}^{(keep)}\right)^* \mathbf{y}\right| < tol \cdot |\theta|, \tag{3.13}$$

where *tol* is some user-defined convergence tolerance, to test for convergence of the approximate eigenpair [130].

Once we determine a Ritz pair has converged, we follow [129, 130] to deflate the corresponding column of $\mathbf{U}^{(keep)}$ by placing it at the beginning of the matrix. We claim an exact Krylov decomposition has been deflated if we can represent it as [129]

$$\mathcal{A}\left[\mathbf{U}^\ell \mid \mathbf{U}^a\right] = \left[\mathbf{U}^\ell \mid \mathbf{U}^a\right] \begin{pmatrix} \mathbf{S}^\ell & \mathbf{S}^{\ell a} \\ \mathbf{0} & \mathbf{S}^a \end{pmatrix} + \mathbf{u}_{m+1}\left[\mathbf{0} \mid \mathbf{b}_2^*\right].$$

We use the superscript ℓ to designate those columns of $\mathbf{U}^{(keep)}$ corresponding to the Ritz values that have been deflated and say the corresponding Schur vectors have been *locked*. The superscript a designates those Ritz values and corresponding vectors that are still active. We say that the matrix $\mathbf{S}_{11}$ is *uncoupled* from the Rayleigh quotient and that $\mathbf{U}^\ell$ spans an invariant subspace of $\mathcal{A}$ [129, 130]. By locking the columns of $\mathbf{U}^\ell$ in the leading portion of $\left[\mathbf{U}^\ell \mid \mathbf{U}^a\right]$, we reduce computational cost during the truncation phase of Krylov-Schur [129]. We still orthogonalize all vectors generated in the Arnoldi expansion against these deflated vectors [100, 129].

Note that Krylov-Schur in general does not compute an exact invariant subspace, but we follow [130, Theorem 2.7, Chapter 5] to show that when (3.11) is sufficiently small, we can deflate those corresponding vectors in $\mathbf{U}^{(keep)}$. Let

$$\mathcal{A}\mathbf{U} = \mathbf{U}\mathbf{S} + \mathbf{u}\mathbf{b}^*$$

and $(\theta, \mathbf{U}\mathbf{y})$ be an approximate eigenpair of $\mathcal{A}$. Define the unitary matrix $\mathbf{Y} = (\mathbf{y}\ \mathbf{Y}_\perp)$, and write the change of basis

$$\mathcal{A}(\mathbf{U}\mathbf{Y}) = (\mathbf{U}\mathbf{Y})\mathbf{Y}^*\mathbf{S}\mathbf{Y} + \mathbf{u}\mathbf{b}^*\mathbf{Y}$$

$$\iff \mathcal{A}(\mathbf{u}_1 \mid \mathbf{U}_2) = (\mathbf{u}_1 \mid \mathbf{U}_2 \mid \mathbf{u}) \begin{pmatrix} \widetilde{s}_{11} & \widetilde{\mathbf{S}}_{12} \\ \widetilde{\mathbf{s}}_{21} & \widetilde{\mathbf{S}}_{22} \\ \widetilde{b}_1 & \mathbf{b}_2^* \end{pmatrix},$$

where $\mathbf{U}\mathbf{Y} = (\mathbf{u}_1 \mid \mathbf{U}_2)$, $\mathbf{Y}^*\mathbf{S}\mathbf{Y} = \begin{pmatrix} \widetilde{s}_{11} & \widetilde{\mathbf{S}}_{12} \\ \widetilde{\mathbf{s}}_{21} & \widetilde{\mathbf{S}}_{22} \end{pmatrix}$, and $\mathbf{b}^*\mathbf{Y} = (\widetilde{b}_1\ \widetilde{\mathbf{b}}_2^*)$. Then, for $(\theta, \mathbf{z}) = (\theta, \mathbf{U}\mathbf{y})$

$$\|\mathcal{A}\mathbf{z} - \theta\mathbf{z}\|_F^2 = \|\widetilde{\mathbf{s}}_{21}\|_2^2 + \left|\widetilde{b}_1\right|^2 + \left|\widetilde{s}_{11} - \theta\right|^2,$$

and it follows that

$$\left\| \begin{pmatrix} \widetilde{\mathbf{s}}_{21} \\ \widetilde{b}_1 \end{pmatrix} \right\|_2 \leq \|\mathbf{r}\|_2.$$

If $\|\mathbf{r}\|_2$ is small enough, $\theta = \widetilde{s}_{11}$, and we can safely set $\widetilde{\mathbf{s}}_{21}$ and $\widetilde{b}_1$ to zero [130].

When solving a sequence of eigenproblems, if the matrices change in such a way that the invariant subspace is not drastically changed, we can recycle the previously obtained approximate invariant subspace when solving the next, closely related eigenproblem in the sequence. Next, we discuss the warm-start Krylov-Schur algorithm, which uses a subspace to start the eigensolver, rather than a single vector. The choice of subspace is flexible, but for a sequence of eigenvalue problems as defined in (3.1), a convenient choice would be to use an previously computed approximate invariant subspace from another, nearby eigenproblem.

When warm-starting, several complications arise. Since we do not have an exact Krylov space for the current matrix, we compute the approximate Krylov-Schur decomposition with minimum norm residual [132]. In doing so, we introduce a residual matrix, $\mathbf{R}$, into the decomposition (3.4), and we must be careful to account for this when expanding and truncating the approximate decomposition, when deflating the decomposition, and when updating the Rayleigh quotient. We address these in Sections 3.3.1 and 3.3.2. Further, the convergence criteria (3.13) will no longer hold for the approximate decomposition, and (3.12) will now include the term $\|\mathbf{R}\mathbf{y}\|_2^2$ on the right-hand side. So, we propose an alternative test in Section 3.3.3.

3.3 Warm-Start Krylov-Schur for a Sequence of Eigenproblems

We introduce a modification of the Krylov-Schur method to start with a subspace, rather than a single vector, in order to improve convergence of the eigensolver. The idea was motivated by the work in [40], where Krylov-Schur is run for a few cycles in order to generate an improved approximate invariant subspace. This improved subspace is intended to provide a better recycle space when solving a linear system using rMINRES [40]. We extend this idea by proposing a complete eigensolver.

Restarting an eigensolver using a set of vectors, rather than a single vector, has been considered before. In [147], a thick-restarted Lanczos method is introduced that restarts using multiple Ritz vectors, rather than a single vector, when solving a symmetric eigenvalue problem. A thick-restarted Lanczos with polynomial filtering is discussed in [103]. This method is used to compute many eigenvalues of a single matrix, and in particular, those within an interval. A hybrid method combining thick-start Lanczos and iteratively refined Ritz vectors when computing extreme eigenvalues is given in [12]. In each of these methods, the exact Krylov space is maintained for the same matrix, but we lose this property when warm-starting the eigensolver using an approximate invariant subspace computed for a different matrix. We address how we accommodate for this in the following sections.

In our numerical experiments, we warm-start the eigensolver using the approximate invariant subspace computed for another, previous matrix in the sequence of eigenproblems we are solving, but the method itself does not restrict the choice of warm-start space.

3.3.1 Computing the Best Approximate Krylov Decomposition

Given the matrix $\mathbf{X}_{1:j} \in \mathbb{C}^{n\times j}$, our method warm-starts Krylov-Schur for the eigenvalue problem

$$\mathcal{A}\mathbf{y} = \mu\mathbf{y}.$$

Unless $\mathbf{X}_{1:j}$ is a Krylov space for $\mathcal{A}$, we can no longer rely on some of the convenient properties of the Krylov-Schur method discussed in Section 3.2. So, we first minimize the size of the initial residual by computing the approximate Krylov decomposition for $\mathcal{A} \in \mathbb{C}^{n\times n}$ with minimum norm residual following the analysis in [132].

There is flexibility in the choice of a warm-start space. For a sequence of eigenvalue problems, a convenient choice is to let $\mathbf{X}_{1:j}$ be a matrix whose columns span the approximate invariant subspace computed for another, close by, matrix in the sequence. In the following analysis, we let $\mathbf{X}_{1:j}$ be any $n \times j$ matrix spanning an arbitrary space.

We first compute the thin QR-decomposition of $\mathbf{X}_{1:j}$ as

$$\mathbf{X}_{1:j} = \widetilde{\mathbf{U}}_{1:j}\widetilde{\mathbf{S}}_{1:j,1:j}, \tag{3.14}$$

and let

$$\widetilde{\mathbf{H}}_{1:j,1:j} = \widetilde{\mathbf{U}}_{1:j}^{*}\mathcal{A}\widetilde{\mathbf{U}}_{1:j}, \tag{3.15}$$

and we define

$$\widetilde{\mathbf{R}}_{1:j} = \mathcal{A}\widetilde{\mathbf{U}}_{1:j} - \widetilde{\mathbf{U}}_{1:j}\widetilde{\mathbf{H}}_{1:j,1:j}. \tag{3.16}$$

We then take the thin singular value decomposition (SVD) of $\widetilde{\mathbf{R}}_{1:j}$

$$\widetilde{\mathbf{R}}_{1:j} = \mathbf{\Phi}_{1:j}\mathbf{\Omega}_{1:j,1:j}\mathbf{\Psi}_{1:j,1:j}^{*}, \tag{3.17}$$

and define

$$\mathbf{V}_{1:j} = [\boldsymbol{\psi}_2 \mid \boldsymbol{\psi}_3 \mid \ \ldots \ \mid \boldsymbol{\psi}_j \mid \boldsymbol{\psi}_1], \quad \mathbf{V}_{1:j-1} = [\boldsymbol{\psi}_2 \mid \boldsymbol{\psi}_3 \mid \ \ldots \ \mid \boldsymbol{\psi}_j].$$

Then we have the approximate Krylov decomposition of $\mathcal{A}$

$$\mathcal{A}\widetilde{\mathbf{U}}_{1:j}\mathbf{V}_{1:j-1} = \widetilde{\mathbf{U}}_{1:j}\mathbf{V}_{1:j}\mathbf{V}_{1:j}^{*}\widetilde{\mathbf{H}}_{1:j,1:j}\mathbf{V}_{1:j-1} + \widehat{\mathbf{R}}_{1:j-1}, \tag{3.18}$$

with

$$\widehat{\mathbf{R}}_{1:j-1} = \widetilde{\mathbf{R}}_{1:j}\mathbf{V}_{1:j-1} \tag{3.19}$$

the minimum norm residual.

For ease of notation, we write $\widehat{\mathbf{U}}_{1:j-1} = \widetilde{\mathbf{U}}_{1:j}\mathbf{V}_{1:j-1}$, $\widehat{\mathbf{u}}_j = \widetilde{\mathbf{U}}_{1:j}\mathbf{v}_j$, and $(\widehat{\mathbf{U}}_{1:j-1}\ \widehat{\mathbf{u}}_j) = \widetilde{\mathbf{U}}_{1:j}\mathbf{V}_{1:j}$, and rewrite (3.18) as

$$\mathcal{A}\widehat{\mathbf{U}}_{1:j-1} = (\widehat{\mathbf{U}}_{1:j-1}\ \widehat{\mathbf{u}}_j)\underline{\widehat{\mathbf{H}}}_{1:j,1:j-1} + \widehat{\mathbf{R}}_{1:j-1}, \tag{3.20}$$

where $\underline{\widehat{\mathbf{H}}}_{1:j,1:j-1} = (\mathbf{V}_{1:j})^{*}\widetilde{\mathbf{H}}_{1:j,1:j}\mathbf{V}_{1:j-1}$. Note that $\underline{\widehat{\mathbf{H}}}_{1:j,1:j-1}$ has the form

$$\underline{\widehat{\mathbf{H}}}_{1:j,1:j-1} = \begin{pmatrix} \widehat{\mathbf{H}}_{1:j-1,1:j-1} \\ \widehat{\mathbf{h}}^{*} \end{pmatrix}. \tag{3.21}$$

In Algorithm 4 we give the implementation for computing the minimum norm residual for an approximate Krylov decomposition derived in lines (3.14)-(3.18). Other choices are possible when preprocessing the warm-start space, including the straightforward choice to not preprocess the space at all and directly expand the space. This may be a reasonable choice if we know a priori the warm-start space results in an approximate Krylov decomposition that is good enough (though, not the best), for instance, if the eigenvalue problems from the sequence are correlated [38, 60].

Algorithm 4 Compute Approximate Krylov Decomposition with Min Norm Residual

Given: $\mathbf{A} \in \mathbb{C}^{n\times n}$, basis $\mathbf{W}_{1:p} = [\mathbf{w}_1 \mid \mathbf{w}_2 \mid \ \dots \ \mid \mathbf{w}_p]$
$[\widetilde{\mathbf{U}}, \widetilde{\mathbf{S}}] = \text{thinQR}(\mathbf{W})$
$\widetilde{\mathbf{W}} = \mathbf{A}\widetilde{\mathbf{U}}$
$\widetilde{\mathbf{H}} = \widetilde{\mathbf{U}}^*\widetilde{\mathbf{W}}$
$\widetilde{\mathbf{R}} = \widetilde{\mathbf{W}} - \mathbf{U}\widehat{\mathbf{H}}$
$[\mathbf{\Phi}, \mathbf{\Omega}, \mathbf{\Psi}] = \text{thinSVD}(\mathbf{R})$
$\mathbf{V}_{1:p} = [\boldsymbol{\psi}_2 \mid \boldsymbol{\psi}_3 \mid \ \dots \boldsymbol{\psi}_p \mid \boldsymbol{\psi}_1]$
$\mathbf{H} = \mathbf{V}_p^*(\widetilde{\mathbf{H}}\mathbf{V}_{1:p-1})$
$\mathbf{U} = \widetilde{\mathbf{U}}\mathbf{V}_p$
$\mathbf{R} = \widetilde{\mathbf{R}}\mathbf{V}_{1:p-1}$
return $\mathbf{U}$, $\mathbf{H}$, and $\mathbf{R}$

3.3.2 Expanding, Truncating, and Deflating an Approximate Krylov Decomposition

Since our decomposition (3.20) is approximate, we must accommodate for the residual matrix when expanding, truncating, and deflating the decomposition. Prior to expanding the decomposition, we can write (3.20) in terms of the locked and active vectors as

$$\mathcal{A}\left[\widehat{\mathbf{U}}^{\ell}_{1:q-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{q:j-1}\right] = \left[\widehat{\mathbf{U}}^{\ell}_{1:q-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{q:j-1}\right] \begin{pmatrix} \widehat{\mathbf{H}}_{11} & \widehat{\mathbf{H}}_{12} \\ \mathbf{0} & \widehat{\mathbf{H}}_{22} \end{pmatrix}$$

$$+\widehat{\mathbf{u}}_j \left[\mathbf{0} \;\middle|\; \widehat{\mathbf{h}}^*\right] + \left[\mathbf{0} \;\middle|\; \widehat{\mathbf{R}}_{q:j-1}\right], \tag{3.22}$$

where $q-1$ represents the number of converged eigenpairs; see Section 3.3.3 for our convergence criteria. Here, $\widehat{\mathbf{H}}_{11} \in \mathbb{C}^{1:q-1\times 1:q-1}$, $\widehat{\mathbf{H}}_{12} \in \mathbb{C}^{1:q-1\times q:j-1}$, and $\widehat{\mathbf{H}}_{11} \in \mathbb{C}^{q:j-1\times q:j-1}$. Immediately after computing the approximate Krylov decomposition with minimum norm residual and prior to expanding the space for the first time, we assume $q-1=0$. In the following analysis, we let $q \geq 1$.

Note the first $q-1$ columns of the residual matrix, $\widehat{\mathbf{R}}_{1:q-1}$, must be zero. In particular, if $q-1$ vectors have converged we have

$$\mathcal{A}\widehat{\mathbf{U}}^{\ell}_{1:q-1} = \widehat{\mathbf{U}}^{\ell}_{1:q-1}\widehat{\mathbf{H}}_{1:q-1,1:q-1} + \widehat{\mathbf{R}}_{1:q-1}.$$

If $\left\|\widehat{\mathbf{R}}_{1:q-1}\right\|_F$ is not zero (or sufficiently small), $\widehat{\mathbf{U}}^{\ell}_{1:q-1}$ does not span an invariant subspace of $\mathcal{A}$.

We extend the decomposition by taking $m-j+1$ Arnoldi steps, recalling that m represents the maximum dimension of the Krylov subspace. In the Arnoldi expansion, we orthogonalize

each new basis vector, $\widehat{\mathbf{u}}_{j+i}$, against all columns of $\widehat{\mathbf{U}}_{1:j-1+i} = \left[\widehat{\mathbf{U}}^{\ell}_{1:q-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{q:j-1+i}\right]$, for $0 \leq i \leq m-j+1$. Equation (3.22) then becomes

$$\mathcal{A}\left[\widehat{\mathbf{U}}^{\ell}_{1:q-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{q:j-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{j:m}\right] = \left[\widehat{\mathbf{U}}^{\ell}_{1:q-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{q:j-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{j:m}\right] \begin{pmatrix} \widehat{\mathbf{H}}_{11} & \widehat{\mathbf{H}}_{12} & \widehat{\mathbf{H}}_{13} \\ \mathbf{0} & \widehat{\mathbf{H}}_{22} & \widehat{\mathbf{H}}_{23} \\ \mathbf{0} & \mathbf{0} & \widehat{\mathbf{H}}_{33} \end{pmatrix}$$

$$+\widehat{\mathbf{u}}_{m+1}\left[\mathbf{0} \;\middle|\; \widehat{\mathbf{h}}^{*}_{32} \;\middle|\; \widehat{\mathbf{h}}^{*}_{33}\right] + \left[\mathbf{0} \;\middle|\; \widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right]. \tag{3.23}$$

Since $\widehat{\mathbf{H}}_{11}$ is uncoupled from the Rayleigh quotient, we need only consider the decomposition [99, 129]

$$\mathcal{A}\left[\widehat{\mathbf{U}}^{a}_{q:j-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{j:m}\right] = \left[\widehat{\mathbf{U}}^{a}_{q:j-1} \;\middle|\; \widehat{\mathbf{U}}^{a}_{j:m+1}\right] \begin{pmatrix} \widehat{\mathbf{H}}_{22} & \widehat{\mathbf{H}}_{23} \\ \mathbf{0} & \widehat{\mathbf{H}}_{33} \\ \widehat{\mathbf{h}}^{*}_{32} & \widehat{\mathbf{h}}^{*}_{33} \end{pmatrix} + \left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right], \tag{3.24}$$

where $\widehat{\mathbf{U}}^{a}_{j:m+1} = \left(\widehat{\mathbf{U}}^{a}_{j:m} \;\middle|\; \widehat{\mathbf{u}}_{m+1}\right)$. In particular, (3.24) has the form

$$\mathcal{A}\widehat{\mathbf{U}}^{a}_{q:m} = \widehat{\mathbf{U}}^{a}_{q:m+1} \begin{pmatrix} \widehat{\mathbf{H}}_{q:j,q:j-1} & \widehat{\mathbf{H}}_{q:j,j:m} \\ \mathbf{0}_{j+1:m+1,q:j-1} & \widehat{\mathbf{H}}_{j+1:m+1,j:m} \end{pmatrix} + \left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right]. \tag{3.25}$$

Hereafter, we drop the superscript a.

Following (3.25), we then update the Rayleigh quotient as

$$\underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} = \widehat{\mathbf{U}}^{*}_{q:m+1}\mathcal{A}\widehat{\mathbf{U}}_{q:m} =$$

$$\widehat{\mathbf{U}}^{*}_{q:m+1}\widehat{\mathbf{U}}_{q:m+1} \begin{pmatrix} \widehat{\mathbf{H}}_{q:j,q:j-1} & \widehat{\mathbf{H}}_{q:j,j:m} \\ \mathbf{0}_{j+1:m+1,q:j-1} & \widehat{\mathbf{H}}_{j+1:m+1,j:m} \end{pmatrix} + \widehat{\mathbf{U}}^{*}_{q:m+1}\left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right]. \tag{3.26}$$

Note that for $q \geq 1$, $\widehat{\mathbf{U}}_{q:j-1}$ and $\widetilde{\mathbf{U}}_{q:j-1}$ have the same range, and so $\widehat{\mathbf{U}}^{*}_{q:j}\widehat{\mathbf{R}}_{q:j-1} = \mathbf{0}$. It follows that

$$\underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} = \begin{pmatrix} \widehat{\mathbf{H}}_{q:j,q:j-1} & \widehat{\mathbf{H}}_{q:j,j:m} \\ \widehat{\mathbf{U}}^{*}_{j+1:m+1}\widehat{\mathbf{R}}_{q:j-1} & \widehat{\mathbf{H}}_{j+1:m+1,j:m} \end{pmatrix}, \tag{3.27}$$

giving the updated decomposition

$$\mathcal{A}\widehat{\mathbf{U}}_{q:m} = \widehat{\mathbf{U}}_{q:m+1}\underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} + \overline{\mathbf{R}}_{q:m}. \tag{3.28}$$

Here, the residual, $\overline{\mathbf{R}}_{q:m}$ has the form

$$\overline{\mathbf{R}}_{q:m} = \mathcal{A}\widehat{\mathbf{U}}_{q:m} - \widehat{\mathbf{U}}_{q:m+1}\begin{pmatrix}\widehat{\mathbf{H}}_{q:j,q:j-1} & \widehat{\mathbf{H}}_{q:j,j:m} \\ \widehat{\mathbf{U}}^*_{j+1:m+1}\widehat{\mathbf{R}}_{q:j-1} & \widehat{\mathbf{H}}_{j+1:m+1,j:m}\end{pmatrix}$$

$$= \left[\mathcal{A}\widehat{\mathbf{U}}_{q:j-1} - \widehat{\mathbf{U}}_{q:j}\widehat{\mathbf{H}}_{q:j,q:j-1} - \widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right]$$

$$= \left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right] - \widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right]$$

$$= \left(\mathbf{I} - \widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)\left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right], \tag{3.29}$$

where $\mathbf{I}$ is the identity matrix of dimension n. Note also that $\widehat{\mathbf{U}}^*_{q:m+1}\overline{\mathbf{R}}_{q:m} = \mathbf{0}$ since

$$\widehat{\mathbf{U}}^*_{q:m+1}\mathcal{A}\widehat{\mathbf{U}}_{q:m} - \widehat{\mathbf{U}}^*_{q:m+1}\widehat{\mathbf{U}}_{q:m+1}\underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} = \underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} - \underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} = \mathbf{0}.$$

In general, $\underline{\widehat{\mathbf{H}}}_{q:m+1,q:m} = \begin{pmatrix}\widehat{\mathbf{H}}_{q:m} \\ \widehat{\mathbf{h}}^*_{m+1}\end{pmatrix}$ in (3.27) is no longer an upper Hessenberg matrix. Next, we take the Schur decomposition of $\widehat{\mathbf{H}}_{q:m,q:m} = \mathcal{X}\mathbf{\Gamma}\mathcal{X}^*$ (with the eigenvalues in the desired order) to obtain

$$\mathcal{A}\widehat{\mathbf{U}}_{q:m} = (\widehat{\mathbf{U}}_{q:m,q:m}\ \widehat{\mathbf{u}}_{m+1})\begin{pmatrix}\mathcal{X}\mathbf{\Gamma}\mathcal{X}^* \\ \widehat{\mathbf{h}}^*_{m+1}\end{pmatrix} + \overline{\mathbf{R}}_{q:m}, \tag{3.30}$$

and write the change of basis

$$\mathcal{A}\widehat{\mathbf{U}}_{q:m}\mathcal{X} = (\widehat{\mathbf{U}}_{q:m}\mathcal{X}\ \widehat{\mathbf{u}}_{m+1})\begin{pmatrix}\mathbf{\Gamma} \\ \widehat{\mathbf{h}}^*_{m+1}\mathcal{X}\end{pmatrix} + \overline{\mathbf{R}}_{q:m}\mathcal{X}. \tag{3.31}$$

Letting $\mathbf{U}_{q:m} = \widehat{\mathbf{U}}_{q:m}\mathcal{X}$, $\widehat{\mathbf{u}}_{m+1} = \mathbf{u}_{m+1}$, $\mathbf{h}^* = \widehat{\mathbf{h}}^*_{m+1}\mathcal{X}$, and $\mathbf{R}_{q:m} = \overline{\mathbf{R}}_{q:m}\mathcal{X}$, we rewrite (3.31) as

$$\mathcal{A}\mathbf{U}_{q:m} = (\mathbf{U}_{q:m}\ \mathbf{u}_{m+1})\begin{pmatrix}\mathbf{\Gamma} \\ \mathbf{h}^*\end{pmatrix} + \mathbf{R}_{q:m}. \tag{3.32}$$

With $\mathbf{\Gamma} \in \mathbb{C}^{(m-q+1)\times(m-q+1)}$ upper triangular, and $\mathbf{h} \in \mathbb{C}^{(m-q+1)}$ (typically) dense, (3.32) is now in quasi-Schur form. Further, the updated residual has the form

$$\mathbf{R}_{q:m} = \overline{\mathbf{R}}_{q:m}\mathcal{X} = \overline{\mathbf{R}}_{q:m}\begin{pmatrix}\mathcal{X}_{q:j-1,q:m} \\ \mathcal{X}_{j:m,q:m}\end{pmatrix}$$

$$= \left(\mathbf{I} - \widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)\left[\widehat{\mathbf{R}}_{q:j-1} \;\middle|\; \mathbf{0}\right]\begin{pmatrix}\mathcal{X}_{q:j-1,q:m}\\ \mathcal{X}_{j:m,q:m}\end{pmatrix}$$

$$= \left(\mathbf{I} - \widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)\widehat{\mathbf{R}}_{q:j-1}\mathcal{X}_{q:j-1,q:m}. \tag{3.33}$$

Finally, rewriting (3.32) as

$$\mathcal{A}\left[\mathbf{U}_{q:j-1} \;\middle|\; \mathbf{U}_{j:m}\right] = \left[\mathbf{U}_{q:j-1} \;\middle|\; \mathbf{U}_{j:m} \;\middle|\; \mathbf{u}_{m+1}\right]\begin{pmatrix}\mathbf{\Gamma}_{q:j-1,q:j-1} & \mathbf{\Gamma}_{q:j-1,j:m}\\ \mathbf{0}_{j:m,q:j-1} & \mathbf{\Gamma}_{j:m,j:m}\\ (\mathbf{h}^*)_{q:j-1} & (\mathbf{h}^*)_{j:m}\end{pmatrix}$$

$$+ \left[\mathbf{R}_{q:j-1} \;\middle|\; \mathbf{R}_{j:m}\right], \tag{3.34}$$

where $\mathbf{\Gamma}_{q:j-1,q:j-1}$ contains the desired eigenvalues along the diagonal, we truncate (3.34) as

$$\mathcal{A}\mathbf{U}_{q:j-1} = [\mathbf{U}_{q:j-1}\ \mathbf{u}_{m+1}]\begin{pmatrix}\mathbf{\Gamma}_{q:j-1,q:j-1}\\ (\mathbf{h}^*)_{q:j-1}\end{pmatrix} + \mathbf{R}_{q:j-1}, \tag{3.35}$$

where

$$\mathbf{R}_{q:j-1} = \left(\mathbf{I} - \widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)\widehat{\mathbf{R}}_{q:j-1}\mathcal{X}_{q:j-1,q:j-1}. \tag{3.36}$$

We then determine those Ritz values and vectors that have converged using the convergence criteria described in the next section, deflate the corresponding columns of $\mathbf{U}_{q:j}$, and update q. With $\widehat{\mathbf{U}}_{q:j-1} = \mathbf{U}_{q:j-1}$, $\widehat{\mathbf{u}}_j = \mathbf{u}_{m+1}$, and $\widehat{\mathbf{R}}_{q:j-1} = \mathbf{R}_{q:j-1}$, (3.22) is satisfied and we repeat the warm-start Krylov-Schur cycle until the desired number of approximate eigenpairs have been computed. The algorithm for warm-start Krylov-Schur is given in Algorithm 5.

3.3.3 Computing Approximate Eigenpairs

We discuss convergence criteria for the warm-start Krylov-Schur method by first considering the error we introduce when warm-starting the eigensolver with a space that is not a Krylov space for the current matrix. Consider the approximate Krylov-Schur decomposition

$$\mathcal{A}\mathbf{U}_{1:j-1} = \mathbf{U}_{1:j}\begin{pmatrix}\mathbf{\Gamma}\\ \mathbf{h}^*\end{pmatrix} + \mathbf{R}. \tag{3.37}$$

Let $\mathbf{S} = \begin{pmatrix}\mathbf{\Gamma}\\ \mathbf{h}^*\end{pmatrix}$ and compute the eigendecomposition $\mathbf{\Gamma V} = \mathbf{VD}$. Given the eigenpair of $\mathbf{\Gamma}$, $(\theta_i, \mathbf{y}_i)$, we compute the Ritz vector $\mathbf{U}_{1:j-1}\mathbf{y}_i$, for $i = q, q+1, \ldots, j-1$. Then, for $(\theta, \mathbf{z}) = (\theta, \mathbf{U}_{1:j-1}\mathbf{y})$ a Ritz pair of $\mathcal{A}$, and following (3.12), we can write

$$\|\mathcal{A}\mathbf{z} - \theta\mathbf{z}\|_F^2 = \|\mathbf{\Gamma y} - \theta\mathbf{y}\|_F^2 + \left|\mathbf{b}_j^*\mathbf{y}\right|^2 + \|\mathbf{Ry}\|_2^2 = \left|\mathbf{b}_j^*\mathbf{y}\right|^2 + \|\mathbf{Ry}\|_2^2. \tag{3.38}$$

Algorithm 5 Warm-Start Krylov-Schur with Convergence Testing

Given: $\mathcal{A} \in \mathbb{C}^{n\times n}$, $\mathbf{W}_j = [\mathbf{w}_1 \mid \mathbf{w}_2 \mid \ldots \mid \mathbf{w}_j]$, max_iter, expansion size m, truncation size j, number of desired eigenpairs $numev \leq j$
{ Obtain $\mathbf{U}$, $\mathbf{H}$, and $\mathbf{R}$ from Algorithm 4 given $\mathbf{W}$ }
$iter = 1$, $q = 1$
while $iter < max\_iter$ and $q \leq numev$ **do**
  for $i = j : m$ **do** { Arnoldi expansion }
    $\mathbf{t} = \mathcal{A}\mathbf{u}_i$
    $\mathbf{v} = \mathbf{U}_{1:i}^{*}\mathbf{t}$
    $\mathbf{t} = \mathbf{t} - \mathbf{U}_{1:i}\mathbf{v}$
    $\mathbf{H}_{1:i,i} = \mathbf{v}$
    $\mathbf{H}_{i+1,i} = \|\mathbf{t}\|_2$
    $\mathbf{u}_{i+1} = \mathbf{t} \;/\; \mathbf{H}_{i+1,i}$
  end for
  { Update Rayleigh quotient and residual }
  $\mathbf{H}_{j+1:m+1,q:j-1} = \mathbf{U}_{j+1:m+1}^{*}\mathbf{R}_{q:j-1}$
  $\mathbf{R}_{q:j-1} = \mathbf{R}_{q:j-1} - \mathbf{U}_{j+1:m+1}\mathbf{H}_{j+1:m+1,q:j-1}$
  { Compute Schur form with eigenvalues in desired order }
  $[\boldsymbol{\mathcal{X}}, \boldsymbol{\Gamma}] = \text{schur}(\mathbf{H}_{q:m,q:m})$
  { Change-of-basis and truncate Schur form }
  $\mathbf{U}_{q:j-1} = \mathbf{U}_{q:j-1}\boldsymbol{\mathcal{X}}_{1:j-q,1:j-q}$, $\mathbf{u}_j = \mathbf{u}_{m+1}$
  $\mathbf{H}_{q:j-1,q:j-1} = \boldsymbol{\Gamma}_{1:j-q,1:j-q}$
  $\mathbf{H} = \mathbf{H}_{1:j,1:j-1}$, $\mathbf{U} = \mathbf{U}_{1:j}$
  $\mathbf{R}_{q:j-1} = \mathbf{R}_{q:j-1}\boldsymbol{\mathcal{X}}_{1:j-q,1:j-q}$
  { Check for convergence using Algorithm 6 and obtain $\widetilde{\mathbf{H}}$, $\widetilde{\mathbf{U}}$, and the number of converged eigenpairs, $numConv$ }
  $\mathbf{H}_{q:j,q:j-1} = \widetilde{\mathbf{H}}$
  $\mathbf{U}_{q:j-1} = \widetilde{\mathbf{U}}$
  $q = q + numConv$
  $iter = iter + 1$
end while
{ For $\mathbf{H}_{1:j-1,1:j-1}\mathbf{V} = \mathbf{V}\mathbf{D}$, the eigendecomposition of $\mathbf{H}_{1:j-1,1:j-1}$ }
Let $\ell = \min(q, numev)$
$\mathbf{W}_{new} = \mathbf{U}_{1:\ell}\mathbf{V}_{1:\ell,1:\ell}$ { Compute the approx. invariant subspace of $\mathcal{A}$ }

While (3.38) gives us a formula for the norm of the residual of the approximate eigenpair $(\theta, \mathbf{z})$ when warm-starting Krylov-Schur, we can no longer use the standard convergence test in (3.13). Certainly, if $\mathbf{R}$ has structure we can exploit, the matrix-vector product $\mathbf{Ry}$ may be cheap to compute. We will show in Section 3.3.4 that the norm of the residual of the approximate Krylov-Schur decomposition monotonically decreases, and in cases where $\|\mathbf{R}\|_2$ is small enough, we may still be able to use (3.38) as a convergence test. How we can use (3.38) to develop a cheap test for convergence is part of ongoing work.

Instead, we test the accuracy of the approximate eigenpairs by directly computing the relative residual following [130, Chapter 2, Section 1]

$$\frac{\|\mathcal{A}\mathbf{z} - \theta\mathbf{z}\|_2}{(\|\mathcal{A}\|_2 + |\theta|)\|\mathbf{z}\|_2}. \tag{3.39}$$

Note that norm estimators can be used to efficiently estimate $\|\mathcal{A}\|_2$ [83]. If (3.39) is below some user-defined tolerance for the Ritz pair $(\theta_i, \mathbf{z}_i)$, we consider that Ritz pair to be converged. Since the Ritz pairs may converge at different rates, they may not correspond to contiguous columns of $\mathbf{U}_{1:j-1}$ (i.e., the Schur vectors). In this case, we can reorder our decomposition as in steps (3.7)-(3.9). Then, our decomposition has the form

$$\mathcal{A}\begin{bmatrix}\mathbf{U}^\ell \mid \mathbf{U}^a\end{bmatrix} = \begin{bmatrix}\mathbf{U}^\ell \mid \mathbf{U}^a\end{bmatrix}\begin{pmatrix}\widetilde{\mathbf{\Gamma}}_{11} & \widetilde{\mathbf{\Gamma}}_{12} \\ \mathbf{0} & \widetilde{\mathbf{\Gamma}}_{22}\end{pmatrix} + \mathbf{u}_j\mathbf{h}_j^*,$$

where $\widetilde{\mathbf{\Gamma}}_{11} \in \mathbb{C}^{r\times r}$ and the columns of $\mathbf{U}^\ell \in \mathbb{C}^{n\times r}$ correspond to the approximate eigenpairs $\mathcal{A}$ that have converged and so are deflated from the decomposition. We update $q = q + r$ and continue warm-start Krylov-Schur at step (3.25) until all desired eigenpairs of $\mathcal{A}$ have been approximately computed. Algorithm 6 summarizes the check for convergence.

This convergence test is generalizable to other types of eigenproblems by simply modifying the relative residual norm considered. For instance, when solving a sequence of GEPs as given in (3.2) with $\mathcal{A}$ as in (3.3), and as is the case in the application we consider in Section 3.4, we can use the relative residual norm [130, Chapter 5, Section 4]:

$$\frac{\|\mathbf{A}\mathbf{z} - \theta\mathbf{B}\mathbf{z}\|_2}{(\|\mathbf{A}\|_2 + |\theta|\|\mathbf{B}\|_2)\|\mathbf{z}\|_2}. \tag{3.40}$$

In applications where the GEP comes from the linearization of a quadratic eigenvalue problem (QEP) of the form

$$P(\lambda)\mathbf{x} = (\lambda^2\mathbf{M} + \lambda\mathbf{D} + \mathbf{K})\mathbf{x} = 0, \tag{3.41}$$

where $\mathbf{M}, \mathbf{D}, \mathbf{K} \in \mathbb{C}^{n\times n}$, we could instead use the relative residual of the QEP. For example, we can use the relative residual [13]

$$\frac{\|(\theta^2\mathbf{M} + \theta\mathbf{D} + \mathbf{K})\mathbf{z}\|_2}{|\theta|^2\|\mathbf{M}\|_1 + |\theta|\|\mathbf{D}\|_1 + \|\mathbf{K}\|_1}, \tag{3.42}$$

Algorithm 6 Check Accuracy of Approximate Eigenpairs

Given $\mathcal{A} \in \mathbb{C}^{n\times n}$, and $\mathbf{H}$, $\mathbf{U}$ such that $\mathcal{A}\mathbf{U} \approx \mathbf{U}\mathbf{H}$, number of eigenvalues already converged $q-1$, accuracy tol, number of desired eigenvalues $numev$, and the Schur decomposition of $\mathbf{H} = \boldsymbol{\mathcal{X}}\boldsymbol{\Gamma}\boldsymbol{\mathcal{X}}^*$
Let $\widetilde{\mathbf{H}} = \mathbf{H}$ and $\widetilde{\mathbf{U}} = \mathbf{U}$
$\mathbf{Z} = \mathbf{U}_{1:j}\boldsymbol{\mathcal{X}}$
for $i = q : j$ **do**
    $\lambda = \boldsymbol{\Gamma}_{i,i}$, $\mathbf{x} = \mathbf{Z}_i$
    $r = \|\mathcal{A}\mathbf{x} - \lambda\mathbf{x}\|_2$
    $denom = (\|\mathcal{A}\|_2 + |\lambda|)\|\mathbf{x}\|_2$
    if $r/denom < tol$ **then**
        { Compute Schur decomposition so that $\widetilde{\mathbf{H}}_{11}$ contains the desired eigenvalues }
        Compute $\widetilde{\mathbf{H}} = \begin{pmatrix} \widetilde{\mathbf{H}}_{11} & \widetilde{\mathbf{H}}_{12} \\ \mathbf{0} & \widetilde{\mathbf{H}}_{22} \end{pmatrix} = \mathbf{Q}^*\widetilde{\mathbf{H}}\mathbf{Q}$
        $\widetilde{\mathbf{U}} = \widetilde{\mathbf{U}}\mathbf{Q}$
        $numConv = numConv + 1$
        if $q + numConv > numev$ **then**
            break
        end if
    end if
end for
return $\widetilde{\mathbf{H}}, \widetilde{\mathbf{U}}, numConv$

or [76]

$$\frac{\|(\theta^2\mathbf{M}+\theta\mathbf{D}+\mathbf{K})\mathbf{z}\|_\infty}{\|(|\theta|^2|\mathbf{M}|+|\theta||\mathbf{C}|+|\mathbf{K}|)|\mathbf{z}|\|_\infty}. \tag{3.43}$$

3.3.4 Properties of Warm-Start Krylov-Schur

In this section we will refer back to the residual as it is updated and transformed at various steps in our algorithm, and so we restate them here for convenience. Recall that $\widetilde{\mathbf{R}}_{1:j}$ as in (3.16) is the initial residual prior to computing the approximate Krylov decomposition with minimum norm residual, and $\widehat{\mathbf{R}}_{1:j-1}$ in (3.19) is the residual after computing this decomposition. $\overline{\mathbf{R}}_{1:m}$ is the updated residual in (3.29) after expanding the space and updating the Rayleigh quotient, $\mathbf{R}_{1:m}$ is the residual in (3.33) after ordering the desired eigenvalues, and $\mathbf{R}_{1:j-1}$ is the truncated residual as given in (3.36).

By computing the approximate Krylov decomposition as guided by [132], we guarantee the minimum norm residual when warm-starting the eigensolver. We formally state this in the following theorem.

Theorem 3.1. *Let $\mathcal{A}\in\mathbb{C}^{n\times n}$ and $\mathbf{X}_{1:j}$ span an approximate invariant subspace. Further, let $\widetilde{\mathbf{H}}_{1:j,1:j}$ be as in (3.15) and $\widetilde{\mathbf{R}}_{1:j}$ be as in (3.16). Then, the globally optimal Krylov residual for the basis $\widetilde{\mathbf{U}}_{1:j}\mathbf{V}_{1:j-1}$ is attained by letting $\mathbf{V}_{1:j-1}$ be the right singular vectors of $\widetilde{\mathbf{R}}_{1:j}$ corresponding to $\sigma_2,\sigma_3,\ldots,\sigma_j$.*

Proof. We refer the reader to [132, Theorem 3.2]. □

In the context of our warm-start Krylov-Schur method, this gives us that $\|\widehat{\mathbf{R}}_{1:j-1}\|_F \leq \|\widetilde{\mathbf{R}}_{1:j-1}\|_F$, and our next theorem guarantees that we can continue to expect a monotonic decrease in the residual.

Theorem 3.2. *Let $\mathbf{R}^{(p-1)}_{1:j-1}$ and $\mathbf{R}^{(p)}_{1:j-1}$ represent the residuals of the approximate Krylov decomposition at the completion of the $(p-1)^{st}$ and p^{th} steps of warm-start Krylov-Schur, respectively. Then, $\|\mathbf{R}^{(p)}_{1:j-1}\|_F \leq \|\mathbf{R}^{(p-1)}_{1:j-1}\|_F$.*

Proof. Note that showing $\|\mathbf{R}^{(p)}_{1:j-1}\|_F \leq \|\mathbf{R}^{(p-1)}_{1:j-1}\|_F$ is equivalent to showing $\left\|\mathbf{R}_{1:j-1}\right\|_F \leq \left\|\widehat{\mathbf{R}}_{1:j-1}\right\|_F$. Recall that

$$\overline{\mathbf{R}}_{1:j-1} = \left(\mathbf{I}-\widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)\left[\widehat{\mathbf{R}}_{1:j-1} \;\middle|\; \mathbf{0}\right].$$

It is easily verified that $\left(\mathbf{I}-\widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)$ is an orthogonal projector. Then

$$\left\|\overline{\mathbf{R}}_{1:m}\right\|_F = \left\|\left(\mathbf{I}-\widehat{\mathbf{U}}_{j+1:m+1}\widehat{\mathbf{U}}^*_{j+1:m+1}\right)\left[\widehat{\mathbf{R}}_{1:j-1} \;\middle|\; \mathbf{0}\right]\right\|_F$$

$$\leq \left\| \left(\mathbf{I} - \widehat{\mathbf{U}}_{j+1:m+1} \widehat{\mathbf{U}}_{j+1:m+1}^* \right) \right\|_2 \left\| \left[\widehat{\mathbf{R}}_{1:j-1} \quad \middle| \quad \mathbf{0} \right] \right\|_F$$

$$= \left\| \left[\widehat{\mathbf{R}}_{1:j-1} \quad \middle| \quad \mathbf{0} \right] \right\|_F = \left\| \widehat{\mathbf{R}}_{1:j-1} \right\|_F.$$

Then with

$$\left\| \mathbf{R}_{1:j-1} \right\|_F \leq \left\| \mathbf{R}_{1:m} \right\|_F = \left\| \overline{\mathbf{R}}_{1:m} \mathcal{X} \right\|_F = \left\| \overline{\mathbf{R}}_{1:m} \right\|_F,$$

we obtain our result. □

3.4 Numerical Simulation of Brake Squeal

In [76], Gräbner, et al., use a finite element model to analyze instability of disk brake squeal using model reduction, such that the smaller-scale problem retains the instability of the large-scale problem. Physically, the instability in the disk manifests as the high-pitched, brake squeal noise. Mathematically, it is associated with the unstable eigenvalues (or those with positive real part) of a parameter dependent QEP of the form

$$P_\Omega(\lambda_\Omega)\mathbf{x}_\Omega = (\lambda^2 \mathbf{M}_\Omega + \lambda \mathbf{D}_\Omega + \mathbf{K}_\Omega)\mathbf{x}_\Omega = 0, \tag{3.44}$$

where $\mathbf{M}_\Omega, \mathbf{D}_\Omega, \mathbf{K}_\Omega \in \mathbb{C}^{n\times n}$ are respectively the mass, damping, and stiffness matrices from the finite element discretization modeling brake squeal and $\Omega \in [\Omega_{min}, \Omega_{max}]$ represents the rotational velocity of the disk measured in radians per second [76]. We provide relevant information from [76] next, but refer the reader to [76, Section 2] for a detailed discussion of the finite element model used to describe the characteristics of the physical system.

In [76], a POD approach is introduced to efficiently analyze this instability by projecting (3.44) to a smaller space, $\mathcal{Q}$, of dimension $d \ll n$ by constructing an orthogonal matrix $\mathbf{Q} \in \mathbb{C}^{n\times d}$ whose range spans $\mathcal{Q}$ and letting $\mathbf{M}_\Omega^r = \mathbf{Q}^T\mathbf{M}_\Omega\mathbf{Q}$, $\mathbf{D}_\Omega^r = \mathbf{Q}^T\mathbf{D}_\Omega\mathbf{Q}$, and $\mathbf{K}_\Omega^r = \mathbf{Q}^T\mathbf{K}_\Omega\mathbf{Q}$. Then, the reduced QEP is defined as

$$P_\Omega^r(\lambda_\Omega)\mathbf{x}_\Omega^r = (\lambda^2 \mathbf{M}_\Omega^r + \lambda \mathbf{D}_\Omega^r + \mathbf{K}_\Omega^r)\mathbf{x}_\Omega^r = 0, \tag{3.45}$$

with $\mathbf{M}_\Omega^r, \mathbf{D}_\Omega^r, \mathbf{K}_\Omega^r \in \mathbb{C}^{d\times d}$. Given the small dimension of the reduced system, the eigenvectors, $\mathbf{x}_\Omega^r$, of (3.45) can be efficiently computed. Approximations of the eigenvectors $\mathbf{x}_\Omega$ of (3.44) are then computed as $\mathbf{Q}\mathbf{x}_\Omega^r$ [76].

To represent the instability of (3.44) in (3.45), the matrix $\mathbf{Q}$ is constructed such that its range contains good approximations to those eigenvectors of (3.44) that correspond to the self-excited modes (i.e., the unstable eigenvalues) [76]. At different rotational velocities, generally there are different eigenvalues that become unstable and so (3.44) is solved for many values of Ω across the domain. In both [76] and the current chapter, $\Omega \in [1, 4] \cdot 2\pi$.

We are interested in approximately computing those eigenvalues with positive real part. However, since iterative projection methods generally compute the exterior eigenvalues, a shift-invert enhancement should be used. In [76], a shift-invert-linearize enhancement is used. First, a shift point τ_i is chosen for each of $\Omega_j \in [1,4] \cdot 2\pi$ in the positive half plane near the eigenvalues of interest (e.g., those eigenvalues of (3.44) with positive real part) [76]. This results in multiple, closely related QEPs of the form

$$((\lambda - \tau_i)^2 \mathbf{M}_{\tau_i} + (\lambda - \tau_i)\mathbf{D}_{\tau_i} + \mathbf{K}_{\tau_i})\mathbf{x} = \mathbf{0}, \tag{3.46}$$

where $\mathbf{M}_{j_{\tau_i}} = \mathbf{M}_{\Omega_j}$, $\mathbf{D}_{j_{\tau_i}} = 2\tau_i \mathbf{M}_{\Omega_j} + \mathbf{D}_{\Omega_j}$, and $\mathbf{K}_{j_{\tau_i}} = \tau_i^2 \mathbf{M}_{\Omega_j} + \tau_i \mathbf{D}_{\Omega_j} + \mathbf{K}_{\Omega_j}$ [76]. We note that since $\mathbf{M}_{\tau_i}$ and $\mathbf{K}_{\tau_i}$ often vary by large orders of magnitude, (3.46) is scaled in [76]. For simplicity, we will omit this from the current discussion, without loss of generality, but refer the reader to [76, Section 4.1] for details on the scaling factor. To compute the interior eigenvalues, the reverse polynomial of (3.46) is computed, giving

$$(\mu_{\tau_i}^2 \mathbf{I} + \mu_{\tau_i} \mathbf{K}_{\tau_i}^{-1} \mathbf{D}_{\tau_i} + \mathbf{K}_{\tau_i}^{-1} \mathbf{M}_{\tau_i})\mathbf{x} = \mathbf{0}, \tag{3.47}$$

where $\mu_{\tau_i} = \frac{1}{\lambda - \tau_i}$.[1] Then, linearizing (3.47) gives a $2n \times 2n$ generalized eigenvalue problem of the form (3.2) where

$$\mathbf{A}_{\tau_i} = \begin{bmatrix} -\mathbf{D}_{\tau_i} & -\mathbf{M}_{\tau_i} \\ \mathbf{I}_n & \mathbf{0}_n \end{bmatrix} \tag{3.48}$$

and

$$\mathbf{B}_{\tau_i} = \begin{bmatrix} \mathbf{K}_{\tau_i} & \mathbf{0}_n \\ \mathbf{0}_n & \mathbf{I}_n \end{bmatrix}. \tag{3.49}$$

In [76], this is solved as a $2n \times 2n$ standard eigenvalue problem of the form (3.1) where

$$\mathcal{A}_{j_{\tau_i}} = \mathbf{B}_{\tau_i}^{-1} \mathbf{A}_{\tau_i} = \begin{bmatrix} -\mathbf{K}_{\tau_i}^{-1} \mathbf{D}_{\tau_i} & -\mathbf{K}_{\tau_i}^{-1} \mathbf{M}_{\tau_i} \\ \mathbf{I}_n & \mathbf{0}_n \end{bmatrix}. \tag{3.50}$$

Given the solution to $\mathcal{A}_{j_{\tau_i}} \mu_{\tau_i} = \mu_{\tau_i} \mathbf{y}$, reversing the spectral transformation identifies the unstable eigenvalues of (3.44) for a specific $\Omega_j \in [\Omega_{min}, \Omega_{max}]$. With several Ω_j chosen over this domain, many QEPs of the form (3.46) must be solved for multiple shifts, τ_i. For each $\mathcal{A}_j$, a subspace of dimension 50 is computed, while for each $\mathcal{A}_{j_{\tau_i}}$, a subspace of dimension 10 is computed. We follow the application as it is presented in [76], but other approaches exist for solving GEPs; see [72, 105, 139].

Then, $\mathbf{X}(\Omega_j)$, the matrix containing those approximate eigenvectors of (3.50), is constructed as [76]

$$\mathbf{X}(\Omega_j) = [Re\,[\mathbf{X}(\Omega_1)\ \mathbf{X}(\Omega_2)\ \ldots \mathbf{X}(\Omega_k)]\ \ Im\,[\mathbf{X}(\Omega_1)\ \mathbf{X}(\Omega_2)\ \ldots \mathbf{X}(\Omega_k)]]\,. \tag{3.51}$$

[1] In [76], the numerator of μ_{τ_i} as it is given in this chapter is replaced with the scaling factor.

Finally, $\mathbf{Q}$ is constructed using the $d \ll n$ left singular vectors of $\mathbf{X}$ corresponding to those singular values above a prescribed threshold [76]. In subsequent discussion and analysis, we denote the unshifted matrix as

$$\mathcal{A}_j = \begin{bmatrix} -\mathbf{K}^{-1}\mathbf{D} & -\mathbf{K}^{-1}\mathbf{M} \\ \mathbf{I}_n & \mathbf{0}_n \end{bmatrix}, \tag{3.52}$$

and $\mathcal{A}_{j_{\tau_i}}$ represents $\mathcal{A}_j$ shifted by τ_i from the shift-invert Arnoldi procedure. We use the approximate invariant subspace computed for $\mathcal{A}_j$ to warm-start the Krylov-Schur method for each of $\mathcal{A}_{j_{\tau_i}}$.

3.4.1 Krylov Subspace Recycling When Solving Generalized Eigenvalue Problems

Since we apply our algorithm to a sequence of GEPs of the form (3.2), which are rewritten (and solved) as standard eigenvalues problems of the form (3.1), the matrix-vector products computed using the matrix (3.3) in Algorithms 4 and 5 require an implicit linear solve, and since we use iterative methods, these matrix-vector products are approximate. For large matrices, line 3 of Algorithm 4 and line 6 of Algorithm 5 should use an iterative (Krylov) method to efficiently compute the matvecs. Since each of these lines is within a loop and each uses the same matrix, this constitutes a sequence of linear systems, or more specifically a linear system with multiple right hand sides. Therefore, Krylov subspace recycling can improve convergence of the iterative method. In our numerical experiments, we use GCRO-DR [110], but GCROT [57] can also be used.

Computing the approximate solution using an iterative solver for the matrix-vector products introduces an error between the approximate and exact matrix-vector product. We do check the accuracy of the space we compute, but analyzing this error in combination with the residual matrix from the approximate Krylov decomposition, and its effect on the quality of the invariant subspace computed using warm-start Krylov-Schur, is future work.

3.5 Numerical Experiments

We analyze the effectiveness of warm-start Krylov-Schur applied to two model: an academic model DEVX6 [76, 93] and an industrial model from [76]. For the academic model, the matrices $\mathbf{M}, \mathbf{K}$, and $\mathbf{D}$ have dimension $n = 4669$ [76, Section 5.1]. To solve the linearized QEP in [76], the MATLAB® function `eigs` is used with an LU factorization computed for each $\mathcal{A}_j$ and $\mathcal{A}_{j_{\tau_i}}$, and we refer to this as the *current strategy*. For matrices with large dimension, such as the industrial model with $n = 842638$, an LU factorization may be very expensive to compute and iterative methods can reduce this cost. We show that Krylov subspace recycling can further reduce the cost of the iterative solver. So, we compare the

current strategy with GMRES [122] and GCRO-DR [110], using Krylov-Schur and warm-start Krylov-Schur.

For both the academic and industrial model, warm-start Krylov-Schur reduces the total number of matrix-vector products computed within the eigensolver, and results in an invariant subspace that is as accurate as the current strategy.

Academic Model

We consider a sequence of eigenvalue problems of the form (3.2) that arise from choosing 17 values of Ω across the domain $[1, 4] \cdot 2\pi$. Each of the corresponding QEPs of the form (3.44) has anywhere from two to four unstable eigenvalues, and therefore is shifted two to four times via shift-invert Arnoldi, giving us a total of 78 eigenvalue problems. We refer to this as the *full sequence*.

For the full sequence, we consider the following strategies, each of which is compared with the current strategy:

1. Krylov-Schur using the exact LU factorization for the matvecs,
2. Warm-start Krylov-Schur using the exact LU factorization for the matvecs,
3. Krylov-Schur using preconditioned, restarted GMRES for the matvecs,
4. Krylov-Schur using preconditioned, GCRO-DR for the matvecs, and
5. Warm-start Krylov-Schur using preconditioned, GCRO-DR for the matvecs.

We use *numev* to denote the dimension of the approximate invariant subspace computed and, next, we give the parameters chosen for each of the methods used in strategies (1)-(5) above.

When computing the exact LU factorization, we use MATLAB's® `lu` function and we use all permutation and scaling matrices available as output from this function. When using restarted GMRES, we set the convergence tolerance to 10^{-10}, maximum iterations to 200, and the restart to 60. For the initial call to GMRES, we use the zero vector to start, and for all subsequent calls, we use the previous solution as the start vector. For GCRO-DR, we use all of the same parameters as with GMRES, and set the dimension of the recycle space to 40.

For Krylov-Schur, we use our own implementation based on the algorithm provided in [129, 130, 132] and choose the start vector of all ones and a relative convergence tolerance of 10^{-16}. We set the maximum Krylov space, or the *expansion space*, dimension to $numev + 20$. Finally, we allow the maximum number of eigensolver iterations to be 20. For warm-start Krylov-Schur, we use the invariant subspace computed for (3.52) as the warm-start space

for each of (3.50) and use the convergence tolerance 10^{-4}.[2] We set the expansion space dimension again to $numev + 20$, and allow a maximum of 5 eigensolver iterations.

Since we solve a sequence of eigenvalue problems of the form (3.1) with the coefficient matrix as in (3.52), multiplication by (3.52) requires an implicit linear solve. Specifically, we solve for $\mathbf{K}_{\tau_i}$ and these matrices can be ill-conditioned; see Table 3.1. So, for those strategies where it is appropriate (e.g., strategies (3)-(5)), we compute an ILUTP preconditioner for $\mathbf{K}_{\tau_i}$ using MATLAB's® function `ilu` with type 'ilutp' and drop tolerance 10^{-3}. For all $\mathbf{K}_{\tau_i}$ in the sequence of eigenvalue problems, the $\mathbf{L}$ and $\mathbf{U}$ factors of the ILUTP factorization have approximately 40 nonzeros per column on average, whereas there is an average of approximately 155 nonzeros per column in each of the exact $\mathbf{L}$ and $\mathbf{U}$ factors when computing the LU factorization. This is compared with approximately 63 nonzeros per column in $\mathbf{K}_{\tau_i}$.

In our numerical results, we give two sets of iterations: iterative solver iterations (e.g., GMRES and GCRO-DR iterations), and eigensolver iterations (e.g., Krylov-Schur and warm-start Krylov-Schur cycles). However, when using an exact LU decomposition, we obviously do not use an iterative solver. So, we provide the eigensolver iterations. In some cases, using recycling increases the number of eigensolver iterations for individual matrices, but in all cases, when recycling, the *total* number of solver iterations decreases.

Matrix	$\mathcal{A}_1$	$\mathcal{A}_{1_{\tau_1}}$	$\mathcal{A}_{1_{\tau_2}}$	$\mathcal{A}_2$	$\mathcal{A}_{2_{\tau_1}}$	$\mathcal{A}_{2_{\tau_2}}$	$\mathcal{A}_{2_{\tau_3}}$
Cond. Num.	10^6	10^{17}	10^{18}	10^6	10^{18}	10^{18}	10^{18}

Table 3.1: Condition number of $\mathcal{A}_1$ and $\mathcal{A}_2$ from the full sequence, along with the corresponding shifted matrices, $\mathcal{A}_{j_{\tau_i}}$ for $j = 1, 2$ and $i = 1, 2, 3$.

We first compare the accuracy of our methods with that of the current method by considering one matrix $\mathcal{A}$, and three shifted matrices, $\mathcal{A}_{\tau_i}$, for $i = 1, 2, 3$. Specifically, $\mathcal{A}$ corresponds to (3.52) for $\Omega = 4\pi$, and $\mathcal{A}_{\tau_i}$ correspond to the three shifted matrices of the form (3.50). There are three unstable eigenvalues, hence the three shifted systems (from shift-invert Arnoldi); see Figure (3.1).

For this example, we focus on comparing strategies (4) and (5) with the current method, and let $numev = 50$ be the total number of eigenpairs we compute for $\mathcal{A}$ as well as for all $\mathcal{A}_{\tau_i}$. Note that for this application, $numev = 50$ for each $\mathcal{A}_{\tau_i}$ is not necessary to construct an accurate POD basis [76]. Rather, we choose a larger subspace to highlight the accuracy of warm-start Krylov-Schur. In fact, even though a subspace of dimension 10 is computed

[2]While this convergence tolerance is quite large compared with that for Krylov-Schur, we note that we compute the eigenvalues of interest to a high accuracy (e.g., the residuals of the approximate eigenvalues in the right half plane are all approximately 10^{-12}).

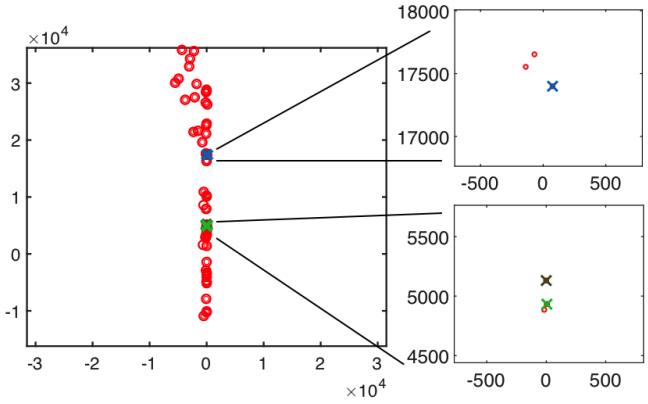


Figure 3.1: The three unstable eigenvalues for $\mathcal{A}$ corresponding to $\Omega = 4\pi$.

for each $\mathcal{A}_{j_{\tau_i}}$, even fewer approximate eigenvectors ultimately contribute to the POD basis as in (3.51). In particular, when comparing the approximate invariant subspace computed using the current strategy against strategies (4) and (5), we see that both Krylov-Schur and warm-start Krylov-Schur produce spaces that are accurate for more than half of the vectors when computing the approximate invariant subspaces for $\mathcal{A}_{\tau_i}$; see Figures 3.2(a) and 3.2(b). We verify that we are accurately approximating the desired number of eigenvectors since they are used to build the POD basis in (3.51). Figure 3.3 shows that we compute the approximate invariant subspace corresponding to the ten smallest approximate eigenvalues in the right half plane as accurately as the current method. We also see that the warm-start Krylov-Schur appears to compute a slightly better space for $\mathcal{A}_{\tau_3}$ corresponding to those 10 eigenvectors. Further, warm-start Krylov-Schur reduces the total number of GCRO-DR iterations compared with Krylov-Schur (see Tables 3.2 and 3.3).

Next we apply all strategies (1)-(5) to the full sequence of eigenvalue problems. In Table 3.4 we provide cumulative results using the parameters listed above. We see that total eigensolver iterations are reduced when using warm-start Krylov-Schur (compare the column labeled **LU/WSKS** with **LU/KS**, and **GCRO-DR/WSKS** with **GCRO-DR/KS**), and iterative solver iterations are reduced when using GCRO-DR (compare **GCRO-DR/WSKS** with **GMRES/KS**).

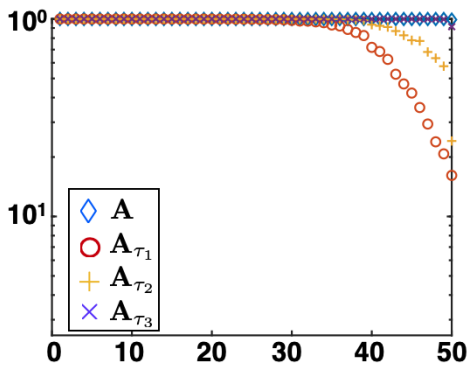
We compare iterations for selected matrices from the full sequence; see Table 3.5. We note that in many cases, warm-starting reduces total iterative solver iterations and eigensolver iterations, and for those systems where iterative solver iterations are not decreased, they are more or less equal (e.g., see $\mathcal{A}_{1_{\tau_2}}$). This may be due to the quality of the warm-start space currently used, or more specifically, the quality of the specific vectors in the matrix spanning the warm-start space. Determining the best vectors to maintain from the matrix spanning the warm-start space is part of future work.

Matvec	$\mathcal{A}$	$\mathcal{A}_{\tau_1}$	$\mathcal{A}_{\tau_2}$	$\mathcal{A}_{\tau_3}$
1	26	212	212	212
2	7	17	11	16
3	7	18	11	16
4	7	18	11	16
5	7	18	11	16
Total Iters	620	2638	1058	1332
Total Matvecs	70	70	70	70
Avg Iter	8.86	37.69	15.11	19.03

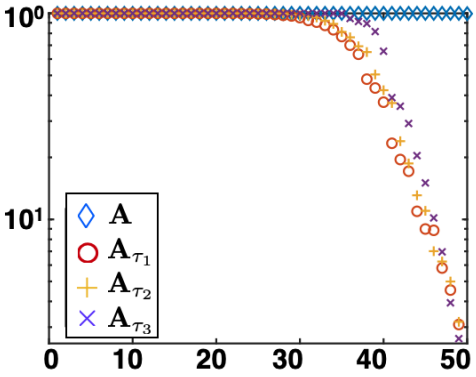
Table 3.2: Total GCRO-DR iterations for each matrix-vector product in Krylov-Schur, with the first five matvecs highlighted. "Total Iters" denotes the total number of GCRO-DR iterations for all calls to the iterative solver for all cycles of Krylov-Schur; "Total Matvecs" denotes the total number of matrix-vector products computed for all cycles of the eigensolver; and "Avg Iter" denotes the average number of iterations per matrix-vector product. Note that the first five matrix-vector products highlighted are computed during the expansion phase of the Krylov-Schur method as in (3.6).

Matvec	$\mathcal{A}$	$\mathcal{A}_{\tau_1}$	$\mathcal{A}_{\tau_2}$	$\mathcal{A}_{\tau_3}$
1	26	25	41	37
2	7	8	6	6
3	8	7	5	6
4	7	6	5	6
5	7	232	5	6
Total Iters	620	1322	504	501
Total Matvecs	70	70	70	70
Avg Iter	8.86	18.89	7.2	7.16

Table 3.3: Total number of GCRO-DR iterations for each matrix-vector product in Krylov-Schur for $\mathcal{A}$ and warm-start Krylov-Schur for $\mathcal{A}_{\tau_i}$, with the first five matvecs highlighted. "Total Iters" denotes the total number of GCRO-DR iterations for all calls to the iterative solver for all cycles of Krylov-Schur; "Total Matvecs" denotes the total number of matrix-vector products computed for all cycles of the eigensolver; and "Avg Iter" denotes the average number of iterations per matrix-vector product. Note that the first five matrix-vector products highlighted are computed as in (3.15) when we build the approximate Krylov-Schur decomposition with minimum norm residual.

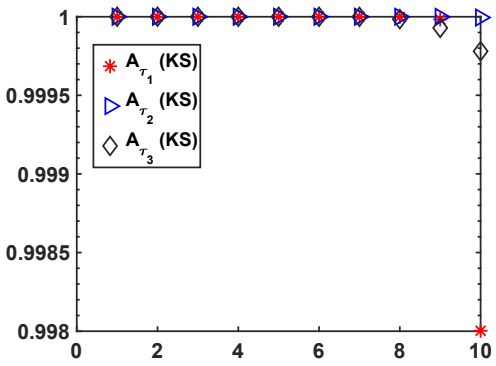


(a) Cosines of the canonical angles between the approximate invariant subspaces computed using the current method and Krylov-Schur with GCRO-DR.

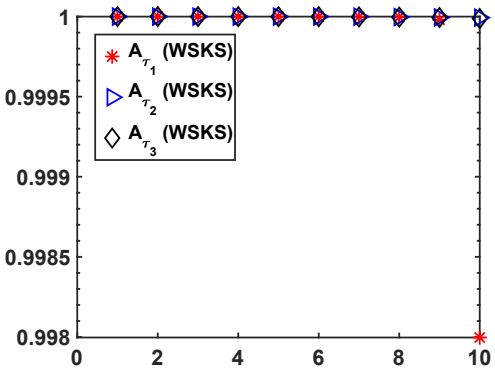


(b) Cosines of the canonical angles between the approximate invariant subspaces computed using the current method and warm-start Krylov-Schur with GCRO-DR.

Figure 3.2: Comparing the quality of the invariant subspace computed for $numev = 50$.



(a) Cosines of the canonical angles between the approximate invariant subspaces computed corresponding to the ten smallest approximate eigenvalues using the current method and Krylov-Schur with GCRO-DR for the shifted systems, $\mathcal{A}_{\tau_i}$.



(b) Cosines of the canonical angles between the approximate invariant subspaces computed corresponding to the ten smallest approximate eigenvalues using the current method and warm-start Krylov-Schur with GCRO-DR for the shifted systems, $\mathcal{A}_{\tau_i}$

Figure 3.3: Comparing the quality of the invariant subspace computed for $numev = 10$.

	LU/KS	LU/WSKS	GMRES/ KS	GCRO-DR/ KS	GCRO-DR/ WSKS
Total Iterative Solver Iters	–	–	599368	145915	88049
Total KS/ WSKS Iters	314	265	519	608	294

Table 3.4: Summary of the total iterative solver iterations and eigensolver iterations when applying (from left to right) the strategies (1)-(5) for the full sequence.

	It. Solver Iters			KS/WSKS Iters		
Matrix $\mathcal{A}_{j_{\tau_i}}$	GMRES/ KS	GCRO-DR/ KS	GCRO-DR/ WSKS	GMRES/ KS	GCRO-DR/ KS	GCRO-DR/ WSKS
$\mathcal{A}_1$	6787	2486	2503	9	8	8
$\mathcal{A}_{1_{\tau_1}}$	12060	923	395	11	2	1
$\mathcal{A}_{1_{\tau_2}}$	9045	2106	2147	7	6	5
$\mathcal{A}_2$	6222	2499	2505	8	8	8
$\mathcal{A}_{2_{\tau_1}}$	3819	4202	279	2	2	3
$\mathcal{A}_{2_{\tau_2}}$	4020	716	416	2	4	1
$\mathcal{A}_{2_{\tau_3}}$	8040	768	651	6	6	5

Table 3.5: Comparison of iterations for seven select matrices when using strategies (3)-(5).

We also vary two parameters: the GCRO-DR convergence tolerance and the Warm-Start Krylov-Schur convergence tolerance (see Table 3.6). All other parameters are as they are listed earlier in this section. When using a larger convergence tolerance (10^{-7}) for the iterative solver, GMRES and GCRO-DR iterations decrease compared with using the smaller convergence tolerance (10^{-10}). When using a larger convergence tolerance (10^{-3}) for Warm-Start Krylov-Schur, we use the smaller convergence tolerance for GCRO-DR (10^{-10}). Details are provided in Table 3.6.

Industrial Model

Next, we consider an industrial model from [76] with $n = 842638$. Due to the size of these matrices, we only test strategies (4) and (5) compared with the current method for two eigenvalue problems corresponding to matrices we denote as $\mathcal{A}$ and $\mathcal{A}_{\tau_1}$, with $\Omega = 4\pi$. For the tests shown, we consider only one unstable eigenvalue and hence one shifted matrix, but we note that there are three unstable eigenvalues of $\mathcal{A}$.

	Larger GCRO-DR Conv Tol		Larger WSKS Conv Tol
	GCRO-DR/ KS	**GCRO-DR/ WSKS**	**GCRO-DR/ WSKS**
Total Iterative Solver Iters	53424	53157	95756
Total KS/ WSKS Iters	239	198	195

Table 3.6: Summary of the total iterations when using (from left to right) a larger convergence tolerance for GCRO-DR (with the smaller convergence tolerance for WSKS) and a larger convergence tolerance for WSKS (with the smaller convergence tolerance for GCRO-DR).

For these matrices, we must compute a very accurate preconditioner. In Figure 3.4, we see that of the 100 smallest (in magnitude) eigenvalues of $\mathbf{K}$, quite a few are clustered close to the origin. Specifically, we use MATLAB's® `ilu` function, with type 'ilutp' and drop tolerance 10^{-8}. For GCRO-DR, we consider two different convergence tolerances: 10^{-7} and 10^{-10}. We refer to these respectively as the *large convergence tolerance* and the *small convergence tolerance*. We set the restart to 200 and the recycle dimension to 100. Finally, we warm-start the eigensolver for $\mathcal{A}_{\tau_1}$ using the approximate invariant subspace computed for $\mathcal{A}$.

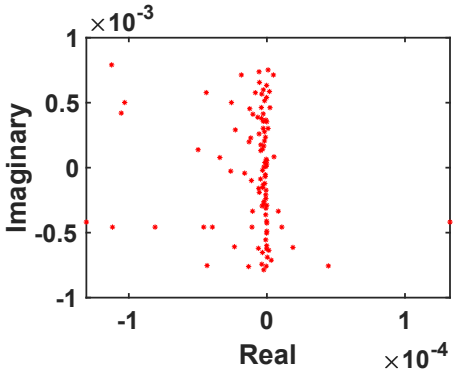


Figure 3.4: The 100 smallest (in magnitude) eigenvalues of $\mathbf{K}_{\tau_i}$ for the industrial model.

Table 3.7 shows the GCRO-DR iterations when solving the eigenvalue problems for $\mathcal{A}$ and $\mathcal{A}_{\tau_1}$ for $numev = 25$. In these tests, we use the smaller convergence tolerance for the iterative solver. In particular, we see that total iterations are reduced when using warm-start Krylov-

	$\mathcal{A}$ using KS/GCRO-DR	$\mathcal{A}_{\tau_1}$ using KS/GCRO-DR	$\mathcal{A}_{\tau_1}$ using WSKS/GCRO-DR
Total GCRO-DR Iterations	1549	1087	1011
Total Matvecs in Eigensolver	200	200	174
Average Iterations per Matvec	7.75	5.44	5.81

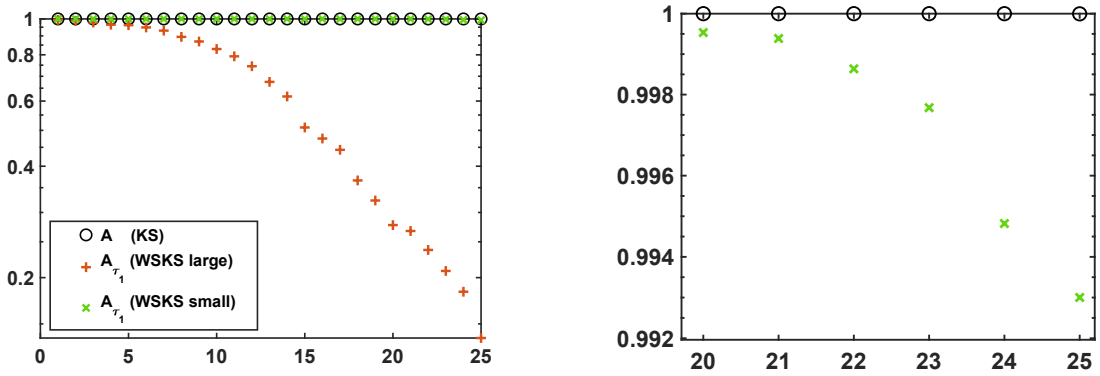
Table 3.7: Total GCRO-DR iterations for each matrix-vector product in the eigensolver, with the first five matvecs highlighted. "Total GCRO-DR Iterations" denotes the total number of GCRO-DR iterations for all calls to the iterative solver for all cycles of the eigensolver; "Total Matvecs in Eigensolver" denotes the total number of matrix-vector products required; and "Average Iterations" denotes the average number of iterations per matrix-vector product. We use the smaller convergence tolerance, 10^{-10}, for GCRO-DR.

Schur compared with Krylov-Schur for $\mathcal{A}_{\tau_1}$, though the difference is not nearly as dramatic as with the academic model. However, we note that we are only considering two matrices of the full sequence, and we expect that when applying warm-start Krylov-Schur to the full sequence, we can reduce total iterations. Continued work with the industrial model is future work.

In Figure 3.5(a), we highlight the effects of a large and small convergence tolerance on the quality of the approximate invariant subspace computed. In particular, when we use a smaller convergence tolerance, we have a significantly more accurate approximate invariant subspace. Figure 3.5(b) gives an enlarged picture of the canonical angles for the last five angles.

3.6 Conclusions

We introduce a warm-start Krylov-Schur algorithm for a sequence of eigenvalue problems. We show that for an application in the numerical simulation of brake squeal, this method is effective in reducing the total number of matrix-vector products in the eigensolver, and produces an approximate invariant subspace that is as accurate as Krylov-Schur. Where the matrix-vector products require an implicit linear solve, as is the case with GEPs rewritten as SEPs, we also show that warm-starting the eigensolver using the approximate invariant subspace computed for one matrix for another in the sequence reduces total iterations when using using a recycling Krylov method, such as GCRO-DR, compared with GMRES. These recycling strategies produce approximate invariant subspaces that are nearly as accurate as



(a) (b)

Figure 3.5: Cosines of the canonical angles between the approximate invariant subspace computed when using the current method and when using warm-start Krylov-Schur with the large and small GCRO-DR convergence tolerances ("WSKS large" and "WSKS small", respectively) for $\mathcal{A}_{\tau_1}$ and when using Krylov-Schur for $\mathcal{A}$. Figure 3.5(b) enlarges the plot for the cosines of the largest five canonical angles for $\mathcal{A}$ and $\mathcal{A}_{\tau_1}$ when using the smaller GCRO-DR convergence tolerance.

an exact solve (i.e., MATLAB's® `eigs` and the exact LU decomposition), and we show this using two sequences of matrices arising in the numerical simulation of brake squeal [76].

3.7 Acknowledgements

We would like to thank Volker Mehrmann for sharing the brake squeal matrices and corresponding POD code considered in this chapter with us, as well as for providing counsel on several occasions as we incorporated our methods into this code.

Chapter 4

An Analysis of Low-Rank and Small-in-Norm Perturbations of the Identity Matrix

In many applications, sequences of linear systems arise where the coefficient matrix takes the form $\mathbf{I}+\mathbf{L}+\mathbf{S}$ (where $\mathbf{I}$ is the identity matrix of dimension n, $\text{rank}(\mathbf{L}) = p \ll n$, and $\|\mathbf{S}\|$ is small). GMRES convergence rates for matrices of the forms $\mathbf{I}+\mathbf{L}$ and $\mathbf{I}+\mathbf{S}$ are well known, but only relatively weak convergence bounds specific to matrices of the form $\mathbf{I}+\mathbf{L}+\mathbf{S}$ exist. In this chapter, we explore the convergence properties of such matrices by considering the pseudospectrum of $\mathbf{I}+\mathbf{L}$, and in particular, we identify those eigenvalues of $\mathbf{I}+\mathbf{L}$ that are sensitive to the introduction of $\mathbf{S}$.

4.1 Introduction

In this chapter, we focus on matrices of the form

$$\mathbf{I}+\mathbf{L}+\mathbf{S}, \tag{4.1}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix, $\text{rank}(\mathbf{L}) = p \ll n$, and $\|\mathbf{S}\|_2 = \epsilon < 1$. In particular, we are interested in applications that involve a sequence of linear systems of the form

$$(\mathbf{I}+\mathbf{L}_k+\mathbf{S}_k)\mathbf{x}_k = \mathbf{b}_k, \tag{4.2}$$

where the right hand side(s) may or may not change, and we explore the convergence behavior of GMRES [122] when solving such a sequence.

This chapter is outlined as follows. In Section 4.2, we analyze GMRES convergence when the coefficient matrix takes the form (4.1) by identifying the sources of sensitivity in matrices

of the form $\mathbf{I} + \mathbf{L}$. In Section 4.3, we highlight a sequence of linear systems of the form (4.2) arising in a Quantum Monte Carlo (QMC) method [4] when reusing a very good initial preconditioner. Finally, in Section 4.4 we provide conclusions.

Throughout this chapter we refer to $\mathbf{L}$ as *low rank* and $\mathbf{S}$ as *small in norm*. We denote $R(\mathbf{A})$ to be the range of an $n \times n$ matrix $\mathbf{A}$, and $N(\mathbf{A})$ to be the null space of $\mathbf{A}$. For all pseudospectra plots shown in this chapter, we use EigTool [146].

4.2 GMRES Convergence for Perturbed Identity Plus Low-Rank Coefficient Matrices

Convergence bounds for small in norm perturbations of the identity matrix, and low rank perturbations of the identity matrix are well known and are given here for convenience.

Theorem 4.1. *Let* $\mathbf{I}, \mathbf{S} \in \mathbb{C}^{n\times n}$ *and* $\mathbf{A} = \mathbf{I} + \mathbf{S}$*, where* $\mathbf{I}$ *denotes the identity and* $\|\mathbf{S}\|_2 \leq \epsilon < 1$*. Let* $\mathbf{r}_m = \mathbf{b} - \mathbf{A}\mathbf{x}_m$ *be the GMRES residual, with* $\mathbf{x}_m \in \mathcal{K}^m(\mathbf{A}; \mathbf{r}_0) = \text{Span}(\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0\})$*. Then* $\|\mathbf{r}_m\|_2 \leq \epsilon^m \|\mathbf{r}_0\|_2$*.*

Proof. We refer to [45, 73] for a proof of this theorem and related discussion. □

In practice n is very large, and so we are interested in fast convergence where $\|\mathbf{r}_m\|_2 \leq \tau$, for $m \ll n$. Here, m is a modest number of GMRES iterations, and τ is the prescribed convergence tolerance. Clearly, in Theorem 4.1, the bound guarantees faster convergence for smaller ϵ. However, ϵ need not be very small in order for GMRES to converge rapidly (consider, for example, $\epsilon = 1/3$).

The next theorem is given in [62] and bounds the convergence of GMRES for matrices that can be expressed as a low-rank perturbation of the identity.

Theorem 4.2. *Let* $\mathbf{I}, \mathbf{L} \in \mathbb{C}^{n\times n}$ *and* $\mathbf{A} = \mathbf{I} + \mathbf{L}$*, where* $\mathbf{I}$ *denotes the identity and rank*$(\mathbf{L}) = p < n$*. Then, GMRES will converge in at most* $p + 1$ *iterations*

Proof. A direct proof can be found in [62]. We also refer the reader to Proposition 2 in [122], noting that in this case the minimum polynomial has degree $1 + p$. □

As we are interested in $p \ll n$, Theorem 4.2 guarantees us very fast convergence in this case.

Some theoretical work has been introduced to describe the convergence behavior of GMRES applied to linear systems with coefficient matrices of the form (4.1). In [96], it is shown that there exists $C > 0$ such that the GMRES residual of (4.1) can be bounded as

$$\|\mathbf{r}_{m(p+1)}\|_2 \leq C^m \|\mathbf{S}\|_2^m, \tag{4.3}$$

for $m > 0$. In particular, for $p(z) = 1 + \sum_{k=1}^{p+1} \beta_k z^k$,

$$C = \sum_{k=1}^{p+1} k|\beta_k| \left\| \mathbf{I} + \mathbf{L} \right\|_2^{k-1} + O\left(\left\| \mathbf{S} \right\|_2^2 \right),$$

satisfies (4.3) [96]. In cases when $\|\mathbf{S}\|_2$ is much less than τ (the GMRES convergence tolerance), GMRES tends to converge in $p+1$ iterations as noted in [96]. However, this severely limits the size of the perturbation to $\mathbf{I} + \mathbf{L}$ we can consider.

Instead, we examine how $\|\mathbf{S}\|_2$ affects the eigenvalues of $\mathbf{A} + \mathbf{S}$, where

$$\mathbf{A} = \mathbf{I} + \mathbf{L}. \tag{4.4}$$

Specifically, we consider the pseudospectrum of (4.4) as guided by the theoretical framework of [125]. Using spectral perturbation theory applied to the resolvents of a general coefficient matrix, $\mathbf{G}$, and the perturbation, $\mathbf{G} + \mathbf{S}$, the norm of the GMRES residual of $\mathbf{G}$ increases by at most $O(\epsilon)$, regardless of the magnitude of the change to the eigenvalues of $\mathbf{G}$ [125]. In this chapter, we are interested in the convergence behavior of GMRES when $\mathbf{G}$ takes the special form of (4.1). We refer to (4.4) as the *unperturbed matrix* and to (4.1) as the *perturbed matrix*. We refer to the linear systems $(\mathbf{I} + \mathbf{L})\mathbf{x} = \mathbf{b}$ and $(\mathbf{I} + \mathbf{L} + \mathbf{S})\mathbf{x} = \mathbf{b}$ as the *unperturbed and perturbed systems*, respectively.

First, define the δ-pseudospectrum for a general coefficient matrix $\mathbf{G}$ as given in [125]:

$$\sigma_\delta(\mathbf{G}) = \left\{ z \in \mathbb{C} \;\middle|\; \left\| (z\mathbf{I} - \mathbf{G})^{-1} \right\|_2 > 1/\delta \right\}. \tag{4.5}$$

Then the spectrum of $\mathbf{G}$, $\sigma(\mathbf{G})$, is contained in $\sigma_\delta(\mathbf{G})$ for all $\delta > 0$ and $\partial\sigma_\delta(\mathbf{G})$ encloses the region containing all eigenvalues of $\mathbf{G}$. When $\delta > \epsilon$, $\partial\sigma_\delta(\mathbf{G})$ encloses the region containing all eigenvalues of $\mathbf{G} + \mathbf{S}$ [125]. The smallest such δ, or δ_0, such that $\partial\sigma_{\delta_0}(\mathbf{G})$ passes through the origin gives the largest pseudospectrum of $\mathbf{G}$, $\sigma_{\delta_0}(\mathbf{G})$, that does not contain the origin. Substituting (4.4) into (4.5), $\sigma_\delta(\mathbf{I} + \mathbf{L})$ is

$$\sigma_\delta(\mathbf{I} + \mathbf{L}) = \left\{ z \in \mathbb{C} \;\middle|\; \left\| ((z-1)\mathbf{I} - \mathbf{L})^{-1} \right\|_2 > 1/\delta \right\}, \tag{4.6}$$

and it follows that $\sigma_\delta(\mathbf{I} + \mathbf{L}) = 1 + \sigma_\delta(\mathbf{L})$.

Denote $\mathbf{r}_m$ as the GMRES residual when solving $(\mathbf{G} + \mathbf{S})\mathbf{x} = \mathbf{b}$, and $\boldsymbol{\rho}_m$ as the GMRES residual when solving $\mathbf{G}\mathbf{x} = \mathbf{b}$.Then, [125, Corollary 2.2] bounds $\|\mathbf{r}_m\|_2$ as

$$\|\mathbf{r}_m\|_2 \leq \|\boldsymbol{\rho}_m\|_2 + \epsilon C_m, \tag{4.7}$$

which is satisfied for

$$C_m = \left(\frac{L_\delta \|\mathbf{b}\|_2}{\pi\delta^2} \right) \sup_{z \in \sigma_\delta(\mathbf{G})} |p_m(z)|. \tag{4.8}$$

$\boldsymbol{\delta}$	10^{-1}	10^{-2}	10^{-3}
$\mathbf{C_3}$	36.85	99.98	1048.19

Table 4.1: Values of C_3 for different δ for $\mathbf{I}+\mathbf{L}$ defined below.

Here, L_δ is the arc length of $\partial\sigma_\delta(\mathbf{G})$ for $\delta > \epsilon$, and $p_m(z)$ is the residual polynomial when applying GMRES to $\mathbf{Gx} = \mathbf{b}$ such that $p_m(0) = 1$ [125]. Replacing $\mathbf{G}$ with (4.4), Theorem 4.2 guarantees that $\|\boldsymbol{\rho}_{p+1}\|_2 = 0$, and so $\|\mathbf{r}_{p+1}\|_2$ is bounded as

$$\|\mathbf{r}_{p+1}\|_2 < \epsilon C_{p+1}. \tag{4.9}$$

Consider an example of a matrix of the form (4.4), with $n = 25$, $\text{rank}(\mathbf{L}) = 2$, and a clustered spectrum away from the origin; see Figure 4.1 for the pseudospectrum of this matrix. In Figure 4.2, we show $\|\mathbf{r}_m\|_2$, $\|\boldsymbol{\rho}_m\|_2$, and the upper bound (4.7) on $\|\mathbf{r}_m\|_2$ for different δ. We choose perturbations of sizes $\epsilon = 10^{-4}$ and $\epsilon = 10^{-5}$, and three values of $\delta \in (\epsilon, 1/\|\mathbf{A}^{-1}\|_2)$, $\delta = 10^{-1}, 10^{-2}, 10^{-3}$. To compute the bound as in (4.7), we replace $\sigma_\delta(\mathbf{A})$ with a slightly larger circle (as is done in [125]), and compute $p_m(z)$ using the harmonic Ritz values of the unperturbed system. We use full GMRES and set the convergence tolerance to 10^{-10}. The unperturbed system converges in three iterations, as guaranteed by Theorem 4.2. For $\|\mathbf{S}\|_2 = 10^{-5}$, the perturbed system converges in four iterations, and for $\|\mathbf{S}\|_2 = 10^{-4}$, it converges in five iterations.

In Table 4.1, we give the values of $C_{p+1} = C_3$ for $\delta = 10^{-1}, 10^{-2}, 10^{-3}$. While these values are much larger than any reasonable τ we would choose for the GMRES tolerance, we make two observations. (1) For matrices of the form (4.4) with tightly clustered eigenvalues away from the origin that are robust to perturbations, L_δ will be smaller for larger values of δ. Thus, C_{p+1} may be much smaller in these cases. Though, note that this is not guaranteed since C_{p+1} also depends on the supremum of $|p_{p+1}(z)|$ and $\|\mathbf{b}\|_2$.[1] (2) While δ must be chosen such that $\delta > \epsilon$, (4.8) itself does not depend directly on ϵ. So, very small perturbations of (4.4) will give small values of ϵC_{p+1} (e.g., consider $\epsilon = 10^{-6}$ for the values of C_3 when $\delta = 10^{-1}, 10^{-2}$). In cases where $\epsilon < \tau/C_{p+1}$, then we can guarantee our perturbed system will converge in $p+1$ steps. This formalizes the observation made in [96], but still restricts the size of $\|\mathbf{S}\|_2$ we can consider. Ongoing work is focused on convergence behavior when $\tau < \epsilon C_{p+1} \ll 1$.

While a clustered spectrum away from the origin is often a good indicator of fast GMRES convergence, convergence may slow when some of those eigenvalues are ill-conditioned. Next,

[1] Even in cases where the pseudospectrum does not contain the origin, $|p_m(z)|$ may be larger than one. See [125, Figure 3.2] for an example.

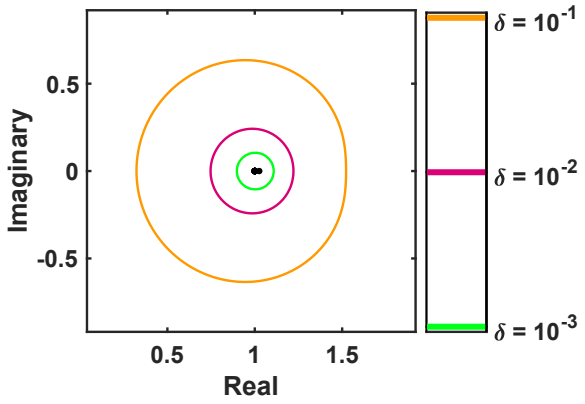
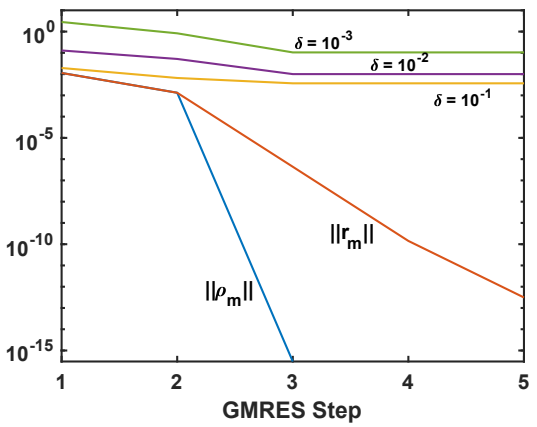
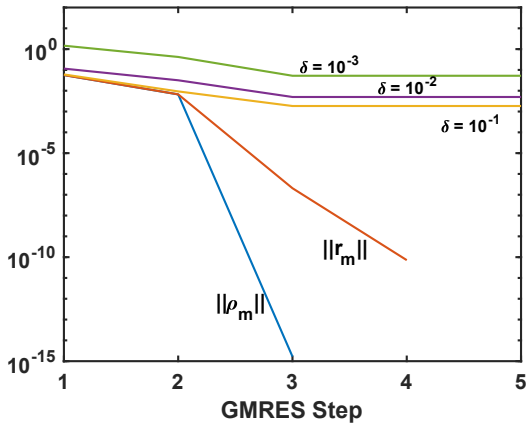


Figure 4.1: Pseudospectrum of a rank-two matrix with dimension $n = 25$. The color-bar on the right gives the δ value for the corresponding $\sigma_\delta(\mathbf{I}+\mathbf{L})$ shown on the left. The black dots indicate the eigenvalues of $\mathbf{I}+\mathbf{L}$.



(a) GMRES convergence for $\mathbf{I}+\mathbf{L}$ (blue line) and $\mathbf{I}+\mathbf{L}+\mathbf{S}$ (red line) with $\|\mathbf{S}\|_2 = 10^{-4}$, bounded above by (4.7) for different values of δ.



(b) GMRES convergence for $\mathbf{I}+\mathbf{L}$ (blue line) and $\mathbf{I}+\mathbf{L}+\mathbf{S}$ (red line) with $\|\mathbf{S}\|_2 = 10^{-5}$, bounded above by (4.7) for different values of δ.

Figure 4.2: $\|\mathbf{r}_m\|_2$, $\|\boldsymbol{\rho}_m\|_2$, and the upper bound (4.7) on $\|\mathbf{r}_m\|_2$ for $\delta = 10^{-1}, 10^{-2}, 10^{-3}$, corresponding to $\sigma_\delta(\mathbf{I}+\mathbf{L})$ in Figure 4.1.

we show that there are at most $2p$ eigenvalues of (4.4) that are sensitive to the introduction of $\mathbf{S}$. For simplicity, we consider complex matrices in our analysis.

First, consider the singular value decomposition (SVD) of $\mathbf{L}$,

$$\mathbf{L} = \left[\begin{array}{c|c} \mathbf{U}_1 & \mathbf{U}_2 \end{array}\right] \left[\begin{array}{c|c} \boldsymbol{\Sigma}_1 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right] \left[\begin{array}{c} \mathbf{V}_1^* \\ \hline \mathbf{V}_2^* \end{array}\right] = \mathbf{U}_1 \boldsymbol{\Sigma}_1 \mathbf{V}_1^*. \tag{4.10}$$

For $\mathbf{A}$ as in (4.4) and the SVD of $\mathbf{L}$ as in (4.10), we can write the eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \iff \mathbf{x} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{x} = \lambda\mathbf{x} \iff \mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{x} = (\lambda - 1)\mathbf{x}, \tag{4.11}$$

and we describe the eigenpairs $(\lambda, \mathbf{x})$ of (4.11) next.

Theorem 4.3. $(1, \mathbf{x})$ *is an eigenpair of* $\mathbf{A} = \mathbf{I} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*$ *if and only if* $\mathbf{x} \in R(\mathbf{V}_2)$.

Proof. Let $(1, \mathbf{x})$ be an eigenpair of $\mathbf{A} = \mathbf{I} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*$. Then,

$$\mathbf{x} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{x} = \lambda\mathbf{x} \Rightarrow \mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{x} = \mathbf{0}.$$

Then, $\mathbf{x} \in N(\mathbf{V}_1^*) = R(\mathbf{V}_2)$. Now assume $\mathbf{x} \in N(\mathbf{V}_1^*) = R(\mathbf{V}_2)$, and so $\mathbf{V}_1^*\mathbf{x} = \mathbf{0}$. Equation (4.11) becomes $(\lambda - 1)\mathbf{x} = \mathbf{0}$ and it follows that $\lambda = 1$. So, $(1, \mathbf{x})$ is an eigenpair of $\mathbf{A} = \mathbf{I} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*$. □

When $\lambda \neq 1$, it follows that the corresponding eigenvector $\mathbf{x} \in R(\mathbf{U}_1)$. In this case, we need only analyze the eigenpairs of the small matrix $\Sigma_1\mathbf{V}_1^*\mathbf{U}_1$, which we formalize in the following theorem.

Theorem 4.4. $\mathbf{x} = \mathbf{U}_1\xi$ *is an eigenvector of* $\mathbf{A} = \mathbf{I} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*$ *if and only if* ξ *is an eigenvector of* $\Sigma_1\mathbf{V}_1^*\mathbf{U}_1$.

Proof. Assume $\mathbf{x} = \mathbf{U}_1\xi$ is an eigenvector of $\mathbf{A} = \mathbf{I} + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*$. Then by (4.11)

$$\mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{U}_1\xi = \gamma\mathbf{U}_1\xi,$$

for $\gamma = \lambda - 1$. Left multiplying by $\mathbf{U}_1^*$ gives

$$\Sigma_1\mathbf{V}_1^*\mathbf{U}_1\xi = \gamma\xi,$$

and ξ is an eigenvector of $\Sigma_1\mathbf{V}_1^*\mathbf{U}_1$ with corresponding eigenvalue $\gamma = \lambda - 1$.

Now assume ξ is an eigenvector of $\Sigma_1\mathbf{V}_1^*\mathbf{U}_1$ with corresponding eigenvalue γ. Then

$$\Sigma_1\mathbf{V}_1^*\mathbf{U}_1\xi = \gamma\xi,$$

and left multiplying by $\mathbf{U}_1$ gives

$$\mathbf{U}_1\Sigma_1\mathbf{V}_1^*(\mathbf{U}_1\xi) = \gamma(\mathbf{U}_1\xi),$$

which is equivalent to (4.11) with $\gamma = \lambda - 1$ and $\mathbf{x} = \mathbf{U}_1\xi$. □

If $\gamma \neq 0$ and $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$ is diagonalizable, then

$$\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1 \xi_j = \xi_j \gamma_j, \tag{4.12}$$

for $j = 1, 2, \ldots, p$, and we characterize the eigenpairs of (4.4) as

$$\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{U}_1\xi_j = \mathbf{U}_1\xi_j + \mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{U}_1\xi_j = \mathbf{U}_1\xi_j(1+\gamma_j). \tag{4.13}$$

For (4.12), for the diagonalizable case, we must consider those potentially sensitive eigenvalues of (4.4) arising from the small angles among the eigenvectors of $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$. We do not consider the case when $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$ is not diagonalizable in detail, but note that the resulting defective eigenvalues are sensitive to perturbations [130]. Combining both the diagonalizable and nondiagonalizable cases, this gives p possibly sensitive eigenvalues of (4.4) from the corresponding p eigenvalues of $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$.

When $\gamma = 0$, eigenvectors $\mathbf{x} = \mathbf{U}_1\xi$ have corresponding eigenvalue $\lambda = 1$, and so $R(\mathbf{U}_1)$ and $R(\mathbf{V}_2)$ intersect nontrivially. Consider $\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta$:

$$(\mathbf{A} - \lambda\mathbf{I})(\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta) = (\mathbf{A} - \mathbf{I})(\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta)$$

$$= \mathbf{U}_1\Sigma_1\mathbf{V}_1^*(\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta) = \mathbf{U}_1\xi \neq \mathbf{0},$$

and

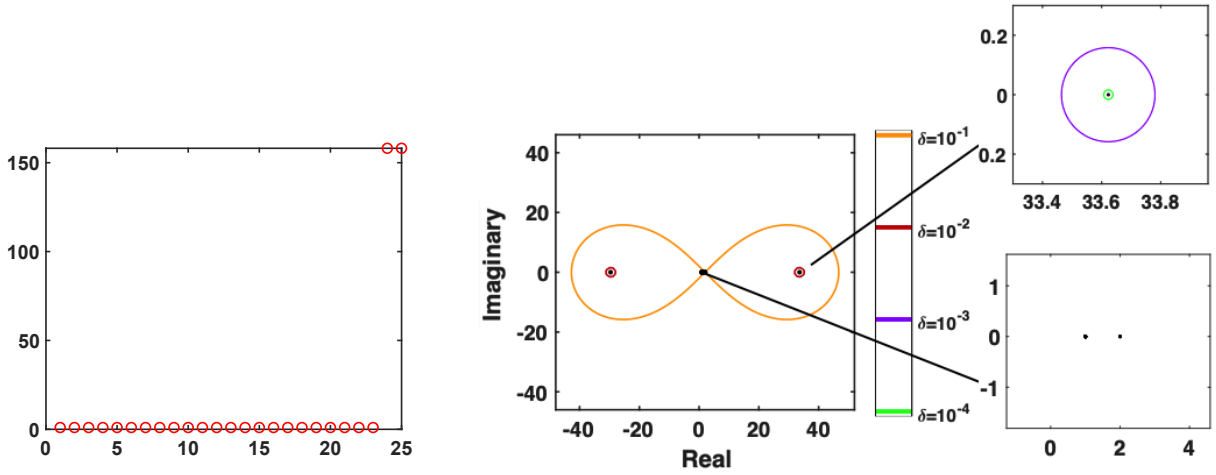
$$(\mathbf{A} - \lambda\mathbf{I})^2(\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta) = (\mathbf{A} - \mathbf{I})^2(\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta) = \mathbf{U}_1\Sigma_1\mathbf{V}_1^*\mathbf{U}_1\xi = \mathbf{0}.$$

So, $\mathbf{V}_1\Sigma^{-1}\xi + \mathbf{V}_2\zeta$ is a generalized eigenvector of order two, and at least one $\lambda = 1$ is defective. Then, we have $n-p$ eigenvectors from $R(\mathbf{V}_2)$, but only $p-1$ (and potentially fewer) eigenvectors from $R(\mathbf{U}_1)$. Again, such defective eigenvalues are sensitive to perturbations.

The other p potentially sensitive eigenvalues of (4.4) arise from small angles between eigenvectors of the forms $\mathbf{V}_2\zeta$ (corresponding to eigenvalue $\lambda = 1$) and $\mathbf{U}_1\xi$ (corresponding to eigenvalue $\lambda \neq 1$), as well as when $\gamma = 0$, described above. In particular, if the angle between $\mathbf{U}_1\xi$ and $\mathbf{V}_2\mathbf{V}_2^*\mathbf{U}_1\xi$ (the projection of $\mathbf{U}_1\xi$ on $R(\mathbf{V}_2)$) is small, this corresponds to a small angle between two right eigenvectors with distinct eigenvalues.

We have developed a prescription for identifying the sensitive eigenvalues of matrices taking the form (4.4): (1) the p potentially sensitive eigenvalues of $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$; and (2) the small angles between eigenvectors of the form $\mathbf{V}_2\zeta$ and $\mathbf{U}_1\xi$, which includes the case when $\Sigma_1\mathbf{V}_1^*\mathbf{U}_1\xi = 0$. Next, we give two small examples, each highlighting how these sources of sensitivity can affect the convergence of GMRES.

First, to visualize the effect of the sensitive eigenvalues of the small eigenvalue problem (4.12) on the GMRES convergence of systems with the coefficient matrix (4.4), we consider an example such that (4.12) has two ill-conditioned eigenvalues. We let the matrix dimension size be $n = 25$ and $\text{rank}(\mathbf{L}) = 5$. We see in Figure 4.3(a) that the resulting matrix has



(a) Condition number of the eigenvalues of $\mathbf{I}+\mathbf{L}$.

(b) Pseudospectrum of $\mathbf{I}+\mathbf{L}$. The color-bar gives the δ value for the corresponding $\sigma_\delta(\mathbf{I}+\mathbf{L})$. The black dots indicate the eigenvalues of $\mathbf{I}+\mathbf{L}$.

Figure 4.3: $\mathbf{I}+\mathbf{L}$ with dimension $n=25$, $\text{rank}(\mathbf{L})=5$, and two ill-conditioned eigenvalues of $\Sigma_1\mathbf{V}_1^*\mathbf{U}_1$.

two sensitive eigenvalues (i.e., those shown with large condition number). In particular, the sensitive eigenvalues of $\mathbf{L}$ are $\gamma_1 = 32.62$ and $\gamma_2 = -30.62$, and so the corresponding sensitive eigenvalues of $\mathbf{A}$ are $\lambda_1 = 33.62$ and $\lambda_2 = -29.62$. The pseudospectrum of $\mathbf{A}$ is given in Figure 4.3(b).

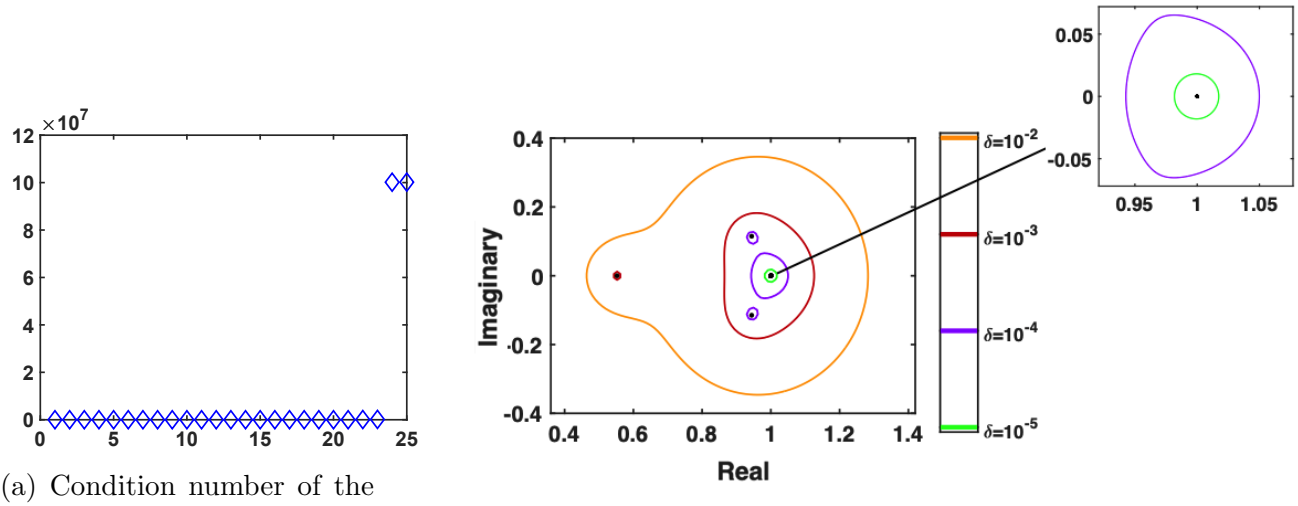
We choose the perturbations $\mathbf{S}_1$ and $\mathbf{S}_2$ such that $\|\mathbf{S}_1\|_2 = 10^{-3}$ and $\|\mathbf{S}_2\|_2 = 10^{-1}$ and consider GMRES iterations for $\mathbf{A}_1 = \mathbf{I}+\mathbf{L}+\mathbf{S}_1$ and $\mathbf{A}_2 = \mathbf{I}+\mathbf{L}+\mathbf{S}_2$. For the unperturbed matrix, GMRES converges in 4 iterations. For $\mathbf{A}_1$, GMRES converges in 7 iterations, and for $\mathbf{A}_2$, iterations further increase to 14. In fact, even for a very small perturbation $\mathbf{S}_3$ with $\|\mathbf{S}_3\|_2 = 10^{-6}$, GMRES converges in 6 iterations.

Next, we analyze a matrix with very small angles between the subspaces spanned by the columns of $\mathbf{U}_1$ and the columns of $\mathbf{V}_2$. Specifically, we let the matrix dimension size be $n = 25$ and let $\text{rank}(\mathbf{L}) = 5$. We construct $\mathbf{L}$ such that there are two small angles, $\theta_{1,2} = 10^{-5}$, between $\mathbf{U}_1$ and $\mathbf{V}_2$. We see in Figure 4.4(a) that the unperturbed matrix we construct has two sensitive eigenvalues. The pseudospectrum is given in Figure 4.4(b).

We choose $\mathbf{S}_1$ and $\mathbf{S}_2$ such that $\|\mathbf{S}_1\|_2 = 10^{-5}$ and $\|\mathbf{S}_2\|_2 = 10^{-2}$ and consider GMRES iterations for $\mathbf{A}_1 = \mathbf{I}+\mathbf{L}+\mathbf{S}_1$ and $\mathbf{A}_2 = \mathbf{I}+\mathbf{L}+\mathbf{S}_2$. For the unperturbed matrix (4.4), GMRES converges in 6 iterations, as guaranteed by Theorem 4.2. For $\mathbf{A}_1$, iterations increase slightly to 7, and for $\mathbf{A}_2$ iterations increase to 10.

There are several future directions we intend to explore. First is to consider

$$(z\mathbf{I}-\mathbf{L})^{-1}, \tag{4.14}$$



(a) Condition number of the eigenvalues of $\mathbf{I}+\mathbf{L}$.

(b) Pseudospectrum of $\mathbf{I}+\mathbf{L}$. The color-bar gives the δ value for the corresponding $\sigma_\delta(\mathbf{I}+\mathbf{L})$. The black dots indicate the eigenvalues of $\mathbf{I}+\mathbf{L}$.

Figure 4.4: $\mathbf{I}+\mathbf{L}$ with dimension $n = 25$, $\text{rank}(\mathbf{L}) = 5$, and two small angles between $\mathbf{U}_1$ and $\mathbf{V}_2$.

as in (4.6) (where for simplicity, we write z in place of $z-1$). Writing the inverse of (4.14) directly may make the analysis of $\sigma_\delta(\mathbf{I}+\mathbf{L})$ easier. In particular, with $\mathbf{L}$ low rank, define $\mathbf{L} = \mathbf{Y}\mathbf{W}^* = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^*$, where $\mathbf{Y}, \mathbf{W} \in \mathbb{C}^{n\times p}$. Then for $\mathbf{Y} = \mathbf{U}_1$, $\mathbf{W}^* = \mathbf{\Sigma}_1\mathbf{V}_1^*$

$$\mathbf{W}^*\mathbf{Y} = \mathbf{\Sigma}_1\mathbf{V}_1^*\mathbf{U}_1. \tag{4.15}$$

Applying the Sherman–Morrison–Woodbury formula to (4.14), and substituting in (4.15) gives[2]

$$\frac{1}{z}\mathbf{I} - \frac{1}{z^2}\mathbf{Y}(\mathbf{I}+\frac{1}{z}\mathbf{W}^*\mathbf{Y})^{-1}\mathbf{W}^* = \frac{1}{z}\mathbf{I} - \frac{1}{z^2}\mathbf{Y}(\mathbf{I}+\frac{1}{z}\mathbf{\Sigma}_1\mathbf{V}_1^*\mathbf{U}_1)^{-1}\mathbf{W}^*. \tag{4.16}$$

With an explicit formula for (4.14), ongoing work focuses on developing the insight (4.16) gives into $\sigma_\delta(\mathbf{I}+\mathbf{L})$, and in particular C_{p+1} as it relates to the bound (4.9).

An interesting observation comes from considering the left and right eigenvectors of (4.4). In our previous analysis, we characterize the right eigenpairs. When considering the left eigenvectors of (4.4), note that

$$\mathbf{y}^*(\mathbf{I}+\mathbf{U}_1\Sigma_1\mathbf{V}_1^*) = \lambda\mathbf{y}^* \;\Leftrightarrow\; (\mathbf{I}+\mathbf{V}_1\Sigma_1\mathbf{U}_1^*)\mathbf{y} = \lambda\mathbf{y}, \tag{4.17}$$

and applying Theorems 4.3 and 4.4 with $\mathbf{U}_1$ and $\mathbf{V}_1$ switched gives the eigenpairs (1) $\lambda = 1$, $\mathbf{y} \in R(\mathbf{U}_2)$, and (2) $\lambda \neq 1$, $\mathbf{y} \in R(\mathbf{V}_1)$. For a general coefficient matrix, the left eigenvectors

[2]This idea came up during a conversation with Mark Embree in June 2019.

can be characterized as the right eigenvectors of $\mathbf{A}^*$ (as was done in (4.17)). In the special case where the coefficient matrix takes the form (4.4), characterizing the left eigenvectors results in analyzing

$$\Sigma_1 \mathbf{U}_1^* \mathbf{V}_1 \xi_j = \xi_j(\lambda_j - 1), \tag{4.18}$$

for $j = 1, 2, \ldots, p$. We note that $\Sigma_1 \mathbf{U}_1^* \mathbf{V}_1$ and $\Sigma_1 \mathbf{V}_1^* \mathbf{U}_1$ are not transposes of one another, but they are similar. Continuing to develop such properties of the eigenpairs of (4.4) is part of future work.

Finally, when considering the angle between $\mathbf{U}_1\xi$ and $\mathbf{V}_2\mathbf{V}_2^*\mathbf{U}_1\xi$, where we assume $\|\mathbf{U}_1\xi\|_2 = 1$, we can write

$$\frac{(\mathbf{U}_1\xi)^*(\mathbf{V}_2\mathbf{V}_2^*\mathbf{U}_1\xi)}{\|\mathbf{V}_2\mathbf{V}_2^*\mathbf{U}_1\xi\|_2} = \frac{\xi^* \left(\mathbf{V}_2^*\mathbf{U}_1\right)^* \left(\mathbf{V}_2^*\mathbf{U}_1\right) \xi}{\|\mathbf{V}_2\mathbf{V}_2^*\mathbf{U}_1\xi\|_2}.$$

Ongoing work seeks to exploit that $\left(\mathbf{V}_2^*\mathbf{U}_1\right)^* \left(\mathbf{V}_2^*\mathbf{U}_1\right)$ is a symmetric matrix when characterizing small angles between $\mathbf{U}_1$ and $\mathbf{V}_2$.

4.3 Sequences of Matrices of the Form I + L + K

Sequences of matrices of the form (4.1) arise in many applications. For instance, in diffuse optical tomography (DOT), in the Gauss-Newton method for a nonlinear least squares problem, linear systems arise of the form $(\mathbf{A}(\mathbf{p}_j) + i\gamma\mathbf{I})\mathbf{x}_{s,w}^{(j)} = \mathbf{b}_s$ [97]. Here, $\mathbf{A}$ comes from the discretization of the diffusion-absorption equation and $\mathbf{p}_j$ parameterizes the diffusion and absorption 'images' [97]. As the Gauss-Newton iteration progresses and begins to localize the regions of diffusion and absorption, the matrices $\mathbf{A}^{(k)}$ take the form,

$$\mathbf{A}^{(k+1)} = \mathbf{A}^{(k)} + \mathbf{E}_1 + \mathbf{E}_2, \tag{4.19}$$

where $\|\mathbf{E}_1\|_2$ is small and $\|\mathbf{E}_2\|_2$ has low rank [97]. Another such example arises in a QMC method, which can be used to study material properties [4]. In particular, if we compute a very good preconditioner $\mathbf{P}$ for a matrix $\mathbf{A}$, we may assume that $\mathbf{AP}$ is a small-in-norm perturbation of the identity matrix. In the QMC application, we describe how, when computing and reusing a very good initial preconditioner, the sequence of matrices takes the form (4.4).

4.3.1 Slater Matrices for Insulators

In [4], a Markov Chain Monte Carlo (MCMC) algorithm is used to study the behavior of many-body systems using the expectation value of the energy (of the many-body system).

This involves approximating high-dimensional integrals of the form

$$E_V = \frac{\int \Psi_\alpha^*(R) H \Psi_\alpha dR}{\int \Psi_\alpha^*(R) \Psi_\alpha dR} \tag{4.20}$$

by sampling, where for efficiency the algorithm preferentially samples in regions of high probability [4]. Here, R represents the current configuration of all the particles; H is the Hamiltonian of the system; α is the vector of variational parameters over which we minimize (the energy); and Ψ_α is the wave function, which gives the probability of a configuration [4] . The Slater matrices, $\mathbf{A}_k$, considered in this section are sparse matrices whose nonzero entries represent the values of (independent) particle wave functions evaluated at the particle positions [4]. We denote $\mathbf{A}_k = \Psi(R)$. Figure 4.5 shows the sparsity pattern of a representative Slater matrix, where $n = 1024$.

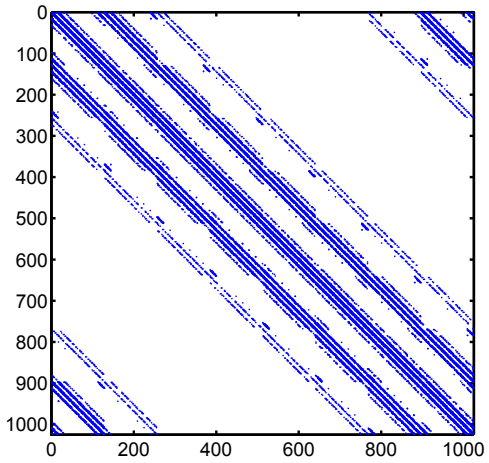


Figure 4.5: Sparsity pattern of a typical Slater matrix for a system with 1024 particles [4].

Importance sampling is done on R (in an MCMC process) and a change to R is proposed to give a new configuration R' [4]. In order to accept or reject the change, the determinant ratio of the two Slater matrices $\mathbf{A}_k = \Psi(R)$ and $\mathbf{A}_{k+1} = \Psi(R')$ corresponding to the k^{th} and $(k+1)^{st}$ Monte Carlo steps,

$$\frac{|\Psi(R')|^2}{|\Psi(R)|^2} = \frac{|\det(\mathbf{A}_{k+1})|^2}{|\det(\mathbf{A}_k)|^2} = |1 + \mathbf{u}_k^T \mathbf{A}_k^{-1} \mathbf{e}_{i_k}|^2, \tag{4.21}$$

must be computed. Here, $\mathbf{u}_k$ is the change in the i_k particle, and $\mathbf{e}_{i_k}$ is the Cartesian basis vector. For each accepted particle change, the resulting Slater matrix changes by only one row in location i_k. So, we can write $\mathbf{A}_{k+1} = \mathbf{A}_k + \mathbf{e}_{i_k} \mathbf{u}_k^T$. If the proposed change is rejected, $\mathbf{A}_{k+1} = \mathbf{A}_k$ [4].

In order to efficiently solve $\mathbf{A}_k^{-1} \mathbf{e}_{i_k}$, an ILUTP preconditioner is computed in [4]. Rather than recomputing a preconditioner for each accepted particle movement, exact updates are

applied to the preconditioner, where the exact update

$$\mathbf{I} - (1 + \mathbf{u}_k^T \mathbf{A}_k^{-1} \mathbf{e}_{i_k})^{-1} \mathbf{A}_k^{-1} \mathbf{e}_{i_k} \mathbf{u}_k^T,$$

comes for free since $\mathbf{u}_k^T \mathbf{A}_k^{-1} \mathbf{e}_{i_k}$ is already computed for the transition probability in the MCMC process [4].

For these Slater matrices, we can take advantage of the (known) row-wise changes by computing a SAM (or even the ideal map) from the left as in (2.18). In this section, however, we consider how reusing a very good initial preconditioner results in a sequence of matrices of the form (4.1), and how the quality of the initial preconditioner affects convergence of GMRES given the size of $\|\mathbf{S}\|_2$.

4.3.2 Reusing Preconditioners for Slater Matrices

In this section, we consider the sequence of Slater matrices for which a particle change is accepted. Recall that in this case $\mathbf{A}_{k+1} = \mathbf{A}_k + \mathbf{e}_{i_k} \mathbf{u}_k^T = \mathbf{A}_k + \mathbf{L}_1$ where $rank(\mathbf{L}_1) = 1$. We can also write

$$\mathbf{A}_{k+1} = \mathbf{A}_1 + \mathbf{L}_k, \tag{4.22}$$

where $\mathbf{L}_k = \sum_{j=1}^{k} \mathbf{e}_{i_j} \mathbf{u}_j^T = \sum_{j=1}^{k} \mathbf{L}_j$ is a matrix with rank at most k.

If we compute a preconditioner for the first matrix in our sequence such that $\mathbf{A}_1 \mathbf{P}_1 = \mathbf{I} + \mathbf{S}$, and then reuse $\mathbf{P}_1$ for all $\mathbf{A}_{k+1}$ ($k \geq 1$), we obtain,

$$\mathbf{A}_{k+1} \mathbf{P}_1 = \mathbf{A}_1 \mathbf{P}_1 + \mathbf{L}_k \mathbf{P}_1 = \mathbf{I} + \mathbf{S} + \mathbf{L}_k \mathbf{P}_k. \tag{4.23}$$

With $\mathbf{L}_k$ at most a rank-k matrix, clearly $\mathbf{L}_k \mathbf{P}_k$ is also (at most) rank-k, and (4.23) takes the form (4.1). For the sequence we consider in this section, there are 62 matrices, and so $\text{rank}(\mathbf{L}_k \mathbf{P}_1) < 62$, which is much less than the matrix dimension $n = 1024$. In the following results, we see the effect the quality of an initial preconditioner has on GMRES convergence when reusing it across the entire sequence of matrices.

In Figure 4.6, we show the results when reusing an ILUTP preconditioner computed using three drop tolerances, 10^{-6}, 10^{-4}, and 10^{-2}. These correspond respectively to perturbations, $\mathbf{S}$, of norm 10^{-3}, 10^{1}, 10^{2}. We note that clearly the last two perturbations are larger than one, and so the form of these matrices do not satisfy the requirement that $\|\mathbf{S}\|_2 < 1$. In particular, the origin is now contained in $\sigma_\delta(\mathbf{A}_{k+1}\mathbf{P}_{k+1})$. However, we include these in our results to highlight that while the analysis in this chapter and that in [125] require $\|\mathbf{S}\|_2 < 1$, the fact that it may be (much) larger than one does not necessarily imply GMRES convergence will suffer. In fact, we see that for $\|\mathbf{S}\|_2 = 10^2$, while iterations are higher than when $\|\mathbf{S}\|_2 = 10^1$ and $\|\mathbf{S}\|_2 = 10^{-3}$, they remain below 120 for all systems in the sequence.

For reference, we also plot the bounds guaranteed by Theorem 4.1 for $\mathbf{I}+\mathbf{S}$ and $\|\mathbf{S}\|_2 = 10^{-3}$, and Theorem 4.2 for $\mathbf{I} + \mathbf{L}_k\mathbf{P}_k$. Neither of these bounds directly applies to matrices of the form (4.1), but for $\|\mathbf{S}\|_2 = 10^{-3}$ iterations remain between these two bounds for most systems in the sequence. In fact, even when $\|\mathbf{S}\|_2 = 10^{1}$, GMRES convergence behavior is nearly the same as when $\|\mathbf{S}\|_2 = 10^{-3}$. This suggests that future development of the theoretical framework focused on the $2p$ potentially sensitive eigenvalues in the context of practical problems such as the QMC application could provide more descriptive bounds for matrices of the form (4.1).

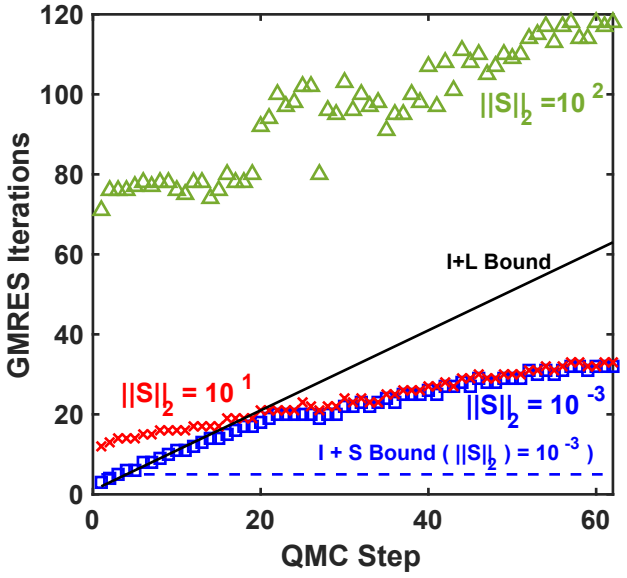


Figure 4.6: GMRES iterations for Slater matrices when reusing an initial ILUTP preconditioner with drop tolerances 10^{-6}, 10^{-4}, and 10^{-2}. $\|\mathbf{S}\|_2$ indicates the small-in-norm perturbations ($\|\mathbf{S}\|_2 = 10^{-3}$, 10^{1}, 10^{2}, respectively).

4.4 Conclusions

In this chapter, we consider GMRES convergence for matrices of the form (4.1) and identify the (potentially) sensitive eigenvalues of matrices of the form (4.4) when perturbed by a matrix $\mathbf{S}$ such that $\|\mathbf{S}\|_2 < 1$. In our numerical results, we give an example where a sequence of linear systems of the form (4.2) arises when reusing a very good initial preconditioner in a QMC application. As part of future work, we aim to further develop how these sources of sensitivity drive a more accurate theoretical framework for the GMRES convergence when applied to linear systems with coefficient matrices of the form (4.4). In particular, we are interested in examining $(z\mathbf{I} - \mathbf{L})^{-1}$ directly in order to better understand and characterize $\sigma_\delta(\mathbf{I} + \mathbf{L})$.

Chapter 5

Conclusions

In this thesis, we introduce two new recycling techniques that reduce the computational cost associated with solving a sequence of linear systems or eigenproblems. These methods are particularly useful when dealing with very large-scale matrices, and/or sequences of many matrices.

The first technique we propose, the SAM update, is an inexpensive method for updating preconditioners when solving a sequence of closely related linear systems. This update technique approximately maps one matrix in the sequence to another nearby matrix for which a good preconditioner is already computed. This approach allows us to amortize the potentially high cost of a good preconditioner over many systems in the sequence while still retaining the good spectral properties of the preconditioner. Given an appropriate choice in sparsity pattern, SAMs can be significantly cheaper to both compute and apply (in the iterative solver), and reduce total computational cost compared with recomputing a preconditioner from scratch or reusing a preconditioner for several (or all) systems. We provide an analysis of choices in sparsity patterns based on salient features of the problem, and show good results when using SAMs with an ILUTP and an AMG preconditioner. We also achieve fast convergence when using SAMs with matrices that are not diagonally dominant or are indefinite, and so SAMs provide a cheap alternative to computing a preconditioner where one may be very difficult to compute. Future work will focus on different strategies for computing the map. For instance, we can consider using hierarchical matrices to approximate the map between two matrices.

For the second method, we introduce a technique for solving a sequence of eigenproblems that we refer to as the warm-start Krylov-Schur algorithm. By warm-starting the eigensolver using an approximate invariant subspace computed for a previous, nearby eigenproblem in the sequence, we reduce the number of matrix-vector products in the eigensolver, while still computing an accurate approximate invariant subspace. Using the backward error analysis for an approximate Krylov space in [132], we first compute the best approximate Krylov decomposition for the current eigenproblem. This introduces a residual matrix of minimum

norm that we account for when expanding, truncating, and deflating the decomposition for each cycle of the eigensolver, and we show that there is a monotonic decrease in the norm of the residual. This method applies to any sequence of standard eigenproblems. We apply our method to an application arising in the numerical simulation of brake squeal that requires an implicit linear solve when computing matrix-vector products within the eigensolver, and incorporate an iterative solver to compute the matrix-vector products for this application. We also show that Krylov subspace recycling reduces the total number of iterations. These recycling strategies produce approximate invariant subspaces that are nearly as accurate as MATLAB's® `eigs` using the exact LU decomposition, and we show this using two sequences of matrices arising in the numerical simulation of brake squeal [76].

Future work will focus on how to exploit the changes between nearby matrices. For instance, for shifted matrices, such as those in the brake squeal application, we will explore how we can use the shifts to efficiently compute more accurate approximate invariant subspaces. We will also analyze the quality of the warm-start space, as well as the interaction between the error introduced when computing an inexact matrix-vector product (using an iterative solver), and the convergence of the eigensolver. We are also interested in extending the idea of warm-starting the eigensolver to methods that solve the quadratic eigenvalue problem. Future work will focus on developing a second-order warm-start Krylov-Schur algorithm based on the concept of a second-order Krylov space introduced in [13]. Further, by considering the sensitivity of the eigenvectors, we may be able to better truncate the expansion in the eigensolver.

We also consider the special case of solving sequences of linear systems where each coefficient matrix can be written as the sum of the identity matrix, a (very) low-rank matrix, and a small-in-norm matrix. Further developing general convergence bounds when solving linear systems with coefficient matrices of the form (4.1) and (4.4) is part of ongoing work. In particular, we are interested in how this theory then applies to practical problems such as the QMC application [4] discussed in Section 4.3.1.

5.1 Attributions

Eric de Sturler and Serkan Gugercin are co-authors of the paper [A. Carr, E. de Sturler, and S. Gugercin, *Preconditioning parametrized linear systems*, SIAM Journal on Scientific Computing, 43 (2021), pp. A2242-A2267], which appears as Chapter 2 in this thesis. Serkan Gugercin wrote the background information on interpolatory model reduction and IRKA (see Section 2.5.2), and both he and Eric de Sturler provided suggestions and feedback on this manuscript throughout several revisions. However, Arielle Carr was the primary author of this manuscript.

Bibliography

[1] A. Aghasi, M. E. Kilmer, and E. L. Miller, *Parametric level set methods for inverse problems*, SIAM Journal on Imaging Sciences, 4 (2011), pp. 618–650.

[2] M. I. Ahmad, D. B. Szyld, and M. B. van Gijzen, *Preconditioned multishift BiCG for H_2-optimal model reduction*, SIAM Journal on Matrix Analysis and Applications, 38 (2017), pp. 401–424.

[3] K. Ahuja, P. Benner, E. de Sturler, and L. Feng, *Recycling BiCGSTAB with an application to parametric model order reduction*, SIAM Journal on Scientific Computing, 37 (2015), pp. S429–S446.

[4] K. Ahuja, B. K. Clark, E. de Sturler, D. M. Ceperley, and J. Kim, *Improved scaling for quantum Monte Carlo on insulators*, SIAM Journal on Scientific Computing, 33 (2011), pp. 1837–1859.

[5] K. Ahuja, E. de Sturler, S. Gugercin, and E. R. Chang, *Recycling BiCG with an application to model reduction*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1925–A1949.

[6] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, SIAM, Philadelphia, PA, USA, 2005.

[7] A. C. Antoulas, C. A. Beattie, and S. Gugercin, *Interpolatory model reduction of large-scale dynamical systems*, in Efficient Modeling and Control of Large-Scale Systems, J. Mohammadpour and K. M. Grigoriadis, eds., Springer, 2010, pp. 3–58.

[8] ——, *Interpolatory Methods for Model Reduction*, Computational Science and Engineering 21, SIAM, Philadelphia, 2020.

[9] H. Anzt, E. Chow, J. Saak, and J. Dongarra, *Updating incomplete factorization preconditioners for model order reduction*, Numerical Algorithms, 73 (2016), pp. 611–630.

[10] H. Anzt, T. K. Huckle, J. Bräckle, and J. Dongarra, *Incomplete sparse approximate inverses for parallel preconditioning*, Parallel Computing, 71 (2018), pp. 1–22.

[11] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, 9 (1951), pp. 17–29.

[12] J. Baglama, T. Bella, and J. Picucci, *Hybrid iterative refined method for computing a few extreme eigenpairs of a symmetric matrix*, SIAM Journal on Scientific Computing, (2021), pp. S200–S224.

[13] Z. Bai and Y. Su, *SOAR: A second-order Arnoldi method for the solution of the quadratic eigenvalue problem*, SIAM Journal on Matrix Analysis Applications, 26 (2005), pp. 640–659.

[14] T. Bakhos, A. K. Saibaba, and P. K. Kitanidis, *A fast algorithm for parabolic PDE-based inverse problems based on Laplace transforms and flexible Krylov solvers*, Journal of Computational Physics, 299 (2015), pp. 940–954.

[15] M. Baumann and M. B. van Gijzen, *Nested Krylov methods for shifted linear systems*, SIAM Journal on Scientific Computing, 37 (2015), pp. S90–S112.

[16] U. Baur, P. Benner, and L. Feng, *Model order reduction for linear and nonlinear systems: a system-theoretic perspective*, Archives of Computational Methods in Engineering, 21 (2014), pp. 331–358.

[17] C. A. Beattie and S. Gugercin, *Realization-independent $\mathcal{H}_2$-approximation*, in Proceedings of 51st IEEE Conference on Decision and Control, 2012, pp. 4953 – 4958.

[18] C. A. Beattie, S. Gugercin, and S. A. Wyatt, *Inexact solves in interpolatory model reduction*, Linear Algebra and its Applications, 436 (2012), pp. 2916–2943.

[19] S. Bellavia, D. Bertaccini, and B. Morini, *Nonsymmetric preconditioner updates in Newton-Krylov methods for nonlinear systems*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2595–2619.

[20] S. Bellavia, V. D. Simone, D. di Serafino, and B. Morini, *Efficient preconditioner updates for shifted linear systems*, SIAM Journal on Scientific Computing, 33 (2011), pp. 1785–1809.

[21] M. P. Bendsøe, *Optimal shape design as a material distribution problem*, Structural Optimization, 1 (1989), pp. 193–202.

[22] M. P. Bendsøe, A. Ben-Tal, and J. Zowe, *Optimization methods for truss geometry and topology design*, Structural Optimization, 7 (1994), pp. 141–159.

[23] M. P. Bendsøe and O. Sigmund, *Material interpolation schemes in topology optimization*, Archives of Applied Mechanics, 69 (1999), pp. 635–654.

[24] ——, *Topology optimization: Theory, Methods, and Applications*, Springer, Berlin, 2003.

[25] P. Benner, S. Gugercin, and K. Willcox, *A survey of projection-based model reduction methods for parametric dynamical systems*, SIAM Review, 57 (2015), pp. 483–531.

[26] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox, *Model Reduction and Approximation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

[27] M. W. Benson, *Iterative solution of large scale linear systems*, Master's thesis, Lakehead University, Thunder Bay, Canada, 1973.

[28] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Mathematica, 22 (1982), pp. 127–140.

[29] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, Journal of Computational Physics, 182 (2002), pp. 418–477.

[30] M. Benzi and D. Bertaccini, *Approximate inverse preconditioning for shifted linear systems*, BIT Numerical Mathematics, 43 (2003), pp. 231–244.

[31] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1318–1332.

[32] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1333–1353.

[33] M. Benzi, R. Kouhia, and M. Tůma, *Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics*, Computer Methods in Applied Mechanics and Engineering, 190 (2001), pp. 6533–6554.

[34] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM Journal on Scientific Computing, 19 (1998), pp. 968–994.

[35] ——, *A comparative study of sparse approximate inverse preconditioners*, Applied Numerical Mathematics, 30 (1999), pp. 305–340.

[36] L. Bergamaschi, *A survey of low-rank updates of preconditioners for sequences of symmetric linear systems*, Algorithms, 13 (2020), p. 100.

[37] L. Bergamaschi, R. Bru, A. Martinez, and M. Putti, *Quasi-newton preconditioners for the inexact newton method*, Electronic Transactions on Numerical Analysis, 23 (2006), pp. 76–87.

[38] M. Berljafa, D. Wortmann, and E. D. Napoli, *An optimized and scalable eigensolver for sequences of eigenvalue problems*, Concurrency and Computation: Practice and Experience, 27 (2014), pp. 905–922.

[39] D. Bertaccini, *Efficient preconditioning for sequences of parametric complex symmetric linear systems*, Electronic Transactions on Numerical Analysis, 18 (2004), pp. 49–64.

[40] M. Bolten, E. de Sturler, and C. Hahn, *Krylov subspace recycling for evolving structures*. Available from https://arxiv.org/abs/2010.11447, 2020.

[41] J. T. Borggaard and S. Gugercin, *Model reduction for DAEs with an application to flow control 2014*, in Active Flow and Combustion Control, R. King, ed., Springer International Publishing, 2015, pp. 381–396.

[42] R. Bru, J. Marin, J. Mas, and M. Tůma, *Balanced incomplete factorization*, SIAM Journal on Scientific Computing, 30 (2008), pp. 2302–231.

[43] A. Bunse-Gerstner, D. Kubalińska, G. Vossen, and D. Wilczek, $\mathcal{H}_2$*-optimal model reduction for large scale discrete dynamical MIMO systems*, Journal of Computational and Applied Mathematics, (2009). doi:10.1016/j.cam.2008.12.029.

[44] C. Calgaro, J.-P. Chehab, and Y. Saad, *Incremental incomplete LU factorizations with applications*, Numerical Linear Algebra with Applications, 17 (2010), pp. 811–837.

[45] S. L. Campbell, I. C. Ipsen, C. T. Kelley, and C. D. Meyer, *GMRES and the minimal polynomial*, BIT Numerical Mathematics, 36 (1996), pp. 664–675.

[46] M. Cardiff and W. Barrash, *3-D Transient hydraulic tomography in unconfined aquifers with fast drainage response*, Water Resources Research, 47 (2011).

[47] A. Carr and E. de Sturler, *MATLAB ® implementation of Yousef Saad's ILUTP algorithm*. Available from https://arxiv.org/abs/1601.05883, 2016. This implementation closely follows the ILUT paper and Sparskit, while exploiting several steps to ensure an efficient implementation in MATLAB ® .

[48] A. Carr, E. de Sturler, and S. Gugercin, *Preconditioning parametrized linear systems*. Available from https://arxiv.org/abs/1601.05883, 2020. This is a previous, unpublished version.

[49] ——, *Preconditioning parametrized linear systems*, SIAM Journal on Scientific Computing, 43 (2021), pp. A2242–A2267.

[50] J. Cerdan, J. Marin, and J. Mas, *Low-rank updates of balanced incomplete factorization preconditioners*, Numerical Algorithms, 74 (2017), pp. 337–370.

[51] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1804–1822.

[52] ——, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, International Journal of High Performance Computing Applications, 15 (2001), pp. 56–74.

[53] E. Chow and A. Patel, *Fine-grained parallel incomplete LU factorization*, SIAM Journal on Scientific Computing, 37 (2015), pp. C169–C193.

[54] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

[55] D. Darnell, R. B. Morgan, and W. Wilcox, *Deflated GMRES for systems with multiple shifts and multiple right-hand sides*, Linear Algebra and its Applications, 429 (2008), pp. 2415–2434. Special Issue in honor of Richard S. Varga.

[56] E. de Sturler, *Nested Krylov methods based on GCR*, Journal of Computational and Applied Mathematics, 67.1 (1996), pp. 15–41.

[57] ——, *Truncation strategies for optimal Krylov subspace methods*, SIAM Journal on Numerical Analysis, 36.3 (1999), pp. 864–889.

[58] E. de Sturler and M. E. Kilmer, *A regularized Gauss-Newton trust region approach to imaging in diffuse optical tomography*, SIAM Journal on Scientific Computing, 33 (2011), pp. 3057–3086.

[59] J. M. Derlaga, T. Phillips, and C. J. Roy, *SENSEI computational fluid dynamics code: a case study in modern Fortran software development*, in 21st AIAA Computational Fluid Dynamics Conference, 2013, p. 2450.

[60] E. Di Napoli, S. Blugel, and P. Bientinesi, *Correlations in sequences of generalized eigenproblems arising in Density Functional Theory*, Computer Physics Communications, 183 (2012), pp. 1674–1682.

[61] H. Dym, *Linear Algebra in Action*, American Mathematical Society, 2 ed., 2013.

[62] R. Ehrig and P. Deuflhard, *GMERR - an error minimizing variant of GMRES*, Tech. Rep. SC-97-63, ZIB, Takustr. 7, 14195 Berlin, 1997.

[63] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 345–357.

[64] H. C. Elman and D. P. O'Leary, *Efficient iterative solution of the three-dimensional Helmholtz equation*, Journal of Computational Physics, 142 (1998), pp. 163–181.

[65] ——, *Eigenanalysis of some preconditioned Helmholtz problems*, Numerische Mathematik, 83 (1999), pp. 231–257.

[66] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation*, Communications on Pure and Applied Mathematics, 64 (2011), pp. 697–735.

[67] Y. A. Erlangga and R. Nabben, *On a multilevel Krylov method for the Helmholtz equation preconditioned by shifted Laplacian*, Electronic Transactions on Numerical Analysis, 31 (2008), pp. 403–424.

[68] Y. A. Erlangga, C. Vuik, and C. W. Oosterlee, *On a class of preconditioners for solving the Helmholtz equation*, Applied Numerical Mathematics, 50 (2004), pp. 409–425.

[69] ——, *Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation*, Applied Numerical Mathematics, 56 (2006), pp. 648–666.

[70] L. Feng, P. Benner, and J. G. Korvink, *Parametric model order reduction accelerated by subspace recycling*, in 48th IEEE Conference on Decision & Control and 28th Chinese Control Conference, 2009, pp. 4328–4333.

[71] ——, *Subspace recycling accelerates the parametric macro-modeling of MEMS*, International Journal for Numerical Methods in Engineering, 94 (2013), pp. 84–110.

[72] D. R. Fokkema, G. L. Sleijpen, and H. A. Van der Vorst, *Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils*, SIAM Journal on Scientific Computing, 20 (1998), pp. 94–125.

[73] N. Gmati and B. Philippe, *Comments on the GMRES convergence for preconditioned systems*, in Large-scale scientific computing, Springer, 2008, pp. 40–51.

[74] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 3 ed., 1996.

[75] A. D. Gordon and C. E. Powell, *On solving stochastic collocation systems with algebraic multigrid*, IMA Journal of Numerical Analysis, 32 (2012), pp. 1051–1070.

[76] N. Grabner, V. Mehrmann, S. Quraishi, C. Schroder, and U. von Wagner, *Numerical methods for parametric model reduction in the simulation of disk brake squeal*, Z. Angew. Math. Mech, 96 (2016), pp. 1388–1405.

[77] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA, 1997.

[78] M. J. Grote and M. Huckle, *Effective parallel preconditioning*, in Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1995, San Francisco, California, USA, February 15-17, 1995, D. H. Bailey, P. E. Bjørstad, J. R. Gilbert, M. Mascagni, R. S. Schreiber, H. D. Simon, V. Torczon, and L. T. Watson, eds., SIAM, 1995, pp. 466–471.

[79] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853.

[80] G. Gu, X. Zhou, and L. Lin, *A flexible preconditioned Arnoldi method for shifted linear systems*, Journal of Computational Mathematics, 25 (2007), pp. 522–530.

[81] S. Gugercin, A. C. Antoulas, and C. A. Beattie, *H_2 model reduction for large-scale linear dynamical systems*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 609–638.

[82] J. S. Hesthaven, G. Rozza, and B. Stamm, *Certified reduced basis methods for parametrized partial differential equations*, Springer Briefs in Mathematics, Springer, Switzerland, 2016.

[83] N. J. Higham, *Estimating the matrix p-norm*, Numerische Mathematik, 62 (1992), pp. 539–555.

[84] J. M. Hokanson and C. C. Magruder, *$\mathcal{H}_2$-optimal model reduction using projected nonlinear least squares*, SIAM Journal on Scientific Computing, 42 (2020), pp. A4017–A4045.

[85] R. M. Holland, A. J. Wathen, and G. J. Shaw, *Sparse approximate inverses and target matrices*, SIAM Journal on Scientific Computing, 26 (2005), pp. 1000–1011.

[86] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, NY, 2 ed., 2013.

[87] X. Hu, J. Lin, and L. T. Zikatanov, *An adaptive multigrid method based on path cover*, SIAM Journal on Scientific Computing, 41 (2019), pp. S220–S241.

[88] T. Huckle, *Efficient computation of sparse approximate inverses*, Numerical Linear Algebra with Applications, 5 (1998), pp. 57–71.

[89] ——, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Applied Numerical Mathematics, 30 (1999), pp. 291–303.

[90] ——, *Factorizaed sparse approximate inverses for preconditioning*, Journal of Supercomputing, 25 (2003), pp. 109–117.

[91] T. Huckle and A. Kallischko, *Frobenius norm minimization and probing for preconditioning*, International Journal of Computer Mathematics, 84 (2007), pp. 1225 – 1248.

[92] T. Huckle, A. Kallischko, A. Roy, M. Sedlacek, and T. Weinzierl, *An efficient parallel implementation of the MSPAI preconditioner*, Parallel Computing, 36 (2010), pp. 273–284.

[93] INTES, *PERMAS Example Manual*, 2014. Version 14.00.147, INTES publication no. 550.

[94] C. W. Jackson, W. C. Tyson, and C. J. Roy, *Turbulence model implementation and verification in the SENSEI CFD code*, in AIAA Scitech 2019 Forum, 2019, p. 2331.

[95] A. Kallischko, *Modified sparse approximate inverses (MSPAI) for parallel preconditioning*, PhD thesis, Technische Universitat Munchen, 2008. Advisor: Thomas Huckle.

[96] C. T. Kelley, I. G. Kevrekidis, and L. Qiao, *Newton-Krylov solvers for time-steppers.* preprint, https://arxiv.org/abs/math/0404374, 2004.

[97] M. E. Kilmer and E. de Sturler, *Recycling subspace information for diffuse optical tomography*, SIAM Journal on Scientific Computing, 27 (2006), pp. 2140–2166.

[98] J. G. Korvink and E. B. Rudnyi, *Oberwolfach benchmark collection*, in Dimension Reduction of Large-Scale Systems, P. Benner, D. C. Sorensen, and V. Mehrmann, eds., Berlin, Heidelberg, 2005, Springer Berlin Heidelberg, pp. 311–315.

[99] D. Kressner, *Structured eigenvalue problems*, Numerical Methods for General and Structured Eigenvalue Problems, (2005), pp. 131–214.

[100] R. Lehoucq and D. Sorensen, *Deflation techniques for an implicitly restarted Arnoldi iteration*, SIAM Journal on Matrix Analysis and Applications, 17(4) (1994), pp. 789–821.

[101] M. Li, *Recycling Preconditioners for Sequences of Linear Systems and Matrix Reordering*, PhD thesis, Virginia Tech, 2015.

[102] M. Li, V. Rao, and E. de Sturler, *Effective truncation of low-rank preconditioner updates.* In preparation.

[103] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad, *A thick-restart Lanczos algorithm with polynomial filtering for Hermitian eigenvalue problems*, SIAM Journal on Scientific Computing, 38 (2016), pp. A2512–A2534.

[104] J. Mackerle, *Topology and shape optimization of structures using FEM and BEM: A bibliography (1999–2001)*, Finite Elements in Analysis and Design, 39 (2003), pp. 243–253.

[105] C. B. Moler and G. W. Stewart, *An algorithm for generalized matrix eigenvalue problems*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 241–256.

[106] C. Moosmann, E. B. Rudnyi, A. Greiner, and J. G. Korvink, *Model order reduction for linear convective thermal flow*, in Proceedings of 10th International Workshops on THERMal INvestigations of ICs and Systems, vol. 29, 2004, pp. 317–322.

[107] R. B. Morgan, *Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1112–1135.

[108] R. B. Morgan, *GMRES with deflated restarting*, SIAM Journal on Scientific Computing, 24.1 (2002), pp. 20–37.

[109] C. C. Paige, B. N. Parlett, and H. A. Van der Vorst, *Approximate solutions and eigenvalue bounds from Krylov subspaces*, Numerical Linear Algebra with Applications, 2 (1995), pp. 115–133.

[110] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti, *Recycling Krylov subspaces for sequences of linear systems*, SIAM Journal on Scientific Computing, 28 (2006), pp. 1651–1674.

[111] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced Basis Methods for Partial Differential Equations: An Introduction*, Springer, 2015.

[112] A. Rafiei, *Left-looking version of AINV preconditioner with complete pivoting strategy*, Linear Algebra and its Applications, 445 (2014), pp. 103–126.

[113] G. I. N. Rozvany, *Aims, scope, methods, history and unified terminology of computer-aided topology optimization in structural mechanics*, Structural and Multidisciplinary Optimization, 21 (2001), pp. 90–108.

[114] G. I. N. Rozvany, M. P. Bendsøe, and U. Kirsch, *Layout optimization of structures*, Applied Mechanics Reviews, 48 (1995), pp. 41–119.

[115] E. B. Rudnyi and J. G. Korvink, *Review: Automatic model reduction for transient simulation of MEMS-based devices*, Sensors Update, 11 (2002), pp. 3–33.

[116] J. W. Ruge and K. Stüben, *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, in Multigrid Methods for Integral and Differential Equations, D. Paddon and H. Holstein, eds., vol. 3, Clarendon Press, Oxford, 1985, pp. 169–212.

[117] ——, *Algebraic multigrid (AMG)*, in Multigrid methods, frontiers in applied mathematics, S. McCormick, ed., SIAM, Philadelphia, 1986.

[118] Y. Saad, *ILUT: a dual threshold incomplete LU factorization*, Numerical Linear Algebra with Applications, 1 (1994), pp. 387–402.

[119] ——, *SPARSKIT: A basic took-kit for sparse matrix computations.* http://www-users.cs.umn.edu/ saad/software/SPARSKIT/, 1994.

[120] ——, *Iterative Methods for Sparse Linear Systems, 2nd Ed.*, SIAM, 2003.

[121] ——, *Numerical Methods for Large Eigenvalue Problems: Revised Edition*, SIAM, 2011.

[122] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[123] A. K. Saibaba, T. Bakhos, and P. K. Kitanidis, *A flexible Krylov solver for shifted systems with application to oscillatory hydraulic tomography*, SIAM Journal on Scientific Computing, 35 (2013), pp. A3001–A3023.

[124] A. H. Sheikh, D. Lahaye, and C. Vuik, *On the convergence of shifted Laplace preconditioner combined with multilevel deflation*, Numerical Linear Algebra with Applications, 20 (2013), pp. 645–662.

[125] J. A. Sifuentes, M. Embree, and R. B. Morgan, *GMRES convergence for perturbed coefficient matrices, with application to approximate deflation preconditioning*, SIAM Journal on Matrix Analysis and Applications, 34 (2013), pp. 1066–1088.

[126] O. Sigmund, *Topology optimization: a tool for the tailoring of structures and materials*, Philosophical Transactions: Mathematical, Physical and Engineering Sciences, 358 (2000), pp. 211–228.

[127] G. L. Sleijpen and H. A. Van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM Review, 42 (2000), pp. 267–293.

[128] K. M. Soodhalter, E. de Sturler, and M. E. Kilmer, *A survey of subspace recycling iterative methods*, GAMM-Mitteilungen, 43 (2020), p. e202000016. https://doi.org/10.1002/gamm.202000016.

[129] G. W. Stewart, *A Krylov-Schur algorithm for large eigenproblems*, SIAM Journal on Matrix Analysis and Applications, 23(3) (2001), pp. 601–614.

[130] ——, *Matrix Algorithms: Volume II: Eigensystems*, SIAM, 2001.

[131] ——, *Addendum to "A Krylov-Schur algorithm for large eigenproblems"*, SIAM Journal of Matrix Analysis and Applications, 24(2) (2002), pp. 599–601.

[132] ——, *Backward error bounds for approximate Krylov subspaces*, Linear Algebra and its Applications, 340 (2002), pp. 81–86.

[133] K. Stüben, *Algebraic multigrid (AMG): Experiences and comparisons*, Applied Mathematics and Computation, 13 (1983), pp. 419–452.

[134] ——, *Algebraic multigrid (AMG): an introduction with applications*, in Multigrid, N. Y. Academic Press, ed., Trottenberg, Ulrich and Oosterlee, Cornelius W and Schüller, Anton, 2000.

[135] ——, *A review of algebraic multigrid*, Journal of Computational and Applied Mathematics, 128 (2001), pp. 281–309.

[136] W.-P. Tang, *Toward an effective sparse approximate inverse preconditioner*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 970–986.

[137] J. D. Tebbens and M. Tůma, *Preconditioner updates for solving sequences of linear systems in a matrix-free environment*, Numerical linear algebra with applications, 17 (2010), pp. 997–1019.

[138] The MORwiki Community, *Convection*. MORwiki – Model Order Reduction Wiki, http://modelreduction.org/index.php/Convection, 2020.

[139] F. Tisseur and K. Meerbergen, *The quadratic eigenvalue problem*, SIAM Review, 43 (2001), pp. 235–286.

[140] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*, Elsevier, 2000.

[141] H. A. van der Vorst, *Iterative Krylov Methods for Large Linear Systems*, Cambridge University Press, 2003.

[142] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 152–163.

[143] S. Wang, *Krylov Subspace Methods for Topology Optimization on Adaptive Meshes*, PhD thesis, Univeristy of Illinois at Urbana-Champaign, 2007. Advisor Eric de Sturler.

[144] S. Wang and E. de Sturler, *Multilevel sparse approximate inverse preconditioners for adaptive mesh refinement*, Linear Algebra and its Applications, 431 (2009), pp. 409–426.

[145] S. Wang, E. de Sturler, and G. H. Paulino, *Large-scale topology optimization using preconditioned Krylov subspace methods with recycling*, International Journal for Numerical Methods in Engineering, 69 (2007), pp. 2441–2468.

[146] T. G. Wright, *EigTool*. http://www.comlab.ox.ac.uk/pseudospectra/eigtool/, 2002.

[147] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM Journal on Matrix Analysis and Applications, 22 (2000), pp. 602–616.

[148] S. A. Wyatt, *Issues in interpolatory model reduction: Inexact solves, second-order systems and DAEs*, PhD thesis, Virginia Tech, 2012.

[149] Y. Xu and T. Zeng, *Optimal $\mathcal{H}_2$ model reduction for large scale MIMO systems via tangential interpolation*, International Journal of Numerical Analysis and Modeling, 8 (2011), pp. 174–188.

Appendices

Appendix A

ILUT(P) Implementation

In Section A.1, we give an algorithm for an efficient MATLAB® implementation of Saad's ILUT [118, 119], and in Section A.2 we provide the modifications necessary to include pivoting. The full MATLAB® code for computing an ILUTP can also be found in [47].

A.1 ILUT Pseudocode

This pseudocode comes directly from the ILUTP implementation in [119]. Bolded line numbers indicate where modifications must be made to include pivoting.

1: Given $\mathbf{A} \in \mathbb{C}^{n \times n}, maxfill, droptol$
2: **for** $row = 1 : n$ **do**
3: $\quad$ $\mathbf{w} = \mathbf{0}$, $\mathbf{jw} = \mathbf{0}$, $\mathbf{rjw} = \mathbf{0}$ { Data structures for working array } [1]
4: $\quad$ $\mathbf{LUval} = \mathbf{0}$, $\mathbf{LUindex} = \mathbf{0}$ { MSR data structures }
5: $\quad$ $\mathbf{Urows} = \mathbf{0}$ { Pointer to the beginning of each row in $\mathbf{U}$ }
6: $\quad$ $Ustart = n + 2$
7: $\quad$ $\mathbf{LUindex}(1) = Ustart$

---

[1] **w** contains the nonzero values in the current row; **jw** contains the column indices in the matrix of the values in **w**; **rjw** contains nonzeros in locations where nonzeros occur in **w** and **jw**. For further description of these data structures, we refer the reader to [120].

```
 8:    currentRow = A(row, :)
 9:    dropNorm = ||currentRow||  { Compute the norm of the current row }
10:    Determine val and col indices of currentRow
11:    idxL = col < row, lenL = length of idxL
12:    idxU = col > row, lenU = length of idxU
13:    diagcoef = row
14:    w(row) = A(row, diagcoef)  { Store diagonal }
15:    jw(row) = row, rjw(row) = row
16:    w(1 : lenL) = val(idxL)  { Store lower part }
17:    jw(1 : lenL) = idxL, rjw(1 : lenL) = 1 : lenL
18:    w(row + 1 : row + lenU) = val(idxU)  { Store upper part }
19:    jw(row + 1 : row + lenU) = idxU, rjw(row + 1 : row + lenU) = 1 : lenU
20:    multiplierLocation = 0
21:    for column = 1 : lenL do
22:        Determine next minCol and corresponding idx in jw(column : lenL)
23:        Update w, jw, and rjw if idx > 1
24:        rjw(minCol) = 0  { Zero out column }  { Perform elimination on previous rows
   }
25:        multiplier = w(column)/LUval(minCol)
26:        if |multiplier| > droptol then  { Apply droptol to L }
27:            for k = Urows(minCol) : LUval(minCol + 1) − 1 do
28:                subtract = multiplier ∗ LUval(k)
29:                j = LUindex(k)
30:                fill = rjw(j)  { Determine corresponding index in w }
31:                fill = rjw(k)  { Find element location in w }
32:                if j ≥ row then  { Upper }
33:                    if fill ≠ 0 then { This is not a fill-in }
34:                        w(fill) = w(fill) − subtract
35:                    else { Append to w, jw, and rjw }
36:                        lenU = lenU + 1
37:                        w(row + lenU) = −subtract
38:                        jw(row + lenU) = j
39:                        rjw(row + lenU) = row + lenU
40:                    end if
```

```
41:                 else { Lower }
42:                     if fill ≠ 0 then { This is not a fill-in }
43:                         w(fill) = w(fill) − subtract
44:                     else { Append to w, jw, and rjw }
45:                         lenL = lenL + 1
46:                         w(lenL) = −subtract
47:                         jw(lenL) = j
48:                         rjw(lenL) = lenL
49:                     end if
50:                 end if
51:             end for
52:         end if
          { Store pivot }
53:         multiplierLocation = multiplierLocation + 1
54:         w(multiplierLocation) = pivot
55:         jw(multiplierLocation) = minCol
56:     end for
57:     Enforce maxfill on w(1 : lenL), update lenL
          { Update L in MSR format }
58:     store = min(lenL, maxfill)
59:     LUval(Ustart : (Ustart + store − 1)) = w(1 : store)
60:     LUindex(Ustart : (Ustart + store − 1)) = jw(1 : store)
61:     Ustart = Ustart + length
          { Apply droptol to U }
62:     store = 0
63:     for k = 1 : lenU do
64:         if |w(k)| ≥ droptol ∗ dropNorm then
65:             w(row + store) = w(row + k)
66:             jw(row + store) = jw(row + k)
67:         end if
68:     end for
```

69: Enforce $maxfill$ on $\mathbf{w}(row+1:row+lenU)$, update $lenU$
{ Update **U** in MSR format }
70: $store = min(lenU, maxfill)$
71: $Uend = length(row+1:row+store-1)$
72: $\mathbf{LUval}(Ustart:Ustart+Uend-1) = \mathbf{w}(Ustart:Ustart+Uend-1)$
73: $\mathbf{LUindex}(Ustart:Ustart+Uend-1) = \mathbf{jw}(Ustart:Ustart+Uend-1)$
{ Store pivot }
74: $\mathbf{LUval}(row) = \mathbf{w}(row)$
75: **end for**
76: Convert MSR to COO
77: Convert COO to MATLAB® `sparse` matrix format
78: Split single **LU** matrix into **L** and **U**

A.2 ILUTP Modifications to ILUT Pseudocode

Line numbers indicate where in the ILUT algorithm in Appendix A.1 to make modifications to incorporate pivoting.

Line 1: $perm = rperm = 1 : n$ { $perm$ and $rperm$ are integer arrays representing the old and new locations of unknowns, respectively. }

Line 10: $col = rperm(col)$, $val = rperm(val)$ { Find the new location of columns according to that permutation and permute the values accordingly. }

Line 13: $diagcoef = perm(row)$ { Determine the new location of the diagonal element. }

Line 29: $j = rperm(\mathbf{LUindex}(k))$ { Determine the new column index. }

Line 60: $\mathbf{LUindex}(Ustart : (Ustart + store - 1)) = perm(\mathbf{jw}(1 : store))$ { Store columns of **L** in old locations. }

(After) Line 69: Determine the next pivot; if it is not $\mathbf{w}(row)$, exchange **w**, $perm$, and $rperm$.

Line 73: $\mathbf{LUindex}(Ustart : Ustart + Uend - 1) = perm(\mathbf{jw}(Ustart : Ustart + Uend - 1))$ { Store columns of **U** in old locations. }

(Before) Line 76: Put the columns of **LU** in new locations (using $rperm$).

Appendix B

Nonlinear Convection-Diffusion Discretization

B.1 Matlab Implementation

Below, we give the MATLAB® implementation of the second order finite difference discretization of the nonlinear (in diffusion) convection-diffusion equation.

```
    function  [F,J,rhsF,Fmat] = cd2d_nonlinear(nx,ny,dx,dy,x0,y0,uvec,...
         p,q,pdx,qdx,r,s,t,f,s_bnd,w_bnd,n_bnd,e_bnd)

% This function generates a sparse matrix and right hand side from the
% 2-dimensional convection-diffusion equation
%
%     - (p*u_x)_x - (q*u_y)_y + r*u_x + s*u_y + t*u = f
%
% using 2nd order central finite differences discretization.  Here p and q
% are functions of x,y, and u (and therefore the convection-diffusion
% equation above is nonlinear in diffusion).
%
% nx: number of grid points in x-direction
%     not counting boundary points for Dir bc
%     counting boundary points for Neumann/Robin bc
% ny: number of grid points in y-direction
%     not counting boundary points for Dir bc
%     counting boundary points for Neumann/Robin bc
% dx: (fixed) mesh width in x-direction; depends on bc
% dy: idem
```

```
% x0: x-coeff lower left corner of domain (south west corner)
% y0: y-coeff lower left corner of domain (south west corner)
% u:  solution at the previous Newton iteration, stored as a vector

% The following are all functions of x, y, and u: p(x,y,u), q(x,y,u), etc.
%
% p:  diffusion coefficient (positive) in x-direction
% q:  idem y-direction
% pdx: derivative of p
% qdx: derivative of q
%
% The following are all functions of x and y:
%
% r:  convection in x-direction (flow velocity)
% s:  idem y-direction
% t:  reaction or absorption rate
% f:  forcing function (rhs); source or sink

% The following are constants (can be modified to be functions of x or y)

% s_bnd: Dirichlet boundary condition - south
% w_bnd: Dirichlet boundary condition - west
% n_bnd: Dirichlet boundary condition - north
% e_bnd: Dirichlet boundary condition - east
%
% Author: Arielle (Grim-McNally) Carr 2017 (modified from cd_2d.m, author
% Eric de Sturler)
% Most recent modifications: February 2018

nrows = nx*ny;
dx2 = dx^2;
dy2 = dy^2;

% Allocate space for A(u)u and f (to put together -F)
mat_tmpF = zeros(nx,ny,5);
rhs_tmpF = zeros(nx,ny);

% Allocate space for J (the Jacobian)
mat_tmpJ = zeros(nx,ny,5);

% Convert the solution vector to a matrix
u_length = length(uvec);
```

```
u_size = sqrt(u_length);
u = zeros(u_size);
ucolAdd = zeros(u_size,1);
ucol = 1;
for i = 1:u_length
    idx = mod(i,u_size);
    if idx ~= 0
        ucolAdd(idx) = uvec(i);
    else
        ucolAdd(u_size) = uvec(i);
        u(:,ucol) = ucolAdd;
        ucol = ucol + 1;
        ucolAdd = zeros(u_size,1);
    end
end
```

```
for i = 1:nx,
  for j = 1:ny,
    xc = x0 + i*dx;   yc = y0 + j*dy;   % center
    xs = xc;          ys = yc - 1/2*dy; % south
    xe = xc + 1/2*dx; ye = yc;          % east
    xn = xc;          yn = ys + dy;     % north
    xw = xe - dx;     yw = yc;          % west

    % Obtain surrounding u-values:
    uc = u(i,j); % Center

    if j ~= 1 % South
        us = u(i,j-1);
    else
        us = s_bnd(xs,ys);
    end

    if i ~= nx % East
        ue = u(i+1,j);
    else
        ue = e_bnd(xe,ye);
    end

    if j ~= ny % North
        un = u(i,j+1);
```

```
else
    un = n_bnd(xn,yn);
end

if i ~= 1 % West
    uw = u(i-1,j);
else
    uw = w_bnd(xw,yw);
end

% Approximate the solution at the midpoints by taking the average of
% the solution at the surrounding points
south = (uc+us)/2; %q
west = (uc+uw)/2; %p
north = (uc+un)/2; %q
east = (uc+ue)/2; %p

% diffusion - A(u)
mat_tmpF(i,j,1) = -q(xs,ys,south)/dy2;  % south
mat_tmpF(i,j,2) = -p(xw,yw,west)/dx2;  % west
mat_tmpF(i,j,4) = -p(xe,ye,east)/dx2;  % east
mat_tmpF(i,j,5) = -q(xn,yn,north)/dy2;  % north
mat_tmpF(i,j,3) = -(mat_tmpF(i,j,1) + mat_tmpF(i,j,2) + ...
                    mat_tmpF(i,j,4) + mat_tmpF(i,j,5));  % center

% convection - A(u)
mat_tmpF(i,j,1) = mat_tmpF(i,j,1) - s(xc,yc)/(2*dy); % south
mat_tmpF(i,j,2) = mat_tmpF(i,j,2) - r(xc,yc)/(2*dx); % west
mat_tmpF(i,j,4) = mat_tmpF(i,j,4) + r(xc,yc)/(2*dx); % east
mat_tmpF(i,j,5) = mat_tmpF(i,j,5) + s(xc,yc)/(2*dy); % north

% reaction - A(u)
mat_tmpF(i,j,3) = mat_tmpF(i,j,3) + t(xc,yc);

% rhs - f
rhs_tmpF(i,j) = f(xc,yc);

% Build the Jacobian:
% South (q)
mat_tmpJ(i,j,1) = 1/dy2*(q(xs,ys,south)+(1/2)*us*qdx(xs,ys,south)-...
            (1/2)*uc*qdx(xs,ys,south));
```

```
    % West (p)
    mat_tmpJ(i,j,2) = 1/dx2*(p(xw,yw,west)+(1/2)*uw*pdx(xw,yw,west)-...
                (1/2)*uc*pdx(xw,yw,west));

    % East (p)
    mat_tmpJ(i,j,4) = 1/dx2*(p(xe,ye,east)-(1/2)*uc*pdx(xe,ye,east)+...
                (1/2)*ue*pdx(xe,ye,east));

    % North (q)
    mat_tmpJ(i,j,5) = 1/dy2*(q(xn,yn,north)-(1/2)*uc*qdx(xn,yn,north)+...
                (1/2)*un*qdx(xn,yn,north));

    % Center (p and q)
    mat_tmpJ(i,j,3) = 1/(dx*dy)*(-q(xs,ys,south)-p(xw,yw,west)-...
                    p(xe,ye,east)-q(xn,yn,north)+...
                    (1/2)*us*qdx(xs,ys,south)+...
                    (1/2)*uw*pdx(xw,yw,west)+...
                    (1/2)*ue*pdx(xe,ye,east)+...
                    (1/2)*un*qdx(xn,yn,north)-...
                    uc*((1/2)*qdx(xs,ys,south)+...
                    (1/2)*pdx(xw,yw,west)+(1/2)*pdx(xe,ye,east)+...
                    (1/2)*qdx(xn,yn,north)));
  end
end

% boundary conditions
xbnd = x0+dx:dx:x0+nx*dx; ybnd = y0+dy:dy:y0+ny*dy;
sbval = s_bnd(xbnd,ybnd)';
ebval = e_bnd(xbnd,ybnd)';
nbval = n_bnd(xbnd,ybnd)';
wbval = w_bnd(xbnd,ybnd)';

% south
rhs_tmpF(1:nx,1) = rhs_tmpF(1:nx,1) - mat_tmpF(1:nx,1,1).*sbval;
mat_tmpF(1:nx,1,1) = 0;
mat_tmpJ(1:nx,1,1) = 0;

% east
rhs_tmpF(nx,1:ny) = rhs_tmpF(nx,1:ny) - mat_tmpF(nx,1:ny,4).*ebval;
mat_tmpF(nx,1:ny,4) = 0;
mat_tmpJ(nx,1:ny,4) = 0;
```

```
% north
rhs_tmpF(1:nx,ny) = rhs_tmpF(1:nx,ny) - mat_tmpF(1:nx,ny,5).*nbval;
mat_tmpF(1:nx,ny,5) = 0;
mat_tmpJ(1:nx,ny,5) = 0;

% west
rhs_tmpF(1,1:ny) = rhs_tmpF(1,1:ny) - mat_tmpF(1,1:ny,2).*wbval;
mat_tmpF(1,1:ny,2) = 0;
mat_tmpJ(1,1:ny,2) = 0;

% create sparse matrix and right hand side
rowsF = zeros(5*nrows,1); rowsJ = rowsF;
colsF = zeros(5*nrows,1); colsJ = colsF;
valsF = zeros(5*nrows,1); valsJ = valsF;
rhsF  = zeros(nrows,1);

idx = 0;
for k = 1:nrows,
  i = mod(k-1,nx)+1;
  j = (k-i)/nx+1;
  rowsF(idx+1:idx+5) = [k;k;k;k;k];
  rowsJ(idx+1:idx+5) = [k;k;k;k;k];

  % For references to boundary points (not in grid/matrix), replace by
  % incorrect but valid references (in matrix) with zero value.
  % Those are later automatically removed by "sparse"
  s_idx = max(1,k-nx);
  w_idx = max(1,k-1);
  e_idx = min(k+1,nrows);
  n_idx = min(k+nx,nrows);
  colsF(idx+1:idx+5) = [s_idx;w_idx;k;e_idx;n_idx];
  colsJ = colsF;
  valsF(idx+1:idx+5) = [mat_tmpF(i,j,1); ...
                        mat_tmpF(i,j,2); ...
                        mat_tmpF(i,j,3); ...
                        mat_tmpF(i,j,4); ...
                        mat_tmpF(i,j,5)];
  valsJ(idx+1:idx+5) = [mat_tmpJ(i,j,1); ...
                        mat_tmpJ(i,j,2); ...
                        mat_tmpJ(i,j,3); ...
                        mat_tmpJ(i,j,4); ...
                        mat_tmpJ(i,j,5)];
```

```
  rhsF(k) = rhs_tmpF(i,j);
  idx = idx+5;
end

Fmat = sparse(rowsF,colsF,valsF);
J = sparse(rowsJ,colsJ,valsJ);

% Build the vector F, containing the entries F_(ij) = A(u)u-f
F = Fmat*uvec - rhsF;

end
```