Efficient rational filter-based interior eigensolvers

Dissertation

Bergische Universität Wuppertal
Fakultät für Mathematik und Naturwissenschaften

eingereicht von
Sarah Huber, M. Sc.
zur Erlangung des Grades eines Doktors der Naturwissenschaften

Betreut durch Prof. Dr. Bruno Lang

Tag der Disputation: 02.06.2021

The PhD thesis can be quoted as follows:

urn:nbn:de:hbz:468-20210706-133415-9
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20210706-133415-9]

DOI: 10.25926/egqy-8b34
[https://doi.org/10.25926/egqy-8b34]

ACKNOWLEDGMENTS

I would like to thank Prof. Bruno Lang for the opportunity to write this thesis, and for his invaluable support and guidance throughout the process.

I further wish to thank my fellow members of the ESSEX-II project, especially Christie Alappat, Andreas Alvermann, Georg Hager and Jonas Thies for helpful software and organizing access to computational resources.

Professors Yasunori Futamura, Akihiro Ida, Akira Imakura, Toshiyuki Imamura, and Tetsuya Sakurai, along with Masatoshi Kawai and other members of their research groups, hosted me in fruitful research visits to Tsukuba, Tokyo and Kobe and provided direction on some of the topics considered in this thesis.

I am grateful to my fellow Angewandte Informatik group members, past and present, for a pleasant work environment and many an interesting discussion during coffee breaks. In particular, Martin Galgon was involved with helpful discussion and assistance on many mathematical, computational, and technical issues related to this thesis. I am also grateful to him, along with Liam Huber, for proofreading this work.

Finally, I wish to thank my family, and in particular, Liam, for their support and encouragement of my studies.

ABSTRACT

This thesis is concerned with improving and expanding projection based methods for Hermitian interior eigenvalue problems. The focus is on methods of this type utilizing a rational function based filtering approach. We outline various strategies and novel variations to improve the efficacy of these methods and their suitability for solving large sparse eigenvalue problems. The potential for improvement is shown in numerical experiments.

This research has been supported by the Deutsche Forschungsgemeinschaft through the priority project Software for Exascale Computing (SPPEXA) within the ESSEX-II project (SPP 1648).

FOREWORD

The research presented in this thesis contains results and discussion that have been previously presented and published. In the introduction of each of Chapters 3 and 5, a brief description of the portions of the respective chapter corresponding to existing publications and distinguishing unpublished results is provided. We also provide a summary here.

Chapter 3 is based on results and discussion shown in

- M. Galgon, S. Huber, and B. Lang, *Mixed precision in subspace iteration-based eigensolvers*, PAMM, 18 (2018)
- A. Alvermann, A. Basermann, H.-J. Bungartz, C. Carbogno, D. Ernst, H. Fehske, Y. Futamura, M. Galgon, G. Hager, S. Huber, T. Huckle, A. Ida, A. Imakura, M. Kawai, S. Köcher, M. Kreutzer, P. Kus, B. Lang, H. Lederer, V. Manin, A. Marek, K. Nakajima, L. Nemec, K. Reuter, M. Rippl, M. Röhrig-Zöllner, T. Sakurai, M. Scheffler, C. Scheurer, F. Shahzad, D. Simoes Brambila, J. Thies, and G. Wellein, *Benefits from using mixed precision computations in the ELPA-AEO and ESSEX-II eigensolver projects*, Japan Journal of Industrial and Applied Mathematics, 36 (2019), pp. 699–717

where the implementation of mixed–precision in the latter publication has been provided by co-author Martin Galgon. These topics have also been previously presented:

- S. Huber, M. Galgon, and B. Lang, *Mixed precision in a large iterative parallel eigensolver framework: BEAST*, International Workshop on Eigenvalue Problems: Algorithms; Software and Applications, in Petascale Computing, Mar. 2018
- ——, *Recent developments and results for the BEAST eigensolver*, in R-CCS Cafe, RIKEN, Kobe, Japan, Oct. 2018

- ——, *Using mixed precision in iterative eigensolvers*, in GAMM Annual Meeting, Munich, Germany, Mar. 2018

Portions of the material and discussion shown in Chapter 4 has been presented:

- S. Huber and B. Lang, *Optimizing rational filters for the scalable solution of large eigenproblems*, in 2021 SIAM Conference on Computational Science and Engineering, Mar. 2021

Chapter 5 is based in part on results and discussion shown in

- S. Huber, Y. Futamura, M. Galgon, A. Imakura, B. Lang, and T. Sakurai, *Flexible subspace iteration with moments for an effective contour integration-based eigensolver*, Submitted to Numerical Linear Algebra with Applications, (2020)
- C. L. Alappat, A. Alvermann, A. Basermann, H. Fehske, Y. Futamura, M. Galgon, G. Hager, S. Huber, A. Imakura, M. Kawai, M. Kreutzer, B. Lang, K. Nakajima, M. Röhrig-Zöllner, T. Sakurai, F. Shahzad, J. Thies, and G. Wellein, *ESSEX: Equipping Sparse Solvers For Exascale*, in Software for Exascale Computing - SPPEXA 2016-2019, H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel, eds., vol. 136, Springer International Publishing, Cham, 2020, pp. 143–187

Again, significant portions of the implementation required for the timing results shown were performed by co–author Martin Galgon. Results have also been presented:

- S. Huber, Y. Futamura, M. Galgon, B. Lang, and T. Sakurai, *Reducing linear system size with moment based methods in the BEAST framework*, in 31st Advanced Supercomputing Environment (ASE) Seminar, University of Tokyo, Sept. 2017
- ——, *Using the Moment to Reduce Linear System Size*, in 18th SIAM Conference on Parallel Processing for Scientific Computing, Tokyo, Japan, Mar. 2018
- ——, *Contour Integration and Moments for the Solution of Large Eigenproblems*, in 2019 SIAM Conference on Computational Science and Engineering, Spokane, USA, Feb. 24

CONTENTS

MOTIVATION AND OUTLINE

This thesis considers strategies for the efficient, accurate and robust solution of large, sparse Hermitian interior eigenvalue problems. Traditionally, interior eigenvalue problems, which involve finding all eigenvalues and associated eigenvectors for a given interval, have been considered particularly challenging in comparison with problems involving finding the largest or even smallest eigenvalues and eigenvectors. Projected subspace methods, and in particular, contour integration-based methods, such as FEAST and Sakurai–Sugiura methods (SSM) have been established over the past two decades as some of the most promising strategies for the scalable solution of these problems.

Efficient strategies for computation are important as these methods are applied in more and more areas of scientific computation. For example, quantum mechanical calculations for materials science and chemistry are a major area of application. A significant portion of core hours worldwide are used to model materials at the atomic level, and the solution of interior eigenproblems is often the most significant bottleneck in these calculations. Improving these methods could save significant computational resources, and enable the computational modelling of larger, more realistic physical systems.

Furthermore, as the computational resources at hand expand, we require methods that are well equipped to utilize them. Methods that are suitable for division into smaller independent sub-problems are of particular interest, as these can be solved in parallel across a computational platform. We will discuss in this thesis how these methods are inherently parallelizable. The main objective in this work is then to show how the efficiency of these methods may be improved while maintaining or improving accuracy, robustness, and scalability.

We refer to *efficiency* as the relative cost of solution for an eigenproblem, which is commonly measured in terms of absolute time, computational resources used, energy usage, or a combination of these. *Accuracy* is defined in terms of the achievable or achieved residual error of a method for a given problem. Higher accuracy is typically associated with a greater cost, particularly for the iterative methods considered in this thesis. We are also concerned with how resilient a method is to error or failure to solve the problem; this is the *robustness* of the method. And finally, the *scalability* of a method describes the degree to which it can be broken into small sub-problems that may be solved simultaneously. Improved scalability is associated with being able to solve problems in a time proportionate to the amount of computational resources available. These aspects of a numerical method control its applicability in high performance computing environments. They are the qualitative or quantitative metrics that we use to evaluate the various algorithmic possibilities explored in this thesis.

We begin in Chapter 1 by outlining basic definitions and ideas for the problems and methods that we consider. This includes the general introduction of several fundamental iterative eigensolvers that form a basis for projected or filtered subspace schemes.

We continue the introduction in Chapter 2, where we define subspace filtering as a form of spectral projection. We focus on filters based on rational functions, which can also be extended to project subspaces with multiple moments. This type of filter is inherent to FEAST and SSM, the foundational algorithms of this thesis. In fact, the original conception of these algorithms focused on the derivation of a rational filter as an approximation of a contour integral, and they were commonly labelled as contour integration-based eigensolvers. Further work has shown that they may be used with general rational filters. We define a rational filter-based eigensolver as a subtype of projected subspace methods, extensible to subspace iteration. In this chapter, we also discuss the computational attributes required and some algorithmic considerations for the efficient implementation of a projected subspace method, and then outline the software framework, `BEAST`, that will be used to implement and evaluate the algorithmic ideas under consideration.

In Chapter 3, we consider how we may save computational effort by reducing the precision used for early subspace iterations, without affecting the overall achievable residual accuracy. This chapter includes experiments and discussion shown in part in [31], where we see that a single step of a subspace iteration computed in a lower precision will generally cause stagnation of the entire method. However, as we have originally shown in [8], convergence up to a certain residual threshold is typically not slowed down by reduced precision for the entire algorithm.

Chapter 4 focuses on improving the applicability of rational filtering schemes for increasing problem size via the interplay between these schemes and iterative linear solvers. The construction of the projected subspace with a rational filter requires the solution of shifted linear systems of equations. As the problems under consideration grow in size, iterative linear solvers become necessary to solve these linear systems in an efficient manner. However, the conditioning of these problems is controlled by the properties of the rational filter as well as the original (unshifted) linear system. We consider whether we can predict the number of iterations required for an iterative linear solver, and use this prediction to create a rational filter that reduces the overall iterative linear solver cost. We show that for the sample problems considered, this may be more efficient and scalable than standard choices of rational filter. This thesis is the first publication of this work.

We consider the flexible use of moments in rational filter-based methods in Chapter 5, bridging the original FEAST and SSM algorithms and reducing the cost of the solution of linear systems. As we have originally discussed in [46], moments can be effectively used to reduce the number of right hand sides needed in the solution of linear systems over all subspace iterations. However, adaptivity in the number of moments used may be helpful in improving the robustness of these methods and preventing stagnation. We outline the flexible strategy used, and extend on [46] by discussing several algorithmic considerations in greater detail. We also discuss the scalability of these schemes, as we have also shown in [4]. Furthermore, we consider the choice of quadrature degree with respect to the linear system solution cost over a rational filter-based eigensolver. The thesis concludes with an outlook for future work.

CHAPTER 1

INTRODUCTION

1.1. Fundamentals from numerical linear algebra

We assume the reader is familiar with the basics of numerical linear algebra, including vectors and matrices, and common operations, such as norms and products. We furthermore assume knowledge of fundamental concepts of numerical and computational mathematics, such as error (e.g., discretization, machine) and convergence. We proceed with a few basic definitions essential to the thesis. These can also be found in any basic numerical methods textbook, e.g., [93].

1.1.1. Hermitian and Hermitian positive definite matrices

A Hermitian matrix $A \in \mathbb{C}^{n \times n}$ has entries that satisfy $a_{i,j} = \overline{a}_{j,i}$. In the real case, this is clearly equivalent to a symmetric matrix. A Hermitian matrix is by definition square. A positive definite matrix satisfies $x^H Ax > 0$ for all vectors $x \neq 0$.

1.1.2. Orthogonality and B-orthogonality

We define two vectors x and y as orthogonal if $x^H y = 0$. If these vectors are also of unit length ($\|\cdot\| = 1$), they are additionally orthonormal. This definition may be extended to matrices or block vectors, $X \in \mathbb{C}^{m \times n}$, which are defined as orthogonal if X's columns x_i for $i = 1, \ldots, n$ are orthonormal, satisfying $x_i^H x_j = \delta_{i,j}$.

We may also define vectors x and y as B-orthonormal when they have unit length with respect to B and $x^H By = 0$. A matrix X is B-orthogonal if the columns are B-orthonormal, that is, $X^H BX = I$ for I the appropriately sized identity matrix.

1.1.3. Sparse matrices

The methods considered in this thesis are designed for *sparse* matrices, for which most elements of the matrix are zero. This means that matrix operations and storage are significantly cheaper (ideally growing linearly with matrix size) than for *dense* matrices, which have mostly or all non-zero elements. In many cases, especially for *iterative* methods, which generate successive approximate solutions to a matrix problem, a method may be much cheaper to apply to a sparse than a dense matrix. This implies that with the same computational power, we can consider much larger sparse problems than dense ones.

1.1.4. Projection

A projector is a matrix $P \in \mathbb{C}^{n\times n}$ satisfying $PPx = Px$ for all $x \in \mathbb{C}^n$. The following properties are also true for general matrices, but are introduced here as they reveal the desired action of a projector. The projected vector $y = Px$ can be written as a linear combination of the columns in P. For a projector P with columns p_i, $i = 1, \ldots, n$, and $x \in \mathbb{C}^n$, $Px = \sum_{i=1}^n x_i p_i$. An alternative definition is that the projected vector lies in the space spanned by the columns of P, $\operatorname{span}\{p_1, \ldots, p_n\}$. Clearly, a set of vectors or matrix can also be projected. The vectors generated by the product of this projection then lie in the space spanned by the columns of P.

1.1.5. Matrix decomposition

When considering eigenvalues and eigenvectors, as we will do next, it is important to be able to decompose a matrix $A \in \mathbb{C}^{m\times n}$. This can be considered as writing A as a product of other matrices, typically in some convenient form. Before turning to eigendecompositions, we define the more general *singular value decomposition* of a matrix. This is a decomposition satisfying

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

where $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary matrices, and Σ is a $\mathbb{C}^{m\times n}$ matrix with non-negative entries $\sigma_i \geq 0$ only along the main diagonal.

1.2. Eigenvalues and eigenvectors

If we consider a *matrix pair* (A, B), that is, two square matrices, $A, B \in \mathbb{C}^{n\times n}$, we can define the *eigenvectors* as the vectors for which multiplication with A is equivalent to multiplication with B and some constant $\lambda \in \mathbb{C}$, an *eigenvalue*. The generalized eigenvalue problem, or *eigenproblem*, is then finding x, λ that satisfy

$$Ax = \lambda Bx. \tag{1.2}$$

This definition, and equivalent definitions in the remainder of this chapter, also apply to *standard eigenvalue problems*, where $B = I$, and multiplication of an eigenvector x with A is equivalent to multiplication with a scalar eigenvalue λ

$$Ax = \lambda x. \tag{1.3}$$

A pair x, λ satisfying (1.2) or (1.3) may be called an *eigenpair* of the generalized or standard eigenproblem respectively. If a matrix pair is *diagonalizable*, it is possible to find a set of linearly independent vectors $X \in \mathbb{C}^{n\times n}$ such that

$$A = BX\Lambda X^{-1} \tag{1.4}$$

where $\Lambda \in \mathbb{C}^{n\times n}$ is a diagonal matrix containing the eigenvalues λ_i along its diagonal, and the i^{th} column of X contains the i^{th} (normalized) eigenvector. This may be referred to as the *eigendecomposition* of a matrix. Equivalently, if B is invertible,

$$X^{-1}B^{-1}AX = \Lambda. \tag{1.5}$$

Invertibility of B further implies that the generalized eigenvalue problem reduces to a standard eigenvalue problem, satisfying

$$B^{-1}A = X\Lambda X^{-1} \tag{1.6}$$

We define an *eigenspace* as a space spanned by a set of eigenvectors, $\operatorname{span}\{x_i \,|\, i = 1, \ldots, n\}$. The *spectrum* is the set of eigenvalues $\{\lambda_i \,|\, i = 1, \ldots, n\} = \operatorname{diag}(\Lambda)$. For the sake of simplicity in definitions and notation, we assume that all eigenvalues and eigenvectors are *simple*; that is, each eigenvalue corresponds to a single eigenvector and vice-versa. However, the theory typically generalizes to problems beyond this case.

1.2.1. Hermitian eigenproblems

If $A \in \mathbb{C}^{n\times n}$ is a Hermitian matrix and $B \in \mathbb{C}^{n\times n}$ is Hermitian positive definite (HPD), then the eigenproblem may be called *Hermitian*, and the matrix pair (A, B) a *definite pair*. If this is the case then the problem is diagonalizable and several convenient properties follow. Firstly, all eigenvalues are real. We can define the largest and smallest eigenvalues in the spectrum as $\lambda_{\max}$ and $\lambda_{\min}$ respectively. Next, the eigenvectors X are B-orthogonal, and thus $X^{-1} = X^H B$. As the inverse must satisfy $X^{-1}X = XX^{-1} = I$, $X^H BX = XX^H B$. Our eigendecomposition may also be written as

$$B^{-1}A = \sum_{i=1}^{n} \lambda_i x_i x_i^{-1} \tag{1.7}$$

where $x_i^{-1} = x_i^H B$. Since B is Hermitian positive definite, it is also invertible. We assume that the eigenproblem is Hermitian in the remainder of this thesis.

1.2.2. Interior eigenproblem

The fundamental problem considered in this thesis is that of finding all eigenvalues λ in a given interval $I_\lambda = \left[\underline{\lambda}, \overline{\lambda}\right]$ with corresponding eigenvectors x, of a given definite pair, (A, B). In this scenario, as stated above, we know that there exists an eigendecomposition with real eigenvalues and B-orthogonal eigenvectors of the form $A = BX\Lambda X^{-1}$. Thus, the fundamental problem is

$$\begin{gathered} Ax = \lambda Bx \\ \lambda \in I_\lambda = \left[\underline{\lambda}, \overline{\lambda}\right]. \end{gathered} \tag{1.8}$$

We will define the block vector X_{I_λ} with columns made up of the eigenvectors x_i such that the corresponding eigenvalue is $\lambda_i \in I_\lambda$. The space spanned by the columns of X_{I_λ} can be defined as

$$\mathcal{X}_{I_\lambda} = \operatorname{span}\{x_i \,|\, \lambda_i \in I_\lambda\}. \tag{1.9}$$

This space, as well as the entire eigenspace, is an *invariant subspace* under $B^{-1}A$. This means that the space does not change through multiplication with this matrix, as $B^{-1}A(\mathcal{X}_{I_\lambda}) \subseteq \mathcal{X}_{I_\lambda}$.

The type of eigenvalue problem under consideration changes the selection of methods that are available. We consider problems where the direct solution of the entire eigenvalue problem, that is, simply finding all eigenvalues and eigenvectors, is not feasible or efficient. We also assume that the interval of interest is not always located at an edge of the spectrum (though this is one use case.) Depending on the size of the problem under consideration and the computational resources at hand, one possibility to refine the problem is to subdivide I_λ and solve each eigenproblem separately. This problem, including the subsequent difficulty of ensuring the B-orthogonality of eigenvectors across different sub-problems, is considered in [30].

1.2.3. Spectral projection

Let us assume that a block vector X is B-orthogonal and the columns of X are the eigenvectors (or a selection thereof) of (A, B). We assume that $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix with entries of either 1 or 0, and thus that $D^2 = D$. Then, XDX^HB is a projector. This follows as

$$\begin{aligned}(XDX^HB)(XDX^HB) &= \\ XD^2X^HB &= \\ XDX^HB.\end{aligned}$$

Thus our matrix satisfies the definition of a projector, and will project into the eigenspace spanned by the columns of X.

1.3. Building blocks for iterative eigensolvers

Most methods for obtaining the eigenvalues and eigenvectors of a matrix, or a matrix pair, can be categorized as “direct,” obtaining a complete eigendecomposition of a matrix, often via decomposition, or “iterative,” obtaining successively improving approximations of one, some, or all eigenvalues and eigenvectors via an iterative scheme, and often relying on the projection of a vector or set of vectors. We will consider eigensolvers of the second type in this thesis, beginning by briefly introducing several methods with foundational ideas and strategies. We will consider only standard eigenvalue problems for the moment.

1.3.1. Power method

The most fundamental iterative method for eigenvalues problems is the power method. A description and proof of convergence can be found in [82]. We outline the method in Algorithm 1. Here, the repeated multiplication of a vector v

with a matrix A generates a sequence where $A^j v$ will approach a scalar multiple of x_1, the eigenvector corresponding with the largest eigenvalue of A, assuming that the associated eigenvalue λ_1 satisfies $\lambda_1 > 0$.

Algorithm 1: Power method

```
Choose a starting vector $v_0$
while not converged do
    $v_j = Av_{j-1}$
    $v_j = \frac{v_j}{\|v_j\|}$
    $j = j + 1$
end
```

The idea of successive applications of a set of vectors, in this case, the matrix A itself, causing iterations to approach a desired eigenspace is an essential concept. A related idea is to consider the entire subspace spanned by iterations of $A^j v$, which collectively form a *Krylov subspace.*

1.3.2. Krylov subspace methods

The j^{th} Krylov subspace built from a matrix A and vector v has the form

$$K_j(A, v) = \text{span}\left\{v, Av, A^2v, \dots A^{j-1}v\right\}. \tag{1.10}$$

Krylov subspaces are a fundamental building block of iterative methods in general; they are defined and discussed in more detail in most standard texts on numerical methods, for example [82]. In this context we could consider the power method as providing only the final vector in $K_j(A, v)$, $A^{j-1}v$. Krylov subspace methods, in contrast, gather an approximate solution based on the subspace spanned by all vectors. Thus they may be more powerful than a scheme considering just the final vector, but the additional costs of storage and orthogonalization are also significant.

The most fundamental Krylov subspace method is the Arnoldi method, as described in [82], which is an orthogonal projection method onto $K_j(A, v)$.

The method begins with a single *Arnoldi vector*, and in each iteration, another direction is added to the basis. In j^{th} iteration, the algorithm will:

- Extend the Krylov subspace by one direction (multiply previous Arnoldi vector v_j by A).
- Orthonormalize the resulting vector against all other Arnoldi vectors v_i, $i = 1, \dots j$.

- End if the new vector is completely linearly dependant on existing Arnoldi vectors (no new directions possible for Krylov subspace).

After j iterations, the resulting vectors form an orthonormal basis of $K_j(A, v)$. From this basis, the j largest eigenvalues and eigenvectors of A may be extracted. Alternative strategies are possible to obtain, e.g., the smallest j eigenvalues.

1.3.3. Basic subspace iteration

Subspace iteration, like the Arnoldi method, also constructs a subspace with j vectors as an approximate basis for the largest j eigenvectors. However, it is in principle more related to the power method in that it saves only the j^{th} application of A to a block of vectors Y. A simple subspace iteration, as introduced in [82] and outlined in Algorithm 2, computes, for a standard eigenvalue problem, a subspace corresponding to the j eigenvectors with largest eigenvalues.

Algorithm 2: Subspace iteration

```
Choose a starting block of $j$ vectors $Y$
while not converged do
    Compute $U = AY$
    Orthogonalize $U$
    Set $Y = U$
end
```

As in the power method, the repeated application of A causes the vectors in U to approach the largest eigenvector. Together with the subsequent reorthogonalization (via, e.g., a QR factorization) the vectors will approach a basis for the j eigenvectors associated with the j largest eigenvalues. A proof of this is given in [82].

1.3.4. Rayleigh–Ritz

Algorithm 3: Rayleigh–Ritz

```
Input  : $A, B \in \mathbb{C}^{n \times n}$, $U$ approximate basis for $\mathcal{X}_{I_\lambda}$
Output: Approximate eigenvectors $\tilde{X}_{I_\lambda}$
Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
Compute full sized approximate eigenvectors $\tilde{X}_{I_\lambda} = UW$
```

The discussion in this section so far has focused on obtaining an approximate subspace for the desired eigenvectors. Another important consideration is how we may extract the eigenpairs from that subspace. A standard combination with a subspace iteration-type scheme is a Rayleigh–Ritz procedure, which, as described in [82], may be used to extract the approximate eigenpairs. The algorithm, as shown in Algorithm 3, computes the *Ritz pairs* $\lambda_i, \tilde{x}_i, i = 1, \ldots, n$, as approximate eigenpairs of (A, B). These are shown in the algorithm in block form, as Λ and $\tilde{X}_{I_\lambda}$. As described in [65], this approximate solution is generated by an approximate similarity transformation of (A, B) into the subspace spanned by the vectors of U.

This is also the standard method we will use for extracting eigenpairs in the remainder of this thesis. If U contains not just an approximate basis for $\mathcal{X}_{I_\lambda}$ but other directions that are not related to eigenvectors of (A, B), these Ritz pairs will also be computed. We refer to these as *spurious* eigenpairs; they are easy to distinguish from the desired eigenpairs when convergence of these eigenpairs has proceeded towards a smaller residual threshold. Further discussion on spurious eigenpairs in the context of subspace iteration methods can be found in [30].

CHAPTER 2

SUBSPACE FILTRATION METHODS

2.1. Subspace filtration

A filter works like a sieve, and subspace iteration like shaking the sieve to separate wanted eigenvector directions from unwanted. We want to know how to make a good sieve that will get rid of the “dirt” quickly while not letting any “gold” fall through the cracks. The general strategies considered in this thesis for solving (1.8) involve constructing an approximate basis for the desired eigenvectors, that is, those with eigenvalues in I_λ. Once this basis has been constructed, we can subsequently use a suitable method to extract the approximate eigenvalues and eigenvectors. Iterations of “filtering” a set of vectors for directions corresponding to those of the desired eigenvectors may be used, as well as combined, in order to construct this approximate basis. The idea of an approximate spectral projector is inherently related to subspace filtration, as directions are “projected towards” or “filtered” for the desired directions in $\mathcal{X}_{I_\lambda}$.

2.1.1. Partial spectral projector

We begin by considering our ideal spectral projector in 1D, h,

$$h(x) = \begin{cases} 1 \text{ for } x \in I_\lambda \\ 0 \text{ for } x \notin I_\lambda, \end{cases} \tag{2.1}$$

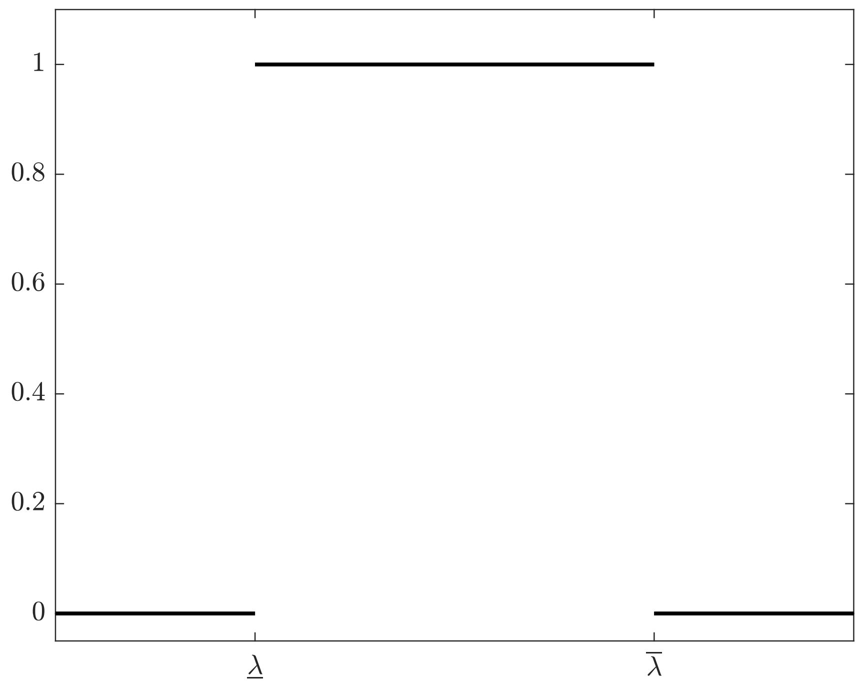


Figure 2.1.: Visualization of (2.1).

which corresponds to the window function, as shown in Figure 2.1.

We can define a corresponding matrix function, $h(B^{-1}A)$. Given the eigendecomposition of $B^{-1}A$, we require our filter to satisfy $h(B^{-1}A) = h(X\Lambda X^{-1}) = Xh(\Lambda)X^{-1}$. With the above definition of $h(x)$,

$$\begin{aligned} &= Xh(\Lambda)X^{-1}, \\ &= \sum_{i=1}^{n} h(\lambda_i)x_i x_i^{-1}, \\ &= X_{I_\lambda} X_{I_\lambda}^{-1} \end{aligned}$$

where again, X_{I_λ} is a block vector containing the exact eigenvectors with eigenvalues in I_λ. Equivalently, given the B-orthogonality of the eigenvectors, $h(B^{-1}A) = X_{I_\lambda} X_{I_\lambda}^H B$. This is a *spectral projector* into $\mathcal{X}_{I_\lambda}$.

As we cannot obtain this exact filter without prior knowledge of the eigendecomposition, we may also consider what happens if we have a function $\tilde{h}(B^{-1}A)$ such that, in 1D,

$$\tilde{h}(x) \approx \begin{cases} 1 \text{ for } x \in I_\lambda \\ 0 \text{ for } x \notin I_\lambda. \end{cases} \tag{2.2}$$

If $\tilde{h}(B^{-1}A) = X\tilde{h}(\Lambda)X^{-1}$, this matrix will be an approximate spectral projector into $\mathcal{X}_{I_\lambda}$. This may be applied to a set of m_U vectors $Y \in \mathbb{C}^{n\times m_U}$ to approximately project them into $\mathcal{X}_{I_\lambda}$. Here we define

$$U = \tilde{h}(B^{-1}A)Y$$

where the projected vectors, $U \in \mathbb{C}^{n\times m_U}$, act as an approximate basis for $\mathcal{X}_{I_\lambda}$.

We may add a filtering scheme to subspace iteration in order to accelerate the process (as discussed in [82]) or to search for eigenpairs with eigenvalues inside a given interval, instead of just the largest. Subspace *filtering* techniques may be discussed synonymously with *projection* or *acceleration* schemes, as the filter usually relies on projection into the desired eigenspace, and accelerates the iteration. The filter used, $\tilde{h}$, controls the quality of the resulting approximate eigenspace, and is the focus of much of the research for subspace iterative methods.

Algorithm 4: Filtered subspace iteration

```
Choose a starting block of $m_U$ vectors $Y$
while not converged do
    Compute $U = \tilde{h}(B^{-1}A)Y$
    Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
    Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
    Compute approximate eigenvectors $X = UW$
    Set $Y = X$
end
```

We observe that methods of this type allow us to reduce a large, sparse partial eigenproblem to a dense problem proportional to the desired slice of the spectrum. We next consider some common techniques for obtaining filtering functions of type $\tilde{h}$. As in general with an interpolation problem, rational or polynomial approximation functions may allow us to obtain a good approximation of h.

2.2. Rational filters

The main filtering method we will consider in this thesis is those arising from rational functions, which we define as a function of degree q that can be written as

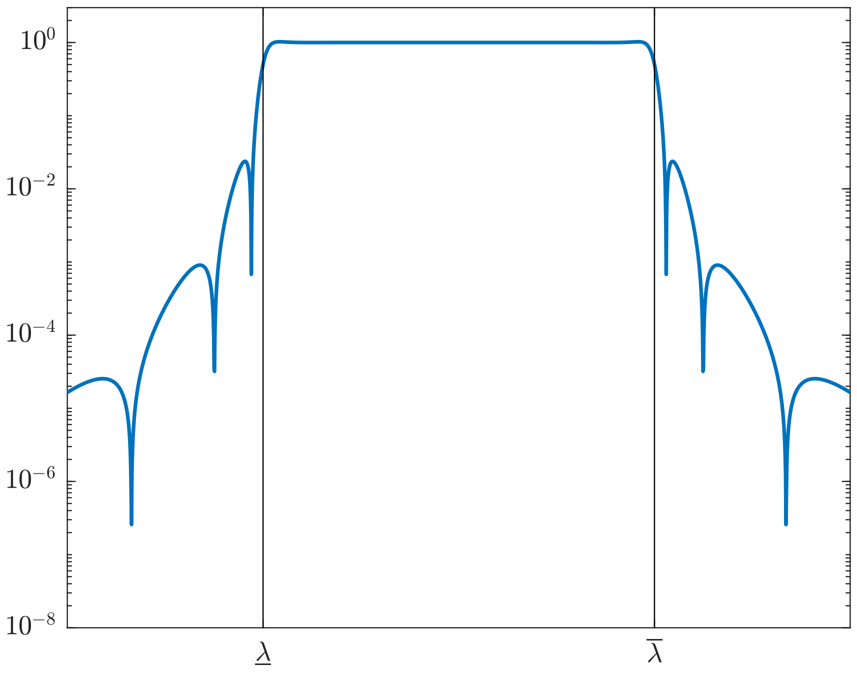


Figure 2.2.: Visualization of absolute value of rational filter (2.3), Gauss–Legendre filter with $q = 16$.

$$r(x) = \frac{f(x)}{g(x)}$$

where $f(x)$ and $g(x)$ are relatively prime monic polynomials with maximum degree q. We will consider functions of the form

$$r(x) = \sum_{i=1}^{q} \frac{\omega_i}{z_i - x} \tag{2.3}$$

where z_i and ω_i, the *poles* and *weights*, are chosen such that $r(x)$ approximates the window function (2.1). Typically z_i and w_i are chosen as complex values, even though the eigenvalues we seek are real. This is in part because of the way rational filters have traditionally been defined as arising from contour integrals, as we will discuss below, but also because the distance between a pole z_i and the closest eigenvalue in the spectrum is significant in determining the cost of the method, as we will discuss in Chapter 4. Note that it is possible to have a repeated pole with power greater than one, but this case will not be considered here; some consideration is given in [97]. The corresponding matrix function is defined as

$$\begin{aligned} r(B^{-1}A) &= \sum_{i=1}^{q} \omega_i (z_i I - B^{-1}A)^{-1} \\ &= \sum_{i=1}^{q} \omega_i (z_i I - B^{-1}A)^{-1} B^{-1} B, \end{aligned}$$

giving us the traditional final matrix form of

$$r(B^{-1}A) = \sum_{i=1}^{q} \omega_i (z_i B - A)^{-1} B. \tag{2.4}$$

Indeed, when we consider the filter applied to the eigendecomposition of (A, B),

$$\begin{aligned} r(B^{-1}A) &= \sum_{i=1}^{q} \omega_i (z_i I - X\Lambda X^{-1})^{-1} \\ &= \sum_{i=1}^{q} \omega_i (X(z_i I - \Lambda)X^{-1})^{-1} \\ &= X(\sum_{i=1}^{q} \omega_i (z_i I - \Lambda)^{-1})X^{-1} \\ &= X r(\Lambda) X^{-1} \end{aligned}$$

Thus, $r(B^{-1}A)$ is an approximate spectral projector into $\mathcal{X}_{I_\lambda}$. When it is applied to a set of vectors Y, the numerical problem becomes the solution of a sequence of linear system of equations with multiple right hand sides

$$r(B^{-1}A)Y = \sum_{i=1}^{q} \omega_i (z_i B - A)^{-1} BY. \tag{2.5}$$

We note that if A, B and Y are real, and if the weights w_i and poles z_i satisfy $w_i = \overline{w}_{i+q/2}$, $z_i = \overline{z}_{i+q/2}$, $i = 1, \ldots, q/2$ (that is, the poles are symmetrically placed above and below the real axis), we only need to solve half the linear systems. This occurs, as pointed out in [77], because $\overline{w}_i(\overline{z}_i B - A)^{-1}BY = \overline{w_i(z_i B - A)^{-1}BY}$, and thus

$$
\begin{aligned}
r(B^{-1}A)Y &= \sum_{i=1}^{q} \omega_i (z_i B - A)^{-1} BY \\
&= \sum_{i=1}^{q/2} \omega_i (z_i B - A)^{-1} BY + \sum_{i=q/2+1}^{q} \omega_i (z_i B - A)^{-1} BY \\
&= \sum_{i=1}^{q/2} \omega_i (z_i B - A)^{-1} BY + \bar{\omega}_i (\bar{z}_i B - A)^{-1} BY \\
&= \sum_{i=1}^{q/2} \omega_i (z_i B - A)^{-1} BY + \overline{\omega_i (z_i B - A)^{-1} BY} \\
&= \sum_{i=1}^{q/2} 2\mathrm{Re}(\omega_i (z_i B - A)^{-1} BY).
\end{aligned}
$$

2.2.1. Contour integration

An alternative approach, also resulting in a rational filter, comes from the perspective of contour integration. The following analysis is based on [65] and [2].

We restate the Cauchy integral formula, as defined in [2], which says that for Γ a continuous, simply closed [2] curve surrounding a simply connected domain $\Omega \in \mathbb{C}$, an analytic function $f(z)$ and a point $x \notin \Gamma$,

$$
\frac{1}{2\pi \mathbf{i}} \int_\Gamma \frac{f(z)dz}{z-x} = \begin{cases} 0, \; x \notin \Omega \\ f(x), \; x \in \Omega. \end{cases} \tag{2.6}
$$

To restate this in simple terms, if a point x lies on the inside of a curve, the integral formula will return the value of the function of f at that point. Otherwise, since $\frac{f(z)}{z-x}$ contains no poles inside the area of integration, the resulting integral is 0.

The simplest example of this is if $f = 1$, returning

$$
c(x) = \frac{1}{2\pi \mathbf{i}} \int_\Gamma \frac{dz}{z-x} = \begin{cases} 0, \; x \notin \Omega \\ 1, \; x \in \Omega. \end{cases} \tag{2.7}
$$

If we consider Ω such that $\mathbb{R}(\Omega) = I_\lambda$, as illustrated in Figure 2.3, this equates to the window function h on I_λ. In order to include the boundaries $\underline{\lambda}$ and $\overline{\lambda}$ in this representation, we must assume that these lie (infinitesimally) inside Γ. We may also make the simplifying assumption that $\underline{\lambda}$ and $\overline{\lambda}$ are not eigenvalues of (A, B).

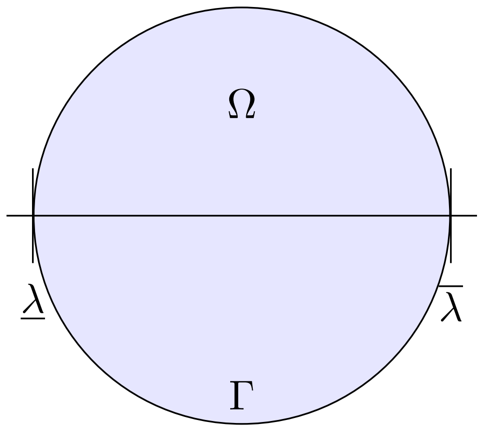


Figure 2.3.: Visualization of Ω, Γ and I_λ in complex plane.

We typically consider contours Γ such as circles or ellipses; the parametrization will be discussed later. If we define the corresponding matrix function $c(B^{-1}A)$ and apply it to the matrix pair, we obtain the exact spectral projector into $\mathcal{X}_{I_\lambda}$:

$$\begin{aligned}
c(B^{-1}A) &= \frac{1}{2\pi i}\int_\Gamma (zB-A)^{-1}B\,dz \\
&= \frac{1}{2\pi \mathbf{i}}\int_\Gamma (zB-BX\Lambda X^{-1})^{-1}B\,dz \\
&= \frac{1}{2\pi \mathbf{i}}\int_\Gamma (zI-X\Lambda X^{-1})^{-1}B^{-1}B\,dz \\
&= \frac{1}{2\pi \mathbf{i}}\int_\Gamma \sum_{i=1}^{n} \frac{1}{z-\lambda_i} x_i x_i^{-1}\,dz \\
&= \frac{1}{2\pi \mathbf{i}}\int_\Gamma \sum_{i=1}^{n} h(\lambda_i) x_i x_i^{-1}\,dz \\
&= X_{I_\lambda} X_{I_\lambda}^{-1}.
\end{aligned}$$

Using numerical quadrature, we may approximate (2.7) and obtain a rational filter of the form (2.3). This is the historical approach to rational filters for interior eigenvalue problems, with initial considerations by Sakurai and Sugiura in [86] and Polizzi in [77]. We will consider their approaches and resulting algorithms in more detail in the remainder of this work.

2.2.2. Numerical integration

Numerical integration is used to compute an approximation for a definite integral, using a sum with the integrand function $f(x)$ evaluated at a sequence of points x_i, $i = 1, \ldots, q$.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{q} w_i f(x_i) \tag{2.8}$$

We describe two simple numerical integration rules below in preparation for the discussion in the context of contour integration. Further discussion can be found in introductory numerical mathematics textbooks, e.g., [20]; introductions with a perspective on contour-integration for eigenvalue problems are [65] and [30].

The simplest rules for numerical integration are Newton-Coates rules, which build an approximation for an integral as an average over a series of points. When the points are equispaced, we obtain a composite quadrature rule. We introduce a simple rule below. This is followed by a rule based on orthogonal polynomials.

2.2.3. Composite midpoint rule

Let $h = \frac{1}{q}$. Then, the composite midpoint rule, for the equispaced points

$$x_i = \left(i - \frac{1}{2}\right) h, \quad i = 1, \ldots, q$$

is the composite form of the open Newton-Cotes formula of degree 1. The estimated value of the integral over each sub-interval is the average value of of the integrand at the interval midpoints, multiplied by the length of the sub-interval. This results in the approximation

$$\int_0^1 f(x)dx \approx h \sum_{i=1}^{q} f(x_i). \tag{2.9}$$

2.2.4. Gauss-Legendre quadrature

For a quadrature rule of degree q, x_i is the i^{th} root of the monic Legendre polynomial of degree q, a set of orthogonal polynomials defined by the recursive formula

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x).$$

where

$$P_0(x) = 1, \quad P_1(x) = x.$$

The weights are defined as

$$w_i = \frac{2}{(1-x_i^2)\left[P_n'(x_i)\right]^2}, \quad i = 1, \ldots, n$$

and quadrature rule is thus

$$\int_0^1 f(x)dx \approx \sum_{i=1}^{q} w_i f(x_i). \tag{2.10}$$

2.2.5. Mapping to a complex contour

The quadrature rules we have seen evaluate integrals over $[0,1]$, of form

$$\int_0^1 f(x)dx. \tag{2.11}$$

To use these approximations, we must define a transformation of our original integral over a contour $\Gamma \in \mathbb{C}$ to this domain. Integration by substitution gives us

$$\frac{1}{2\pi\mathbf{i}} \int_\Gamma f(z)dz = \frac{1}{2\pi\mathbf{i}} \int_0^1 s'(t)f(s(t))dt. \tag{2.12}$$

where we require s to be a function of the form $s\colon \mathbb{R} \to \mathbb{C}$, satisfying the parametrization $\Gamma = s(t)$, $t = [0,1)$.

Let us assume that Γ is a circle in the real/imaginary plane, as shown in Figure 2.4 with center

$$c = \frac{\overline{\lambda} + \underline{\lambda}}{2} \tag{2.13}$$

and radius

$$r = \frac{\overline{\lambda} - \underline{\lambda}}{2}. \tag{2.14}$$

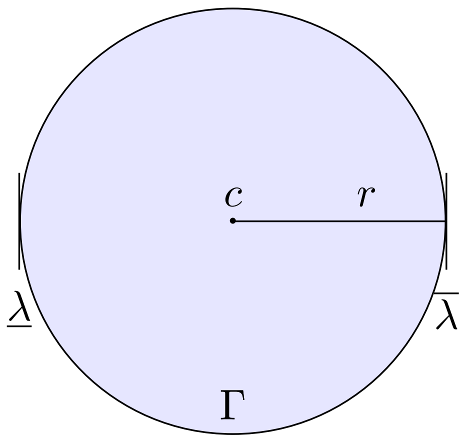


Figure 2.4.: Circular contour

Then, the parametrization

$$s = c + re^{2\pi\mathbf{i}t} \quad t = [0, 1) \tag{2.15}$$

satisfies the condition for substitution. We also require the derivative,

$$s'(t) = 2\pi\mathbf{i}re^{2\pi\mathbf{i}t} \quad t = [0, 1)\,. \tag{2.16}$$

Thus,

$$\frac{1}{2\pi\mathbf{i}}\int_0^1 s'(t)f(s(t))dt = \int_0^1 re^{2\pi\mathbf{i}t}f(c + re^{2\pi\mathbf{i}t})dt \tag{2.17}$$

This is also possible for other contours Γ, assuming a parametrization exists. We also consider elliptical contours. In this case, given an ellipse with height $2a$, center c and width $2r$, we can consider the parametrization

$$s(t) = c + r\cos(2\pi t) + a\mathbf{i}\sin(2\pi t) \quad t = [0, 1)\,. \tag{2.18}$$

Here we have the derivative

$$s'(t) = -2\pi r\sin(2\pi t) + 2\pi a\mathbf{i}\cos(2\pi t) \quad t = [0, 1)\,. \tag{2.19}$$

We typically define the ellipse with regards to its *eccentricity*, which is defined as $\sqrt{1 - \frac{a^2}{r^2}}$.

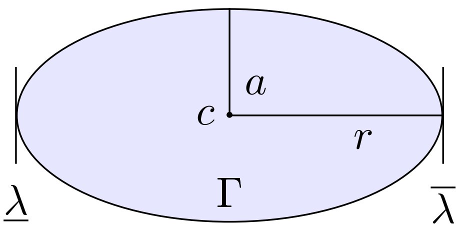


Figure 2.5.: Elliptical contour

Other contours, with, e.g., rectangular shapes, are also possible. Here, the possibility of a computational advantage arises as the transformed quadrature points are equally distanced from the real axis. This is a defining characteristic in the difficulty of the linear systems, as we will discuss in Chapter 4.

2.3. Moments and rational filters

For a given function $f(x)$, each *moment* extracts a different, related function or measurement. For a function $f(x)$, the p^{th} moment is defined as $\int z^p f(z)dz$. For example, moments are commonly defined in statistics with regards to the probability density function where, in rough terms, the first moment corresponds to the mean, the second to the variance, and the third to the skewness of the function, etc. Moments are also seen in physics, where the value of the p^{th} moment at some point refers to the integral over space of the distance from that point multiplied by the density of some physical quality (e.g., mass): $\int r^p \rho(r)dr$. If we consider moments in the context of our filtering functions, they might be seen as giving us different approximated "views" of the desired eigenspace $\mathcal{X}_{I_\lambda}$. If some directions of this space are not well represented in the filter corresponding to one moment, they might be better represented in another.

Just as shown in the last section, a rational filter can be defined either independently as an approximation of the window function on I_λ, given in (2.1), or as a numerical quadrature rule approximating a contour integral around I_λ. The use of moments in a projected subspace–based eigensolver has historically been defined in terms of the latter case. In this thesis, we will only consider multi-moment filters arising from contour integration rules. We consider, in 1D, the filters with different values of p satisfying

$$\frac{1}{2\pi\mathbf{i}} \int_\Gamma \frac{z^p}{z-x} dz \approx r^p(x) \tag{2.20}$$

where

$$r^p(x) = \sum_{j=1}^{q} \frac{\omega_j z_j^p}{z_j - x}. \tag{2.21}$$

As before, the Cauchy integral formula, as defined in (2.6), may be used for this integral, with $f(x) = x^p$. Thus,

$$c^p(x) = \frac{1}{2\pi\mathbf{i}} \int_\Gamma \frac{z^p}{z-x} dz = \begin{cases} x^p, \, x \in I_\lambda \\ 0, \, x \notin I_\lambda. \end{cases} \tag{2.22}$$

In this way, r^p is a function representing an approximation of an alternative version of the window function (2.1).

When we consider the resulting approximate spectral projector, the key feature we desire from each of our filters is to preserve or amplify eigendirections with eigenvalues inside I_λ and to reduce directions with eigenvalues outside of I_λ. Theoretically, the value of the window function inside of I_λ must simply be non-zero in order to satisfy this requirement. Furthermore, we do not consider the different moment-based filters in isolation, but as a whole, projecting a set of vectors according to each filter, then gathering the entire set of vectors as an approximate basis for $\mathcal{X}_{I_\lambda}$. We begin by considering the application of (2.21) to the matrix pair (A, B), as

$$r^p(B^{-1}A) = \sum_{j=1}^{N} \omega_j z_j^p (z_j B - A)^{-1} B. \tag{2.23}$$

As before in (2.4), in what can now be seen was the case of the 0^{th} moment, this filter will act on a set of random vectors and project them into the desired approximate eigenspace. Indeed, in [58], the use of multiple moments was shown to equate arithmetically to a single filter acting as an approximate projector on a block-Krylov subspace to create a projected subspace. We highlight key points of this analysis here, following the logic shown in [58], to guide the reader's understanding.

We begin by re-defining $r(x)$ as the (0^{th} moment) rational function in (4.1). Then, $r(B^{-1}A)$ is applied to Y to obtain U^0

$$U^0 = \sum_{i=1}^{q} w_i (z_i B - A)^{-1} BY. \tag{2.24}$$

If we consider the subspace resulting from the combination of s moments, the full subspace is constructed as

$$U = \left[U^0, \sum_{j=1}^{q} w_i z_i (z_i B - A)^{-1} BY, \dots, \sum_{j=1}^{q} w_i z_i^{s-1} (z_i B - A)^{-1} BY \right], \tag{2.25}$$

with the vectors corresponding to the p moment defined as

$$U^p = \sum_{j=1}^{q} w_i z_i^p (z_i B - A)^{-1} BY. \tag{2.26}$$

Thus, the equivalent definition of the full subspace is

$$U = \left[U^0, U^1, \dots, U^{s-1} \right]. \tag{2.27}$$

We can decompose U^0 into a the sum of the n eigenvectors of (A, B) as

$$\begin{aligned} U^0 &= \sum_{j=1}^{n} r(\lambda_j) x_j x_j^H BY \\ &= \sum_{j=1}^{n} \sum_{i=1}^{q} \frac{w_i}{z_i - \lambda_j} x_j x_j^H BY. \end{aligned}$$

Similarly, for the p^{th} moment,

$$U^p = \sum_{j=1}^{n} \sum_{i=1}^{q} \frac{w_i z_i^p}{z_i - \lambda_j} x_j x_j^H BY. \tag{2.28}$$

Finally, we assume that

$$\sum_{i=1}^{q} w_i z_i^p \begin{cases} \neq 0, (p = -1) \\ = 0, (p = 0, 1 \dots, q-2) \end{cases}. \tag{2.29}$$

The quadrature rules considered in this thesis (midpoint and Gauss–Legendre) have positive weights, and are exact for polynomials up to at least degree $q-2$. Furthermore, according to Cauchy's integral theorem, the integral of z^p over the closed curve Γ is 0, since this function is analytic in this region [2]. Therefore, this condition should be satisfied for all quadrature rules considered in this chapter. Then, as shown in [85],

$$\sum_{i=1}^{q} \frac{w_i z_i^p}{z_i - \lambda} = \lambda^k \sum_{i=1}^{q} \frac{w_i}{z_i - \lambda}. \tag{2.30}$$

This means that

$$U^p = \sum_{j=1}^{n} \lambda_j^p r(\lambda_j) x_j x_j^H BY \tag{2.31}$$

and thus in block form,

$$U^p = X r(\Lambda) \Lambda^p X^H BY. \tag{2.32}$$

If we define the s^{th} block-Krylov subspace $K_s = K_s(B^{-1}A, Y)$, where Y is a block vector, we obtain, via the eigendecomposition of $B^{-1}A$,

$$K_s = \left[Y, X\Lambda X^H BY, (X\Lambda X^H B)^2 Y, (X\Lambda X^H B)^{s-1} Y\right] \tag{2.33}$$

or equivalently

$$K_s = \left[Y, X\Lambda X^H BY, X\Lambda^2 X^H BY, \ldots, X\Lambda^{s-1} X^H BY\right], \tag{2.34}$$

then the subspace U can be written as

$$(X r(\Lambda) X^H B) K_s. \tag{2.35}$$

Again, $X r(\Lambda) X^H B$ is the approximate projector for the rational filter with 0^{th} moment into $\mathcal{X}_{I_\lambda}$ Thus, the generated subspace U is equivalent to the result of a projector, acting on the vectors of an order-s block-Krylov subspace formed with the matrix pair (A, B) and the block vector Y.

However, the full rank of the block Krylov subspace is not guaranteed, and it is possible that linear dependence may occur, hampering the effectiveness of these methods. This may be offset by orthogonalizing U after construction, increasing subspace size, or limiting the number of moments. The effects and potential countermeasures are discussed further in Chapter 5.

2.4. Rational filter-based eigensolvers

We define a rational filter-based eigensolver, or RFE, as a projected subspace method for solving (1.8) that utilizes a rational filter for approximate projection into the desired eigenspace. This class of methods has often been summarized with the label of "contour integral–based eigensolvers," but we generalize the name here to include rational filters beyond those deriving explicitly from contour integration. As above, we consider rational filters $r(x)$ of degree q with shifts, z_i and weights w_i, $i = 1, \dots, q$, such that for $x \in \mathbb{R}$

$$r(x) = \sum_{i=1}^{q} \frac{w_i}{z_i - x}.$$

As we have seen up to this point, a rational filter, and thus an RFE may involve multiple moments or simply the 0^{th} moment. The FEAST algorithm, as introduced by Polizzi [77], is the fundamental method for the single–moment case. In this algorithm, a set of m_U initial vectors $Y \in \mathbb{C}^{n \times m_U}$, usually chosen randomly, is projected by the approximate spectral projector based on the rational filtering scheme $r(B^{-1}A)$. The result of this projection is an approximate basis for the desired space, gathered into a block vector which we call U. A Rayleigh–Ritz step is used to extract the eigenvectors and eigenvalues from U. If convergence is not satisfactory, iterations may continue with the starting vectors of the next iteration chosen as the approximate eigenvectors of the last iteration.

Algorithm 5: FEAST

```
Choose a starting block of $m_U$ vectors $Y$
while not converged do
    Compute $U = \sum_{i=1}^{q} w_i (z_i B - A)^{-1} B Y$
    Compute Rayleigh quotients $A_U = U^H A U$, $B_U = U^H B U$
    Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
    Compute approximate eigenvectors $X = UW$
    Set $Y = X$
end
```

2.4.1. RFEs with moments

Sakurai–Sugiura methods (SSM) were introduced in [86] as the first rational filtering based eigensolver (though the methods were not introduced as iterative). The initial conception of the algorithm extracted the eigenpairs via a Hankel matrix [86]. A later version of the scheme using a Rayleigh–Ritz step for a more accurate extraction of the eigenpairs was given in [87]; this was extended to use a more stable block Rayleigh–Ritz (SS–RR) procedure in [55]. The possibility of subspace iteration has also been introduced in [58, 84] and will be discussed further in Chapter 5. We introduce the method in Algorithm 6 with the possibility of multiple iterations for completeness, but omit the details of how Y is selected for the time being. We define Φ_i as $(z_i B - A)$

Algorithm 6: Block SS–RR with subspace iteration

```
Choose a starting block of $\frac{m_U}{s}$ vectors $Y$
**while** *not converged* **do**
    Compute $U = \left[\sum_{i=1}^{q} w_i \Phi_i^{-1} BY, \quad \ldots, \quad \sum_{i=1}^{q} w_i z_i^{s-1} \Phi_i^{-1} BY\right]$
    Orthogonalize $U$
    Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
    Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
    Compute approximate eigenvectors $X = UW$
    Set $Y$ for next iteration
**end**
```

2.4.2. Convergence of an RFE

In [89], convergence bounds for the FEAST algorithm and all equivalent RFEs compatible with Algorithm 5 were given. As seen in that work, if we assume the n eigenvalues of (A, B) are ordered according to $r(\lambda_1) \geq r(\lambda_2) \geq \cdots \geq r(\lambda_n)$, and define P^t as the B-orthogonal projector into the space spanned by the vectors of U in the t^{th} iteration of Algorithm 5, we obtain a convergence bound

$$\|(I - P^t)x_i\|_B \leq c \left|\frac{r(\lambda_{m_U+1})}{\lambda_i}\right|^t \quad i = 1, 2, \ldots, m_U. \tag{2.36}$$

We can observe that the convergence of the initial starting vectors towards the desired eigenvectors is controlled by the value of the rational function at the eigenvalues. The value of $r(\lambda_{m_U})$ is clearly also important; a larger value of m_U should improve overall convergence, though the degree to which this occurs is controlled by the spectrum, as will be discussed later. Of course, the numerical solution of the linear systems of equations is an important factor to consider in determining the overall convergence of the RFE. In [89] it was shown that, beyond a given degree of accuracy, the error added by a (possibly iterative) inexact solution to the linear systems of equations should not affect the rate of convergence as a whole. The degree to which error in the linear systems can affect the solution has been shown in more detail in [38].

2.5. History of the RFE

In 2003, Sakurai and Sugiura released their contour–integral scheme with multiple moments [86], which involved the construction of a Hankel matrix in order to extract the approximate eigenpairs. Several updates have followed with a modified extraction type: Rayleigh–Ritz [87], and block versions of the Rayleigh–Ritz [55], Arnoldi [57], and Hankel [56] extraction methods for improved stability. These works also include the generalization to non-Hermitian problems. Furthermore, Sakurai–Sugiura methods have been extended to nonlinear eigenvalue problems [12, 98]. Beyn methods have also been introduced as a rational scheme with multiple moments for nonlinear eigenvalue problems [17]; these also have been considered further [94].

In 2009, Polizzi independently developed the FEAST method [77] for Hermitian generalized eigenvalue problems, using a contour integration scheme within a projected subspace iteration method with Rayleigh–Ritz extraction. This scheme has also been further developed for non-Hermitian [63] and nonlinear [37] problems, as well as combined with Beyn methods for the solution of nonlinear problems [19]. Error bounds have been shown [89], including with respect to the accuracy of solution of the linear systems of equations [38]. The extension from contour–integral based schemes to more general rational filters has also been considered, for Zolotarev functions [42], and optimized rational functions [64, 96]. In [97], a weighted least–squares formula for the weights of a rational function was also provided. An overview of possible choices of rational functions for an RFE was given in [30]. This topic will be discussed further in Chapter 4.

2.6. Polynomial filters

A subset of rational filters that do not require the solution of a linear system of equations is polynomial filters, which can be defined for a generic polynomial of degree d as

$$p(x) = \sum_{i=0}^{d} \omega_i x^i \tag{2.37}$$

When applied to a Hermitian nonsingular matrix $A = X\Lambda X^H$, the subsequent matrix function is

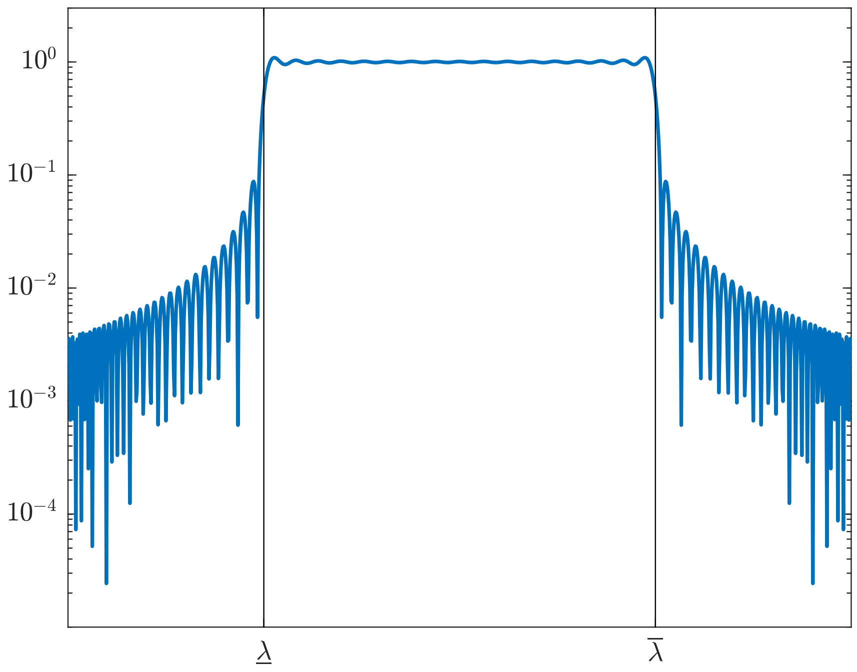


Figure 2.6.: Visualization of Chebyshev polynomial filter (2.37) with $d = 100$.

$$\begin{aligned} p(A) &= \sum_{i=0}^{d} \omega_i (X \Lambda X^H)^i \\ &= \sum_{i=0}^{d} \omega_i X (\Lambda)^i X^H \\ &= X p(\Lambda) X^H. \end{aligned}$$

When applied to a set of vectors Y, this acts as an approximate spectral projector,

$$U = \sum_{j=0}^{d} \omega_i A^i Y \tag{2.38}$$

Assuming that $p(x)$ is an approximation for the window function, the resulting projected vectors satisfy

$$U \approx X_{I_\lambda} X_{I_\lambda}^H Y \tag{2.39}$$

We note that the application of the filter for a generalized eigenvalue problem would require inversion of the matrix B; thus we only consider the standard eigenvalue problem here. A general overview of polynomial filtering may also be found in [82].

Algorithm 7: Polynomial filter-based subspace iteration

```
Choose a starting block of $m_U$ vectors $Y$
while not converged do
    Compute $U = \sum_{i=0}^{d} \omega_i A^i Y$
    Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H U$
    Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
    Compute approximate eigenvectors $X = UW$
    Set $Y = X$
end
```

2.6.1. Chebyshev polynomials

Chebyshev polynomials, as defined in, e.g., [95], are a common approximating polynomial due to their optimal approximation properties in the uniform norm and their ability to limit oscillatory activity in comparison to other polynomials, especially at higher degrees. In this thesis, we use Chebyshev polynomials of the first kind; that is

$$P(x) = \sum_{i=0}^{d} c_i T_i(x) \tag{2.40}$$

where $T_i(x)$ satisfies $T_i(\cos(\theta)) = \cos(i\theta)$. The values of c_i are selected such that $P(x)$ approximates the window function on $[-1, 1]$. Further details on calculating the values of c_i and $T_i(x)$ for this approximation, as well as additional filters to improve the approximation, are given in [34].

2.7. Additional algorithmic considerations

We have so far discussed how to apply a filtering scheme to a set of vectors to obtain a basis, U, for the desired eigenspace $\mathcal{X}_{I_\lambda}$. We now describe several other important considerations of a subspace iterative algorithm.

As described in Algorithm 2, at the end of a subspace iteration, we have a set of approximate eigenvectors and eigenvalues. Unsurprisingly, these eigenpairs may converge at different rates. To reduce computational expense, eigenpairs that have converged to the desired tolerance may be removed from the starting subspace vectors for the next iteration; the subspace may shrink accordingly [82]. It is important to continue to orthogonalize the approximate eigenvectors in future iterations against these "locked" vectors. This topic is discussed in more detail in [33, 65, 66].

The number of vectors included in U, m_U, is also important in ensuring convergence. Indeed, as shown in [65, 66], it is essential that m_U is greater than m, the true number of eigenvalues in I_λ. As discussed in [30], if this condition is not met, it is essentially impossible to find all desired eigenpairs. Given $\tilde{m}$, an estimate for m, the standard choice of $m_U \approx 1.5 \times \tilde{m}$, the expected number of eigenvalues in I_λ, has been shown in [32] and [89] to be an acceptable choice. We will discuss the choice of subspace size with respect to a multiple-moment RFE in Chapter 5.

Updating the value of $\tilde{m}$ over subspace iterations is important for maintaining this ratio and determining when all eigenvalues have been found. As discussed in [32, 65, 89], the eigenvalues of $U^H BU$ approach the values of $r(\lambda_i)^2$ or $p(\lambda_i)^2$, and thus may be used to count the number of eigenvalues in I_λ. These eigenvalues correspond to the "filtered" value of the corresponding direction, so values closer to 1 are presumably directions that should be kept, and values close to 0 are presumably spurious. Thus, the number of eigenvalues of $U^H BU$ with a value greater than $1/4$, or alternatively, singular values with a value greater than $1/2$, may be used as an estimate for $\tilde{m}$. As discussed in [32, 65], the rank of U also coincides in exact arithmetic with m. In [32], the authors show that again, counting the singular values of U with a value greater than $1/2$ is a good metric for counting eigenvalues.

Another important aspect is forming this initial estimate for $\tilde{m}$. If the eigenvalue problem is standard, the Kernel Polynomial Method (KPM) may be used to obtain the approximate density of the spectrum [95]. Strategies for generalized problems involving polynomial and rational filters have also been explored [27, 72].

2.8. Solving linear systems of equations

The most expensive part of a subspace iterative scheme is the construction of the subspace basis U; for a rational filter this cost is based in the cost of solving linear systems of equations of the form

$$\Phi u = y. \tag{2.41}$$

The matrices in these linear systems are *shifted*, such that $\Phi = A - zB$, as required to construct U. Again, the value z corresponds to a pole of the rational function. Note that we consider u and y as belonging to block vectors; depending on the solver in use, the linear systems corresponding to the different pairs of columns can be solved together or separately. In the context of this thesis, y is replaced by By; we skip this notation in this section for the sake of simplicity. We note that the number of columns of each of these blocks is expected to be small relative to the matrix size.

The choice of linear solver is very important for the overall performance of an RFE. The field of linear solvers is extremely broad and an area of active research. Furthermore, performance is typically dependant on the properties of the linear system under consideration. The shifted linear systems that we consider in this thesis have in common that they are typically ill-conditioned (due to their shifted nature) large, and sparse. In some cases we can limit our problem set to enforce characteristics such as the normality of a matrix or real entries outside of the diagonal (which itself contains a complex shift). Certain conditions, such as being banded or based on a grid, or knowing details of the matrix structure can help in choosing a particular solver or preconditioner. However, depending on such conditions or knowledge is extremely limiting for the generality of the RFE and so are avoided in this thesis. We discuss below some basics categories and principles of linear solvers, as well as some general subtypes, which will arise as we consider strategies for their efficient use within an RFE.

2.8.1. Direct solvers

Direct solvers for linear systems of equations are typically based on factorization. The most common is the LU factorization; given the decomposition of a matrix Φ into factors $\Phi = LU$, with L lower and U upper triangular, the solution of (2.41) may be given as

$$u = U^{-1}L^{-1}y \tag{2.42}$$

As L and U are triangular matrices, the solution of these linear systems requires $\mathcal{O}(n^2)$ operations. In the general case, such a factorization requires $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ storage [93], even if the original matrix is sparse. This cost quickly becomes prohibitive. Various mitigation schemes are available; depending on the underlying matrix, reordering, approximation, or a differing factorization may be appropriate. Some of these strategies are included in the software packages we discuss in Section 2.10.4, but a general discussion is beyond the scope of this work.

2.8.2. Iterative solvers

Iterative linear solvers generate approximate solutions such that, after k iterations, $\Phi u_k \approx y$. There is a large variety to the way these approximate solutions may be generated, including the basic relaxation schemes, Krylov subspace methods and multigrid methods. Furthermore, the choice of preconditioner, while specific to the problem under consideration, is often essential to obtaining good performance. Generally we may hope for solution to a linear system of equations in $\mathcal{O}(n)$ operations and storage for a sparse matrix. In practice, iterative methods may only outperform direct methods for larger matrix sizes. Their convergence may depend on the conditioning of the problem under consideration, slowing or even diverging for the ill-conditioned shifted linear systems under consideration. At this point we will focus on a few basic Krylov subspace methods which will be revisited in the remainder of this thesis, mainly in Chapter 4.

2.8.2.1. GMRES

The Generalized Minimum Residual Method (GMRES) is a *Krylov subspace method*; that is, the provided solution u_j satisfying $\Phi u_j \approx b$ in the j^{th} iteration lies in $u_0 + K_j(\Phi, r_0)$, where $K_j(\Phi, r_0)$ is the j^{th} Krylov subspace and r_0 is the initial residual vector $b - \Phi u_0$.

This method specifically minimizes the residual norm in the j^{th} iteration over all solution vectors in $u_0 + K_j(\Phi, r_0)$. We define the method below in Algorithm 8, as given in [39]. We note that this involves the definition of the upper Hessenberg matrix $H_j \in \mathbb{C}^{j+1,j}$

$$H_j = \begin{bmatrix} h_{1,1} & h_{1,2} & \dots & \dots & h_{1,j} & \\ h_{2,1} & h_{2,2} & \dots & \dots & h_{2,j} & \\ 0 & \ddots & \ddots & & \vdots & \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & h_{j,j-1} & h_{j,j} & \\ 0 & \dots & \dots & 0 & h_{j+1,j} & \end{bmatrix} \tag{2.43}$$

As j increases, the cost of a GMRES iteration also increases. As n iterations may be required (in exact arithmetic), this may become computationally infeasible. The restarted GMRES method begins building a new Krylov basis every c iterations, using the solution of the previous iteration as a starting guess for the solution.

Algorithm 8: GMRES for the solution of $\Phi u = y$

```
Compute $r_0 = y - \Phi u_0$, $h_{1,0} = \|r_0\|_2$
$j = 0$
while $\|r_j\| >$ tol do
    $v_{j+1} = r_j / h_{j+1,j}$
    $j = j + 1$
    $r_j = \Phi v_j$
    for $i = 1, \dots, j$ do
        $h_{i,j} = v_i^H r_j$
        $r_j = r_j - h_{i,j} v_i$
    end
    $h_{j+1,j} = \|r_j\|_2$
    if $h_{j+1,j} = 0$ then
        break
    end
    $y_j = \text{argmin}_y \|H_{j+1,j} y - \|r_0\|_2 e_1\|_2$
    $u_j = u_0 + V_j y_j$, where $V_j = [v_1, \dots, v_j]$
end
```

2.8.2.2. Conjugate Gradients

The conjugate gradient (CG) method is also a Krylov subspace method, suitable for symmetric positive definite matrices. The resulting approximate solution u minimizes the energy norm, $u^H \Phi u$ of the error in the Krylov subspace $K_j(\Phi, r_0)$ [81]. The algorithm relies on the iterative search for a solution in "conjugate" (Φ-orthogonal) directions, and is presented, as in [81], in Algorithm 9. The impressive convergence rate bound for the CG method, related to the square root of the condition number of the matrix, is a reason why it is a foundational and leading method for iterative solvers.

2.9. Orthogonalization

Orthogonalization is an important component of a filtered subspace scheme. We will now highlight some areas of these schemes where orthogonalization may be helpful, and explain some basic strategies for orthogonalization. This is a brief overview; a comprehensive discussion including recommended strategies and usage for projected subspace methods may be found in [30].

There are two important points where orthogonality is considered in an RFE or other projected subspace scheme. The first is regarding the vectors of U. These

Algorithm 9: CG for the solution of $\Phi u = y$

```
Compute $r_0 = y - \Phi u_0$, $p_0 = r_0$
$j = 0$
while $\|\Phi u_j - y\| >$ tol do
    $\alpha_j = \frac{r_j^H r_j}{p_j^H \Phi p_j}$
    $u_{j+1} = u_j + \alpha_j p_j$
    $r_{j+1} = r_j - \alpha_j \Phi p_j$
    $\beta_j = \frac{r_{j+1}^H r_{j+1}}{r_j^H r_j}$
    $p_{j+1} = r_{j+1} + \beta_j p_j$
    $j = j + 1$
end
```

are expected to form an approximate basis for $\mathcal{X}_{I_\lambda}$. However, if the vectors of U approach linear dependence, the robustness of the method is jeopardized. Thus, an extra step to ensure stability is to orthogonalize U, and if needed, to remove any rank-deficient columns. This is shown in Algorithm 10. This topic is also discussed in more detail in Chapter 5, as RFEs with multiple moments are more sensitive to rank deficiency, and the orthogonalization of U is required.

Algorithm 10: Filter based subspace iteration with orthogonalization of U

```
Choose a starting block of $m_U$ vectors $Y$
while not converged do
    Compute $U = \tilde{h}(B^{-1}A)Y$
    Orthogonalize $U$
    Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
    Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
    Compute approximate eigenvectors $X = UW$
    Set $Y$ based on $X$
end
```

It is important to ensure the eigenvectors found by the algorithm are orthogonal to each other. Within a subspace iterative method, the Rayleigh–Ritz algorithm should return orthogonal eigenvectors; the resulting full-sized vectors should also be orthogonal. However, if the spectrum is sub-divided into separate eigenproblems, and these are solved separately, the resulting eigenvectors from different intervals may not be orthogonal to each other. This problem and strategies for ensuring orthogonalization are discussed in [30]. Orthogonalization strategies for RFEs with multiple moments are discussed in Chapter 5.

2.10. A software framework for iterative subspace filtration

When considering the efficient solution of RFEs and other filtered subspace methods, it is helpful to define the software framework we are considering. The BEAST eigensolver is a framework for large, sparse interior eigenvalue problems. It relies on filtered subspace iterative schemes, as described above. These strategies are implemented for use in a a high-performance computing environment. We will give a general overview of the framework here, as it implements many of the ideas provided and has generated results shown in the remainder of this thesis. Further description of the framework is provided in [30], as well as in the context of the larger ESSEX (Equipping Sparse Solvers for eXascale) [4, 9] and ELPA (Eigenwert-Löser für Petaflop-Anwendungen) [73] projects. The BEAST framework has provided the solution to a variety of eigenvalue problems, including very large problems [4, 8, 31, 34, 35, 65].

Support for efficient (sparse) matrix and vector operations is provided by the background libraries, GHOST [67, 68, 69] and PHIST [90, 91, 92]. Efficient operations, such as sparse matrix vector products are essential to the overall efficacy of the framework. We present here the overall framework of BEAST, as relevant to the remainder of this thesis, as well as highlighting some important computational considerations. An overview is given in Figure 2.7. However, a complete description of the software and its many options is outside the bounds of this work. Noteworthy is that a significant advantage to the iterative schemes implemented here is the ability to adjust parameters over iterations. This can include the choice of degree for a polynomial or rational filter [33], the precision used [8, 31], or even the type of filter under consideration [46]. These topics will be discussed further in the remainder of this thesis as well.

2.10.1. Main algorithmic choices

Within BEAST, the user may choose which filtering scheme should be used. The three main choices currently included in the framework are:

2.10.1.1. BEAST-C

BEAST-C is named for the original application of the BEAST framework to filters arising from contour integration, based off of the FEAST algorithm [77], cf. Algorithm 5. Implementations of filters arising from numerical quadrature using either a Gauss–Legendre or midpoint quadrature rule are available, as described in [30]. Furthermore, a user defined filter, specifying the poles and weights of a generic

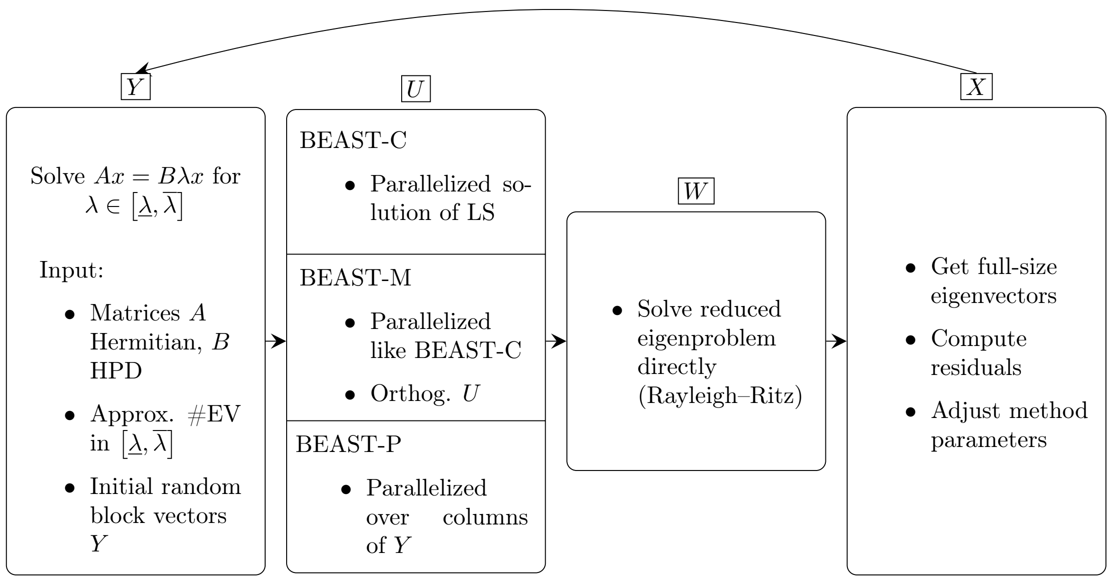

$\boxed{Y}$
$\boxed{U}$
$\boxed{W}$
$\boxed{X}$
Solve $Ax = B\lambda x$ for $\lambda \in [\underline{\lambda}, \overline{\lambda}]$
Input:
Matrices A Hermitian, B HPD
Approx. #EV in $[\underline{\lambda}, \overline{\lambda}]$
Initial random block vectors Y
BEAST-C
Parallelized solution of LS
BEAST-M
Parallelized like BEAST-C
Orthog. U
BEAST-P
Parallelized over columns of Y
Solve reduced eigenproblem directly (Rayleigh–Ritz)
Get full-size eigenvectors
Compute residuals
Adjust method parameters

Figure 2.7.: Algorithmic outline of `BEAST`.

rational filter, may be provided for this method. This option will be further explored in Chapter 4. This portion of the framework incorporates ideas from a large body of work beyond the general algorithm described thus far, including the estimation of the number of eigenvalues in the spectrum and choice of subspace size [30, 32, 65, 66], as well as the adaptive choice of quadrature degree, [33].

2.10.1.2. `BEAST-M`

With `BEAST-M`, the moment based rational filtering scheme described in Algorithm 6 is implemented. As described further in Chapter 5 and [46], the choice of number of moments is flexible, and may be chosen adaptively to minimize overall cost of the eigensolver. A number of details regarding the implementation specific to this scheme, such as orthogonalization and convergence, as well as an adaptive switch to `BEAST-C`, are also described in Chapter 5.

2.10.1.3. `BEAST-P`

`BEAST-P` implements a polynomial filter, as described in Algorithm 7. As described above, this is only available for standard eigenvalue problems. Further description of some features and strategies for optimization of the polynomial are given in [33, 34].

2.10.2. Parallelism

A driving factor in the compatibility of subspace iteration schemes with large scale computation is their capacity for parallelism. `BEAST` is a parallel, distributed memory library, using an MPI+X paradigm [74], where X is typically Open-MP [75]. Multiple levels of parallelism include:

- Parallelism over sub-intervals of I_λ. These become independent eigenproblems. Orthogonalization of the solutions over the different intervals may be required [30].
- Parallelism over quadrature nodes (for rational filters). In the case of rational filters, the linear systems solved are independent of each other. The solutions must, however, be gathered centrally after solution.
- Parallelism over columns of Y. In the case of rational filters, the solution of linear systems over a block of right hand side may be independent, depending on the linear solver in use. For polynomial filters, matrix multiplication is independent over the columns of Y.
- Internal parallelism (data distribution, matrix and block vector operations) provided by background libraries.

The scalability of these algorithms is a prerequisite for increasing problem size, and at least the first three cases are very parallelizable. Further details on the implementation of the parallel framework can be found in [30].

2.10.3. Precision

The precision of computations and storage of values within `BEAST` may be specified to single (32 bit) or double (64 bit) machine precision. `BEAST` may begin in single precision and switch to double precision after a convergence threshold has been reached. Further details are described in Chapter 3 and [4, 8, 30]

2.10.4. Linear solvers

For `BEAST-C` and `BEAST-M`, the choice of linear solver is very relevant to the efficacy of the algorithm as a whole. As previously mentioned, the cost of solving linear systems of equations is expected to be the limiting factor of the overall eigensolver cost. It is very important, therefore, to choose a linear solver that is equipped for the solution of large, sparse linear systems of equations in a scalable and robust manner. Furthermore, we desire flexibility in the precision used, as described above. If an iterative linear solver is being considered, the difficulty of the shifted linear systems of equations becomes relevant. We therefore turn now to the choice of direct and iterative linear solvers used in the remainder of this thesis. Several other options are implemented in `BEAST`, including a direct linear solver for banded systems, and a callback framework for extension to a user-defined solver package that also implements the call within `BEAST` to the external packages discussed here. Further details on these can be found in [30].

2.10.4.1. Direct solvers

Direct solvers are less sensitive to the shifted and thus ill-conditioned systems of equations, making them a good choice if the problem under consideration is not too large. With this solver type, the resulting factorization of $(z_i B - A)$ may be stored, reducing the cost of multiple `BEAST` iterations, but requiring additional storage proportional to $O(n^2)$. A number of software packages are available for the direct solution of large, sparse linear systems. `BEAST` includes the option of calling either MUMPS [10, 11] or STRUMPACK [80]. Both of these are distributed-memory, parallel direct solvers for the solution of large linear systems. They allow for the use of single or double precision arithmetic.

2.10.4.2. Iterative solvers

Iterative solvers are typically much better suited to sparse matrices than direct methods, as they are able to take advantage of the reduced storage and computational costs associated with sparsity. They are therefore also typically suitable for much larger problems than a direct method. The `BEAST` eigensolver allows for a user defined iterative solver, possibly from another library via the callback scheme described in [30]. A couple of solvers are included with the framework, including the CARP-CG [41] implementation provided by PHIST, as described in [36]. More details on the iterative solvers used this thesis can be found in Chapter 4.

2.11. Conclusion

We have outlined the underlying theory and definition of subspace filtration methods. We have focused particularly on rational filters and defined an RFE, which will be a central theme of this thesis. We have also defined computational details for the implementation of these methods, as required for efficient and scalable computation. Next, we will consider ways to improve the efficiency of these schemes.

CHAPTER 3

MIXED PRECISION

3.1. Introduction

In this chapter, we consider the effect of precision on the projected subspace eigensolvers implemented in `BEAST`. The "default" precision in modern computation is IEEE double precision [54], using a 64 bit representation of each floating point number. In contrast, single precision uses a 32 bit representation. The use of other precisions is increasing, but still less common. A change in precision is associated with a change in cost; with a lower precision, less data must be stored and computed. Reduced precision could result in reduced overall accuracy levels. We explore this within the filtered subspace schemes considered, for given steps within subspace iterations, as we have discussed and shown in part in [31], and over subspace iterations, as we have originally shown in [8]. Related results have also been shown in [30], discussed with the implementation of mixed precision in the `BEAST` framework.

3.2. Background

The topic of mixed precision within numerical methods has been explored within a variety of contexts [1], most comprehensively within linear solvers [13, 21, 22, 70]. These works have shown that using a combination of precisions will not, under certain conditions, affect overall convergence, and may allow for significant cost reduction of an algorithm. Many software libraries enable computations in either single or double precision, and some allow for extended precisions as well. The FEAST software package [78], which implements the FEAST algorithm as described in Algorithm 5, is among the libraries offering this flexibility.

Explorations in varying precision within a projection based algorithm has mainly focused on the effect of error within the construction U, the approximate basis for $\mathcal{X}_{I_\lambda}$. In [89] and [38], the authors showed that, beyond a given threshold, the convergence of the FEAST algorithm is not affected by a change in accuracy in the solution of linear systems required to construct U, as described in Algorithm 5. Furthermore, the current version of the FEAST software package utilizes an inverse residual iteration to allow for lower precision solution of linear systems without affecting overall convergence [78].

In [44] and [60], it was shown that with a large enough basis for the subspace $\mathcal{X}_{I_\lambda}$, that is, enough linearly independent columns in U, the convergence of Sakurai–Sugiura methods, as described in Algorithm 6, is not affected by a certain amount of error in the solution of linear systems of equations required to generate U.

Changing the precision of a computation can have dramatic effects on the speed of computation, as well as the size of storage required. As discussed in [21], switching from the standard double precision to single may significantly impact the performance of sparse matrix computations. Halving the precision halves the storage requirements, which not only reduces overall memory required, but allows for double the values to be stored in the cache, resulting in fewer cache misses. The reduced size also results in faster memory transfer. A vector unit can typically perform twice as many operations per clock cycle. Understandably, many libraries have incorporated support for different precisions.

3.3. Varying precision within a projected subspace iteration

Does a change in precision in a single part of the algorithm affect convergence as a whole? Theoretically, the loss of accuracy in single precision could cause it to slow; an idea we will explore here.

In [21], we see that mixed-precision computation can speed computation while maintaining overall accuracy levels. In this case, selected steps of various sparse matrix algorithms (e.g., CG, iterative refinement for direct linear solvers) were performed in single precision, and the remainder in double precision. Significant speed-ups were achieved while maintaining double precision accuracy. Similar themes have previously been partially explored for the FEAST eigensolver. Most significantly, in [38], it was observed that an inexact linear system solution within the quadrature rule of FEAST may not affect the overall rate of convergence, if the systems are solved to a sufficient residual tolerance. The presentation here builds on what has previously been presented in [31] by exploring additional algorithmic steps and results.

3.3.1. Algorithmic components

We refer back to Chapter 2 where the `BEAST` framework has been introduced. We can consider the effect of a change in precision on a single component of the algorithm for each algorithmic type.

Algorithm 11: `BEAST` framework, with isolated steps considered for single precision execution numbered and in bold

```
Choose desired subspace size $m_U$ ($>$ number of eigenvalues in $I_\lambda$) and initial
  vectors $Y$
while not converged do
    (1) Construct subspace $U \leftarrow Y$ with BEAST-* scheme
    (2) Orthogonalize $U$ (optional for BEAST-P, -C) and (3) compute
      singular values
    Resize $U$ according to rank and subspace size estimate
    (4) Solve reduced eigenproblem $A_U W = B_U W \Lambda$, where $A_U := U^H AU$,
      $B_U := U^H BU$
    Compute full approximate eigenvectors as $X := UW$
    (5) Orthogonalize against converged eigenvectors, lock converged
      eigenvectors
    Set $Y := BX$ (BEAST-P, -C) or $Y := BXR$ (BEAST-M), where $R$ is a
      random matrix
end
```

3.3.2. Methodology

We consider the effect of single precision upon the different steps of the algorithms contained within the `BEAST` framework. The outline in Algorithm 11 shows steps that may be performed in single precision, while still allowing the method as a whole to continue in double precision. For these experiments, computations were performed in MATLAB. `BEAST` iterations were performed with a selected operation in single precision. All variables for this step are cast to single precision, using the MATLAB function `single()`. After the operation is completed, all modified variables are cast to double precision (with `double()`) and the iteration continues in double precision. A specified number of `BEAST` iterations are completed in this manner, before remaining iterations continue in double precision. We compare with tests performed entirely in double precision.

3.3.3. Numerical experiments

We consider finding the solution of (1.3) for a 256×256 matrix A, Graph256, as described in Table A.2. This matrix is derived from graphene modeling [23]. The algorithm begins with random vectors such that $m_U = 30$ ($m_U = 32$ for `BEAST-M`). `BEAST-C` and `BEAST-M` both used an elliptical contour with eccentricity 0.4 with 8 quadrature points on the half contour. `BEAST-P` ran with polynomial degree 300. `BEAST-M` used 4 moments for all iterations. In Figures 3.1 through 3.4 we illustrate some characteristic results of single precision on convergence. In each case, for the first four iterations, the selected step is performed in single precision. The resulting convergence behaviour is compared with all steps being performed in double precision for all iterations.

We observe the effect of constructing U in single precision for `BEAST-C` in Figure 3.1 (marked as **(1)** in Algorithm 11.) This involves using single precision to solve the linear systems and sum the terms of the rational filter. In the first iteration, the difference in residual values when single or double precision is used for this operation is not significant; convergence of the minimum residual in the subspace appears to be $\approx 10^{-5}$ in both cases. In subsequent iterations (2-4), there is an observable difference in residuals depending on the precision used; at this point, the single precision computation appears to stagnate at a residual threshold, and presumably no further convergence can be expected. In iteration 5, all computations are performed in double precision again. Following the switch, we see that convergence begins again, with no apparent impact from error in the previous computations. Similar behaviour was seen for computing **(1)**, **(4)**, including with other `BEAST` projection types. This implies that single precision can be used for at least these operations up to some residual threshold without impacting overall achievable convergence.

There are also operations for which single precision may have no effect at all on overall convergence of the algorithm. As discussed in [32, 65, 89], and summarized in Chapter 2, the singular values computed in **(3)** may be used to update the value of m_U, either by removing rank deficient columns, or by updating the estimate for the number of eigenvalues in the interval and thus the number of columns needed in m_U. This topic will also be covered in more detail for moment-based methods in Chapter 5. If all columns of U are linearly independent, and the subspace is not undersized, which causes stagnation of convergence [32, 65], this value is unlikely to be sensitive to the extra error induced by single over double precision. This is seen in Figure 3.2, where single vs. double calculation of **(3)** does not appear to significantly affect convergence. On the other hand, if the linear independence of the vectors is not assured, as is more likely to occur for `BEAST-M`, the overall convergence may indeed be damaged when the orthogonalization and computation of the singular values are computed in single precision **(2-3)**.

We also explore the interplay of varying precision with locking vectors over iterations, which requires orthogonalization against the locked vectors. Here, the orthogonalization (**(5)** in Algorithm 11) takes place in single precision for the first four iterations. As we see in Figure 3.4, any vectors "locked" after this operation in reduced precision may be less accurate. Even when all computations proceed in single precision in iteration 5, the final vectors must be orthogonalized against those locked at reduced precision, reducing the accuracy achievable for the algorithm as a whole. However, these results are easily avoidable if locking does not take place in single precision iterations.

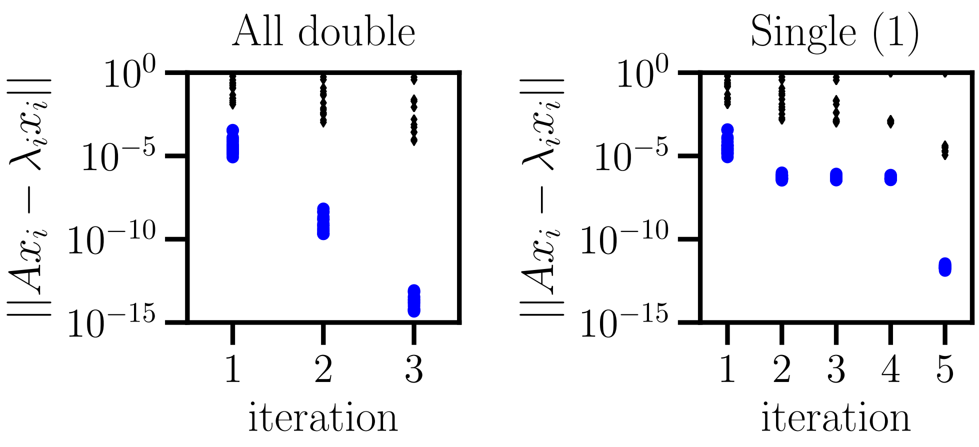


Figure 3.1.: Double precision in all iterations (left) vs single precision (right) for **(1)** in first four iterations of Algorithm 11 using type `BEAST-C`. Residuals of all Ritz pairs computed are shown; the $m = 16$ values with minimal residual are shown as blue circles, and spurious values as black diamonds. A version of this figure also appears in [31].

3.4. Precision changes over subspace iterations

From the previous section, we can conclude, in rough terms, the effects of the lowest precision used in a single step of the iteration apply to the achievable accuracy of the iteration as a whole. Therefore, performance may be improved not by selecting steps in the iteration to perform in a reduced (or increased) precision, but by selecting iterations to perform in reduced precision. In this section, we make

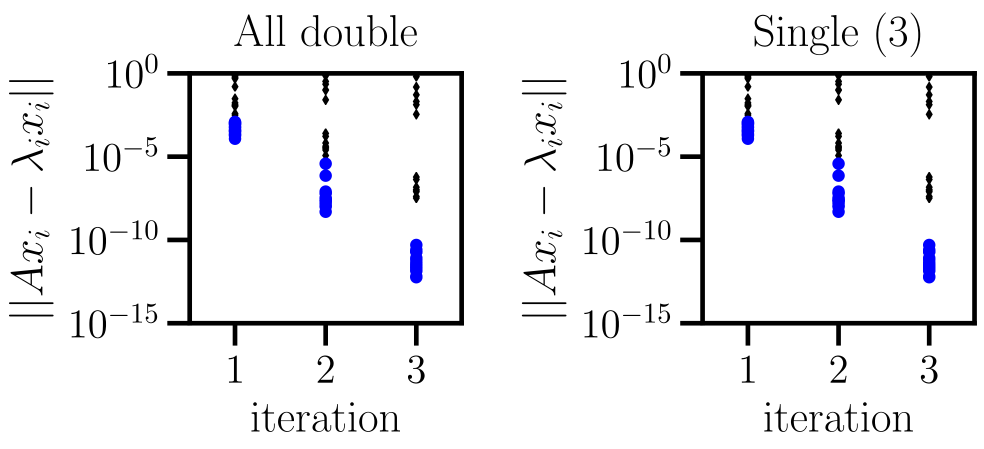


Figure 3.2.: As in Figure 3.1, with double precision in all iterations (left) vs single precision (right) for **(3)** in first four iterations of Algorithm 11 using type `BEAST-P`. A version of this figure also appears in [31].

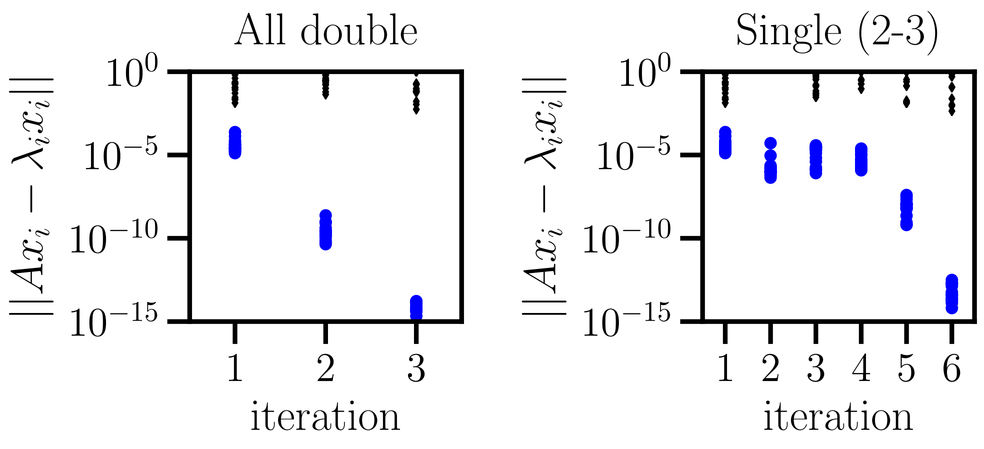


Figure 3.3.: As in Figure 3.1, with double precision in all iterations (left) vs single precision (right) for **(2-3)** in first four iterations of Algorithm 11 using type `BEAST-M`.

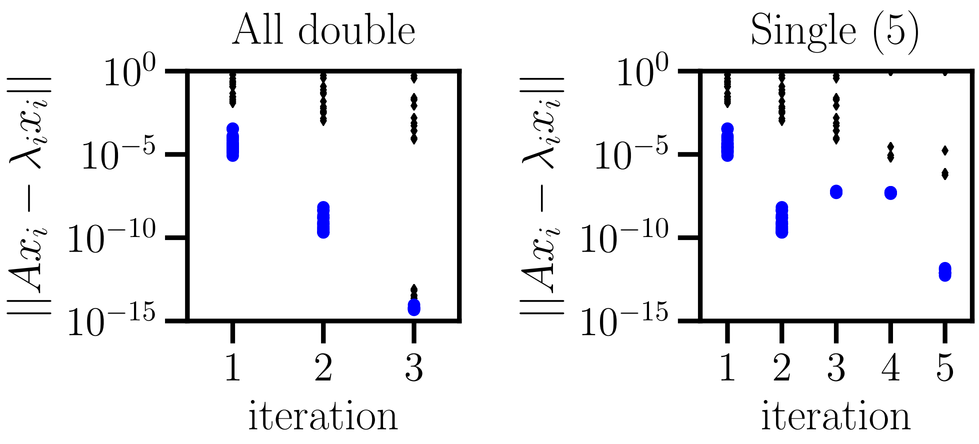


Figure 3.4.: As in Figure 3.1, with double precision in all iterations (left) vs single precision (right) for **(5)** in first four iterations of Algorithm 11 using type `BEAST-C`.

observations about the achievable convergence and performance behaviour of this strategy.

We are concerned with how the convergence of a `BEAST` algorithm changes when all steps of the iteration are performed in single precision. As observed in [8] and [30], convergence with a given precision is not affected above the given precision limit. For example, iterations in single precision may proceed to a given residual tolerance before stagnating. Convergence may continue unaffected if the precision is changed between iterations before the residual threshold of a given precision is reached. We observe that in single precision, a residual threshold of 10^{-5} is typically achievable before stagnation begins, but this may change depending on the problem and the algorithm used.

In the below numerical experiment, which may also be found in [8], we test this strategy, as well as observing the achievable performance gains from mixing precisions. The latter is highly variable depending on the computational setup used, and thus the potential gain from mixed precision cannot be definitively deduced from these experiments.

3.4.1. Numerical experiments

The `BEAST` algorithmic scheme for switching precision over iterations is shown in Algorithm 12. We observe that if iterations begin in single precision, we may decide at the end of each subspace iteration whether to switch from single to double precision. In this experiment we set a residual threshold that, once reached

Algorithm 12: BEAST framework with inter–iteration precision change

```
Choose desired subspace size $m_U$ (> number of eigenvalues in $I_\lambda$) and initial
  vectors $Y$
while not converged do
    Construct subspace $U \leftarrow Y$ with BEAST-* scheme
    Orthogonalize $U$ (optional for BEAST-C, BEAST-P) and compute singular
      values
    Resize $U$ according to rank and subspace size estimate
    Solve reduced eigenproblem $A_U W = B_U W \Lambda$
        $A_U := U^H AU$, $B_U := U^H BU$
    Compute full approximate eigenpairs $X := UW$
    Set $Y := BX$ (BEAST-P, -C) or $Y := BXR$ (BEAST-M), with $R$ random
    if single precision threshold reached then
        Switch to double precision
    end
end
```

by the smallest (non-converged) eigenpair, induces a change from single to double precision, as described in Figure 3.5. We consider a standard eigenvalue problem (1.3) for the Graph16M problem described in Table A.2.

The computation ran using 32 nodes of the Emmy HPC cluster at Friedrich-Alexander-Universität Erlangen-Nürnberg. The method began in each case with $m_U = 480$ and random columns of Y. The residual tolerance was set to 10^{-10}. In the mixed precision case, all computations and storage took place in single precision; upon a switch to double precision, all vectors were cast from single to double and the matrices were reloaded. This is associated with a small extra cost for the mixed precision case. A polynomial degree of 10,000 was used for BEAST-P, and precision was switched from single to double once the smallest residual was smaller than 10^{-5}. The same precision tolerance was used for BEAST-C, which used Gauss–Legendre quadrature with a circular contour and $q = 8$. Both BEAST-C and BEAST-M used STRUMPACK for the direct solution of linear systems. The BEAST-M method was somewhat more sensitive to the effects of single precision in this case; a higher threshold of 10^{-4} was used as the threshold for the switch to double precision, and the quadrature scheme used $q = 16$ for a circular contour. BEAST-M ran with 4 moments in all iterations.

In Figure 3.5, we observe the convergence of the smallest residual in each of the above cases. If the switch from single to double precision is well timed, convergence may not be impeded at all by single precision in early iterations, and a small "catch-up effect" may even appear directly after the switch, as also observed in [30]. The best choice of threshold appears somewhat method dependant; in the case of

BEAST-M we observe a difference in residual from the first iteration and stagnation even before the threshold of 10^{-4} has been reached, an effect that occurs earlier and more strongly than for BEAST-P or BEAST-C. It is possible that generosity in other parameters, e.g., increasing m_U, would reduce the effect of error in this case [44, 60].

The difference in time between mixed and double precision is shown in Table 3.1. For BEAST-P in particular, the example where the number of iterations was equal in both the mixed and double precision case, the reduction in cost from early iterations conducted in single precision is clearly observable. For BEAST-C and BEAST-M, we may assume that a similar reduction in cost took place for each RFE iteration performed in single precision, but the extra iteration for the all–double run of BEAST-C (apparently a random effect due to some spurious directions being removed in early iterations) adds some uncertainty to this value. In any case, the qualitative reduction in cost with mixed precision computation is clear. In the case of BEAST-M, we observe an extra iteration in the mixed–precision case, due to stagnation in early iterations, and thus a slow-down, rather than a speed-up, from mixed precision is observed. We note that these effects are not expected to generalize to all hardware or software situations; the use of accelerators or optimized vector instructions could be used to generate a stronger effect for mixed precision computations.

3.5. Conclusions

The precision used for storage and computation is an important aspect in controlling the cost and convergence of a subspace projection scheme. Particularly when the method is iterative, we can see that single precision can be used in early iterations to reduce costs without impacting later convergence; later iterations in double precision may continue to the desired residual threshold. It is particularly interesting to observe a strategy to reduce cost without "adding back" costs in reduced convergence rates; suggesting that double precision in early iterations may even be considered "overspending." However, the extra knowledge required to use these methods successfully in order to ensure stagnation does not occur does prevent universal suitability. Furthermore, as discussed in [30], the results we have seen suggest that higher precision computations will not have any effect unless all steps proceed in higher precision and a reduced residual tolerance is required.

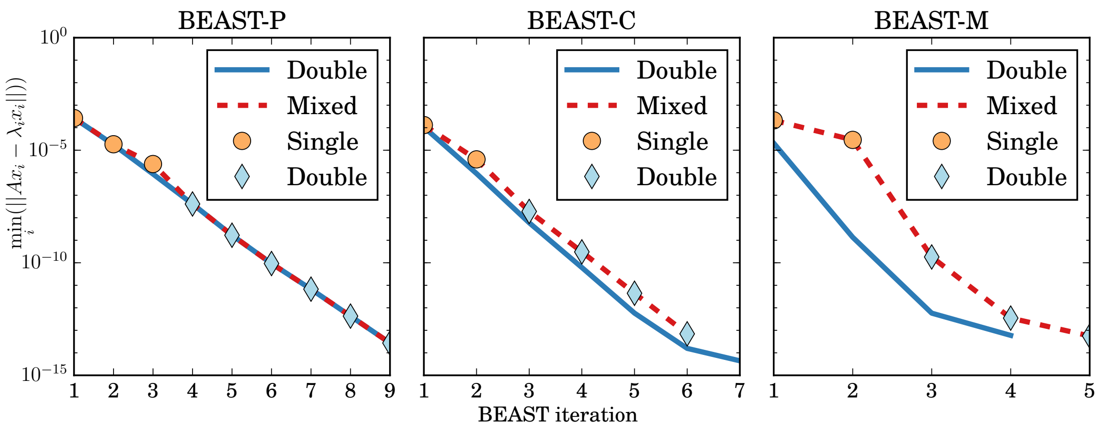


Figure 3.5.: Smallest residual over `BEAST` iterations. This figure originally appeared in [8].

Solver	Time Mixed / Time Double
`BEAST-P`	0.8
`BEAST-C`	0.7
`BEAST-M`	1.2

Table 3.1.: Time ratio between a mixed precision solve and full double precision solve (reference time)

CHAPTER 4

OPTIMIZING RATIONAL FILTERS

4.1. Introduction

In this chapter, we consider how the rational filter used in an RFE affects the cost of the method, and how we might improve the efficiency of these schemes through the choice of rational filter. We focus on the cost of building the basis U for our desired subspace through the solution of shifted linear systems of equations $(z_iB-A)^{-1}BY$, which is expected to be the most expensive portion of an RFE iteration. Here, the shift z_i corresponds to a pole of the rational filter used. We are interested in the solution of large and sparse eigenproblems, implying that an iterative linear solver would be appropriate for the linear systems that arise in these methods. However, these shifted linear systems are typically very poorly conditioned, as the shifts z_i are chosen to be close to the eigenvalues of the matrix pair to profit convergence of the RFE. A trade-off arises, where the cost of solving the linear systems in a single RFE iteration must be balanced with the number of RFE iterations required. In this chapter, we attempt to construct rational filters that will reduce the overall cost of solution with iterative linear solvers.

We begin by exploring how convergence of an RFE is controlled by the rational filter used and the relative location of the eigenvalues. This relationship has been used in the past by other authors to guide the choice of rational filter in an RFE beyond the use of quadrature rules, which we have seen already in Chapter 2. We will introduce some of the pre-existing choices defined in other works and discuss how they may affect the convergence of the resulting RFE and the iterative solver for the linear systems. Then we turn to exploring how we should choose our rational filter. We introduce two iterative linear solvers and generate predictive models for the expected iteration count of each of these, based on the underlying rational filter. This prediction is also expanded to include the cost of linear solves over all RFE iterations, based on the expected convergence rate for said rational filter. Given these models, we may then generate a cost function and optimize over possible rational filters. This is performed for a number of test problems, followed by a comparison of the cost of using the resulting filter with filters arising from a standard contour integration scheme. We conclude with a discussion of the benefits of this work and ideas for future research.

4.2. Convergence of an RFE

As introduced in Chapter 2, RFEs use a rational filter to iteratively project an initial subspace in directions corresponding to the desired eigenvectors. The rational filter $r(x)$ of degree q consists of a series of shifts, z_i, and weights w_i, $i = 1, \dots, q$, forming the function

$$r(x) = \sum_{i=1}^{q} \frac{w_i}{x - z_i}. \tag{4.1}$$

In matrix form, we consider the function $r(B^{-1}A) = \sum_{i=1}^{q} w_i(A - z_iB)^{-1}B$, or, when applied to a (block) vector, $r(B^{-1}A)Y = \sum_{i=1}^{q} w_i(A - z_iB)^{-1}BY$.

As discussed in [89], the values of $r(\lambda)$ for the eigenvalues λ_i, $i = 1, ..., n$ of (A, B) determine the convergence of the RFE. Specifically, the convergence of the i^{th} eigenpair depends on the ratio

$$\frac{r(\lambda_{m_U+1})}{r(\lambda_i)} \tag{4.2}$$

assuming that the eigenvalues are ordered according to $r(\lambda_1) \geq r(\lambda_2) \geq ... \geq r(\lambda_N)$, and that the subspace $\tilde{X}_{I_\lambda}$ contains m_U vectors.

As previously discussed, we can consider the behaviour of the filter in 1D as corresponding to its action on vectors; directions corresponding to eigenvectors with eigenvalues inside or outside I_λ will be amplified or reduced respectively according to the value of $r(x)$ at that eigenvalue.

The quality of the filter, therefore, defines the convergence of FEAST. It should also be noted that the (presumably unknown a-priori) location of the eigenvalues has a significant effect on the actual convergence behaviour. By observing (4.2), we realize that the value $r(\lambda_{m_U+1})$ controls the behaviour of FEAST. If $r(\lambda_{m_U+1})$ is small, particularly in comparison with $r(\lambda_m)$, with m the true number of eigenvalues inside I_λ, the method will converge more quickly. It is easier to find a function $r(x)$ is that is (relatively) small at $r(\lambda_{m_U+1})$ if more distance lies between λ_{m_U+1} and λ_m; this occurs if eigenvalues are well separated, or if m_U is larger relative to m.

Conversely, dense sections of the spectrum and narrow subspaces may have an adverse effect on convergence. A steep drop of $r(x)$ outside of I_λ will minimize the worst-case convergence rate, which we observe when the spectrum and choice of m_U places λ_{m_U+1} very close to λ_m. Sharp bounds of the filter will conversely have less of an effect on convergence if these values are well separated.

Given this relationship, several significant research findings have explored the choice of rational filter. Though contour integration, usually with the Gauss-Legendre (GL) rule, remains a standard choice, several other functions have been explored, as we discuss in Section 4.3.

4.2.1. Numerical example - subspace size and convergence

The effect of the relative location of λ_{m_U+1} and λ_m on convergence is illustrated in Figure 4.1. In this figure, we compare $\frac{r(\lambda_{m_U+1})}{r(\lambda_i)}$ for two values of m_U, $m_U = 220$ and $m_U = 320$, for an equispaced spectrum

$$\lambda_i = -2 + ih, \quad i = 0, \dots, 400$$

with $h = 0.01$.

We consider a typical choice of $r(\lambda)$, the Gauss–Legendre quadrature rule, as described in (2.10), with 16 poles on a circular contour surrounding $I_\lambda = [-1, 1]$. We see that a larger value of $\frac{r(\lambda_{m_U+1})}{r(\lambda_i)}$, which may occur due to poorly separated eigenvalues, or simply to a smaller choice of m_U, affects the estimated convergence rate of all eigenvalues inside I_λ.

In Figure 4.2, we illustrate how the convergence of an RFE may actually be affected by the change in this ratio. We consider A a diagonal matrix with entries λ, and consider the resulting estimated eigenpairs after a single `BEAST-C` iteration with $m_U = 320$ or $m_U = 220$ initially random vectors. The filter is the same of that in Figure 4.1. Here, we see that the change in theoretical convergence rate corresponds to the actual difference.

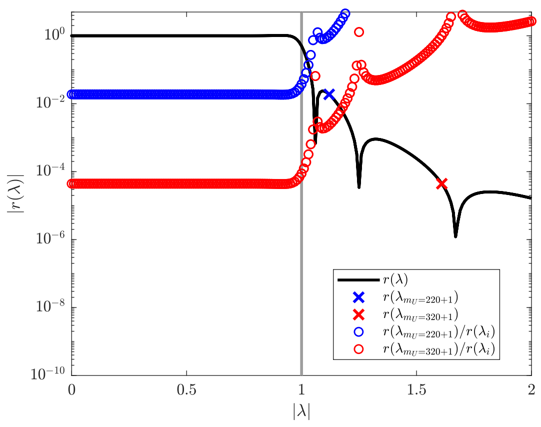


Figure 4.1.: (a)

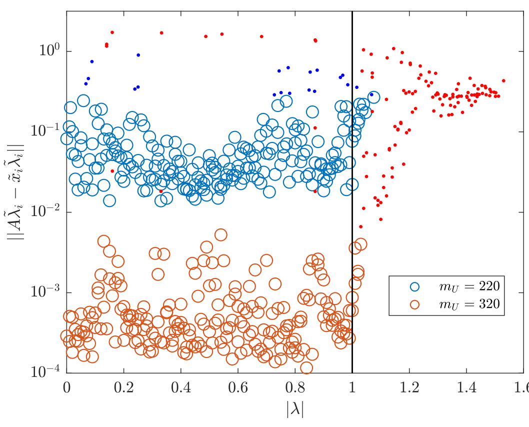


Figure 4.2.: (b)

Theoretical (a) and actual (b) effect of subspace size on convergence rate. In (b), the smallest $m = 201$ residuals are visualized as circles and the remaining $m_U - m$ residuals as dots.

4.3. Other rational filters

While SSM and FEAST were originally conceived as contour-integration based eigensolvers, they can also be considered as relying on a more general rational filter, as discussed in Chapter 2.

Recent research in this direction has included several types of rational filters; most notably with Zolotarev approximation [42] and SLiSe/WiSe optimal filters [64, 96], which will both be explained in more detail below. A definition for least-squares rational filters [97] also generates weights for a filter from a set of poles. The field of rational filter approximation in general is beyond the scope of this work; we will focus here on the most prominent research directions so far. A broader discussion applicable to RFEs can be found in [30].

4.3.1. Zolotarev

In [42], the authors considered a rational function for FEAST arising from an elliptic filter: Zolotarev approximation [99].

In order to describe the properties of these functions, we introduce a modified indicator function

$$\mathrm{ind}_{[-G,G]}(x) = \begin{cases} 0 & |x| > G \\ 1 & |x| \leq G \end{cases} \tag{4.3}$$

where $[-G, G] \subset (-1, 1)$.

We restrict our choice of rational functions of type (q, q); that is, functions that can be written as $\frac{f(x)}{g(x)}$, where $f(x)$ and $g(x)$ are real polynomials of degree at most q.

We define the Zolotarev function of this type as

$$r_q^{(Z)} = \frac{s_q(t(z)) + 1}{2} \tag{4.4}$$

where R satisfies

$$G = \frac{\sqrt{R} - 1}{\sqrt{R} + 1}. \tag{4.5}$$

We define

$$t(z) = \sqrt{R}\frac{1+z}{1-z}$$

and

$$s_q(x) = xD\frac{\prod_{j=1}^{q-1}(x^2 + c_{2j})}{\prod_{j=1}^{q}(x^2 + c_{2j-1})}, \quad c_j = \frac{\mathrm{sn}^2(jK(\kappa)/(2q); \kappa)}{1 - \mathrm{sn}^2(jK(\kappa)/(2q); \kappa)}$$

Here, $\mathrm{sn}(w, \kappa) = x$ is the Jacobi elliptic function, and $K(\kappa)$ is the elliptic integral for the modulus κ. These are defined in [42]. The constant D is chosen to satisfy

$$\min_{x \in [-R,-1]} s_q(x) + 1 = \max_{x \in [1,R]} -s_q(x) + 1$$

and $\kappa = \sqrt{1 - 1/R^2}$.

Due to its equioscillating behaviour, $r^Z(x)$ is shown in [42] to be the best uniform approximation of the indicator function $\mathrm{ind}_{[-G,G]}(x)$ on $[-G, G]$ and $[-\inf, -G^{-1}] \cup [G^{-1}, \inf]$. This behaviour was more generally described by Zolotarev in his original definition of these approximations [99], which were chosen to meet these criteria.

4.3.2. SliSe and WiSe

In contrast to filters arising from existing rational functions, such as those based on Zolotarev or quadrature rules, SLiSe and WiSe rational filters are chosen by optimization to minimize RFE iterations [64, 96]. These methods focus on sharp drops outside of I_λ, minimizing the worst-case convergence ratio, and/or requiring fewer vectors in the subspace U. As both weights and poles are determined by non-convex optimization, their solution is considerably more complex than, e.g., least-squares minimizing weights for a provided set of poles. The original SLiSe library [96] obtained rational functions minimizing the weighted least–squares error between the window function over the desired interval, $h(x)$ (2.1), and a rational function, $r(x)$:

$$\int_{-\infty}^{\infty} v(t)|h(t) - r(t)|^2 dt \tag{4.6}$$

The authors discretized this problem and solved it using a residual level function approach. The resulting optimization problem is non-linear and non-convex, but has a differentiable objective function. The choice of weight function, $v(t)$ may rely on box constraints in the optimization problem to prevent poles from approaching the real axis and support compatibility with iterative linear solvers.

This approach was iterated upon to give the WiSe rational functions in [64], where the authors focused on finding a weight function v in the least squares problem minimizing the worst case convergence ratio with a fixed gap parameter $G \in (0, 1)$, as (4.3):

$$\min_{v(t)} \frac{\max_{v(t)\in[-\infty,-G^{-1}]\cup[G^{-1},\infty]} |r(v(t))|}{\min_{v(t)\in[-G,G]} |r(v(t))|}$$

and using this fixed choice of function $v(t)$ in the optimization of (4.6). The choice of G, as for Zolotarev functions, controls the size of the interval over which the rational function must drop from ≈ 1 to ≈ 0; a value closer to 1 corresponds to a narrower gap. This parameter is fixed; these filters may be distinguished by this value. The two minimization problems are solved in a self-consistent fashion, and thus the weight function does not need to be explicitly selected. However, the second optimization problem is non-linear and the gradient is not known; the authors use existing non–linear derivative–free optimization schemes in order to solve find a global minima, which we will also utilize later in this chapter. The initial filter is chosen as the corresponding Zolotarev filter for that degree; the original SLiSe library also considers the Gauss–Legendre filter as a starting point. As we see in Figure 4.3, the resulting poles approach the real axis. The resulting optimized worst–case convergence ratio means that a smaller subspace size is likely needed for these filters. It is possible that a change in box constraints would produce filters better suited for iterative methods; however, the filter may be inhibited by the size of constraint needed to produce a filter truly suitable for iterative schemes.

4.3.3. Least-squares filters

Xi and Saad introduced a method [97] for determining weights based on a set of poles to minimize the weighted least-squares error of a filter $r(x)$ from the window function, $h(x)$ (2.1). To define this in more detail, let us assume that the spectrum has been normalized such that $I_\lambda = [-1, 1]$. Then $h(x)$ takes the value 1 on I_λ and the value 0 elsewhere. As above, we consider a weighted least-squares error function (4.6). Here we define the weight function $v(t)$ such that

$$v(t) = \begin{cases} 0 & |t| > \alpha \\ \beta & |t| \leq 1 \\ 1 & \text{otherwise.} \end{cases} \tag{4.7}$$

Typical values for α are 10 or greater, and β is typically set to a small value, between 10^{-3} and 0.5. Assuming $\beta < 1$, the error of the filter $r(x)$ is weighted most heavily in $[-\alpha, \alpha] \setminus [-1, 1]$. This makes intuitive sense because the specific amplitude of the eigenvectors corresponding to eigenvalues in $[-1, 1]$ is less important; it is more important that they are preserved in some magnitude while eigenvectors outside the interval are dampened.

The definition for the weights w_i, as seen below, is provided in general form in [97].

4.3.3.1. Calculation of least-squares weights

We assume that α, β, z_i for $i = 1, \dots q$ and I_λ are all defined. We additionally define, for $r = \frac{\overline{\lambda}-\underline{\lambda}}{2}$ and $c = \frac{\overline{\lambda}+\underline{\lambda}}{2}$,

$$\begin{aligned} \alpha_L &= c - \alpha r \\ \alpha_R &= c + \alpha r. \end{aligned}$$

Then, the weights w_i for $i = 1, \dots q$ can be found as the solution to the linear equation

$$Gw = d. \tag{4.8}$$

where $G \in \mathbb{C}^{q\times q}$:

$$G_{i,j} = \begin{cases} \frac{1}{\alpha_L - z_j} - \frac{1}{\alpha_R - z_j} + (\beta - 1)\left(\frac{1}{\underline{\lambda} - z_j} - \frac{1}{\overline{\lambda} - z_j}\right) & \bar{z}_i = z_j \\ \frac{1}{z_j - \bar{z}_i}\left(\log\left(\frac{\alpha_R - z_j}{\alpha_L - z_j}\right) - \log\left(\frac{\alpha_R - \bar{z}_i}{\alpha_L - \bar{z}_i}\right)\right) & \\ + \left(\frac{\beta - 1}{z_j - \bar{z}_i}\right)\left(\log\left(\frac{\overline{\lambda} - z_j}{\underline{\lambda} - z_j}\right) - \log\left(\frac{\overline{\lambda} - \bar{z}_i}{\underline{\lambda} - \bar{z}_i}\right)\right) & \text{otherwise} \end{cases} \tag{4.9}$$

and $d \in \mathbb{C}^q$ is defined as

$$d_i = \beta \log\left(\frac{\overline{\lambda} - \bar{z}_i}{\underline{\lambda} - \bar{z}_i}.\right) \tag{4.10}$$

We assume that the shifts z_i, $i = \frac{q}{2} + 1, \dots, q$ are complex conjugates of z_i, $i = 1, \dots, q$, the weights w_i, $i = \frac{q}{2} + 1, \dots, q$ are also complex conjugates of w_i, $i = 1, \dots, \frac{q}{2}$.

4.3.4. Pole placement and iterative solvers

As we observe in Figure 4.3, the poles of the Zolotarev and WiSe filters come much closer, relatively speaking, to the real axis than those for Gauss-Legendre quadrature. This effect is compounded by narrow intervals and dense sections of the spectrum. As matrix size grows, the density of the spectrum will also increase. To gather m eigenvalues, a relatively smaller length I_λ will be required. For $I_\lambda \neq [-1, 1]$ the filters are scaled according to $r = |\overline{\lambda} - \underline{\lambda}|$. This helps them obtain a comparatively very steep drop-off around $x = \left\{\underline{\lambda}, \overline{\lambda}\right\}$, and a resulting improved worst case convergence rate. If the resulting linear systems $(A - z_i B)^{-1}BY$ are solved with a direct solver, this is irrelevant, as long as the shifted system is not numerically singular. However, direct solvers become impractical or impossible to use as problem size and the resulting computational and memory requirements increase. Even if a matrix is sparse, its factorized components may not be. For an $n \times n$ problem, the computational and storage requirements of an LU factorization are $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ respectively [93]. For a scalable solution, we must then look to iterative solvers, which hopefully are $\mathcal{O}(n)$ in both memory and computational requirements. Here, the distance of a pole z_i from the eigenvalues, which in the case of Hermitian eigenproblems are confined to the real axis, control the behaviour of the iterative linear solver with the shifted linear system, as we discuss in the following section. These factors make the shifted linear systems of an RFE more difficult to solve with an iterative method; many methods diverge. But it is also with increased problem size that iterative methods for solving linear systems become

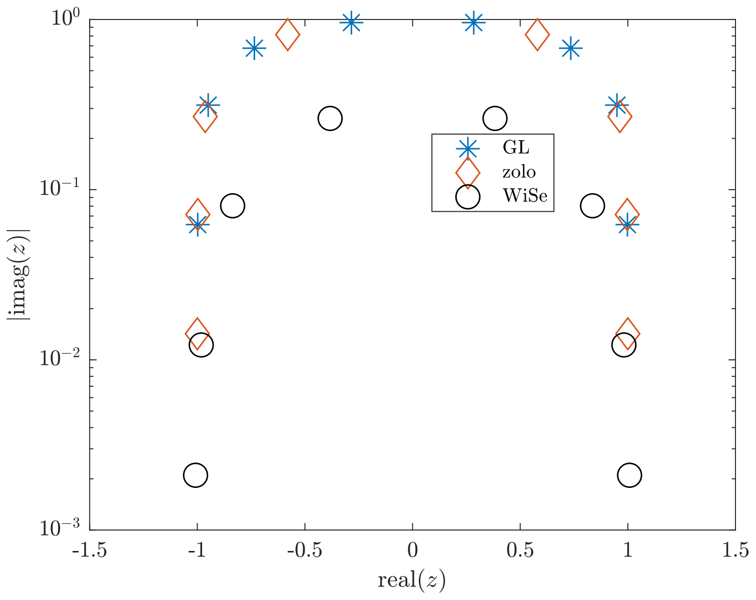


Figure 4.3.: Comparison of poles for rational filters, $I_\lambda = [-1, 1]$: Gauss-Legendre quadrature (GL), Zolotarev (zolo) and WiSe filter. $q = 16$ (the 8 poles with positive imaginary part are shown; the others are reflected across the y axis), $G = 0.98$ for Zolotarev and WiSe filters.

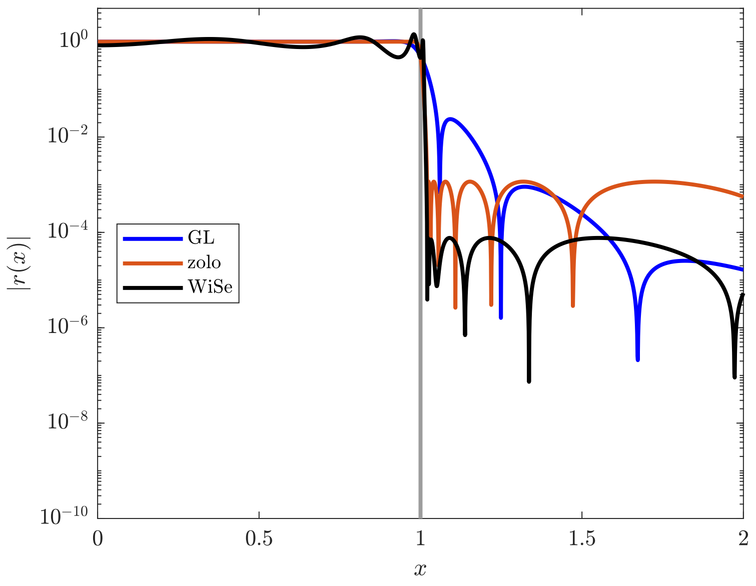


Figure 4.4.: Various rational filters, $I_\lambda = [-1, 1]$: Gauss-Legendre quadrature (GL), Zolotarev (zolo) and WiSe filter. $q = 16$, $G = 0.98$ for Zolotarev and WiSe filters.

competitive in comparison with direct methods for larger matrix size. Indeed the memory requirements for direct methods alone typically make them infeasible past a given problem size. In addition, the desired problem size continues to increase with general computational capacity. For this reason, filters such as Zolotarev and SLiSe/WiSe may be extra challenging for a iterative solver with increasing problem size. We also explore this numerically in Section 4.8.

The other existing strategy to reduce the cost of using an iterative linear solver is to reduce the tolerance to which the linear systems are solved, as seen for the FEAST algorithm in [38] and the inverse iteration in [79]. These authors demonstrate that in early iterations, a larger residual error in the solution of the linear systems may not affect the overall convergence of a subspace iterative method. In [79], the authors show that for subsequent subspace iterations, a smaller residual error for the linear systems may be achieved without additional linear solver iterations. However, if the number of iterations required to reach even a large residual error is high, this strategy may still be cost prohibitive. By considering the underlying rational filter, we hope to reduce the "starting cost" for our iterative linear solver, improving the efficacy of the strategies presented in these works.

4.4. Iterative solver convergence

As described above, the construction of a rational filter in an RFE requires the solution of the shifted linear systems $(A - z_i B)U = BY$, $i = 1, \dots q$. Let us define $\Phi_i = (A - z_i B)$. In order to easily relate the eigenvalues of Φ_i to the eigenvalues being sought, we limit ourselves for the rest of this chapter to standard Hermitian eigenvalue problems, with $B = I$.

These shifted linear systems are considered difficult to solve with iterative solvers; we will now discuss why this is and consider techniques for reducing this difficulty. As defined previously, z_i is a complex shift. If z_i is chosen close to the real axis, it may be very near one of the eigenvalues of A. Let us define the eigenvalues of Φ as γ_i, $i = 1, \dots, n$, ordered according to $|\gamma_1| \geq |\gamma_2| \geq |\gamma_n|$. This means the eigenvalue of the shifted system with smallest magnitude, γ_n, will be the distance between z_i and the closest eigenvalue in the spectrum of A:

$$|\gamma_n| = \min_j |\lambda_j - z_i| \quad j = 1, \dots, n. \tag{4.11}$$

This may be significantly smaller than λ_n, the smallest magnitude eigenvalue of A. In general, we may expect eigenvalues close to 0 to slow convergence of an iterative method. In particular, if a Krylov method is used, convergence at the k^{th} iteration depends in the worst case on the maximum value of the minimizing monic polynomial (that is, the polynomial of degree k with value $p(0) = 1$; this is referred to as the "normalizing condition") at the eigenvalues of Φ_i.

From [71], we obtain the following bound for the residual for iterative methods like GMRES, as well as a further discussion of general bounds for Krylov methods. A similar bound holds for the conjugate gradient (CG) method

$$\frac{\|r_k\|}{\|r_0\|} \leq \kappa(V) \min_{p_k} \max_{\gamma_i} |p_k(\gamma_i)| \tag{4.12}$$

where r_k is the residual at the k^{th} iteration and in the case of normal matrices, $\kappa(V) = 1$.

The normalizing condition means that if an eigenvalue d is very close to 0, obtaining a value close to 0 at d and 1 at 0 is very difficult. Convergence may be inhibited. We will explore this topic further for selected iterative methods, but the results may be applicable for Krylov schemes in general.

Given our restriction to standard Hermitian eigenvalue problems, we expect Φ to be normal and thus $\kappa = |\frac{\gamma_1}{\gamma_N}|$. Many of the results are likely applicable to generalized eigenproblems as well, but such an extended analysis is beyond the scope of this work.

4.5. Kaczmarz sweeps and CG acceleration

Finding a method that can handle the shifted linear systems as described above is in general very challenging. The CGMN method has shown the ability to provide reliable convergence for the shifted linear systems arising from contour-integration based eigensolvers [3, 36]. We discuss this method and its implementation here, stepping through the ideas and algorithms it is based on in chronological order. In the remainder of this section, we consider the solution of a linear system of equations, $\Phi u = y$, where $\Phi \in \mathbb{C}^{n\times n}$, $u \in \mathbb{C}^n$, $y \in \mathbb{C}^n$. We refer to the i^{th} row of Φ as ϕ_i and the i^{th} entry of y as y_i.

4.5.1. Kaczmarz sweeps

The Kaczmarz method [61] applies orthogonal projections of the approximate solution iteratively onto the hyperplanes associated with each of the rows of a matrix. At each step, the orthogonal projector associated with the i^{th} linear equation is applied. The iterative solution at the $k+1^{st}$ step, u^{k+1}, is thus defined as

$$u^{k+1} = u^k - \omega \frac{y_i - \langle \phi_i, u^k \rangle}{\|\phi_i\|} \bar{\phi}_i \quad i = k \bmod n \tag{4.13}$$

Algorithm 13: Kaczmarz sweep for the solution of $\Phi u = y$.

```
Input  : $\Phi \in \mathbb{C}^{n\times n}$, $y \in \mathbb{C}^n$, $u \in \mathbb{C}^n$, $\omega \in \mathbb{R}$
Output: $u^n \in \mathbb{C}^n$
Function kaczsweep($\Phi$,$y$,$u$,$w$):
    Set $u^0 = u$
    for each $k$, $0 \le k < n$ do
        $u^{k+1} = u^k + \omega \frac{(y_i - \langle \phi_i, u^k \rangle)\bar{\phi}_i}{\|\phi_i\|}, \quad i = k$
    end
```

In fact, ω could be chosen as a distinct value ω_i for each row. In this thesis, the simplifying assumption of a single constant is made.

We can also perform this sweep through the rows of Φ in a different order, e.g., backwards. We call this method `backwardkaczsweep`. In anticipation of the coming subsection, we also introduce here the symmetric double sweep `doublekaczsweep`, consisting of successive forwards and backwards Kaczmarz sweeps.

4.5.2. CGMN

The acceleration of Kaczmarz sweeps with the conjugate gradient (CG) method was first introduced as CGNM [18, 40]. We note that the definition and subsequent analysis require a symmetric semidefinite matrix. The symmetry requirement is satisfied for Φ when only the diagonal has a nonzero imaginary component, which we assume to be the case for the remainder of this chapter. Here, the method relies on the equivalence of Kaczmarz sweeps with successive overrelaxation (SOR) on the normal equations; a coupling of a forward and backward sweep is thus equivalent to symmetric successive overrelaxation (SSOR) and is symmetric, thus compatible with CG.

4.5.3. Parallelization and block multicoloring

We utilize a shared memory implementation of CGMN as described in [6]. To avoid write conflicts and performance bottlenecks in shared memory, we could use a multicoloring approach, as has been previously considered [36]. In this strategy, columns of the same color have no indices in common; this is called *structural*

Algorithm 14: CGMN for the solution of $\Phi u = y$

```
Input  : $\Phi \in \mathbb{C}^{n\times n}$, $y \in \mathbb{C}^n$, $\omega \in \mathrm{Re}$
Output: $u \in \mathbb{C}^n$
Function cgmn($\Phi$,$y$,$w$):
    Choose a starting block of $m$ vectors $Y$
    Set $u^0 \in \mathbb{C}^N$ to arbitrary value
    Set $p^0 = r^0 =$ doublekaczsweep$(\Phi, y, u^0, \omega) - u^0$
    while not converged do
        $q^k = p^k -$ doublekaczsweep$(\Phi, \mathbf{0}, p^k, \omega)$
        $\alpha_k = \|r^k\|^2 / \langle p^k, q^k \rangle$
        $u^{k+1} = u^k + \alpha_k p^k$
        $r^{k+1} = r^k - \alpha_k q^k$
        $\beta_k = \|r^{k+1}\|^2 / \|r^k\|^2$
        $p^{k+1} = r^{k+1} + \beta_k p^k$
        $k = k + 1$
    end
```

orthogonality. However, better performance has been shown from a block multi-coloring approach [3]. In this approach, structural orthogonality is only enforced between different blocks of the same color. We rely on such an implementation here, utilizing the improved RACE multicoloring strategy [5].

The matrix is first permuted to minimize bandwidth, using the Reverse-Cuthill-McKee algorithm [24]. Then, the RACE block-coloring algorithm is performed, as described in [5]. Kaczmarz sweeps may then be performed in parallel over blocks of the same color, with each block assigned to a thread. Within a block, the sweeps are performed sequentially, cycling through the rows. The full CGMN algorithm with block-multicolored parallel Kaczmarz sweeps would then perform a forward sweep over each color, followed by a backward sweep; the `doublekaczsweep` algorithm. Other details associated with block multicoloring for CGMN, such as the possibility of load balancing, are discussed further in [3].

A distributed memory parallelization has also been introduced. A strategy called CARP-CG [41] involves averaging results obtained from different distributed processes to obtain a final result. Although the use of this strategy is promising for RFE methods[36], it is beyond the scope of this work.

4.5.4. Implementation of CGMN

The Kaczmarz sweeps are implemented in the GHOST library [67], and incorporated with the CGMN method as implemented in [6]. The weighting of $\omega = 1$ is used for the Kaczmarz sweeps in the remainder of this work. As discussed above, block-multicoloring occurs a-priori with the RACE library [5].

We also note that the definition above is for a single right hand side vector b and solution x. The extension of the above definition to our use case with block vectors requires only their substitution for the single vectors; the analysis is the same.

4.6. Predicting cost

For a given rational function, we would like to be able to predict the “cost" of using CGMN to solve all the linear systems. We re-emphasize that this is by far the most significant expense in projection-based methods. We may consider the cost as the total number of CGMN iterations required over all poles and all RFE iterations, or, keeping parallelism in mind, the maximum number of CGMN iterations required for a pole over all RFE iterations. As described in Chapter 2, the linear systems of FEAST can be solved independently and in parallel. When choosing to minimize the maximum number of CGMN iterations per pole, we are focusing on the possible reduction in cost or time for this parallel solution. It is the slowest of these linear solves, with presumably the most costly solution of the linear system, that acts as a barrier for the rest of the iteration.

4.6.1. Analyzing the behaviour of CGMN

We analyze the behaviour of CGMN on the shifted linear systems $\Phi U = Y$, where $\Phi = A - zI$ for A a 1000×1000 generated [30] (see Section 4.1.1) sparse matrix. A, which we will name Gen1000-40pt01, has 40 eigenvalues inside $[-0.01, 0.01]$ and the remainder of its eigenvalues equidistantly located in $[-1, 1] \setminus [-0.01, 0.01]$. Thus, the spectrum is relatively dense in a small interval around 0, and less dense away from this interval. We consider the number of CGMN iterations required for all 64 random right hand side vectors in Y to reach a relative residual tolerance of 10^{-8} or 10^{-12}. This is plotted in Figures 4.5 and 4.6 vs. the condition number of the shifted system, that is,

$$\|\Phi\| \cdot \|\Phi^{-1}\| = \frac{\max_i |z - \lambda_i|}{\min_i |z - \lambda_i|} \tag{4.14}$$

If λ_i is close to real(z), $\min_i |z - \lambda_i|$ will be close to imag(z). The values are plotted on lines associated with the distinct real values of z. These are also associated with color, blue for shifts with small real value, moving to red away from the origin. We observe that after a period of roughly (log-)linear growth, the number of iterations approaches a threshold, which appears correlated with the density of the spectrum at that value of real(z).

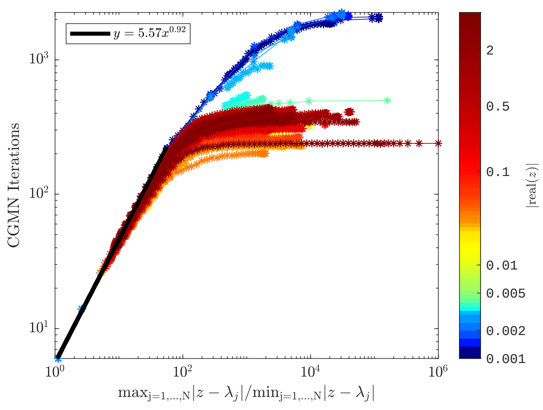


Figure 4.5.: (a)

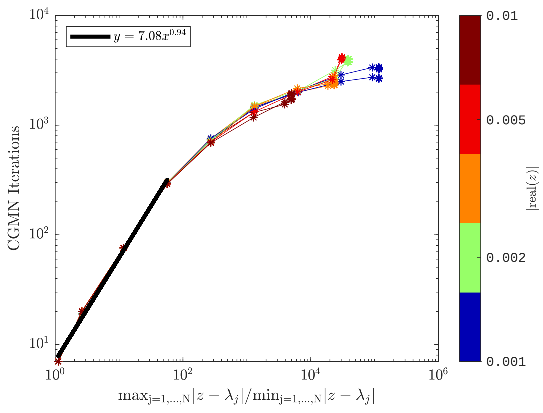


Figure 4.6.: (b)

$\kappa(\Phi)$ vs. CGMN iterations for Gen1000-40pt01 with residual tolerance 10^{-8} (a) and 10^{-12} (b). The distance of $\mathbb{R}(z)$ from 0 is shown as a color gradient. A line of best fit for the linear growth portion of the (logarithmically scaled) data is also shown, transformed back to its exponential form.

Next, we analyse the relationship for a larger matrix, now using an approximation for the minimal distance between the pole and an eigenvalue of A. Here, A is a graphene matrix, GraIII-11k-b, with 40 eigenvalues in I_λ. More specific details of this system can be found in Table 4.1. Again, we consider the number of CGMN iterations required to solve the system $\Phi U = Y$ to reach a relative residual tolerance of 10^{-8} or 10^{-12} with 64 random right hand side vectors in Y. We now use the approximation $|h/4 + \text{imag}(z)\mathbf{i}|$ to approximate the distance from the smallest eigenvalue, where h is the local density in I_λ, that is, $h = \frac{\overline{\lambda}-\underline{\lambda}}{\tilde{m}}$. Since we predict eigenvalues with spacing h, the expected value of the distance from an eigenvalue along the real axis is $\frac{h}{4}$. The separation in the imaginary plane is clearly just the imaginary part of the pole z, as our eigenvalues are real. We thus form the estimation for the condition number

$$\kappa_{est}(\Phi) \approx \frac{\max\{|\lambda_1 - z|, |\lambda_n - z|\}}{|h/4 + \text{imag}(z)\mathbf{i}|} \tag{4.15}$$

In Figures 4.7 and 4.8, we observe the linear relationship (up to some threshold) between this estimate for the condition number and the number of CGMN iterations required to meet this threshold. Here $\mathbb{R}(z)$ is in I_λ; the results shown in this figure include various values of $\mathbb{R}(z)$. Unlike the change in $\mathbb{C}(z)$, this variation shows little effect on the number of iterations as the density of eigenvalues is similar close to all values of $\mathbb{R}(z)$ considered.

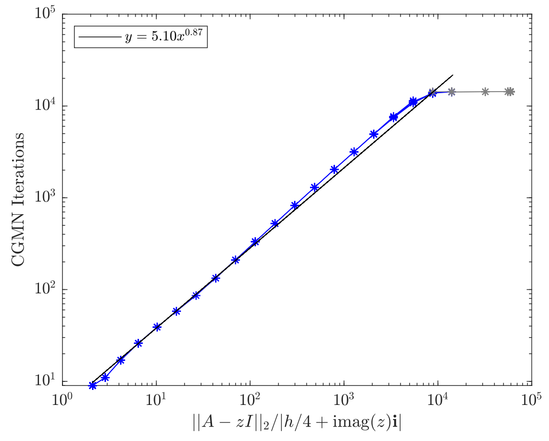


Figure 4.7.: (a)

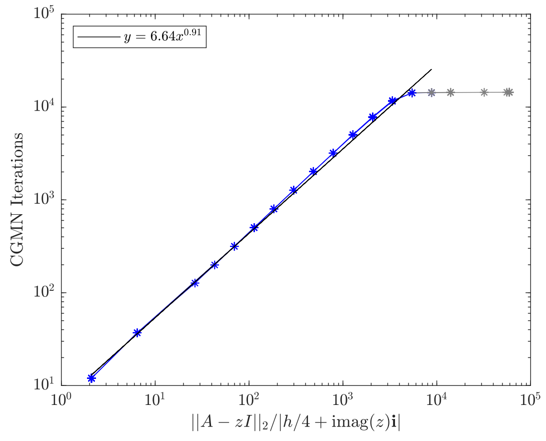


Figure 4.8.: (b)

$\kappa_{est}(\Phi)$ vs. CGMN iterations for GraIII-11k-b matrix with residual tolerance 10^{-8} (a) and 10^{-12} (b). A line of best fit for the linear growth portion of the (logarithmically scaled) data is also shown, transformed back to its exponential form. Values past the threshold of stagnation are shown in grey.

4.6.2. Condition number relationship

We have observed that for a sample shifted matrix $\Phi = A - zI$, the number of CGMN iterations required increases steadily as the shift approaches the real axis, at least until some threshold for iterations is reached. We have used a linear fitting to the (logarithmically scaled) linear growth portions of data for the GraIII-11k-b problem in Figures 4.7 and 4.8. The line of best fit (an exponential expression when transformed from logarithmic scaling) is similar, for a matching residual tolerance, to the relationship shown between $\kappa(\Phi)$ and CGMN iterations for the Gen1000-40pt01 problem. Thus, it appears that we are able to form a rough estimate for the number of CGMN iterations for a general linear system of equations with the relative residual tolerance 10^{-8}

$$_{C}k_{pred}(\Phi) = 5.5 \times \kappa_{est}(\Phi)^{0.9} \tag{4.16}$$

or 10^{-12}

$$_{C}k_{pred}(\Phi) = 7 \times \kappa_{est}(\Phi)^{0.9}. \tag{4.17}$$

We can observe the efficacy of this estimate for more matrices, ranging in size from 1000×1000 to $\approx 33000 \times 33000$. Details for these systems can be found in Table

Matrix	n	$\|A\|_2$	$\overline{\lambda} - \underline{\lambda}$
Gen1000-40pt01	1000	5	0.02
Gen10000-1	10000	1	0.02
Gen10000-100	10000	100	2
rgg_n_2_15_s0-a	32768	1	0.00058
SiO-b	33401	1	0.005
GraIII-11k-b	11604	1	0.0039

Table 4.1.: Sizes, norm, and length of search interval I_λ for 6 test matrices. Matrices Gen* were generated [30]. GraIII-11k-b was generated from graphene modeling [23] and two test matrices from the SuiteSparse Matrix Collection [25]. More information for the last three matrices may also be found in Table A.1.

4.1. Each system $\Phi U = Y$ was solved to a relative tolerance of 10^{-8} using CGMN for a block vector Y with 64 random columns.

These values are shown in Figure 4.9. We observe that this fit is reasonably good for the linear growth (in logarithmic scaling) portion of the data. As we will see in later sections, we are typically interested in predicting the relatively small number of linear solver iterations required from a (relatively) small value of κ_{est}. Furthermore, this relationship is not the only factor in determining the number of linear solver iterations required for an RFE. Therefore, the rough estimate gained thus far is sufficient for the time being.

We expect that although the fit is acceptable for our intended purpose with these problems, it will lose predictive power if the linear systems under consideration are very different from the ones considered here, especially if κ_{est} does not fall into the range of values tested. However, if a new set of problems under consideration requires a new estimate, the fit function for the expected number of linear solver iterations is relatively cost effective to obtain, requiring the solution of some hundreds of block linear systems of equations. It may be done a-priori for a subset of the test problems under consideration.

We also observe that the number of CGMN iterations reaches a threshold for some value of κ_{est} with each matrix problem. A predictive form for this threshold is not known; though it appears to be correlated with the density of the spectrum around real(z), or similarly, the overall matrix size. We expect to be mainly interested in predicting values in the constant growth portion of linear solver iterations. This is especially true for large matrices, where this threshold is expected to be beyond the bound of a reasonable number of CGMN iterations. Furthermore, inclusion of a bound would result in additional complications for our upcoming optimization scheme. Therefore, explorations into this topic remain subjects of future research.

We note that this correlation between the number of linear solver iterations and

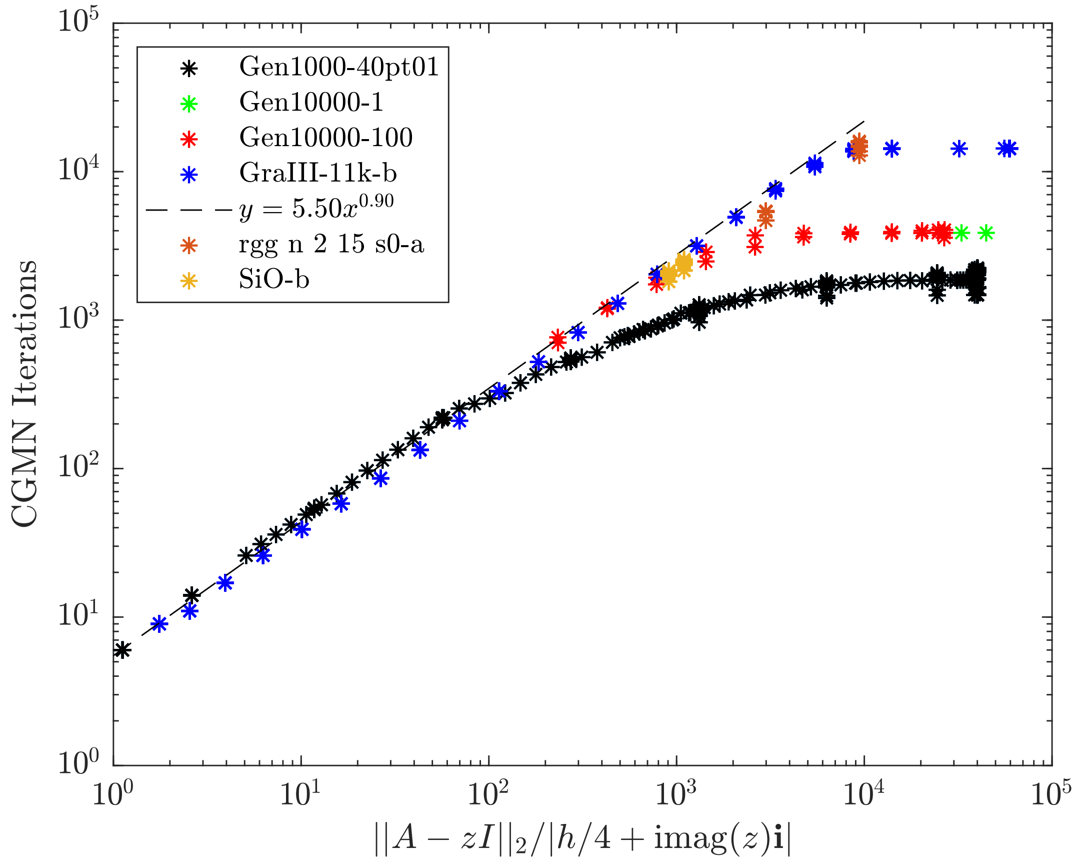


Figure 4.9.: $\kappa_{est}(\Phi)$ vs. CGMN iterations for multiple eigenproblems (listed in Table 4.1) with residual tolerance 10^{-8}, along with fitting function provided in (4.16).

the condition number up to some threshold is likely true for other Krylov iterative solvers. Galgon [30] observed an increase in the number of GMRES iterations required for the solution of shifted linear systems with poles close to larger clusters of eigenvalues. We know that the convergence behaviour of all Krylov subspace methods is bounded by the maximum value of the minimizing polynomial on the eigenvalues [93]. Therefore, we do anticipate that these results generalize, at least in theory. However, this is difficult to explore, as most iterative solvers are unable to solve these shifted linear systems, and often diverge. We perform a rudimentary analysis for GMRES in the following section. Future research could include testing to see if a relationship exists for other linear solvers, potentially combined with appropriate preconditioners.

4.6.3. Analyzing the behaviour of GMRES

We next explore the prediction of number of iterations for a different iterative linear solver, GMRES. A description of this solver is given in Section 2.8.2.1. We used an implementation of restarted GMRES from the PETSc KSP library [14, 15, 16], with GMRES restarted every 30 iterations and using modified Gram-Schmidt orthogonalization.

We observe the behaviour of restarted GMRES on the shifted linear systems $\Phi U = Y$ with $\Phi = A - zI$ for the same test matrices as above. Our sample problems were Gen1000-40pt01, and GraIII-11k-b, as described in Table 4.1. We show, for a variety of shifts z, with various values of $\mathbb{R}(z)$ within I_λ, the average number of GMRES iterations required for 64 problems to reach an absolute residual tolerance of 10^{-12}, when starting with a random right hand side vector. This is again compared with the condition number of the shifted matrix Φ, (4.14), for the Gen1000-40pt01 matrix, as shown in Figure 4.10 and our estimate for the condition number $\kappa_{est}(\Phi)$, (4.15) for the GraII-11k-b matrix, as shown in Figure 4.11.

We observe a linear relationship between the values, as shown by the line of best fit provided in each graph. We also observe some distinction in the slope of this graph between different problems. This is not too surprising; as with any linear solver, we cannot expect a uniform estimate to be obtained across all matrices. However, the slopes are similar, suggesting that with a rough estimate, we can obtain a reasonable prediction for the number of GMRES iterations required, especially for similar problems. Indeed, in Figure 4.12 we see the estimated vs actual GMRES iterations plotted along with two possible linear estimations. Again, the average number of GMRES iterations required for 64 right hand sides to reach a absolute tolerance of 10^{-12} is shown. Both linear estimates appear reasonable, especially when a rough estimate is sufficient.

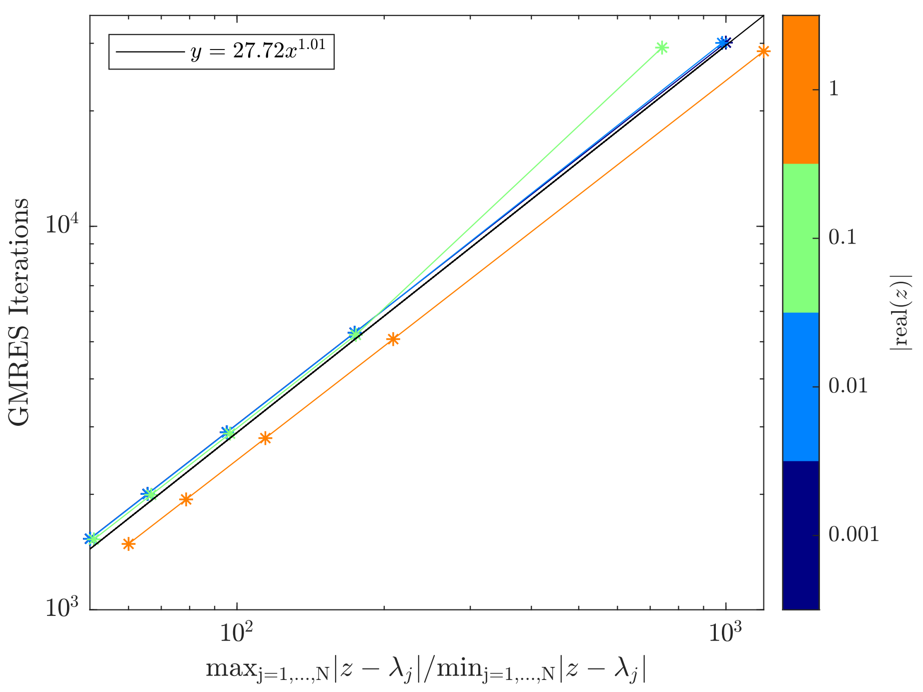


Figure 4.10.: GMRES iterations vs $\kappa(\Phi)$ for Gen1000-40-pt01 matrix. The distance of $\mathbb{R}(z)$ from 0 is shown as a color gradient.

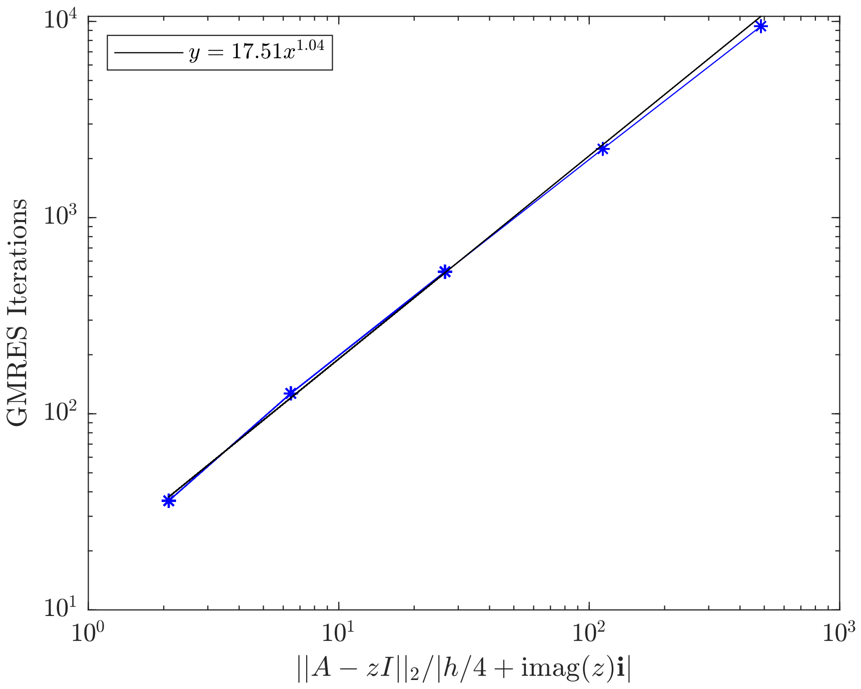


Figure 4.11.: $\kappa_{est}(\Phi)$ vs. GMRES iterations for GraIII-11k-b matrix.

Matrix	n	$\|A\|_2$	$\overline{\lambda} - \underline{\lambda}$
Gen1000-40pt01	1000	5	0.02
SiH4-b	5041	1	0.0076
GraIII-1k-a	11604	1	0.085
GraIII-11k-b	11604	1	0.0039

Table 4.2.: Sizes, norm, and length of search interval I_λ for 4 test matrices. Matrix Gen100-40pt01 was generated [30]. GraIII-11k-b was generated from graphene modeling [23] and SiH4-b obtained from the SuiteSparse Matrix Collection [25]. More information for the last three matrices may also be found in Table A.1.

Interestingly, and without an obvious explanation, no threshold is observable for the number of GMRES iterations as for the number of CGMN iterations. This means that the linear relationship is likely more accurate for larger condition numbers, in comparison to CGMN, though we are unlikely to be interested in the resulting high iteration counts. It is certainly possible that this simply appears for much larger values of κ_{est}. The expected growth rate of GMRES iterations is also much faster than for CGMN. Perhaps with an appropriate preconditioner, the number of GMRES iterations would be similarly decreased. However, from a theoretical point of view, it is helpful to observe the behaviour of this basic iterative linear solver.

We will use the following estimate for the number of GMRES iterations, ${}_Gk_{pred}$, in the remainder of this chapter, unless otherwise stated,

$$ {}_Gk_{pred}(\Phi) = 30\kappa_{est}(\Phi), \tag{4.18}$$

as it appears slightly more accurate for increasing numbers of iterations. As with CGMN, if this approach was to be applied to a different set of matrices (e.g., with large changes in matrix size, matrix sparsity, etc.) it would likely be valuable to generate a new estimate for this linear relationship for the set of matrices under consideration. It is also possible that GMRES will not converge at all, and may even diverge for some problems. The use of an appropriate preconditioner would also potentially reduce the number of linear solver iterations required and increase robustness, though this would again require a new estimate for the relationship between condition number and linear solver iterations, which may be non-linear.

4.6.4. Predicting RFE iterations

As previously discussed, the convergence of an RFE algorithm in exact arithmetic is a known function (4.2) of the value of the rational filter $r(x)$ at the eigenvalues. We would like to be able to predict the convergence of an RFE without full

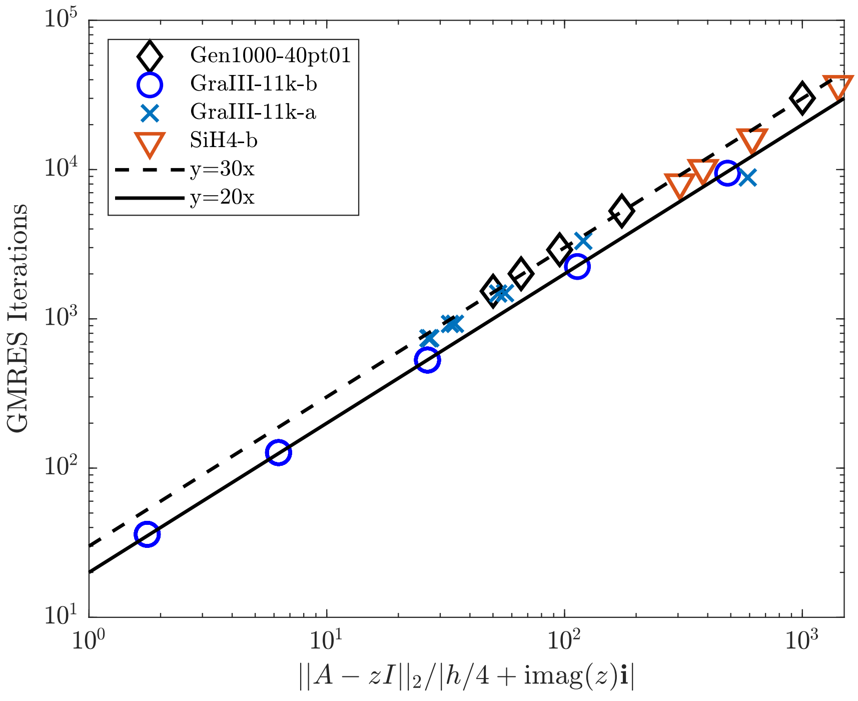


Figure 4.12.: $\kappa_{est}(\Phi)$ vs. GMRES iterations for multiple systems.

knowledge of the eigenvalues. A prerequisite for an RFE is an approximation for the number of eigenvalues in I_λ, $\tilde{m}$. With this value, we have an approximation for the local density of eigenvalues in and around I_λ, and thus can generate approximations for $\lambda_1, \ldots, \lambda_{m_U+1}$ (sorted such that $r(\lambda_1), \geq r(\lambda_2), \geq \ldots, r(\lambda_{m_U+1})$). Since $r(x)$ cannot be expected to be monotonically decreasing, we need more than $m_U + 1$ evaluations of the function to estimate convergence. Furthermore, we want to better capture oscillations in the underlying function r by additional evaluations. We generate a sequence of approximate eigenvalues δ_j, $j = 1, \ldots, N$ with equidistant spacing twice the expected eigenvalue density $h = \frac{I_\lambda}{2\tilde{m}}$ in the window $[\max(\lambda_{\min}, c - 10r), \min(\lambda_{\max}, c + 10r)]$. Again, $\lambda_{\min}$ and $\lambda_{\max}$ are the global minimum and maximum eigenvalues (or estimates of these); they are included in the expression here to avoid unnecessary function evaluations outside of the spectrum. We expect this expanded window to contain an approximation for $r(\lambda_{m_U+1})$, assuming that the filter does not increase in value again far from I_λ. After evaluating $r(\delta)$ and sorting according to $r(\delta_1) \geq r(\delta_2) \geq \ldots r(\delta_N)$, we can determine the number of iterations expected for all eigenvalues inside I_λ to converge to the desired residual tolerance, limited by the eigenvalue expected to converge most slowly, $\lambda_{\tilde{m}}$. As we have doubled the frequency of function evaluations within the interval, we use a modified ratio:

$$l_{pred} = \frac{\log(\text{tol}_{\text{RFE}})}{\log\left(\frac{r(\delta_{4\lfloor m_U/2\rfloor+1})}{r(\delta_{4\lfloor \tilde{m}/2\rfloor})}\right)}. \tag{4.19}$$

4.6.5. Choosing weights

A significant parameter in the choice of filter is the choice of weights, w_i in equation (4.1).

We choose weights that, given a set of poles z_i, minimize the least-squares norm from the (parametrized) window function, as defined in [97] and described in Section 4.3.3. By using a weighted least-squares error function, we focus on ensuring that outside of I_λ, the resulting rational function is close to 0, which is relatively more important than ensuring values close to 1 inside I_λ. As previously noted, the resulting least-squares problem that must be solved to compute the weights can become very ill-conditioned. To reduce error, we use SVD-regularization, which is effective to a given degree of instability. However, in our search for poles z_i, we penalize sets which induce a problem that is ill-conditioned to the point of numerical singularity. The penalty is chosen to be linearly proportional to the relative loss of rank. This process is described in full in Algorithm 15.

Algorithm 15: Finding least-squares rational filter weights

```
Input  : $z_1, \dots, z_q \in \mathbb{C}$, $\beta$, $\alpha \in \mathbb{R}$
Output: $w \in \mathbb{C}^q$, $\frac{1}{\sum_{\text{rank}}} \in \mathbb{R}$
Function lsrational($z_1, \dots, z_q$,$\alpha$,$\beta$):
    Form $G, d$ per (4.8)
    if $\kappa(G) > 10^{12}$ then
        $[U, \Sigma, V] = \text{svd}(G)$
        $\Sigma = \text{diag}([\sigma_1, \sigma_2, \dots, \sigma_q])$
        $\text{rank} = \sum_{i=1}^{q} 1\,[\sigma_i > q\epsilon_{mach}(\max_i(\sigma_i)))]$    ▷ Calculate rank of $G$
        $\frac{1}{\sum_{\text{rank}}} = \frac{q}{\text{rank}}$    ▷ Inverse rank number
        $U = U\,[:, 1 : \text{rank}]$
        $V = V\,[:, 1 : \text{rank}]$
        $\Sigma = \text{diag}\,(\sigma_i) \quad i = 1, \dots, \text{rank}$
        $w = V\Sigma^{-1}U^H d$
    else
        $w = G^{-1}d$
        $\frac{1}{\sum_{\text{rank}}} = 1$
    end
```

4.7. Optimization

4.7.1. Cost function

We seek a filter $r(x)$ that minimizes the total number of linear solver iterations for the RFE solver, either for the most expensive pole or for the sum over all poles, as described above. We assume an estimation has been obtained for the number of iterations of the linear solver for solving a single (block) linear system of equations, k_{pred}. Here, the linear solver may be either CGMN or GMRES, and the estimate is based on the condition number of the shifted matrix, as described in Section 4.6. We begin by fixing the value of q; optimization over this value is a separate problem.

For the normalized eigenproblem, $I_\lambda = [-1, 1]$, the values of the poles and weights of the rational function, z_i and w_i, are chosen to obey:

$$\begin{aligned}
\operatorname{Re}(z) > 0 \quad \& \quad \operatorname{Im}(z) > 0, &\quad i = 1, \ldots, \frac{q}{4} \\
&\quad \alpha > 0, \beta > 0 \\
\operatorname{Re}(z_{i+\frac{q}{4}}) = -\operatorname{Re}(z_i), &\quad i = 1, \ldots, \frac{q}{4} \\
\operatorname{Re}(z_{i+\frac{q}{2}}) = -\operatorname{Re}(z_i), &\quad i = 1, \ldots, \frac{q}{4} \\
\operatorname{Re}(z_{i+\frac{3q}{4}}) = \operatorname{Re}(z_i), &\quad i = 1, \ldots, \frac{q}{4} \\
\operatorname{Im}(z_{i+\frac{q}{4}}) = \operatorname{Im}(z_i), &\quad i = 1, \ldots, \frac{q}{4} \\
\operatorname{Im}(z_{i+\frac{q}{2}}) = -\operatorname{Im}(z_i), &\quad i = 1, \ldots, \frac{q}{4} \\
\operatorname{Im}(z_{i+\frac{3q}{4}}) = -\operatorname{Im}(z_i), &\quad i = 1, \ldots, \frac{q}{4} \\
[w_1, \ldots, w_q] = \texttt{lsrational}&([z_1, \ldots, z_q], \alpha, \beta)
\end{aligned}$$

We may additionally define upper limits on our values of z_i, to prevent poles from travelling unreasonably far from the real axis or outside of I_λ:

$$10^{-3} < \operatorname{Re}(z) < 1.1 \quad \& \quad 10^{-10} < \operatorname{Im}(z) < 10, \quad i = 1, \ldots, \frac{q}{4}.$$

That is, we optimize over $\frac{q}{4}$ poles z_i in the upper right-hand quadrant of the complex plane. These are then symmetrically reflected over the complex plane to form the remaining poles.

We will re-frame the optimization problem with a number of simplifying assumptions.

$$\min_{z_i, i=1\dots\frac{q}{4}, \alpha, \beta} \sum k_{pred}^{\Sigma} \tag{4.20}$$

where the total expected number of linear solver iterations, $\sum k_{pred}^{\Sigma}$, is defined as

$$\sum k_{pred}^{\Sigma} = \sum_{i=1}^{q} k_{pred}(\Phi_i) l_{pred}(\Phi_i, z_i, w_i)$$

or

$$\min_{z_i, i=1\dots\frac{q}{4}, \alpha, \beta} \sum k_{pred}^{\max} \tag{4.21}$$

where the maximum number of linear solver iterations for a single pole, $\sum k_{pred}^{\max}$ is defined as

$$\sum k_{pred}^{\max} = \max_{i=1\dots q} k_{pred}(\Phi_i) l_{pred}(\Phi_i, w_i, z_i).$$

The weights are chosen to minimize the least-squares error, as described in Section 4.3.3 according to (4.8). We additionally restrict our choice of weights by constricting $\beta \in [0.0001, 2.0]$ and $\alpha \in [0.1, 100.0]$.

These can then be shifted and scaled to fit the original I_λ according to

$$\begin{aligned} z_i &= c + r z_i, \quad i = 1, \dots, q \\ w_i &= r w_i, \quad i = 1, \dots, q. \end{aligned}$$

We can then evaluate $k_{pred}(\Phi_i)$ and $l_{pred}(\Phi_i, w_i, z_i)$ to determine the cost at $\left[z_1, \dots, z_{\frac{q}{4}}\right]$.

An algorithmic summary of the evaluation of the cost function is shown in Algorithm 16

4.7.2. Visualization of cost function

In the simplest case of $\frac{q}{4} = 1$ we can visualize our cost function over the upper right-hand quadrant of the complex plane.

We do this for a sample eigenproblem, GraIII-11k-b, defined in Table 4.1 with additional matrix information in Table A.1. We visualize the cost for the unscaled poles, z_i in $I_\lambda = [-1, 1]$. The cost is actually calculated for the poles shifted and

Algorithm 16: Evaluate cost function. Input values that are mutable are shown in red.

```
Input  : $z = [z_1, \dots, z_{q/4}] \in \mathbb{C}$ poles of $r(x)$
         $\beta, \alpha \in \mathbb{R}$ Parameters for least-squares rational weights
         $\underline{\lambda}, \overline{\lambda}$ Bounds for $I_\lambda$
         $\lambda_{\min}, \lambda_{\max}$ Estimates for global min and max eigenvalue
         $\tilde{m}$ Expected number of eigenvalues in $I_\lambda$
         $\delta$ Estimate for eigenvalues in and around $I_\lambda$
Output: $\sum k_{pred}^{\sum}$ or $\sum k_{pred}^{\max}$
Function evaluatecost($z_1, \dots, z_{q/4}$,$\beta$, $\alpha$,$\underline{\lambda}$, $\overline{\lambda}$,$\lambda_{min}$, $\lambda_{max}$,$\tilde{m}$,$\delta$):
    $z = [z; -\bar{z}; \bar{z}; -z]$
    $[w, \frac{1}{\sum_{\text{rank}}}]$ = lsrational($z_1, \dots, z_q, \beta, \alpha$)
    $h = \frac{\overline{\lambda} - \underline{\lambda}}{\tilde{m}}$
    for $i = 1, \dots, q/4$ do
        $\text{dist}_{\text{exp}} = |h/4 + \text{Im}(z_i)\mathbf{i}|$          ▷ Expected distance from closest ew
        $\kappa_{est}(\Phi_i) = \max(|\lambda_{\max} - |z_i|\,|), |\lambda_{\min} - |z_i|\,|)/\text{dist}_{\text{exp}}$
        Estimate $k_{pred}(\Phi_i)$                ▷ Evaluate (4.16), (4.17) or (4.18)
    end
    Evaluate $r(\delta_i)$, sort such that $r(\delta_1) \geq r(\delta_2) \geq \dots \geq r(\delta_{\text{end}})$
    $l_{pred} = \log(\text{tol}_{\text{RFE}}) / \log\left(\frac{r(\delta_{m_U+1})}{r(\delta_{\tilde{m}})}\right)$          ▷ Evaluate (4.19)
    $\sum k_{pred}^{\sum} = \sum_{i=1}^{q/4} k_{pred}(\Phi_i) \times l_{pred}(\Phi_i, z_i, w_i) \times \frac{1}{\sum_{\text{rank}}}$
    $\sum k_{pred}^{\max} = l_{pred}(\Phi_i, z_i, w_i) \times \frac{1}{\sum_{\text{rank}}} \times \max_{i=1 \dots q/4} k_{pred}(\Phi_i)$
```

scaled to fit the interval I_λ specified in Table 4.1. We predict, using the cost function described in Algorithm 16, the number of CGMN and RFE iterations required at (unscaled) poles z_i scattered in $[0, 1] \times [0, 2]$. At each iteration, $m_U = 64$, while the actual number of eigenvalues in I_λ is 40. The residual tolerances of the RFE and the linear solver were each chosen as at 10^{-8}. The weights of the rational function $r(x)$ were determined by Algorithm 15, with fixed values of $\alpha = 4.5$ and $\beta = 0.01$.

As we can see in Figure 4.15 even in two dimensions, we are optimizing over a highly variable surface. Indeed, in optimizing $\sum_C k_{pred}^{\sum}$ or $\sum_C k_{pred}^{\max}$ we are optimizing over the product of two surfaces. We visualize each of these distinctly; the CGMN iterations per RFE iteration as seen in Figure 4.13, the estimated number of RFE iterations, l_{pred} in Figure 4.14, and the total number of CGMN iterations ($\sum_C k_{pred}^{\sum}$ and $\sum_C k_{pred}^{\max}$ are equivalent here.)

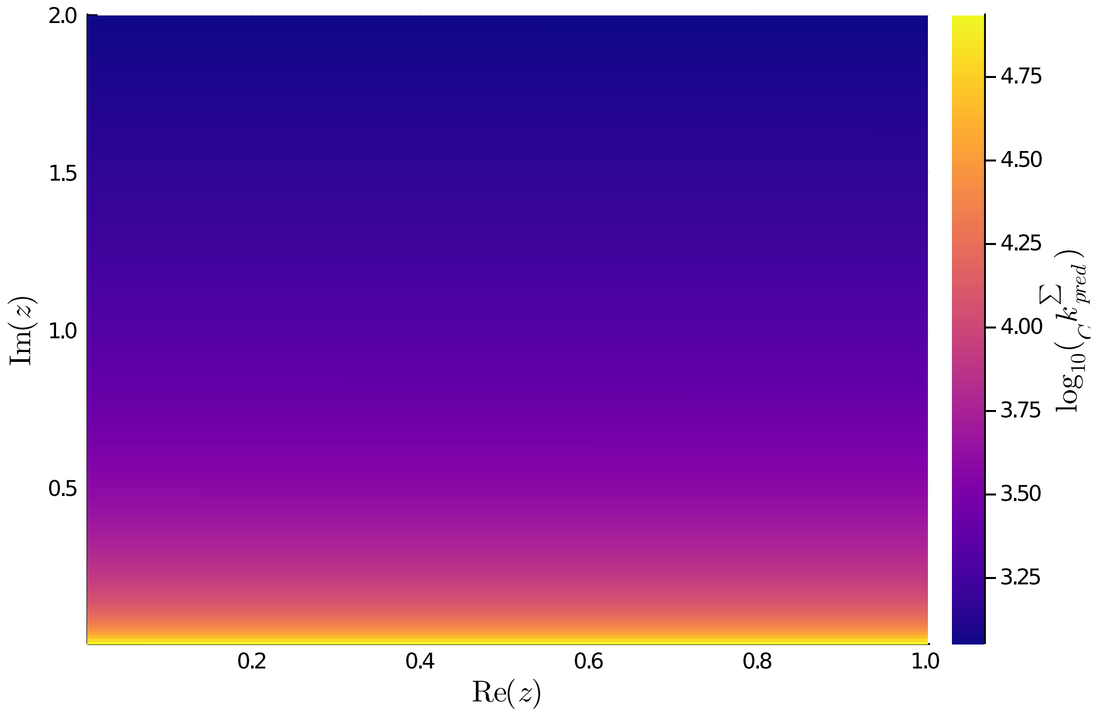


Figure 4.13.: Total CGMN iterations per RFE iteration as a function of z.

4.7.3. Optimization scheme

We are optimizing a multi-dimensional function without an analytic form for the gradient. Furthermore, our optimization space is bounded. These properties limit the choice of optimization scheme. We search for a global minimum with a fixed value of q using the derivative-free algorithm Adaptive Differential Evolution/rand/1/bin (`aderand1bin`) implemented in the BlackBoxOptim.jl library [28]. In [64], this library was shown to work well in finding global minima in a non-linear, gradient-free manner. This method allows a multidimensional search of variables, and searches for a global minimum, not just a local one. This is important, as we are not sure of the shape of our surface, especially in high dimensions and could inadvertently start at a poor choice, landing far from the global minimum. The scheme also allows for bounded variables.

If the number of processes is not predetermined, we may be interested to know what choice of q will minimize (4.21). To this end, we may search for a minimum over the valid values of q (where $q \bmod 4 = 0$). The overall algorithm is described in Algorithm 17.

Initial optimization results are more reliable for the minimization of $\sum k_{pred}^{\max}$ than $\sum k_{pred}^{\Sigma}$. This is the more relevant problem for parallel execution, due to the desire

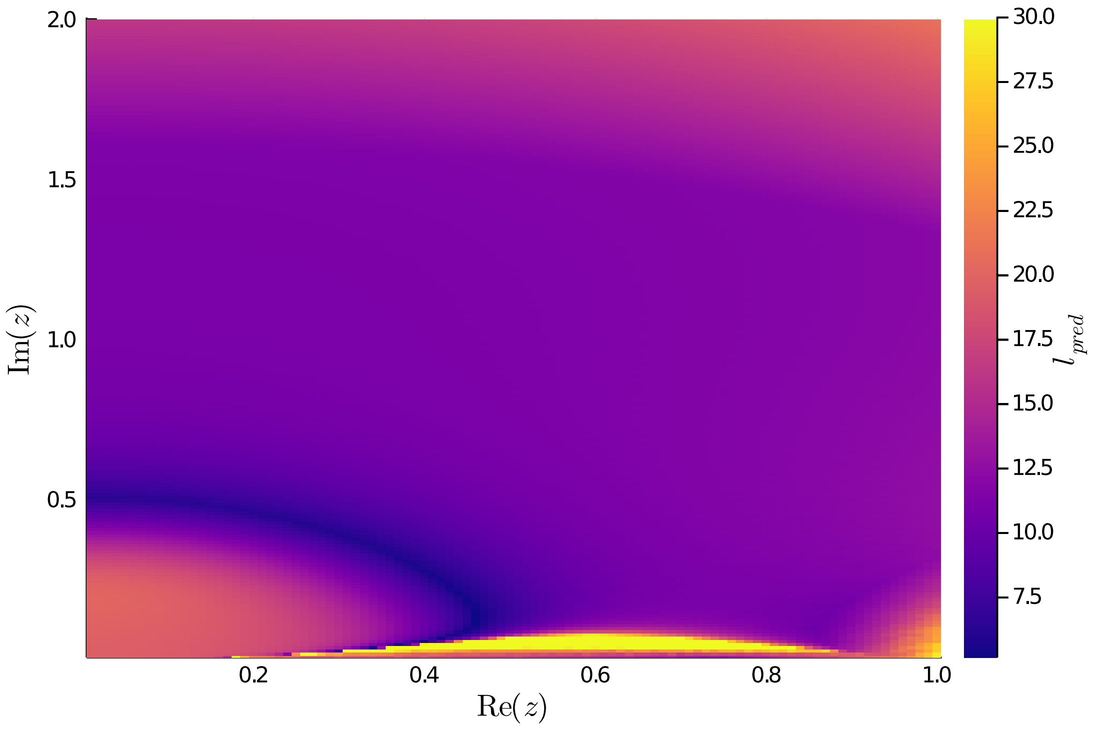


Figure 4.14.: RFE iterations as a function of z.

Algorithm 17: Optimize rational filter over degree

```
Input  : Maximal degree for rational function $q_{\max}$
         $z_{\min}, z_{\max}$, bounds for poles $z_i$
         $\alpha_{\min}, \alpha_{\max}$, $\beta_{\min}$, $\beta_{\max}$ bounds for LS-weight parameters
         $\underline{\lambda}, \overline{\lambda}$ Bounds for $I_\lambda$
         $\lambda_{\min}, \lambda_{\max}$ Estimates for global min and max eigenvalue
         $\tilde{m}$ Expected number of eigenvalues in $I_\lambda$
         $\delta$ Estimate for eigenvalues in and around $I_\lambda$
Output: Optimal degree $q$, poles $z_1, \dots, z_q$, weights $w_1, \dots, w_q$
Function optimizeoverdegree($z_{\min}, z_{\max}, \alpha_{\min}, \alpha_{\max}, \beta_{\min}, \beta_{\max}$,
 $\underline{\lambda}, \overline{\lambda}, \lambda_{min}, \lambda_{max}, \tilde{m}, \delta$):
    for $q/4 = 1, \dots, q_{\max}/4$ do
        $\left[\sum k_{pred}^{\max}, z_{opt}, w_{opt}\right]$= aderand1bin(evaluatecost(), $z_{\min}, z_{\max}$,
         $\alpha_{\min}, \alpha_{\max}, \beta_{\min}, \beta_{\max}, \underline{\lambda}, \overline{\lambda}, \lambda_{\min}, \lambda_{\max}, \tilde{m}, \delta$)
        if $\sum k_{pred}^{\max} < \min_q \sum k_{pred}^{\max}$ then
            $\min_q \sum k_{pred}^{\max} = \sum k_{pred}^{\max}$
        end
    end
```

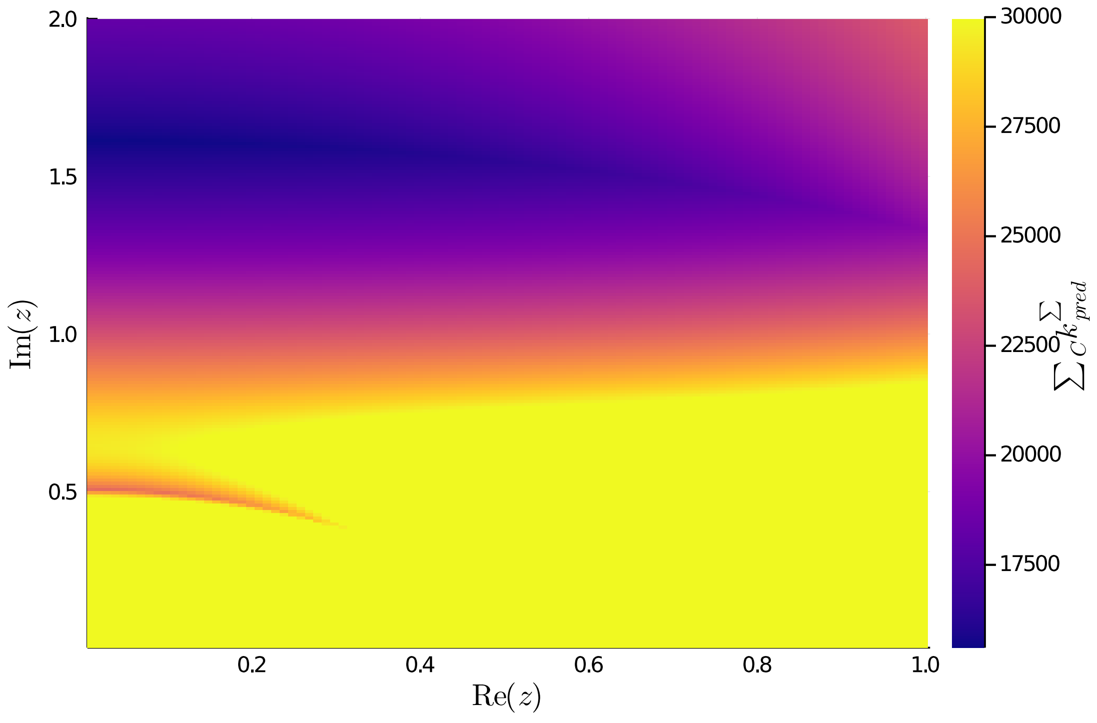


Figure 4.15.: Total CGMN iterations over all RFE iterations as a function of z.

for a balanced load over processes. Therefore we will use this as our optimization problem going forward. We note that this also provides comparatively good results in terms of the total number of overall linear solver iterations required, as we will observe in Section 4.8.

4.8. Numerical Results

We can now consider testing our optimization scheme on a selection of eigenvalue problems. We consider separately the two iterative linear solvers that were analyzed earlier in this chapter, beginning with CGMN.

4.8.1. Numerical results for CGMN

We now consider the application of our filter optimization scheme to a variety of standard eigenproblems. Three matrices come from graphene modeling [23] and 11 from the SuiteSparse Matrix Collection [25]. These test matrices have also appeared in [33, 46] and will be used again in Chapter 5. Each interval contains 40 eigenpairs. Information on I_λ is given in Table 4.3. The entire spectrum for each matrix is scaled to fit the interval $[-1, 1]$. More information for the test matrices can be found in Table A.1.

The optimized rational filter is obtained a–priori using an implementation in the Julia programming language, where the external optimization library described above may be easily utilized. The optimization scheme was time-limited to 80 seconds for each degree, after which further time did not seem to provide substantial improvements in the quality of filter. It is likely that this step could be accelerated with additional cores or parallelization.

An example filter can be seen in Figure 4.17. We see that the filter shows much weaker drop-off away from the edges of I_λ relative to the Gauss–Legendre filter of the same degree. The poles are, however, all much further from the real axis, as seen in Figure 4.16. Furthermore, the poles of the optimized filter are all a similar distance from the real axis, which should result in a more balanced number of CGMN iterations over the different poles.

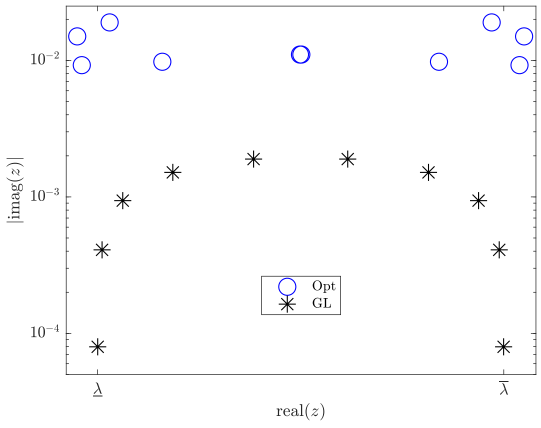


Figure 4.16.: (a)

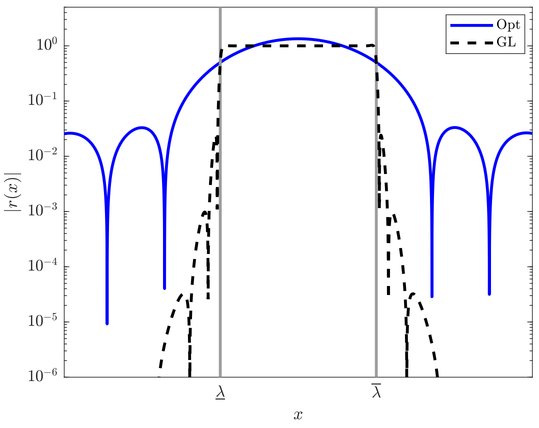


Figure 4.17.: (b)

Poles (a) and filter (b) of the optimized filter (Opt) for GraIII-11k-b problem with $q/2 = 10$. The poles and filter of the Gauss–Legendre filter (GL) for the same degree are shown for comparison.

The filters were tested within the `BEAST` framework on the Emmy cluster at Friedrich-Alexander-Universität Erlangen–Nürnberg. For each eigenproblem, we compare the Gauss-Legendre filter with our optimized filter generated for that problem from Algorithm 17. We begin with $m_U = 64$ randomly generated columns in Y. All linear systems were solved to a relative residual tolerance of 10^{-12}. Each problem was considered solved when 40 eigenpairs were found in I_λ to an absolute residual tolerance of 10^{-8}. Since all problems considered here are real, only $\frac{q}{2}$ (block) linear systems need to be solved, assuming a symmetric rational function. As discussed in Chapter 2, we solve for poles z_i in the upper half of the complex plane and use symmetry to obtain the result over the whole rational function. The results below accordingly contain the actual number of CGMN iterations required for a block linear system solve; those corresponding to $\frac{q}{2}$ poles. More specifically, the number of CGMN iterations reported for each block linear system solve is the maximal number of CGMN iterations required over the $m_U = 64$ columns in the block.

In Figures 4.18 and 4.19, we observe the actual total number of CGMN iterations required for all poles $\sum_C k^{\Sigma}$, or the actual maximum number of CGMN iterations required for a single pole, $\sum_C k^{\max}$, over all RFE iterations. The value of $\sum_C k^{\Sigma}$ and $\sum_C k^{\max}$, as given by (4.20) and (4.21), is also shown in these figures for each value. We show for each eigenproblem the Gauss-Legendre filter for degree q that minimized $\sum_C k^{\Sigma}$ or $\sum_C k^{\max}$, according to the quantity in question. This degree is listed together with the rest of the results in Table 4.3. We observe that the degree minimizing these two quantities is typically very different. The RFE solver was run with Gauss-Legendre filters up to a degree at which only one RFE iteration was required, at which point no further decrease in cost is possible. For comparison, the optimized filter gained (a-priori) from Algorithm 17 is optimized simply to minimize $\sum_C k^{\max}$; here, the value of q predicted to minimize cost is chosen in advance. The value of q is also listed in Table 4.3.

To highlight the differences in values, we also show in Figures 4.20 and 4.21 the values normalized by the number of iterations required for the (optimal degree) Gauss-Legendre filter. We observe that the value of $\sum_C k^{\max}$ is typically reduced to 1/2 or 1/3 for the problems under consideration. The value of $\sum_C k^{\Sigma}$ is also typically reduced, even though this is not the quantity under consideration for optimization. The reduction in $\sum_C k^{\max}$ is promising, as it signifies a proportionate reduction in cost when all linear systems in an RFE iteration are solved in parallel. Here, as predicted in Section 4.3, the differing distances from the real axis for most rational filters under consideration is expected to cause a significant imbalance in the number of linear solver iterations required for the different poles. As the RFE iteration requires all linear systems to be solved before continuing, an imbalance in cost for the different poles means that if all linear systems are solved in parallel, some processes will be waiting for the completion of the most expensive linear solves. Therefore, the reduced value of $\sum_C k^{\max}$ indicates improved load balancing, which is important for the overall scalability of the algorithm. This suggests that these strategies may result in RFEs that are better suited for large problems solved in high performance computing environments.

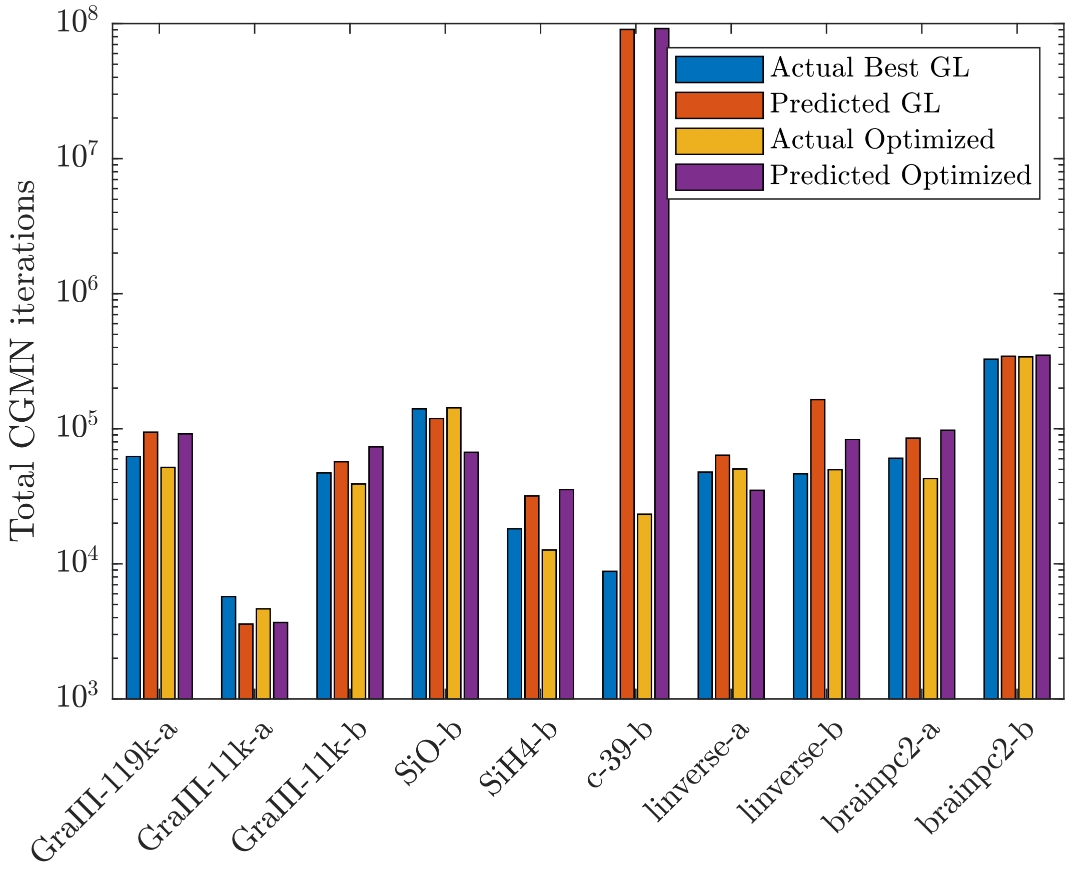


Figure 4.18.: Comparison of actual and predicted number of total CGMN iterations, $\sum_C k^{\Sigma}$ and $\sum_C k^{\Sigma}_{pred}$ for all poles required for all eigenpairs in the interval to converge. Results are shown for the Gauss-Legendre degree minimizing this value, and the optimized filter.

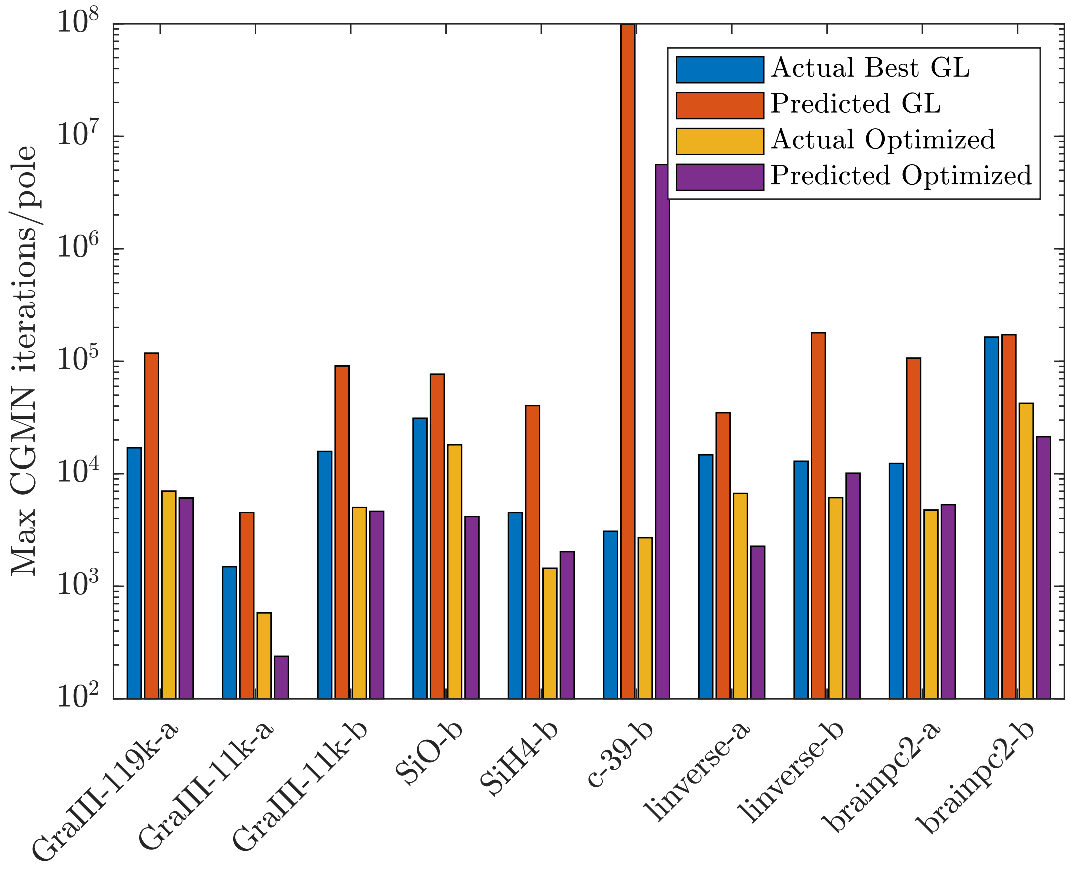


Figure 4.19.: Comparison of actual and predicted number of maximum CGMN iterations for a single pole, $\sum_C k^{\max}$ and $\sum_C k_{pred}^{\max}$ required for all eigenpairs in the interval to converge. Results are shown for the Gauss-Legendre degree minimizing this value, and the optimized filter.

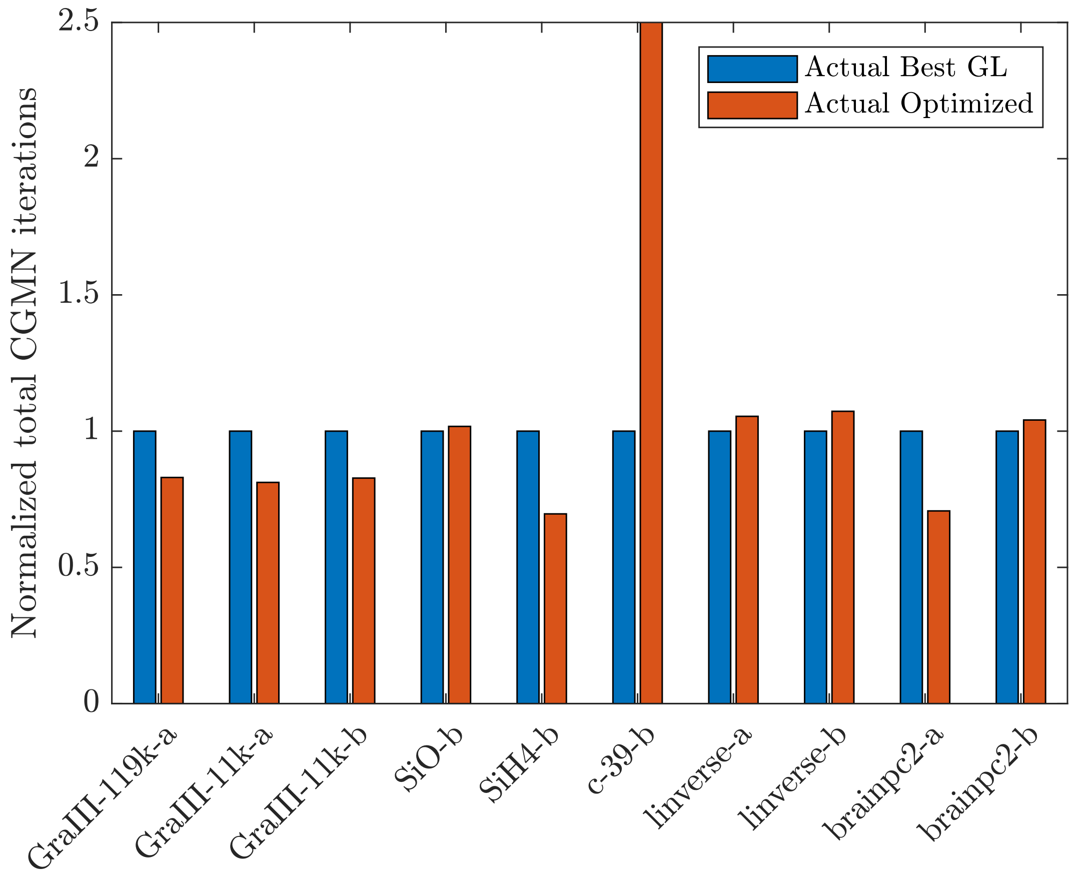


Figure 4.20.: Comparison of actual total number of CGMN iterations for all poles, $\sum_C k^{\Sigma}$ required for all eigenpairs in the interval to converge. Results for the optimized filter are shown relative to those for the Gauss-Legendre degree minimizing this value.

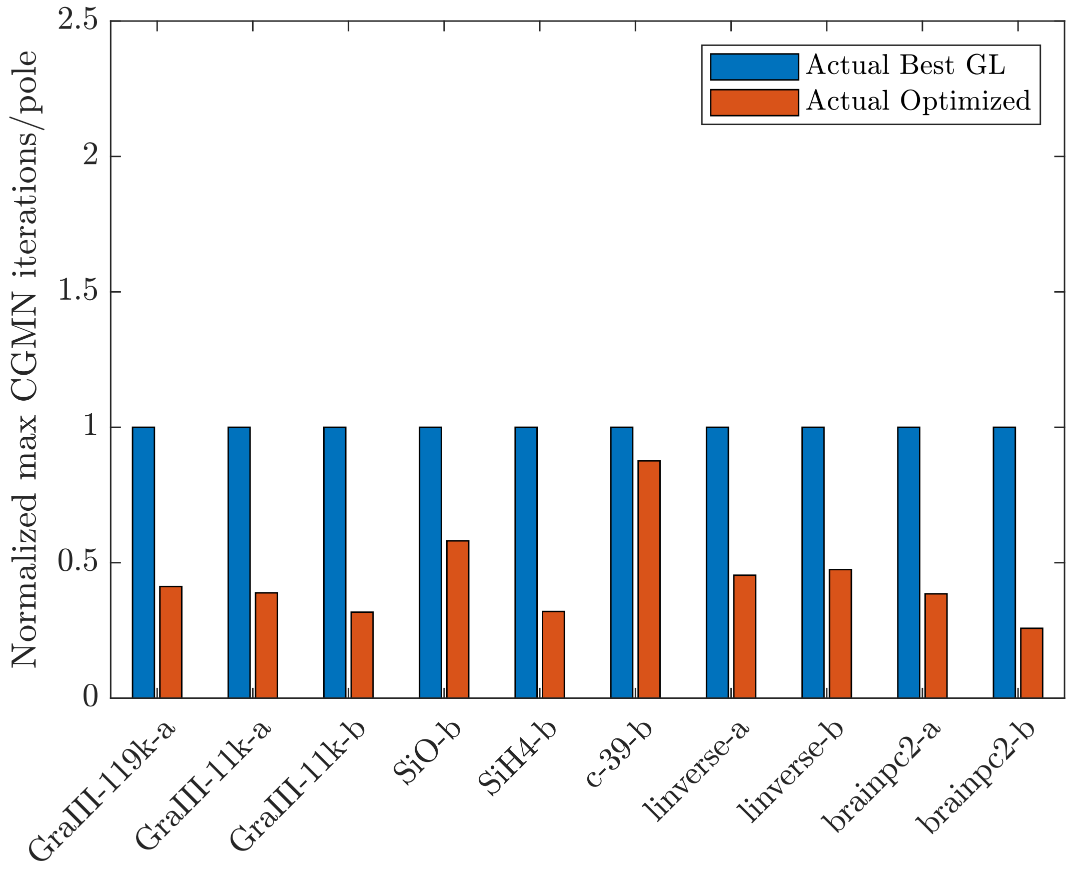


Figure 4.21.: Comparison of actual number of maximum CGMN iterations, $\sum_C k^{\max}$ for a single pole required for all eigenpairs in the interval to converge. Results are shown relative to those for the Gauss-Legendre degree minimizing this value.

		GL : $\min_q \sum_C k^{\Sigma}$			GL : $\min_q \sum_C k^{\max}$			Opt			
Problem	$\overline{\lambda} - \underline{\lambda}$	$q/2$	$\sum_C k^{\Sigma}$	l	$q/2$	$\sum_C k^{\max}$	l	$q/2$	$\sum_C k^{\Sigma}$	$\sum_C k^{\max}$	l
SiH4-a	0.019	2	24733	14	20	4568	1				
SiH4-b	0.0076	2	18180	6	16	4520	1	10	12659	1446	6
c-39-b	2×10^{-6}	2	8801	4	10	3084	1	10	23308	2702	8
linverse-a	0.0091	4	47773	6	30	14748	1	10	50371	6693	31
linverse-b	0.0014	2	46392	5	10	12924	1	10	49772	6133	6
bcsstk37-a	0.00036	2	128285	6	16	28839	1				
bcsstk37-b	0.0043	2	23863	8	20	5983	1				
brainpc2-a	0.0029	2	60498	9	18	12363	1	12	42786	4761	10
brainpc2-b	0.00058	2	327578	4	2	164142	4	10	340961	42325	7
SiO-b	0.005	4	140503	6	24	31173	1	10	142950	18104	34
GraIII-11k-a	0.085	2	5719	19	22	1492	1	10	4643	580	20
GraIII-11k-b	0.0039	2	47131	7	18	15813	1	10	39012	5019	7
GraIII-119k-a	0.0023	2	62357	5	18	17023	1	10	51754	7014	8

Table 4.3.: Comparison of CGMN and RFE iterations needed for Gauss-Legendre (GL) and optimized (Opt) rational functions. For each system, the (relative) interval length is shown along with the number of CGMN ($_C k$) and RFE iterations (l) for the degree ($q/2$; the number of quadrature points in the upper half of the complex plane) for Gauss-Legendre minimizing the total number of CGMN iterations required ($\sum_C k^{\Sigma}$) and the maximum number of iterations per pole ($\sum_C k^{\max}$). In the latter case, $\sum_C k^{\max}$ refers to the maximum number of iterations for a single pole over all RFE iterations. These are compared with the optimized filter, when it successfully found all eigenpairs (otherwise blank). The name "-a" or "-b" refers to the different intervals chosen for the matrices described in Table A.1.

In Table 4.3 we present a summary of results for all problems tested. We observe here that the optimized filter is not perfect, and sometimes fails to find all eigenpairs to the desired tolerance. This is observed to occur when the convergence of the RFE stagnates, often for only a couple eigenpairs, above the desired residual tolerance. This stagnation is not very surprising, as the optimizer is under pressure to place poles further from the real axis, where they are "cheaper," though this results in a less effective filter. Furthermore, our estimate within the optimizer for the number of RFE optimizations is based on an assumption of evenly distributed eigenvalues in and around the interval, even though this is very likely not the case. A cluster of eigenvalues around the edges of I_λ could cause significantly worse convergence than expected.

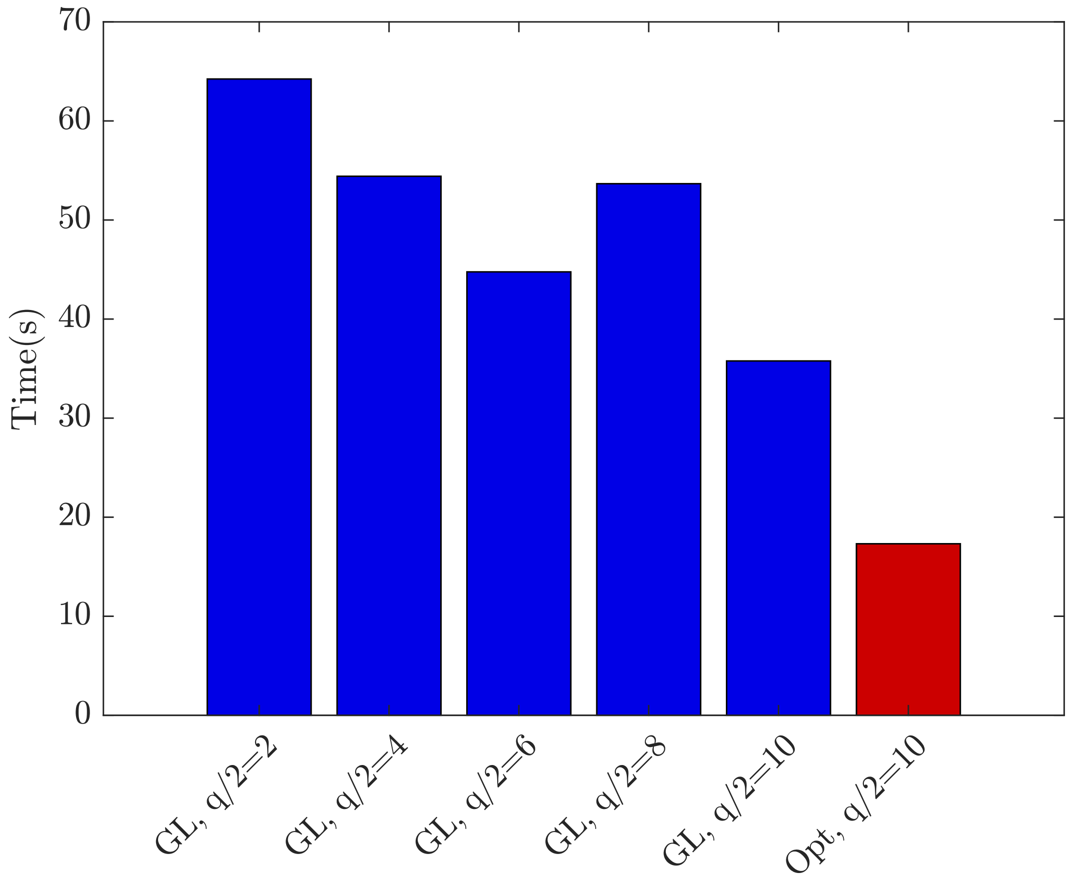


Figure 4.22.: Time required for `BEAST-C` on linverse-b problem to converge with Gauss–Legendre (GL) filter of various degrees vs. optimized (Opt.) filter. Results are shown in red for the optimized filter and blue for Gauss-Legendre filters.

We also observe how the performance of an eigensolver may depend on the performance of the linear solver with a given filter. We show in Figure 4.22 the time taken for a run of `BEAST-C` with a Gauss–Legendre filter over various degrees, up to $q/2 = 10$, for which `BEAST-C` converges in a single iteration and has minimal CGMN iterations per pole for a Gauss–Legendre filter, as well as the optimized filter, also with degree $q/2 = 10$. All linear systems were solved in parallel during each `BEAST` iteration, with one computational node for each linear system; thus $q/2$ nodes were used. Each node contains Xeon 2660v2 CPUs and a total of 20 cores. We see that the difference in CGMN iterations, as given in Table 4.3, is associated with a proportionate difference in time.

4.8.2. Numerical results for GMRES

The efficacy of the optimization scheme can also be explored with the GMRES iterative linear solver. Since the number of GMRES iterations required increases so much more quickly than for CGMN, making the experiments computationally expensive, if not infeasible, the experiments here are limited to a couple of "easier" problems.

We consider the eigenproblems SiH4-b and GraIII-11k-a, as described in Table 4.2, with general matrix information in Table A.1. For each eigenproblem, we compare our optimal filter from Algorithm 17 with the Gauss-Legendre filter over various degrees (up to a degree required to reach convergence in a single RFE iteration.) Here, the 40 eigenvalues for this problem within the interval listed in Table 4.2 are found to a tolerance of 10^{-6}, with $m_U = 64$ initially randomly generated columns in Y and all linear systems solved to an absolute residual tolerance of 10^{-12}. The average number of iterations required for all columns in the block linear system of equations to reach the desired tolerance is reported. Again, only $\frac{q}{2}$ linear systems need to be solved, assuming a symmetric rational function, and the results contain the actual (averaged over the block) number of GMRES iterations required; those corresponding to $\frac{q}{2}$ poles.

In Section 4.6.3 we observed variance across matrices in the line of best fit for estimated condition number vs. GMRES iterations. In Figure 4.23 we observe results for a filter with optimized poles generated with an estimated slope of 30 in this linear relationship (as in (4.18)) as well as an estimated slope of 20; i.e.

$$_{G}k_{pred}(\Phi) = 20\kappa_{est}(\Phi),$$

The number of actual iterations over all metrics tested for the two filters is very similar, confirming the idea that even a rough estimate for the number of linear solver iterations is useful for this optimization strategy, and can outperform a standard rational filter choice.

In Figure 4.23 and 4.24 we compare a number of metrics for the various filters tested for each eigenproblem: the total number of GMRES iterations over all iterations of the RFE, $\sum {}_{G}k^{\Sigma}$, the maximum value of GMRES iterations, ${}_{G}k$, for a single RFE iteration, ${}_{G}k^{\max}$, the number of RFE iterations, l, and the maximum number of GMRES iterations over all RFE iterations, $\sum {}_{G}k^{\max}$. For both eigenproblems, the reduction in GMRES iterations for the optimized vs. Gauss–Legendre filter is massive.

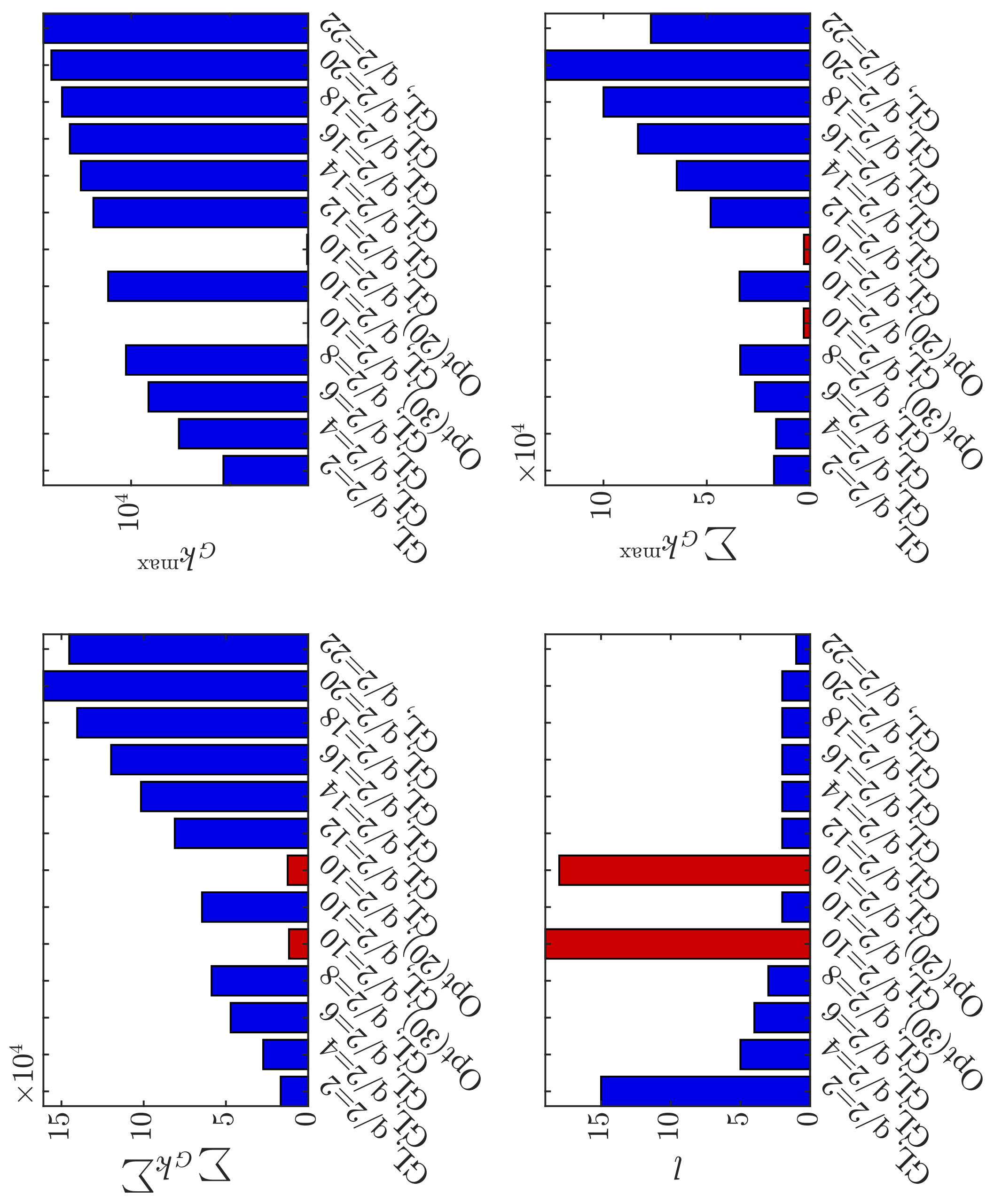


Figure 4.23.: Comparison for GraIII-11-a eigenproblem of $\sum_G k^\Sigma$, maximum $_G k$ for a single RFE iteration, l and $\sum_G k^{\max}$ required for all eigenpairs in the interval to converge. Results are shown in red for the optimized filter and blue for Gauss-Legendre filters of differing degrees.

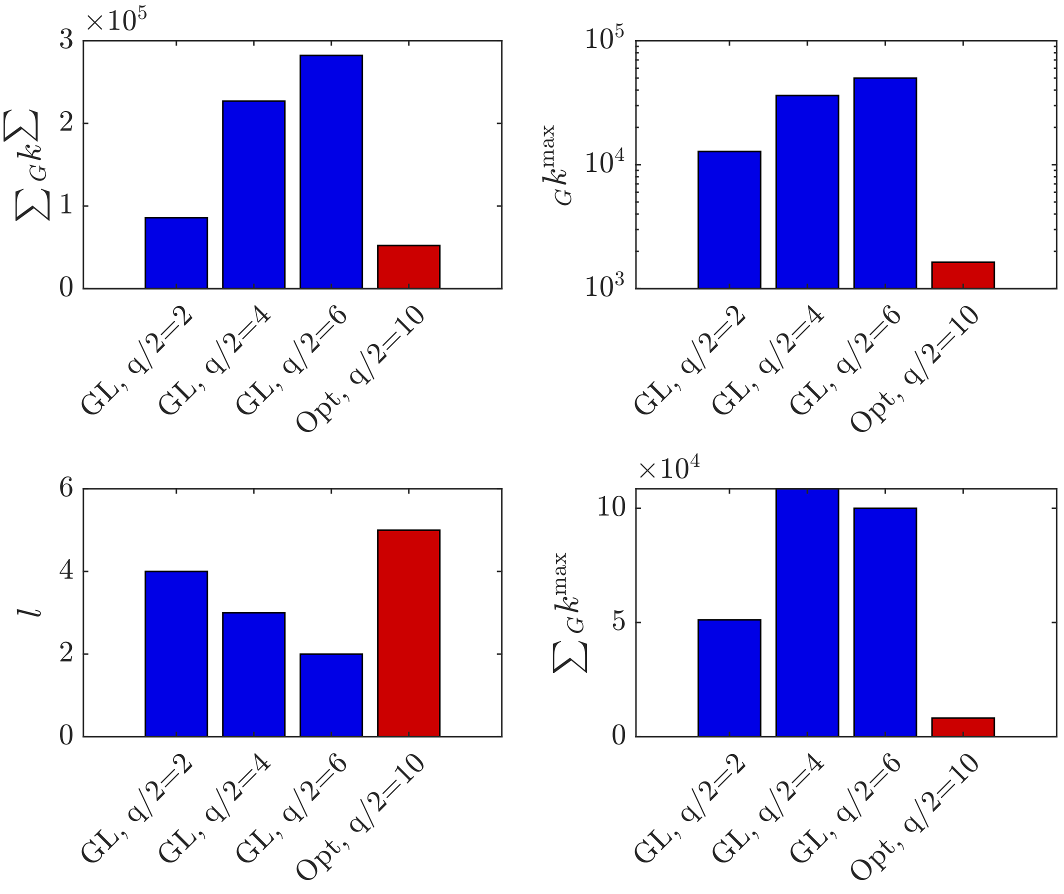


Figure 4.24.: Comparison for SiH4-b eigenproblem of $\sum_G k^{\Sigma}$, maximum $_G k$ for a single RFE iteration, l and $\sum_G k^{\max}$ required for all eigenpairs in the interval to converge. Results are shown in red for the optimized filter and blue Gauss-Legendre filters of differing degrees.

Due to the significantly steeper increase in GMRES iterations with increasing matrix condition number, the benefits of placing poles further from the real axis are even stronger than for CGMN. As in the case of CGMN, the benefits of the optimized poles also have powerful implications for load balancing, with a possible order of magnitude reduction for $\sum_G k^{\max}$, as observed for the GraIII-11k-a problem.

However, the stronger rate of growth for GMRES vs CGMN with increasing condition number makes the use of the former as a linear solver more complicated. As problem size increases, so does the relative density of eigenvalues in the spectrum, shrinking h and I_λ, and requiring the poles of a filter to lie closer to the real axis for convergence of the RFE. This in turn drives up the relative condition number, resulting in a greater increase in iterations for GMRES than for CGMN. Again, is possible that an appropriate preconditioner could change the iteration growth rate, and make GMRES more competitive as a choice of iterative solver. This, as well as testing other iterative linear solvers, remains a topic of future research.

4.8.3. Comparing to other optimized filters

The reason for comparison with Gauss-Legendre quadrature becomes clear after comparison with some of the “optimal” schemes. Though Zolotarev and SLiSe/-WiSe functions may provide fast convergence, the cost of the linear systems with poles close to the real axis is unlikely to justify their use.

We illustrate this for one problem, as shown in Figure 4.25. Here, the 40 eigenvalues for the matrix bcsstk37-a (as described in Table A.1) are solved with either a Zolotarev or Gauss-Legendre filter of differing degrees. The RFE began with $m_U = 64$ randomly generated columns of Y and solved all linear systems to a relative residual tolerance of 10^{-12}. We consider the maximum number of CGMN iterations required for a column in the block linear system to reach the desired tolerance. All 40 eigenpairs were found to a residual tolerance of 10^{-8}. The Zolotarev weights and poles were generated using a value of $R = 10^6$, as defined in (4.5) except for one comparison point with $R = 10^4$.

We observe that the total number of CGMN iterations required is much higher when using Zolotarev functions. The maximum number of iterations per pole is roughly equivalent when only one iteration is needed, but this requires a much higher degree for Zolotarev functions than for Gauss-Legendre. Better behaviour might be expected with a differing value of R. A larger value is associated with a better worst case convergence rate, as the filter improves for eigenvalues lying close to the boundaries of I_λ. However, the average case may not be improved, as we see in Figure 4.25 for the Zolotarev filter with $q/2 = 4$. Here, a decreased value of $R = 10^4$ improves the RFE convergence rate (not always to be expected) while also decreasing the overall and per RFE iteration cost of CGMN. In general, we expect that the average case behaviour of Gauss-Legendre will continue to outperform Zolotarev functions when considering iterative solvers like CGMN.

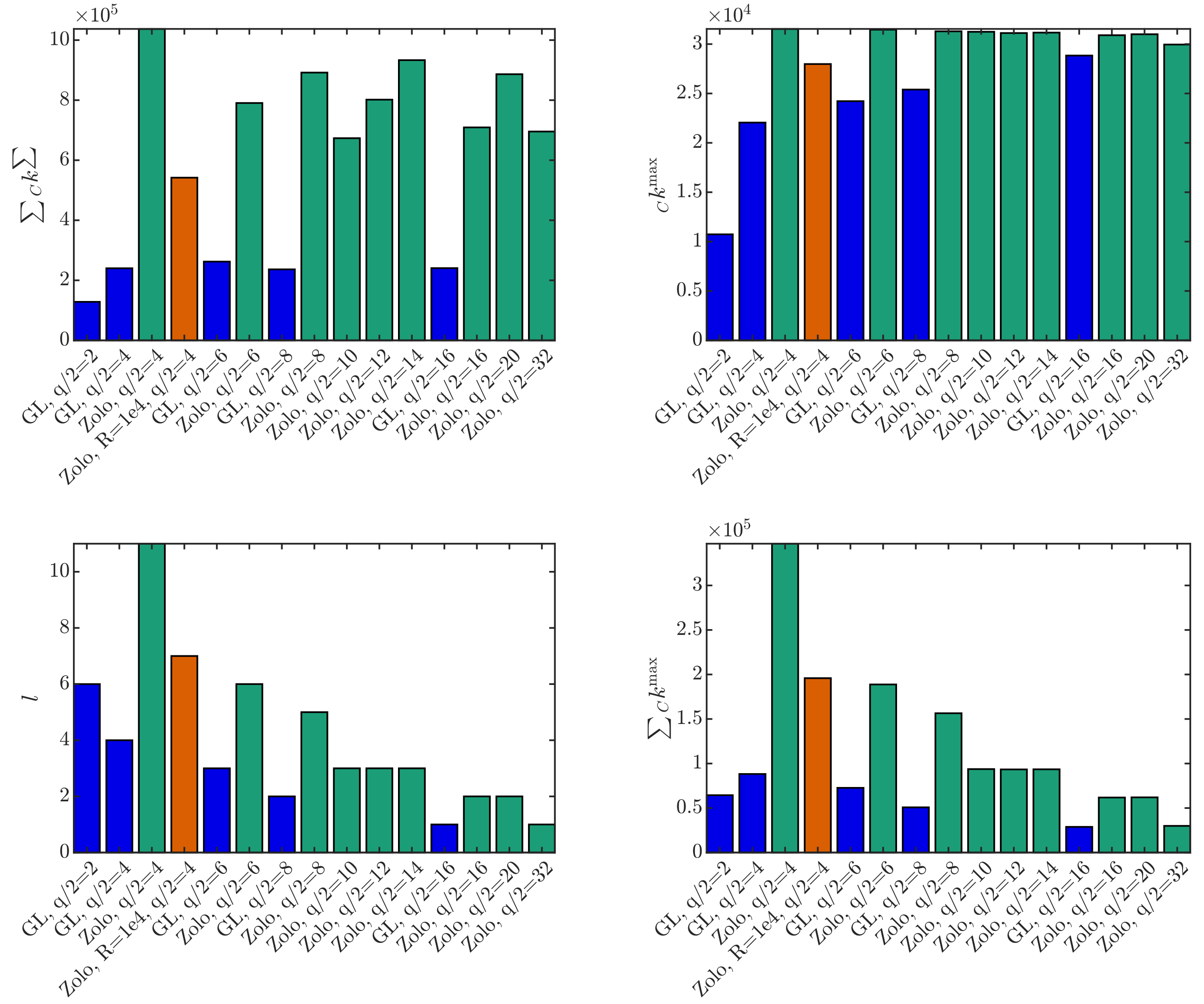


Figure 4.25.: Comparison of maximum CGMN iterations for a single pole, $\sum_C k^{\max}$ and $\sum_C k^{\max}_{pred}$ required for all eigenpairs in the interval to converge. Results are shown in blue for Gauss-Legendre and green Zolotarev filters with value $R = 10^6$ of differing degrees. We also show results in orange for one Zolotarev filter with $R = 10^4$, $q/2 = 4$.

4.9. Conclusions

In this chapter, we have considered the relationship between the rational filter in an RFE and the resulting shifted linear systems required for the construction of the basis of the desired eigenspace. Here, we have explicitly focused on the use of iterative solvers for these linear systems, and explored whether the number of linear solver iterations required may be predicted. We developed a simple optimization problem, based on the predicted cost of solving all shifted linear systems over all iterations. Previous work in optimizing filters for Hermitian eigenvalue problems has focused on reducing the worst–case convergence ratio of the RFE [42, 64, 96].

The optimization problem is simple to describe, but its solution is not straightforward, as it is a bounded non-linear problem without a known gradient. Future work could include a modification of the cost function such that a wider range of optimization schemes are applicable, and perhaps improve the quality of filter that arises. The improvement in linear system solver cost that we have seen so far is substantial, but for the solution of very large problems, a further gain may be needed in order to make the solution using an iterative linear solver truly feasible. This could also include expanding the prediction of linear solver iterations to other iterative solvers, perhaps with appropriate preconditioners. A block incomplete Cholesky approach [62] has previously been used for the linear systems arising from an RFE [4]. Furthermore, our filter optimization strategy could be combined with the graduated residual strategies seen in [38, 79], increasing the residual tolerance in early RFE iterations and thus presumably reducing the overall number of linear solver iterations required.

One promising possibility for future research is to re-run the optimizer and generate a new filter after each RFE iteration with an updated estimate of the location of the eigenvalues, information that is "free." The computational cost of running the optimizer is vastly outweighed by a single RFE iteration, especially with growing problem size, as it runs in minutes on a workstation with a few cores, and presumably seconds on a single node in a high performance computing environment. Whether the optimization procedure takes place a-priori or after each RFE iteration, the increase in cost for the RFE should be basically insignificant, and vastly outweighed by the presumed reduction in cost for the linear system solves.

An alternative strategy is to compute a set of filters a-priori with the properties seen thus far. Up to this point, we have used information about the width of the spectrum in the optimization of the filter. However, it is possible that a table of filters for a range of expected spectral widths could be gathered in advance and provided within `BEAST` for reasonably good results without requiring the user to perform the optimization step. This would first require evaluating how well these filters generalize to similar problems.

The practical use of iterative linear solvers in an RFE is a long-standing and significant problem. Though the approach we have shown in this chapter is a simple one, and further work is likely needed to improve the feasibility of the approach for truly large problems, the direction seems promising.

CHAPTER 5

RFES WITH MULTIPLE MOMENTS

In Chapter 2, rational filter-based eigensolvers were introduced, including the definition of RFEs with multiple higher order moments. In this chapter, we discuss the various algorithmic choices that arise in these methods, as well as the subsequent consequences. We are interested in how these choices affect the accuracy, robustness, and scalability of the overall scheme particular to a RFE with multiple moments. As with all RFEs, a significant consideration is the solution of the multiple linear systems that arise. One advantage immediately apparent from the use of multiple moments is the ability to re-use the solution of linear systems in the construction of U, a basis for a subspace containing the desired eigenvectors. As the solution of linear systems is by far the most expensive component of these schemes, this is a promising prospect for the overall cost reduction of our scheme. There are two possible viewpoints, given the re-use of the solution of linear systems: either to construct a larger subspace overall, hopefully obtaining a better approximate basis for space spanned by the desired eigenvectors, or to use a constrained subspace size, and reduce the cost per RFE iteration. Sakurai–Sugiura methods have traditionally focused on the former strategy [59], but in this thesis, the latter perspective will be extensively explored for potential cost reduction. As we will see in this chapter, a constrained subspace size has implications for convergence. We explore the expected heuristic behaviour with respect to various parameters and algorithmic choices, and propose multiple schemes to reduce cost while maintaining accuracy and robustness. Numerical results showing the potential efficiency of these various strategies are shown. This chapter is built upon work that is available elsewhere. In [46], we consider RFEs with a flexible number of moments, including discussions and comparisons of iteration types, strategies for changing the number of moments over iterations, the combined effect of subspace size and quadrature rules, and numerical experiments showing the reduction in cost for linear systems possible with a flexible choice of moments. In [4], we explore the scalability of multi-moment RFEs with respect to quadrature degree. Here, we take the opportunity to expand on several important topics included in the implementation of a multi-moment RFE, including the estimation of the number of eigenvalues inside the interval and orthogonalization of the constructed basis for the desired eigenspace. We also explore choosing and predicting quadrature degree to decrease the cost of solving linear systems over an RFE.

5.1. Algorithmic overview

In Chapter 2, we introduced RFEs with multiple moments. We have considered the construction of U, the basis for the desired eigenspace, from the perspective of a contour integral as well as a general rational filter. We have also highlighted how the approximate eigenpairs may be extracted from U. We will now introduce additional algorithmic considerations, such as RFE iterations. We highlight the general flow of an RFE with multiple moments in Figure 5.1. Corresponding sections discussing relevant choices in detail are listed. The algorithmic flow is similar to a general RFE, as described in Chapter 2.

5.1.1. Extraction of eigenvalues and eigenvectors

Sakurai–Sugiura methods were first introduced with the extraction of the desired eigenvalues and eigenvectors from the generated subspace basis U via a Hankel matrix composed with the distinct moments [85]. This is referred to as the *block SS–Hankel method.*

Alternatively, and as is now common practice, the eigenpairs can be extracted via a Rayleigh–Ritz technique, as described in Chapter 2. This method has the advantage of preserving numerical stability if some eigenvalues are close together, in comparison with methods using Hankel matrices [87]. This practice is called the *block SS–RR* technique [56, 87]. For this approach, it is necessary that U be of full numerical rank in order to provide a non-singular eigenproblem. Therefore, the pre-processing of U via orthogonalization and removal of rank deficient columns is imperative. Strategies for orthogonalization will be discussed later in this chapter. The remainder of this chapter will assume approximate eigenpairs are extracted via the Rayleigh–Ritz procedure.

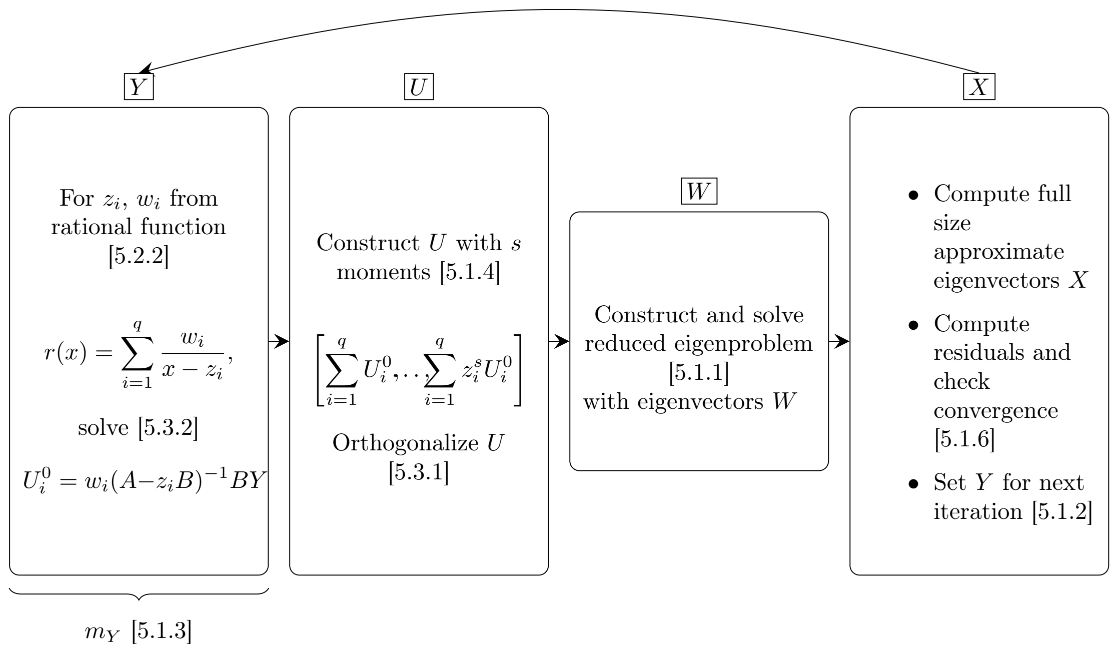

$\boxed{Y}$
$\boxed{U}$
$\boxed{X}$
$\boxed{W}$
For z_i, w_i from rational function [5.2.2]
$r(x) = \sum_{i=1}^{q} \frac{w_i}{x - z_i}$,
solve [5.3.2]
$U_i^0 = w_i(A - z_iB)^{-1}BY$
Construct U with s moments [5.1.4]
$\left[\sum_{i=1}^{q} U_i^0, .. \sum_{i=1}^{q} z_i^s U_i^0\right]$
Orthogonalize U [5.3.1]
Construct and solve reduced eigenproblem [5.1.1] with eigenvectors W
• Compute full size approximate eigenvectors X
• Compute residuals and check convergence [5.1.6]
• Set Y for next iteration [5.1.2]
m_Y [5.1.3]

Figure 5.1.: Algorithmic choices for multi-moment RFEs. The respective (sub)sections detailing the different steps and choices are shown in brackets.

5.1.2. Subspace iteration

As shown in Chapter 2, a multi-moment RFE involves constructing a subspace U, as defined in (2.25), formed by collecting the vectors from the application of different moment-based filters to Y. When s moments are used, this may again be defined as

$$U = \left[U_0, U_1, \ldots, U_{s-1}.\right] \tag{5.1}$$

The portion of U corresponding to the p^{th} subspace is again

$$U_p = \sum_{i=1}^{q} w_i z_i^p \Phi_i^{-1} BY \tag{5.2}$$

where $\Phi_i = (A - z_iB)$. We note that in practice, z_i^p should be replaced by a factor ζ_i^p which is not scaled by I_λ in order to improve numerical stability [84]. For ease of notational understanding, this is omitted in our description.

If this subspace does not allow us to extract accurate approximations of the desired eigenvectors from its basis, we can repeat the application of our filter(s) in a subspace-iterative based manner. In a single moment scheme (i.e. FEAST) we use our approximate eigenvectors $\tilde{X}_{I_\lambda}$ as the initial vectors Y in the following iteration

[77]. When multiple moments are considered, we only need $1/s$ of the number of vectors in $\tilde{X}_{I_\lambda}$ (assuming a full rank U) to form Y for the next RFE iteration. How should we select our new vectors for Y? We discuss two options below.

5.1.2.1. Outer iteration

In the so-called "outer iteration", the m_Y initial vectors needed in the following iteration are chosen as a linear combination of the current approximate eigenvectors, $\tilde{X}_{I_\lambda}$, as only $m_Y = \frac{m_U}{s}$ are needed in Y. This strategy was introduced for SSM in [84]. Here, the linear combination is traditionally random, via multiplication with a random matrix with values uniformly chosen in $[0, 1]$ as

$$Y = \tilde{X}_{I_\lambda} R \tag{5.3}$$

where $\tilde{X}_{I_\lambda} \in \mathbb{C}^{n \times m_U}$, $R \in \mathbb{R}^{m_U \times m_Y}$, and thus $Y \in \mathbb{C}^{n \times m_Y}$. This random combination ensures that directions according to all approximated eigenvectors are included in the new approximate subspace, where they will again be filtered. However, as observed in [30], the choice of random values in R perturbs the vectors and may lead to more "noise" in the filtered vectors of the next subspace. The evaluation of different choices of R is beyond the scope of this work; some ideas are illuminated in [30]. The linear combination of vectors also makes the analysis of the error more difficult, as we will discuss next.

Algorithm 18: Block SS-RR with outer iteration

```
Choose a starting block of $m_Y$ vectors $Y$
while not converged do
    Compute $U = \left[\sum_{i=1}^{q} w_i \Phi_i^{-1} BY, \ \ldots, \ \sum_{i=1}^{q} w_i z_i^s \Phi_i^{-1} BY\right]$
    Orthogonalize $U$
    Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
    Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
    Compute approximate eigenvectors $X = UW$
    Set $Y = XR$
end
```

5.1.2.2. Inner iteration

In the so-called "inner iteration" [58], the choice of Y in subsequent subspace iterations of the RFE is U^0, the subspace formed by applying the 0th moment filter $r(B^{-1}A)$ to Y,

$$U^0 = \sum_{i=1}^{q} w_i \Phi_i^{-1} BY.$$

The resulting algorithm, as shown as Algorithm 19, does not require the approximate eigenpairs for subsequent RFE iterations. If convergence to the desired tolerance is not yet expected, the extraction of the approximate eigenpairs may be skipped. This direct connection between the solution of the linear systems in one RFE iteration and the right hand sides BY in the next may be contrasted with the other iteration type we have seen so far. This is also reflected in the respective names of "inner" and "outer" iteration types.

Algorithm 19: Block SS-RR with inner iteration

```
Choose a starting block of $m$ vectors $Y$;
while not converged do
    Compute $U = \left[U^0,\ \sum_{i=1}^{q} w_i z_i \Phi_i^{-1} BY,\ \ldots,\ \sum_{i=1}^{q} w_i z_i^s \Phi_i^{-1} BY\right]$
    if checking convergence then
        Orthogonalize $U$
        Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
        Solve reduced eigenproblem for $(W, \Lambda)$ $A_U W = B_U W \Lambda$
        Compute approximate eigenvectors $X = UW$
        Compute residuals, $Ax_i - \lambda_i Bx_i$, $[\lambda_1, \ldots \lambda_{m_U}] = \mathrm{diag}(\Lambda)$, $\lambda \in I_\lambda$
    end
    Set $Y = U^0$
end
```

As shown in Chapter 2, $U^0 = Xr(\Lambda)X^H BY$. We provide here illuminating highlights of the analysis in [58] to obtain the error bound for multiple inner iterations, which is related to the error bound for single moment RFE iterations [89]. In the second inner RFE iteration, the subspace U_2^0 is formed as:

$$U_2^0 = (Xr(\Lambda)X^H B)U_1^0$$
$$U_2^0 = (Xr(\Lambda)^2 X^H B)Y$$

and subsequently the t^{th} single moment RFE iteration is equivalent to applying the approximate projector $(Xr(\Lambda)X^H B)$ t times. The subspace U_t^0 is thus defined as

$$U_t^0 = (Xr(\Lambda)X^H B)^t Y. \tag{5.4}$$

As shown in [58] and restated in Chapter 2, a single iteration of a multi-moment RFE is equivalent to applying the single moment filter to the s^{th} block-Krylov subspace based on the initial vectors:

$$U = Xr(\Lambda)X^H BK,$$

where

$$K = \left[Y, X\Lambda X^H BY, X\Lambda^2 X^H BY, \ldots, X\Lambda^{s-1} X^H BY\right].$$

Substituting U_t^0 for Y in the expression for K, we obtain in the t^{th} iteration the subspace K_t:

$$K_t = \left[(Xr(\Lambda)X^H B)^t Y, \ldots, X\Lambda^{s-1} X^H B (Xr(\Lambda)X^H B)^{t-1} Y\right] \tag{5.5}$$

$$K_t = (Xr(\Lambda)X^H B)^t K. \tag{5.6}$$

Thus, in the t^{th} inner iteration,

$$U = (Xr(\Lambda)X^H B)^t K. \tag{5.7}$$

This is equivalent to projected subspace iteration, now applied to a block Krylov space of initial vectors. Since $Xr(\Lambda)X^H B = Xr(\Lambda)X^{-1}$ is by definition a diagonalizable matrix, the error theorem from [81], Theorem 5.2, holds when the eigenvalues are extracted via Rayleigh–Ritz (i.e. Algorithm 19). We assume that the set of vectors resulting from the application of the spectral projector with invariant subspace span$\{x_1, x_2, \ldots, x_{m_U}\}$ onto Y are of full rank. Thus we obtain the following expression for the error after t iterations:

$$\|(I - P^t)x_i\|_2 \le c \left|\frac{r(\lambda_{m_U+1})}{r(\lambda_i)}\right|^t \quad i = 1, 2, \ldots, m_U, \tag{5.8}$$

where P^t is the orthogonal projector onto the space spanned by the columns of U.

This is equivalent, in the single moment case, to the convergence behaviour of the FEAST algorithm over multiple iterations. Since the outer iteration uses a linear combination of the extracted vectors $\tilde{X}_{I_\lambda}$ to form Y in the next iteration, this error analysis is only appropriate for a single iteration of Algorithm 18. Although no theoretical error analysis for the outer iteration is given, we can observe the similarities in convergence between the two methods in subsequent experiments.

5.1.3. Subspace size

The choice of subspace size, m_U, for an RFE with multiple moments is a critical determinant of convergence, as it is for RFEs in general. As has previously been shown for a single moment method [65, 66], m_U should be at least as large as the number of eigenvalues in I_λ, m. If this condition is not met, convergence will either stagnate, if the filter is a close approximate of the window function, or proceed slowly for some eigendirections, while failing to find all eigenpairs, as described in [30]. This is of course also true in the case of multiple moments, as without enough directions, the space spanned by U has too few dimensions to contain the complete eigenspace spanned by X_{I_λ}.

The construction of the subspace basis U is defined in (2.25). By definition, $m_U = s \times m_Y$. Therefore, in the linear systems defined in an RFE, only $\frac{1}{s}$ as many right hand sides are needed as for an iteration with a single moment RFE. How should m_U be chosen? Historically, a factor of enlargement, or *subspace factor* of $1.5 \times \tilde{m}$ has been recommended for the FEAST algorithm, with $\tilde{m}$ the expected number of eigenvalues in I_λ. An analytical argument for this choice was given in [89], with initial explanations in [32].

SSM, with multiple moments, has instead traditionally relied on a subspace size large enough to converge in a single iteration. Indeed, vectors are traditionally added to U until the smallest singular values are below a given threshold, a sign that the space is large enough to contain an accurate basis for the desired eigenvectors [84]. This strategy may make convergence to a desired residual tolerance more difficult, as the “break point” of the algorithm is chosen by numerical rank and not according to a residual threshold.

Currently, our focus is on reducing the cost of linear system solves by reducing the overall number of right hand sides summed over all RFE iterations, $\mathrm{RHS}_{\mathrm{ovl}}$. To this end, we constrain m_U as in the FEAST algorithm to approximately $1.5 \times \tilde{m}$. This is expected to have consequences for the number of RFE iterations required. Numerical experiments are considered in Section 5.1.5.

5.1.4. Multiple moments

As seen above, a subspace created with s moments is equivalent to a single moment rational filter applied to a set of initial vectors expanded by an order-s block-Krylov subspace. An alternative perspective is to consider the action of the filter belonging to each moment separately, with form for the p^{th} moment

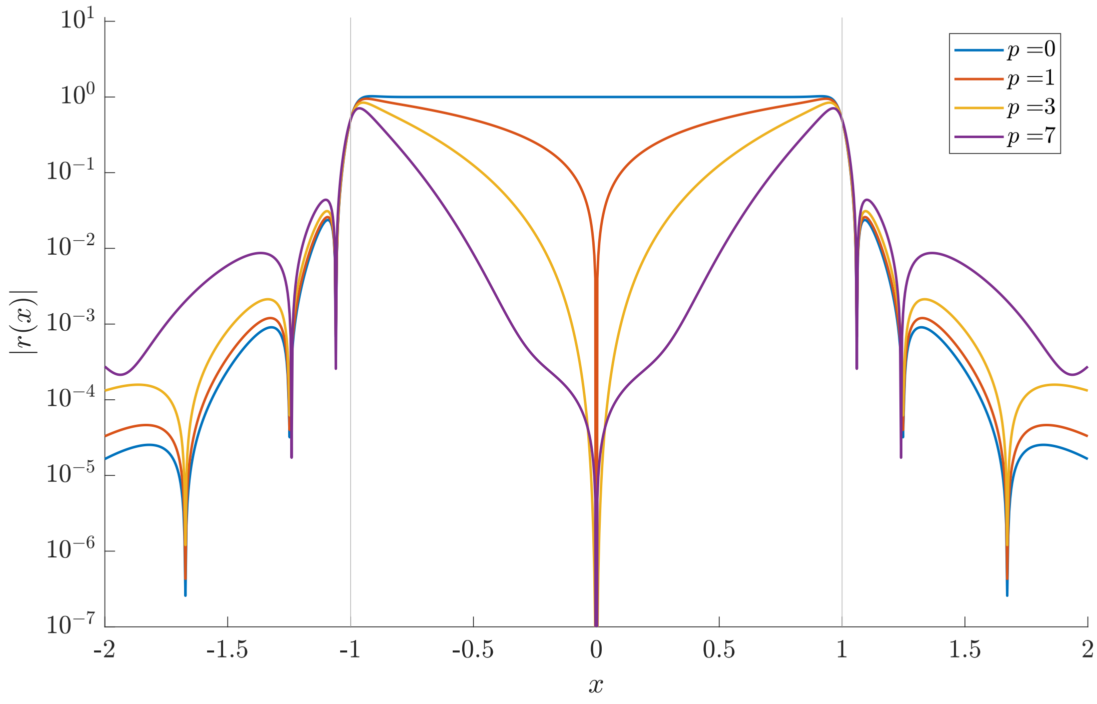


Figure 5.2.: Visualization of filter functions (5.9) for various moments p.

$$r^p(x) = \sum_{i=1}^{q} \frac{w_i z_i^p}{z_i - x}. \tag{5.9}$$

We can also anticipate the effect of a multi-moment RFE by observing the distinct filter functions for various moments. In Figure 5.2, we see the filter function resulting from various distinct moments. Here, we see that the filter function returns to 0 inside I_λ for larger portions of the interval with increasing p. Realistically, we must expect the projected subspace corresponding to each moment to decrease in quality as p increases. How exactly this effect applies to the overall generated subspace is not analytically known, however, as the overall filter function with multiple moments does not have a known form.

How many moments is a "good" choice for a subspace? A larger number of moments leads to a larger overall subspace size, or if this is constrained, a smaller value of m_Y needed in an RFE iteration. However, the behaviour and analysis with increasing numbers of moments encourages careful choice. As shown in [84], eigendirections in the space spanned by the columns of U with eigenvalues outside of I_λ are reduced by a factor of their scaled distance from the interval, raised to the power of $-q+p$. The number of moments is recommended to be $q/4$, to keep this factor large enough to speed up convergence.

When we consider the error analysis previously shown, we observe that at least for the inner iteration, the error is independent from the number of moments. The same initial starting vectors Y are treated by similar filter functions; as in the generation of a traditional Krylov space, it is to be expected that the vectors of this space approach linear dependency. The achievable residual error may also be limited when multiple moments are used, as we observe in Section 5.1.5. The error given by (5.8) depends on m_U. Even if the initial choice of m_U is sufficiently large, a reduction due to low rank will also affect the subsequent expected error reduction.

We can avoid stagnation of the residual above our desired tolerance by decreasing the number of moments in later iterations. In a similar way to the adaptive strategy described in [34], we can choose the number of moments adaptively. We can monitor the value of the "smallest-not-converged" residual pair- that is, the smallest value of $A\tilde{x}_i - B\tilde{\lambda}_i\tilde{x}_i$, $i \in 1, \ldots, m_U$ that lies above the chosen residual threshold. If, in repeated iterations, this value does not shrink by a factor of 100, stagnation is determined to have set in, and the number of moments is decreased, typically with a reduction to a single moment scheme.

5.1.5. Numerical experiments

We will begin by exploring the expected behaviour of SSM with regards to the choice of number of moments and choice of initial vectors. This experiment has been considered and discussed in [46].

We consider a small standard eigenproblem

$$A\mathbf{x}_i = \lambda_i \mathbf{x}_i, \quad \lambda_i \in I_\lambda = [-1, 1] \tag{5.10}$$

with the diagonal matrix

$$A = \operatorname{diag}(-2.99, -2.89, ..., 6.91) \in \mathbb{R}^{100\times 100} \tag{5.11}$$

A has $m = 20$ eigenvalues in I_λ.

We use the contour-integration based filter of Gauss–Legendre type with $q = 16$ quadrature points on the unit circle surrounding $I_\lambda = [-1, 1]$. In the first iteration, the m_Y columns of $Y \in \mathbb{C}^{n\times m_Y}$ were chosen randomly. In subsequent iterations, $Y = U^0$ or $Y = \tilde{X}_{I_\lambda} R$ depending on whether the inner or outer iteration type (see Section 5.1.2) was currently in use. The subspace size was constrained to $m_U = 32$, and thus $m_Y = 4, 8$, or 32, depending on whether s was currently chosen as 8, 4, or 1.

We wish to observe the behaviour induced by switching from inner to outer iterations, as well as from changing the number of moments in use over RFE iterations.

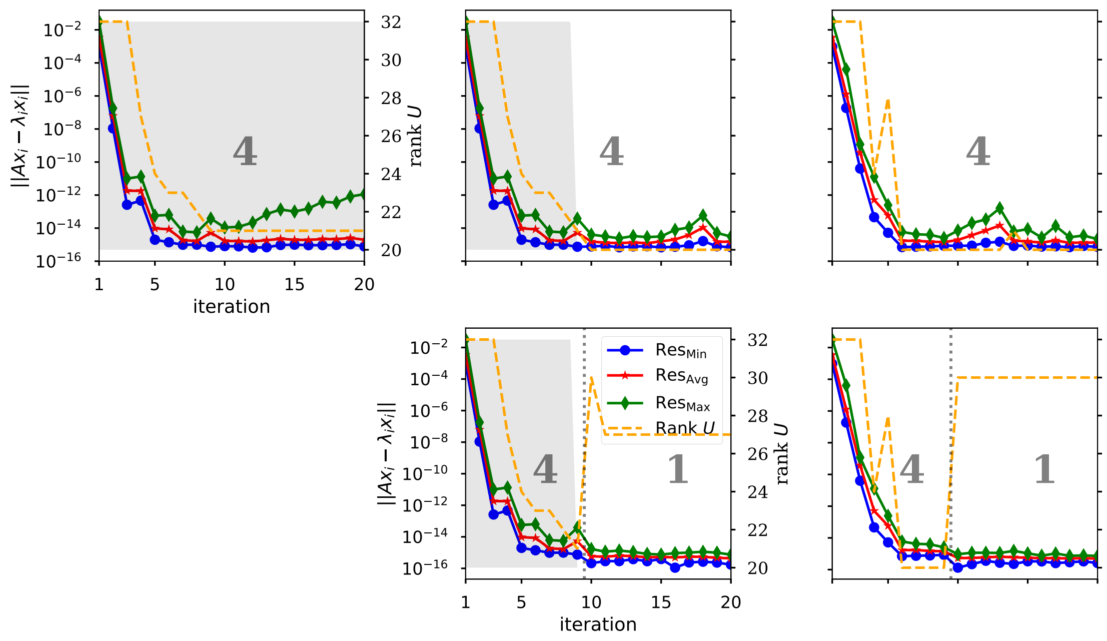


Figure 5.3.: Absolute maximum, minimum, and average of the 20 smallest residuals for problem (5.10), as well as rank. m_U is 32. The number of moments is written in the background, with a vertical line indicating a change in number of moments. A grey background indicates inner iteration types (see Algorithm 19); white outer iteration types (Algorithm 18). Left: Inner iteration type for all RFE iterations. Center: Inner iteration type for 8 RFE iterations, then switch to outer iteration type. Right: Outer iteration type for all RFE iterations. Top row: 4 moments for all RFE iterations. Bottom row: 4 moments for first 9 RFE iterations, then 1 moment for subsequent iterations. This figure also appears in [46].

The values of the smallest $m = 20$ residuals are shown for various configurations of moments and iteration types, for an initial 4 moments in Figure 5.3 and 8 moments in Figure 5.4. Here we observe that using more moments, particularly in conjunction with outer iterations, leads to stagnation of the residual error at a given threshold. This effect, and the height of the stagnation threshold, may be even stronger for larger, non-diagonal matrices. Furthermore, the numerical rank of U when multiple moments are in use is reduced, restricting the theoretical error bound for that iteration. These effects may be combated by switching to a single moment RFE in later iterations. If this switch occurs, we observe that convergence continues to the maximum expected quality.

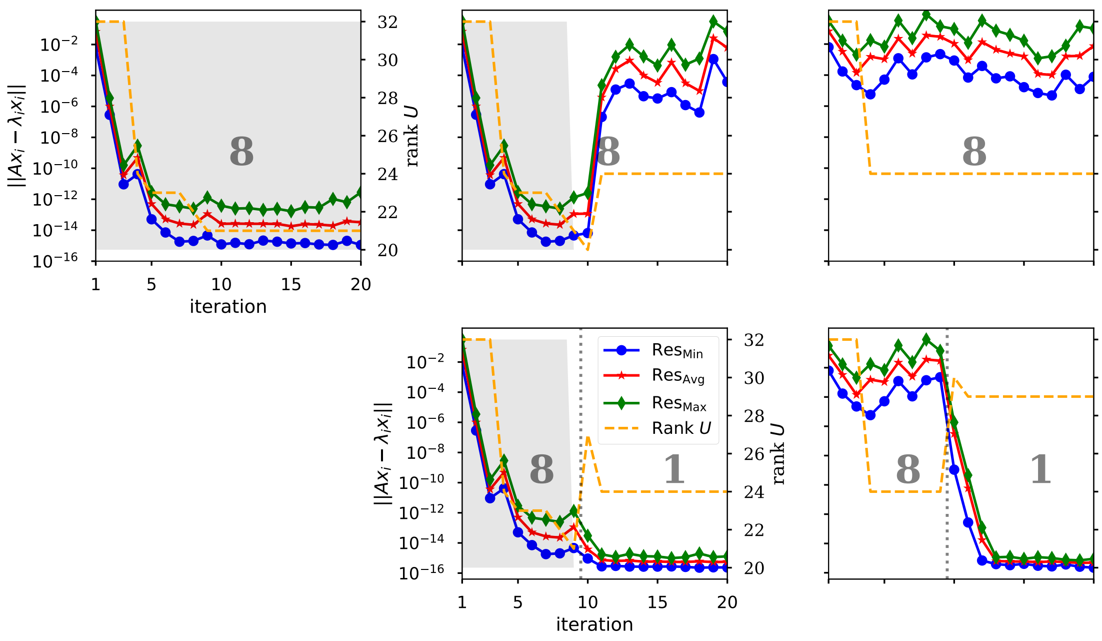


Figure 5.4.: As Figure 5.3, but with 8 moments. This figure also appears in [46].

5.1.6. Eigenvalue counting

An important remaining step is to be able to determine whether all eigenpairs in I_λ have been computed to an acceptable tolerance. We need to know how many eigenpairs are in the interval. We assume that a reasonable estimate is given a-priori. However, this estimate may need to be updated over the course of running the eigensolver. Fortunately, we can use information gained during an iteration to generate a new estimate, typically with reasonably good accuracy, especially as the quality of approximation of the eigenpairs improves over RFE iterations.

5.1.6.1. Singular values of U

Various techniques for counting the number of eigenvalues in the FEAST algorithm are available, as illustrated in [32]. As introduced in Chapter 2, based on [32, 65, 89], the singular values of U may be used to provide an estimate of this count. In floating point arithmetic, we can count the number of singular values, σ_i, $i = 1, \ldots, m_U$, of U above a tolerance (σ_t). A best choice was shown to be $\sigma_t = 0.5$ in [32].

For moment based methods, orthogonalization is an essential step in ensuring the full rank of U before a Rayleigh–Ritz step is performed. The singular values of U can be obtained as a not-too-expensive by-product of this orthogonalization. If an SVD decomposition is used to obtain an orthogonalization of U, the singular values are computed as part of this process. If an (economy sized) QR decomposition is used, the singular values can be extracted from the (smaller) matrix R.

5.1.6.2. Generation of τ values

An alternative metric for estimating the number of eigenvalues in I_λ was described in [56], with the singular values scaled by the absolute values of the Ritz vectors.

Specifically, we define $\mathbf{w}_i$ as the Ritz vector arising from the solution of the reduced eigenproblem in the Rayleigh–Ritz problem, $U^H A U \mathbf{w}_i = \lambda_i U^H B U \mathbf{w}_i$, $i = 1, \ldots, m_U$. We again define σ_i as the i^{th} singular value of U.

Then, our metric τ_i is defined as

$$\tau_i = \frac{\sum_{j=1}^{m_U} |\mathbf{w}_{i,j}|^2}{\sum_{j=1}^{m_U} \sigma_j |\mathbf{w}_{i,j}|^2} \quad i = 1, \ldots, m_U. \tag{5.12}$$

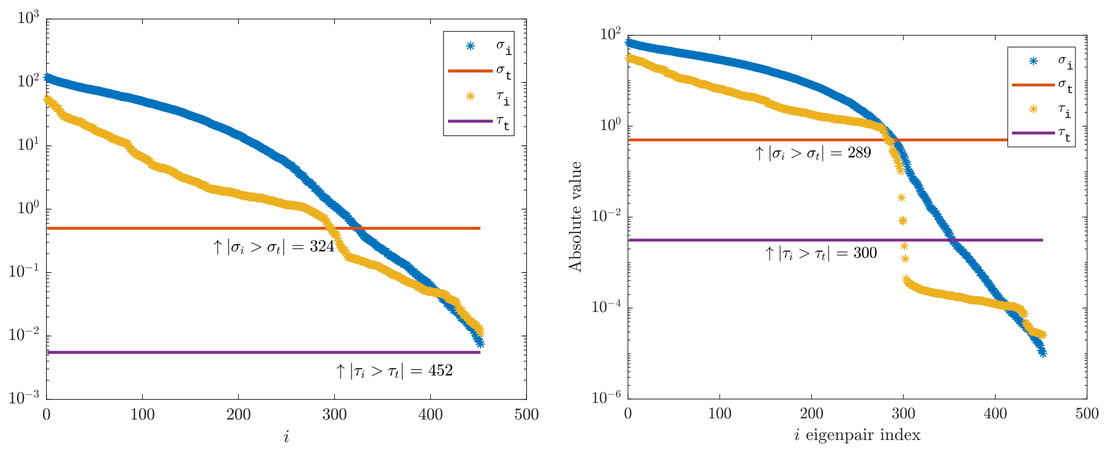
This metric was originally defined as a quality control measure for the computed eigenpairs, to determine whether or not they are spurious. But it can equivalently be used to track how many of the computed eigenpairs are associated with true eigenpairs within the interval [83]. Assuming $m_U > m$, and each true eigenpair is associated with a computed eigenpair, a good estimate should result. Furthermore, we expect the quality of the estimate to improve over RFE iterations.

5.1.6.3. Numerical experiment

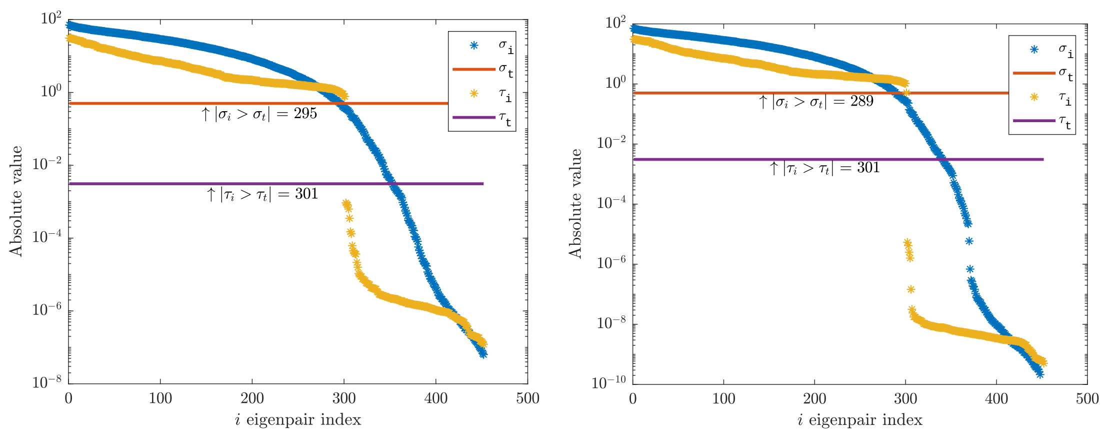
These two metrics can now be compared in a numerical experiment. We consider a standard eigenproblem with A = GraphI-1k, a 1152×1152 sparse, symmetric real matrix with 301 eigenvalues in $I_\lambda = [-0.7575, 1.1025]$. This problem is also described as No. 20 in Table 5.2.

We compare the values of τ_i and σ_i over RFE iterations with either 4 (as shown in Figure 5.5) or 1 moment (as shown in Figure 5.6). We use an initially random block vector Y, along with either 4 or 1 moment to obtain a full subspace size of m_U = 452 or 450 respectively. All linear systems are solved using MATLAB's backslash function. The `svd()` function was used for the generation of the singular values, and the `eig()` function for the calculation of Ritz values and vectors. Gauss–Legendre quadrature was used, with 16 quadrature points on the entire circular contour.

The values τ_i and σ_i are generated as described above. We count the number of values below the thresholds [83].



(a) Iteration 1 (L) and 2 (R)



(b) Iteration 3 (L) and 4 (R)

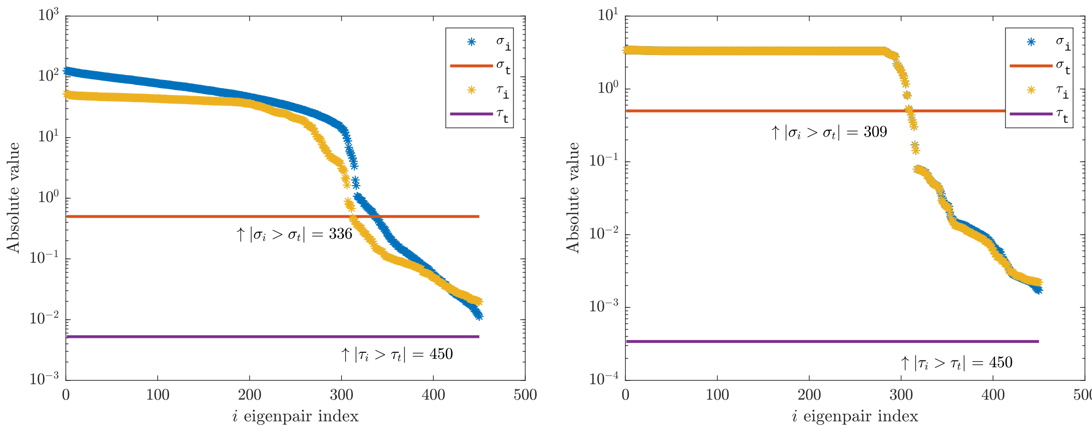
Figure 5.5.: τ_i vs. σ_i over 4 iterations. $s = 4$

$$\tau_t = 10^{-4} \times \max_{i=1,\dots,m_U} \tau_i$$

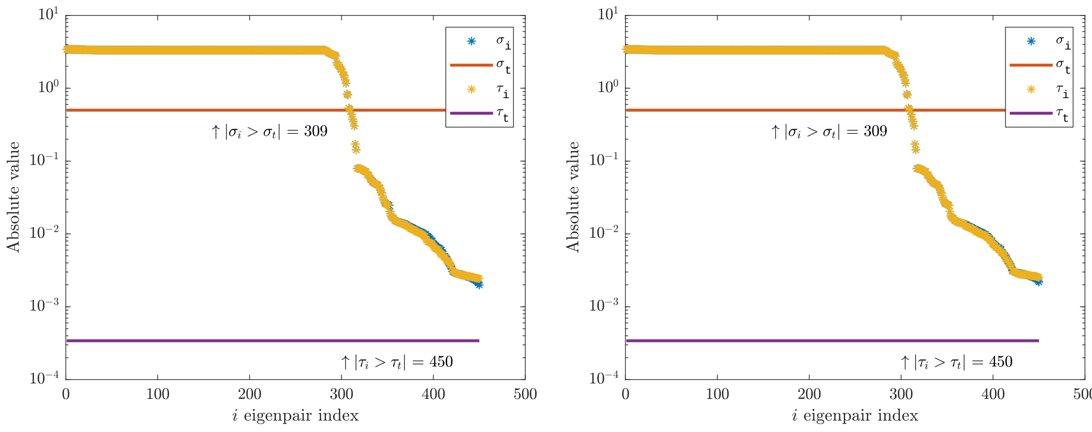
and

$$\sigma_t = 0.5.$$

As we see in Figures 5.5 and 5.6, the number of moments used makes an observable difference in the quality of the metric. With 4 moments, the count resulting from the $\tau_i's$ is clearly superior, especially after the first iteration. We can see a marked drop off and good separation between the "good" values of τ_i, corresponding to real eigendirections, and "bad" values, corresponding to spurious vectors. With 1 moment, the difference in quality disappears between σ_i and τ_i after the first couple



(a) Iteration 1 (L) and 2 (R)



(b) Iteration 3 (L) and 4 (R)

Figure 5.6.: τ_i vs. σ_i over 4 iterations. $s = 1$.

of iterations. Obviously the threshold chosen for counting τ_i is not well suited to the single moment case, but this is less relevant than the possibility of a well placed threshold; that is, a point at which the values exhibit a steep drop off. This is more obvious for the values of σ_i in the 1 moment case.

5.2. Extension of the BEAST framework

The BEAST framework, as introduced in Chapter 2, has been extended beyond its original form to include (iterative) SSM. This extension is also introduced in [46]. As discussed previously, we consider the SSM with a constrained subspace size, meaning a change in the number of moments may be required to reach the desired convergence threshold. Several extensions of a traditional SSM are therefore added within the BEAST solver to support its iterative and adaptive nature. In particular, the considerations discussed in this chapter are included. Most parameters may be set or changed by the user as options. Suggested parameters are listed in this chapter. In the cases where a collection of fixed algorithmic choices have been made, this is stated in their respective description.

A variety of schemes are possible when multiple moments are used, including the options discussed up to this point. We consider the possible combinations of the two "switches" that we have discussed so far: the inner vs. outer iteration type, and the flexible vs. fixed number of moments. These are labelled as BEAST-M$_{\mathtt{i},*}$, BEAST-M$_{\mathtt{o},*}$, BEAST-M$_{*,\mathtt{x}}$ and BEAST-M$_{*,\mathtt{n}}$ respectively. An overview of the BEAST-M schemes is given in Algorithm 20. In Section 5.2.3 we describe again the distinguishing features between these methods along with some related RFEs.

Algorithm 20: BEAST-M

```
Choose a starting block of $m_Y$ vectors $Y$;
while not converged do
    Compute $U = \left[U^0,\ \sum_{i=1}^{q} w_i z_i \Phi_i^{-1} BY,\ \ldots,\ \sum_{i=1}^{q} w_i z_i^s \Phi_i^{-1} BY\right]$
    if checking convergence then
        Orthogonalize $U$
        Compute Rayleigh quotients $A_U = U^H AU$, $B_U = U^H BU$
        Solve reduced eigenproblem $A_U W = B_U W \Lambda$ for $(W, \Lambda)$
        Compute approximate eigenvectors $X = UW$
        Compute residuals $Ax_i - \lambda_i Bx_i$, $[\lambda_1, \ldots \lambda_{m_U}] = \mathrm{diag}(\Lambda)$, $\lambda \in I_\lambda$
    end
    optional Lock converged eigenpairs
    optional Check if change in $s$ needed (BEAST-M$_{*,\mathtt{x}}$), otherwise BEAST-M$_{*,\mathtt{n}}$
    Set $Y = U^0$ (BEAST-M$_{\mathtt{i},*}$) or $XR$ (BEAST-M$_{\mathtt{o},*}$) for $R \in \mathbb{R}^{m_U \times m_Y}$
end
```

5.2.1. Definition of cost metrics

The main metric we will use for comparing of the cost of solving linear systems of equations in an RFE is $\mathrm{RHS}_{\mathrm{ovl}}$, the number of right-hand sides in all linear systems and over all iterations of the RFE. We are also interested in reducing the number of block linear systems solved over all iterations of the RFE, which we label $\mathrm{BLS}_{\mathrm{ovl}}$. These two metrics are both important for evaluating the cost of solving linear systems. If a direct method is used, assuming the factorized components can be stored, there is a single up-front factorization cost, followed by the repeated cost of applying the factorized components to the (multiple) right hand sides. An example of this approach is summarized in Algorithm 21. Here, the number of $\mathrm{RHS}_{\mathrm{ovl}}$ is significant as it represents the repeated component of solving the linear systems: matrix-vector multiplication with the factorized components and the columns of BY.

Algorithm 21: Example of direct solution of linear systems over iterations of an RFE. Here, the direct linear solution is shown to rely on an `lu` decomposition [93], but other factorizations may be used.

```
for $i = 1, ..., q$ do
    $[L, U] = \texttt{lu}(z_i B - A)$
end
while RFE not converged do
    Solve $U^0 = U^{-1} L^{-1} BY$
end
```

In this chapter, we focus on the solution of linear systems with a direct method. If an iterative method is used, however, the metric of $\mathrm{RHS}_{\mathrm{ovl}}$ is still effective in measuring relative cost of linear system solves. If a non block-iterative method is used, we must consider the separate iterative solution of $\Phi\mathbf{u} = \mathbf{y}$ for each column $\mathbf{y}$ in Y. Even when a method that does allow for block right hand sides is used, the method and its computational design will control the relative importance of $\mathrm{RHS}_{\mathrm{ovl}}$ and $\mathrm{BLS}_{\mathrm{ovl}}$.

5.2.2. Quadrature rule

Sakurai–Sugiura methods have traditionally been introduced as contour integration-based schemes. As discussed in Section 5.1.2, we can analyse the error in a subspace generated with the numerical approximation of a contour integral and the combination of s moments. The choice of quadrature rule, including the value of q, remains open to the user, assuming certain conditions are met, as discussed in Chapter 2. Traditionally, SSM have relied on the midpoint quadrature rule, as defined in (2.9).

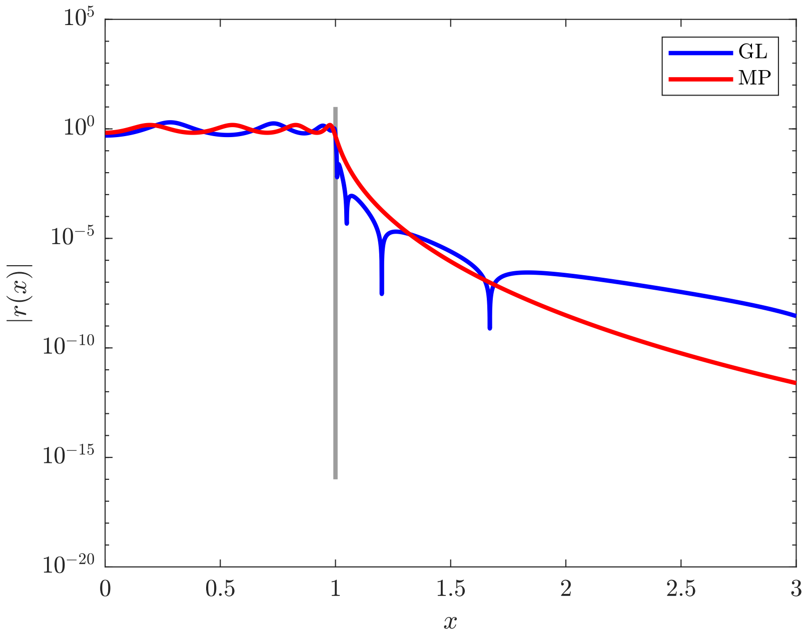


Figure 5.7.: Filters arising from quadrature, $q = 16$ with Gauss–Legendre (GL) and midpoint (MP) rules. Contour is an ellipse with eccentricity $= 0.1$, radius 1, and center 0.

To avoid confusion: the definition of the midpoint rule given in (2.9) matches the definition of the trapezoidal rule in [85], and subsequent works defining extensions of SSM. On the other hand, the Gauss–Legendre quadrature rule, as defined in (2.10), is more popular with FEAST. As originally shown and discussed in [46], there is a reason for this difference in "standard" quadrature rules.

Interestingly, the effectiveness of a subspace generated with a given rule may depend on the subspace size. We see that Gauss–Legendre has a steeper drop at the boundary of the interval, but that the midpoint rule continues to decay further from the boundary, eventually reaching a lower value than Gauss–Legendre. This indicates that methods with a smaller subspace size may see faster convergence with Gauss–Legendre quadrature, but beyond a certain threshold, convergence with the midpoint rule may outperform it.

For further illumination, we show a numerical experiment from [46], performed using MATLAB. We consider 37 standard eigenproblems, as defined in Table 5.2, with additional matrix details in Table A.1, with $\texttt{BEAST-M}_{\texttt{i,x}}$ and $\texttt{BEAST-M}_{\texttt{o,x}}$. In all cases, the method began with 4 moments. The subspace factors of 1.5 and 4 were tested. In all cases, Y was initially chosen randomly. The respective quadrature rules were tested with $q = 16$ on the full elliptical contour with eccentricity 0.1. Only the 8 linear systems corresponding to the upper half of the contour were

Table 5.1.: Average RHS_{ovl} for `BEAST` with midpoint and Gauss–Legendre quadrature rules and different subspace sizes. The average was taken over the RHS_{ovl} for 31 sample problems listed in Table 5.2, for which all schemes found all eigenpairs. This condition was not met for problems labelled (7,14,20,26,34); they are omitted from the average. A version of this table appears in [46].

Iteration Type	Quadrature Rule	Subspace Factor	Average RHS_{ovl}
$\texttt{BEAST-M}_{\texttt{i,x}}$	Gauss–Legendre	1.5	3587
$\texttt{BEAST-M}_{\texttt{i,x}}$	Midpoint	1.5	3514
$\texttt{BEAST-M}_{\texttt{i,x}}$	Gauss–Legendre	4.0	5229
$\texttt{BEAST-M}_{\texttt{i,x}}$	Midpoint	4.0	4545
$\texttt{BEAST-M}_{\texttt{o,x}}$	Gauss–Legendre	1.5	3201
$\texttt{BEAST-M}_{\texttt{o,x}}$	Midpoint	1.5	3723
$\texttt{BEAST-M}_{\texttt{o,x}}$	Gauss–Legendre	4.0	5218
$\texttt{BEAST-M}_{\texttt{o,x}}$	Midpoint	4.0	4366

solved, as all eigenproblems are real symmetric. All problems were solved to a tolerance of 10^{-13}.

The average RHS_{ovl} over all problems is shown for the different options in Table 5.1. We observe that in general, Gauss–Legendre quadrature leads to a lower RHS_{ovl} for a subspace width of $1.5 \times \tilde{m}$ and the midpoint rule leads to a lower RHS_{ovl} for a larger subspace width of $4 \times \tilde{m}$. This effect is increasingly pronounced for the outer iteration. These results fit well with our observations from Figure 5.7. The reasoning for the difference in default choice of quadrature rule for FEAST and SSM is now more clear.

5.2.3. Numerical experiments

We are now prepared to test the efficacy of the strategies outlined so far. The data and discussion shown here follows the original appearance in [46]. A summary chart illustrating the distinguishing features of the various methods considered is shown in Figure 5.8. We consider various permutations of the `BEAST` framework. Under consideration are all four of our `BEAST-M` variants resulting from combinations of $\texttt{BEAST-M}_{*,\texttt{n}}$, with a fixed number of moments, $\texttt{BEAST-M}_{*,\texttt{x}}$, with an adaptive value, as well as $\texttt{BEAST-M}_{\texttt{o},*}$, with the outer iteration type and $\texttt{BEAST-M}_{\texttt{i},*}$, with the inner iteration type. `BEAST-C` is again the single moment scheme described in Chapter 2 with both an adaptive $\texttt{BEAST-C}_{\texttt{ad}}$ and fixed number $\texttt{BEAST-C}_{\texttt{n}}$ of quadrature nodes. These strategies follow the definitions given in [33]. The results shown for $\texttt{BEAST-C}_{\texttt{n}}$ are for the degree providing the minimum RHS_{ovl}, assuming all eigenpairs are found to the desired tolerance.

Table 5.2.: Search intervals, along with respective eigenvalue counts m for 9 test matrices from graphene modeling [23] and 10 test matrices from the SuiteSparse Matrix Collection [25]. Problems are labeled by number (No.). More information for these matrices can be found in Table A.1.

Matrix	Interval A	m	No.	Interval B	m	No.
laser	$[-0.100, 0.357]$	307	1	$[1.0000, 4.2389]$	304	NA
SiH4	$[25.0, 28.4]$	301	2	$[15.3, 16.4]$	290	3
linverse	$[0.00, 0.62]$	308	4	$[2.62, 2.77]$	304	5
Pres_Poisson	$[3.7, 10.0]$	302	6	$[1.000, 1.182]$	300	7
Si5H12	$[24.2, 24.7]$	320	8	$[41, 42]$	274	9
bcsstk37	$[8.0e5, 9.3e5]$	297	10	$[1.15e7, 1.30e7]$	305	11
brainpc2	$[275, 345]$	313	12	$[1900, 1920]$	309	13
rgg_n_2_15_s0	$[-2.00, -1.95]$	282	14	$[5.0, 5.5]$	337	15
SiO	$[32.0, 32.4]$	316	16	$[57.0, 57.8]$	283	17
Andrews	$[21.0, 21.4]$	300	18	$[11.20, 11.26]$	298	19
GraI-1k	$[-0.7575, 1.1025]$	301	20	$[0.42, 1.58]$	301	21
GraI-11k	$[-0.0475, 0.3925]$	298	22	$[0.935, 1.065]$	289	23
GraI-119k	$[0.1375, 0.2075]$	306	24	$[0.9957, 1.0043]$	304	25
GraII-1k	$[-0.7575, 1.1025]$	292	26	$[0.42, 1.58]$	304	27
GraII-11k	$[-0.1375, 0.4825]$	299	28	$[0.949, 1.051]$	299	29
GraII-119k	$[0.0975, 0.2475]$	313	30	$[0.9956, 1.0044]$	303	31
GraIII-1k	$[-0.7575, 1.1025]$	300	32	$[0.42, 1.58]$	331	33
GraIII-11k	$[-0.0975, 0.4425]$	305	34	$[0.952, 1.048]$	319	35
GraIII-119k	$[0.1395, 0.2055]$	310	36	$[0.996, 1.004]$	311	37

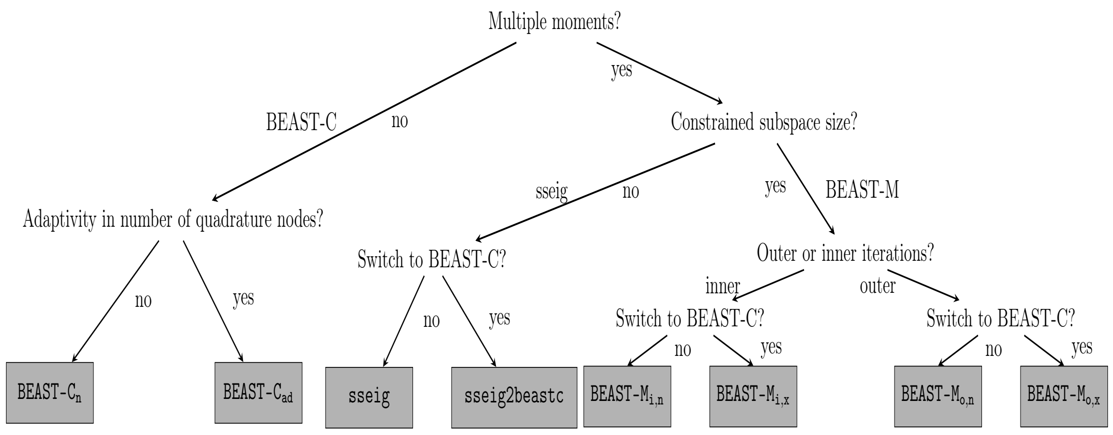


Figure 5.8.: Overview of all computation schemes and choices considered in section 5.2. This figure also appears in [46].

We also compare with a larger subspace size, as is typically considered in SSM. This is encapsulated in the `sseig` implementation of SS-RR, as well as with the method `sseig2beastc`, where the method switches after a single outer iteration from `sseig` to `BEAST-C`, and using the computed eigenvectors $\widetilde{X}_{I_\lambda}$ as the starting vectors Y for `BEAST-C`. The details of SS–RR implemented in the `sseig` software are further described in [84].

We consider 37 sample eigenproblems as described in Table 5.2. For all problems shown here, the eigenproblem is standard, with a real symmetric matrix A. `BEAST` schemes used Gauss–Legendre quadrature and `sseig` schemes used the midpoint rule with $q = 16$ on an elliptical contour with eccentricity 0.1. As all eigenproblems are real, only $\frac{q}{2} = 8$ linear systems need to be solved in each iteration of the RFE. The counts of $\mathrm{RHS}_{\mathrm{ovl}}$ and $\mathrm{BLS}_{\mathrm{ovl}}$ include only counts from these linear systems. All methods with multiple moments began with $s = 4$. The initial subspace size m_U was set for all `BEAST` methods to 452 (when multiple moments were used) or 450 (otherwise), approximately $1.5 \times \tilde{m}$ for all problems. For `sseig` and `sseig2beastc` the subspace size was set to a maximum of $m_U = 1024$. Locking of converged eigenvectors, as described in Chapter 2, was enabled for all `BEAST` methods. MATLAB was used for these numerical experiments.

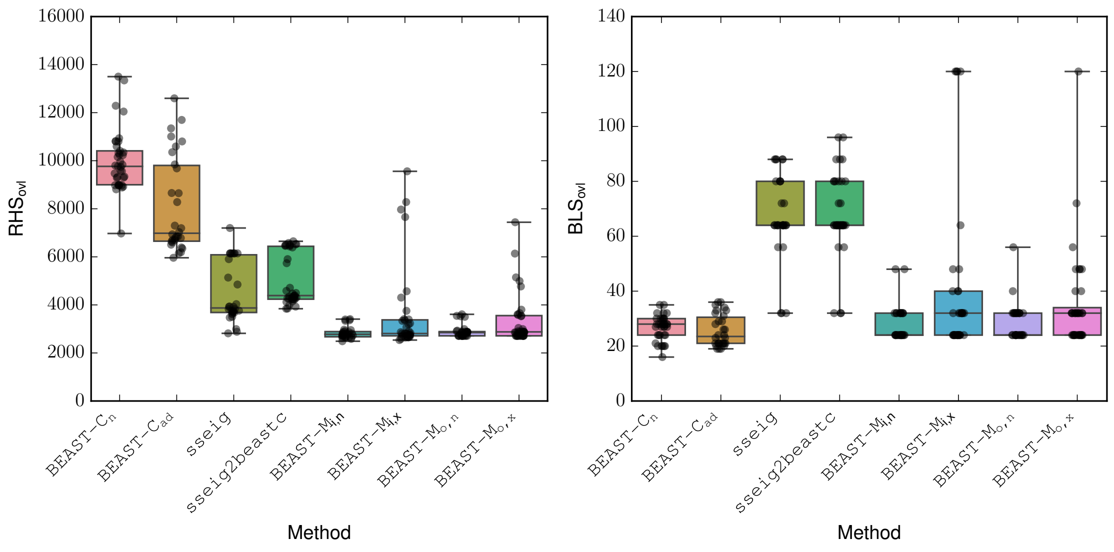


Figure 5.9.: $\mathrm{RHS}_{\mathrm{ovl}}$ (left) and $\mathrm{BLS}_{\mathrm{ovl}}$ (right) for 37 eigenproblems solved by all schemes under consideration. The respective value for each eigenproblem is shown as a translucent circle, assuming all eigenpairs were found using the method under consideration. Statistical analysis of the data is shown, with the median given by a horizontal black line. The surrounding box extends from the first to third quartiles, and whiskers to the largest and smallest observations. Full numerics are available in Table 5.3. This figure also appears in [46].

Table 5.3.: $\mathrm{RHS}_{\mathrm{ovl}}$: $\mathrm{BLS}_{\mathrm{ovl}}$ corresponding to Figure 5.9. No. corresponds to problem number listed in Table 5.2. A dash indicates that not all eigenpairs were found to given tolerance. A version of this table appears in [46].

	BEAST-C		sseig		BEAST-M			
No.	`n`	`ad`	no switch	`2beastc`	`i,n`	`i,x`	`o,n`	`o,x`
1	6972 : 16	-	5904 : 80	6392 : 80	-	8280 : 120	-	4992 : 40
2	10320 : 24	-	3744 : 64	4280 : 64	2968 : 32	7968 : 120	2992 : 32	2992 : 32
3	9585 : 27	6703 : 21	7200 : 80	4120 : 64	2672 : 24	2672 : 24	2712 : 24	2712 : 24
4	10234 : 28	6781 : 21	3680 : 64	4304 : 64	2776 : 24	2776 : 24	2856 : 24	2856 : 24
5	8885 : 20	6784 : 21	3920 : 64	4392 : 64	2752 : 24	2752 : 24	2712 : 24	2712 : 24
6	9835 : 35	10803 : 33	6144 : 88	6576 : 96	3336 : 32	3336 : 32	3504 : 56	3584 : 48
7	9765 : 27	6697 : 21	3648 : 64	4224 : 64	2728 : 24	2728 : 24	2712 : 24	2712 : 24
8	10410 : 30	-	-	4584 : 64	2808 : 24	2808 : 24	2848 : 32	2848 : 32
9	9800 : 30	6637 : 20	3472 : 64	3928 : 64	2488 : 24	2536 : 32	2888 : 24	2888 : 24
10	8980 : 30	-	3712 : 64	4232 : 64	2584 : 24	2584 : 24	2712 : 24	2712 : 24
11	10818 : 27	6189 : 19	3824 : 64	4352 : 64	2808 : 32	2808 : 32	2880 : 32	2880 : 32
12	10936 : 32	7194 : 24	3936 : 72	4472 : 72	2792 : 24	2792 : 24	2712 : 24	2712 : 24
13	9297 : 27	6390 : 20	3824 : 64	4384 : 64	2696 : 24	2696 : 24	2712 : 24	2712 : 24
14	8980 : 30	6839 : 23	-	-	-	-	-	-
15	8995 : 20	8649 : 29	4032 : 64	4712 : 64	2904 : 32	2904 : 32	2928 : 32	3040 : 48
16	10596 : 30	8657 : 29	-	4496 : 64	2832 : 24	2832 : 24	2856 : 32	2856 : 32
17	8922 : 24	6348 : 19	-	3992 : 64	2648 : 24	2648 : 24	2712 : 24	2712 : 24
18	9744 : 28	7299 : 24	3920 : 72	4360 : 72	2744 : 24	2744 : 24	2712 : 24	2712 : 24
19	9963 : 27	7029 : 23	3696 : 64	4232 : 64	2728 : 48	3072 : 40	2880 : 32	2880 : 32
20	10255 : 35	12598 : 36	-	6456 : 80	-	4568 : 40	-	5144 : 72
21	8810 : 20	6936 : 26	2880 : 32	3848 : 32	2664 : 24	2664 : 24	2848 : 32	2848 : 32
22	9000 : 20	6848 : 21	6144 : 88	6480 : 88	2952 : 32	7656 : 120	2712 : 24	2712 : 24
23	13500 : 30	10594 : 32	6144 : 88	6520 : 96	3392 : 32	3392 : 32	-	3656 : 48
24	8975 : 20	9845 : 31	3616 : 64	4256 : 64	2864 : 32	2864 : 32	2872 : 32	2872 : 32
25	9471 : 28	6505 : 21	3744 : 64	4304 : 64	-	3192 : 40	2888 : 24	2888 : 24
26	12047 : 28	-	-	6432 : 80	-	4304 : 48	3528 : 40	3528 : 40
27	8976 : 24	-	2816 : 32	3840 : 32	2656 : 24	2656 : 24	-	3800 : 32
28	10800 : 24	9682 : 28	6144 : 88	6488 : 88	-	3232 : 64	3544 : 32	3544 : 32
29	13344 : 32	11347 : 34	6144 : 80	6488 : 80	3360 : 32	3360 : 32	-	4768 : 56
30	9338 : 21	10355 : 35	4848 : 56	5736 : 56	-	9560 : 120	-	7440 : 120
31	9562 : 28	6599 : 21	3760 : 64	4304 : 64	2672 : 24	2672 : 24	2712 : 24	2712 : 24
32	12288 : 32	11698 : 33	-	6496 : 80	3408 : 32	3408 : 32	3608 : 32	3608 : 32
33	10157 : 28	6150 : 21	2992 : 32	4144 : 32	2640 : 24	2640 : 24	2904 : 32	2904 : 32
34	10800 : 24	11008 : 36	6144 : 88	6536 : 88	-	-	-	6136 : 48
35	10346 : 28	8279 : 26	6144 : 80	6648 : 80	2808 : 32	2808 : 32	2840 : 32	2840 : 32
36	9317 : 28	-	5136 : 64	5904 : 64	2736 : 24	2736 : 24	2712 : 24	2712 : 24
37	9289 : 28	5961 : 19	3504 : 56	4240 : 56	2888 : 48	3752 : 48	2712 : 24	2712 : 24

Table 5.4.: Number of times a method failed to find all eigenpairs to desired tolerance (out of 37 problems).

Solver	Total failed
$\texttt{BEAST-C}_{\texttt{ad}}$	7
`sseig`	7
`sseig2beastc`	1
$\texttt{BEAST-M}_{\texttt{i,n}}$	8
$\texttt{BEAST-M}_{\texttt{i,x}}$	2
$\texttt{BEAST-M}_{\texttt{o,n}}$	8
$\texttt{BEAST-M}_{\texttt{o,x}}$	1

The absolute residual tolerance was set to 10^{-13}. A method was deemed to have succeeded if all eigenpairs were computed to this tolerance, otherwise it was determined to have failed. A summary of the number of failures for each method is shown in Table 5.4. As we see here, $\texttt{BEAST-M}_{*,\texttt{x}}$ and `sseig2beastc`, the methods allowing a switch to `BEAST-C`, behaved the most robustly in terms of ensuring that all eigenvalues were found to the desired tolerance.

We observe the number of RHS_{ovl} and BLS_{ovl} in Figure 5.9. Clearly, increasing the number of moments has a strong impact on the number of RHS_{ovl}. A reduced subspace size also seems to have an impact, as we see a reduction in RHS_{ovl} for the `BEAST-M` schemes when compared to `sseig` and `sseig2beastc`. It is therefore $\texttt{BEAST-M}_{\texttt{i,x}}$ and $\texttt{BEAST-M}_{\texttt{o,x}}$ that stand out as the best choices of methods. Comparing these two, we see more outlier cases where the number of RHS_{ovl} is large for $\texttt{BEAST-M}_{\texttt{i,x}}$. Here, we must consider the interplay of multiple effects; namely the consideration of locking of converged eigenvalues, as discussed in Chapter 2. When locking is enabled, the expected number of eigenpairs remaining in the interval sinks as convergence begins. This is not a theoretical issue, but when only a few eigenpairs remain, the $1.5 \times \tilde{m}$ factor guarantees a very small subspace. We consider the example case shown in Figure 5.10. After convergence begins, eigenpairs are locked, and the subspace shrinks. This reduction in subspace size appears be correlated with a reduction in convergence rate, and the number of RFE iterations required for the method increases.

5.3. Considerations for larger problems

Our considerations thus far have mainly been for smaller problems and numerical results without significant consideration for performance and scalability. We now turn to issues that arise as problem size and performance expectations increase.

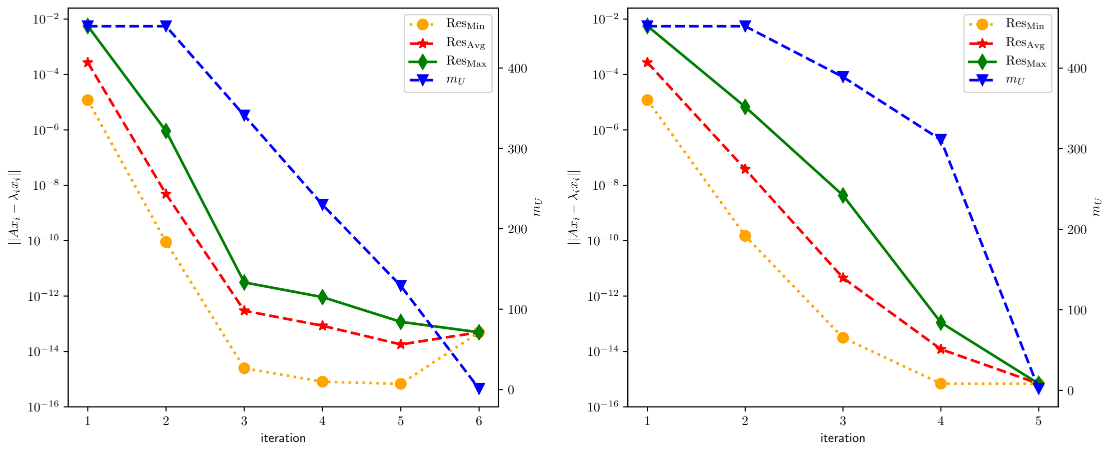


Figure 5.10.: Convergence and subspace size m_U with `BEAST-M`$_{\mathtt{i,x}}$ (left) and `BEAST-M`$_{\mathtt{o,x}}$ (right) for Problem 26, Table 5.2. A version of this figure also appears in [46].

5.3.1. Rank and orthogonalization

One particular challenge of using multiple moments is correctly preserving the linear independence of the basis for the generated subspace, U. As we see in Figures 5.3 and 5.4, the rank of U may fall below $s \times m_Y$. To avoid a rank-deficient eigenproblem in the subsequent Ritz step, m_U should be shrunk to rank(U). Furthermore, U should be orthogonalized to avoid near-rank deficiency, and the determination of rank at this point (via, e.g., a rank-revealing QR decomposition) ensures that a rank-deficient eigenproblem can be avoided. However, removing linearly independent directions may remove components of the basis of $\tilde{X}_{I_\lambda}$, so over-zealous trimming of the subspace must also be avoided.

One important question is how the numerical rank of a set of vectors can be accurately determined. Given that a rank-revealing QR or SVD decomposition has been computed, and we have obtained the singular values of U, these may be used to determine the rank of U. In the performance-based implementation of `BEAST`, experimentation showed that keeping columns of U associated with singular values larger than $2 \times \epsilon_{mach}$ (a stricter cut-off than a standard ϵ_{mach}) may be helpful. Stricter cut-off values were also tested, but an over-reduction of the rank may also be detrimental, as discussed above. This cut-off factor is modifiable in the implementation.

The QR decomposition of U is also a matter of consideration. Since U is often rank deficient, a basic, e.g., Cholesky-QR may not be sufficient. Even a repeated Cholesky-QR [29] may not be enough for the orthogonalization of these block vectors if the condition number surpasses 10^8, as is the case for a rank-deficient subspace. For the performance-based implementation of BEAST-M, we tested with a Tall-Skinny QR (TSQR) [26], as provided by the Trilinos library [45] via the PHIST library [91] and a numerically stabilized SVQB [88] routine, described in Algorithm 22. The latter has been incorporated into BEAST [30]. The TSQR relies on Householder reflections and is communication avoiding. Both methods appear numerically stable for the problems tested, but the custom implementation of a SVQB decomposition ran faster in some test cases and reduced dependence on external libraries. This is therefore the default technique used in the the current performance-based implementation of BEAST-M.

5.3.1.1. SVQB

The SVQB technique used returns a B-orthogonal matrix $Q \in \mathbb{C}^{m \times n}$ and square matrix M, not necessarily upper triangular, such that $U = QM$. After a SVQB decomposition is performed, the singular values of U can be found via an SVD of the (small) $n \times n$ matrix M. These can be used to determine the rank of U. Further details of this method, including the B-orthogonality of Q, are shown in [30].

Algorithm 22: SVQB orthogonalization

```
Set $Q = U$
Set $M = I_{n \times n}$
for $x$ iterations do
    $S = Q^H B Q$
    Compute $D = \text{diag}(S)$ for the normalization of $S$
    $S = D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$
    Solve the eigenproblem $SW = W\Lambda$
    (optional) Replace all $\lambda \in \text{diag}(\Lambda) < \tau_{\text{thresh}}$ with $\lambda = \epsilon_{mach}$
    Compute $Q = QD^{-\frac{1}{2}} W \Lambda^{-\frac{1}{2}}$
    Compute $M = \Lambda^{\frac{1}{2}} W^H D^{\frac{1}{2}}$
end
```

As suggested in the original introduction of the SVQB [88], we include several techniques to deal with a possibly rank-deficient U and to ensure its orthogonalization. These are:

- Replacing eigenvalues below a given threshold τ_{thresh}(we use $\tau_{\text{thresh}} = 0$; see the optional step in the algorithm.)
- Repeating SVQB iterations, according to the number of iterations specified in the algorithm, typically up to three times for very ill-conditioned vectors.

5.3.2. Solution of linear systems

As discussed in Chapter 2, the solution of linear systems is an algorithmic requirement for the `BEAST-M` framework. Either a direct or iterative linear solver may be used, depending on the qualities of the linear systems to be solved. In this chapter, we rely on direct solvers, which are less sensitive to the shifted and thus ill-conditioned systems of equations. If a direct solver is used, the resulting factorization of $(z_iB - A)$ may be stored, reducing the cost of multiple `BEAST` iterations. As stated in Chapter 2.10.4, and [30], the `BEAST` framework incorporates support for the STRUMPACK and MUMPS libraries for the direct solution of linear systems.

5.3.3. Numerical experiments

We have shown in the previous section that it is possible to reduce the number of RHS_{ovl} in an RFE using `BEAST-M`. The natural question, originally considered and discussed in [46], is what effect this reduction may have on the reduction in computational cost for the RFE, e.g., in time taken to reach convergence. This question is naturally dependant on a number of factors; the linear solver used (direct or indirect), whether the factorization of a direct solver is stored, parallelization, the kernel libraries used, etc. Needless to say, the difference in performance is highly variable, and not all cases may be considered here. However, we can illustrate here that a definitive difference in performance is possible.

We compare $\texttt{BEAST-C}_\texttt{n}$ and $\texttt{BEAST-M}_\texttt{o,x}$ for two larger standard eigenproblems, as described in Table A.2. Testing was performed with 32 nodes of the Emmy cluster at Friedrich-Alexander-Universität Erlangen–Nürnberg. Eigenpairs were computed to a tolerance of 10^{-12}, and locking was enabled. The Gauss–Legendre quadrature rule with a circular contour was used with 16 quadrature nodes on the whole contour. The direct solver STRUMPACK [80] was used to solve all linear systems. The number of columns of U was set to $1.5 \times \tilde{m}$. $\texttt{BEAST-M}_\texttt{o,x}$ started with four moments.

We observe the results in Figure 5.11. Here, fewer RHS_{ovl} are required for $\texttt{BEAST-M}_\texttt{o,x}$ than `BEAST-C`, for both problems, and this reduction is also reflected in the time taken for the RFE. The reduction in time seems to correspond with the reduction in RHS_{ovl}, and this in turn appears problem dependant. To summarize, it is clear that the use of multiple moments can lead to noticeable reductions in cost for an RFE when a good selection of parameters is made.

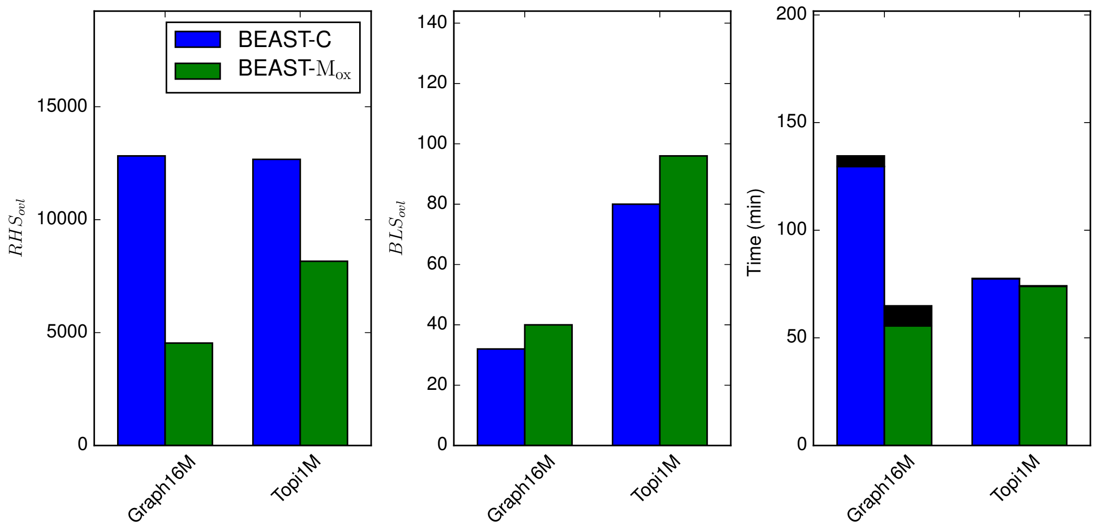


Figure 5.11.: Linear solve count and timings of BEAST-$M_{o,x}$ vs. BEAST-C. In the right-most figure, time for building U is shown in color and time for all other components is shown in black. This figure also appears in [46].

5.3.4. Parallelism and quadrature nodes

As discussed previously, the construction of U via a rational filter requires the solution of multiple linear systems of the form $(z_i B - A)^{-1}BY$ for different values of z_i. Clearly, these linear systems can be solved independently, meaning that this algorithmic step is embarrassingly parallelizable. Given increasing physical problem sizes and computational resources, improvements to a method may come in the form of increased *numerical efficiency* (as discussed previously) or *speed*. In the previous section, we showed that the additional flexibility in the BEAST algorithm may lead to improved numerical efficiency via a reduction in $\mathrm{RHS}_{\mathrm{ovl}}$. Now, as shown and discussed in [4], we consider the potential for speed-up when running in parallel with respect to the number of quadrature nodes. As the solution of linear systems is by far the most costly effort in BEAST-M, parallel speed-up is likely to have a significant effect. The additional efficiency from a reduced number of $\mathrm{RHS}_{\mathrm{ovl}}$ may also increase overall speed.

We consider a standard eigenproblem with matrix A=Graph1M, a real, symmetric $10^6 \times 10^6$ matrix. Further details may be seen in Table A.2. BEAST-M found 260 eigenpairs in the interval $[-0.01, 0.01]$. The BEAST-$M_{o,n}$ variant was used with a fixed number of moments to focus on the effect of changing the number of quadrature nodes and scaling computational resources. Testing was performed on the Emmy HPC cluster at Friedrich-Alexander-Universität Erlangen-Nürnberg. MUMPS was used for the direct solution of all linear systems. BEAST-M ran with 4 moments, and started with a random initial block vector Y with $m_Y = 100$ columns. A residual tolerance of 10^{-8} was reached for all eigenpairs.

We observe the strong scaling of `BEAST-M` for different number of quadrature nodes in Figure 5.12. We tested with two Gauss-Legendre quadrature rules, with 16 and 64 quadrature nodes on a circular contour. MUMPS was used to solve the linear systems. As the problem is real and symmetric, only $\frac{q}{2}$ (8 or 32) linear systems must actually be solved. We observe that overall scaling is dominated by the construction of U, especially for smaller numbers of processes N_p. For both schemes, scaling for $N_p \leq q$ is presumably dominated by being able to solve N_p linear equations simultaneously. After this point, parallelism in MUMPS allows $\frac{N_p}{q}$ processes to work in parallel to solve each linear equation, leading to further scaling. The number of quadrature nodes also controls the accuracy of the quadrature scheme and thus the quality of the filter function; we therefore expect the number of iterations to drop as the number of quadrature nodes increases. Indeed, for $q = 64$, only one iteration was required for `BEAST-M` to find all eigenpairs. However, for some values of N_p, random spurious eigenpairs caused multiple iterations to occur, causing some oscillatory behaviour in the graph. This occurred particularly for low values of N_p with $q = 64$. Regardless, we observe that when additional computational resources are available, further speed-up may be possible with a higher number of quadrature points.

This available increase in speed with additional quadrature points only occurs up to a certain point; once only one iteration is required for `BEAST-M`, additional quadrature points will only cause additional effort. Knowing how many quadrature nodes are required to obtain full convergence in a single iteration is, however, complicated, and the user may find that the increase of time due to additional iterations may be easier to control than the number of integration points needed to guarantee convergence in a single iteration without “wasted effort”. This will be explored further in the next section. Another interesting consideration is comparison with the behaviour of `BEAST-C`. In [33], an adaptive strategy was determined for the quadrature degree q of `BEAST-C`, usually resulting in a relatively low number of q and multiple iterations. This was shown to improve the efficiency of the method with reduced $\mathrm{RHS}_{\mathrm{ovl}}$ and $\mathrm{BLS}_{\mathrm{ovl}}$. It would be interesting to consider if this effect carries over to speed-up in a parallel system.

5.4. Choosing the quadrature degree

In Chapter 4, we estimated the number of RFE iterations required for a given rational filter. This was used to generate new filters, based on reducing the cost of iterative linear solvers over an RFE. We also wish to consider whether this prediction is appropriate for established rational filters, such as those based on contour

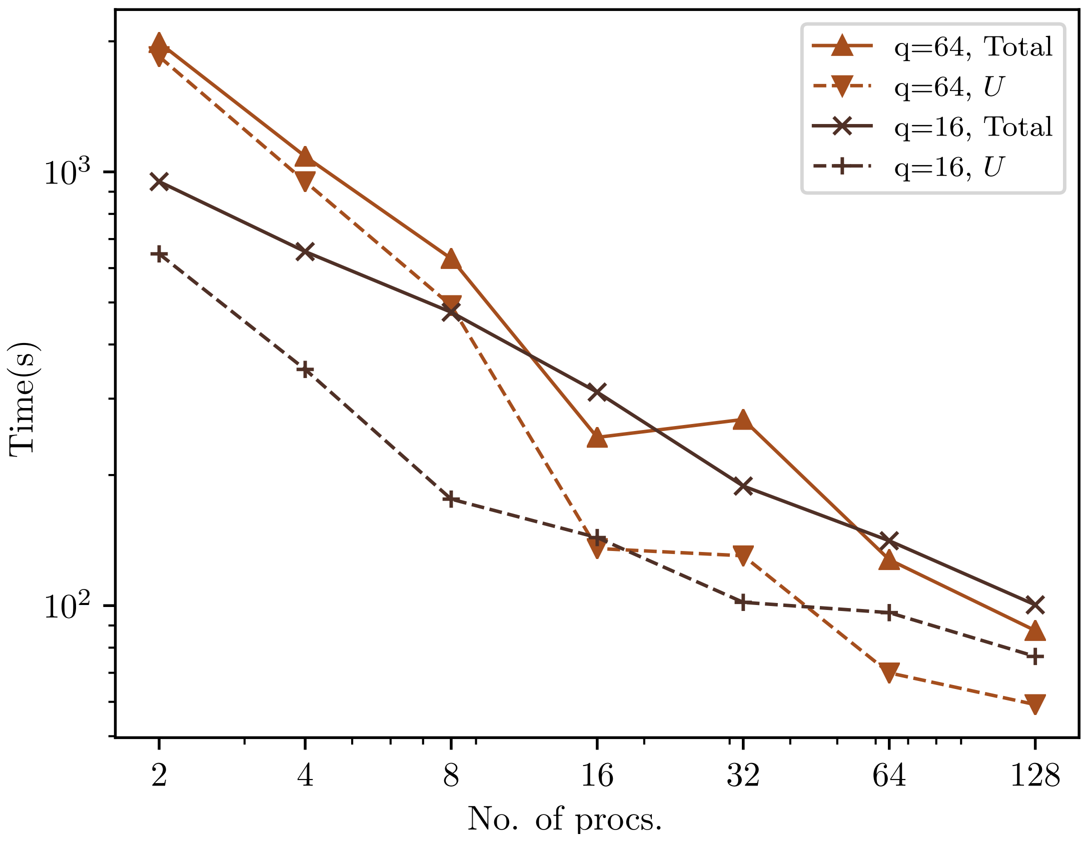


Figure 5.12.: Strong scaling of `BEAST-M`. The time for the total computation as well as for the construction of U (summed over RFE iterations) is shown. A version of this figure also appears in [4].

integration, in an RFE with multiple moments. Specifically, can we predict the degree of a contour integration rule that will have a low (ideally minimum) cost for the solution of linear systems? Previous work [33] has focused on adaptively finding the optimal degree for a single moment (FEAST based) RFE. This works well for single moment solvers, where for a good quadrature rule, the degree minimizing the number of RHS_{ovl} or BLS_{ovl} is typically quite low. Is this also true for RFEs with multiple moments? And how well can we predict the degree required?

Predicting the "optimal" degree for a problem with a given rational filter is actually quite difficult. An under-prediction may result in many extra RHS_{ovl} from an additional iteration, while an over-prediction results in extra RHS_{ovl} from the "unnecessary" degree. This is represented in Figure 5.13. A simpler problem is to predict the degree required such that only one RFE iteration is needed. Here, we may accept a certain amount of "over-prediction", or a higher degree than is really needed, in order to avoid an extra RFE iteration in most cases. Our prediction also relies on having a numerical bound for the expected error of our results. Such a bound exists for the inner iteration type, (5.8), a bound equivalent to the single moment RFE. We consider this in a numerical experiment, after first defining how the single-iteration degree we have described may be predicted.

5.4.1. Predicting degree for a single iteration

As in Chapter 4, we consider formulating an estimation of (5.8) based on a sequence of approximate eigenvalues δ_j. Without prior knowledge of the spectrum of the matrix, we generate an equidistant spectrum according to the known approximate density of I_λ. As before, we generate sample values for the rational filter at twice the expected average frequency of the eigenvalues, and adjust the index in (5.8) accordingly to obtain a prediction based on the estimated number of eigenvalues in the interval. This means that the approximated eigenvalues δ_j are placed with even spacing $h = \frac{I_\lambda}{2\tilde{m}}$ inside the interval $[\max(\lambda_{\min}, c - 10r), \min(\lambda_{\max}, c + 10r)]$. Again, $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of the entire spectrum. Given that the evaluated values are sorted according to $r(\delta_1) \geq r(\delta_2) \geq \ldots r(\delta_{m_U+1})$, we can evaluate (4.19) to calculate the predicted number of RFE iterations (l_{pred}) (restated here for convenience)

$$l_{pred} = \frac{\log(\text{tol}_{\text{RFE}})}{\log\left(\frac{r(\delta_{4\lfloor m_U/2\rfloor+1})}{r(\delta_{4\lfloor \tilde{m}/2\rfloor})}\right)}.$$

We can use this to predict a value of q for our quadrature rule such that only one RFE iteration is required. Indeed, as l_{pred} is not calculated as an integer, we can search for the smallest value of $q/2$ such that $l_{pred} < 0.7$. By choosing this smaller value, we are more sure that the degree selected will actually obtain convergence in a single iteration.

5.4.2. Numerical experiments

We consider solving 29 eigenvalue problems, a subset of the problems seen in Table 5.2. These are referenced by matrix name and problem number in Table 5.5 and problem number in Figure 5.14. As in our previous experiments in this chapter, we use `BEAST-M` (specifically `BEAST-M`$_{\mathtt{i,n}}$) with 4 moments. Here, we use the inner iteration type so that we may predict the theoretical convergence rate, as described in Section 5.1.2. The filter under consideration is Gauss–Legendre quadrature on an elliptical contour with eccentricity 0.1. For each problem, we perform computations with all integer values of $\frac{q}{2}$ up to a degree such that all eigenpairs in I_λ were computed to a residual tolerance of 10^{-10} in a single RFE iteration. Again, as all eigenproblems are real, only $\frac{q}{2}$ linear systems need to be solved in each iteration; this "actual count" is reflected in the results. The subspace size m_U was set to 452 for each iteration of the RFE and locking was disabled. MATLAB was used for all computations.

An example of how the value of $\mathrm{RHS}_{\mathrm{ovl}}$ changes over values of q along with the number of RFE iterations required is shown in Figure 5.13, for the sample matrix GraII-119k (problem 25) described in Table 5.2. Here we see that multiple local minima appear when an increase in quadrature degree results in a decrease in RFE iterations. We may now consider these local minima for our collection of problems.

In Figure 5.14 and Table 5.5 we compare the number of $\mathrm{RHS}_{\mathrm{ovl}}$ required for three different quadrature degrees, selected from the results for all degrees considered for each problem. These are: the degree that actually minimizes $\mathrm{RHS}_{\mathrm{ovl}}$, the degree such that our convergence criterion is satisfied in a single `BEAST-M` iteration, and the degree that we predict will enable convergence in a single `BEAST-M` iteration, as described above. Here we see that the degree that minimizes $\mathrm{RHS}_{\mathrm{ovl}}$ is often (though not always) low compared to the degree that achieves convergence in a single iteration. There may also be problems for which multiple values of q obtain a minimal value of $\mathrm{RHS}_{\mathrm{ovl}}$. As we see in Figure 5.14, even if $\mathrm{RHS}_{\mathrm{ovl}}$ is higher with the filter obtaining results in a single iteration than the filter degree with minimal $\mathrm{RHS}_{\mathrm{ovl}}$, these results are typically similar. In the case of the predicted degree being greater than the smallest degree such that only one RFE iteration is required, the value of $\mathrm{RHS}_{\mathrm{ovl}}$ was not actually computed. We assume in this case that the number of RFE iterations remains 1, and can calculate the number of $\mathrm{RHS}_{\mathrm{ovl}}$ required as $m_Y \times \frac{q}{2}$. We see that in some cases, our prediction is not large enough, and a substantial "overpayment" occurs from the required second iteration. But otherwise we obtain what is typically an overestimate, but still a relatively reasonably low cost for $\mathrm{RHS}_{\mathrm{ovl}}$.

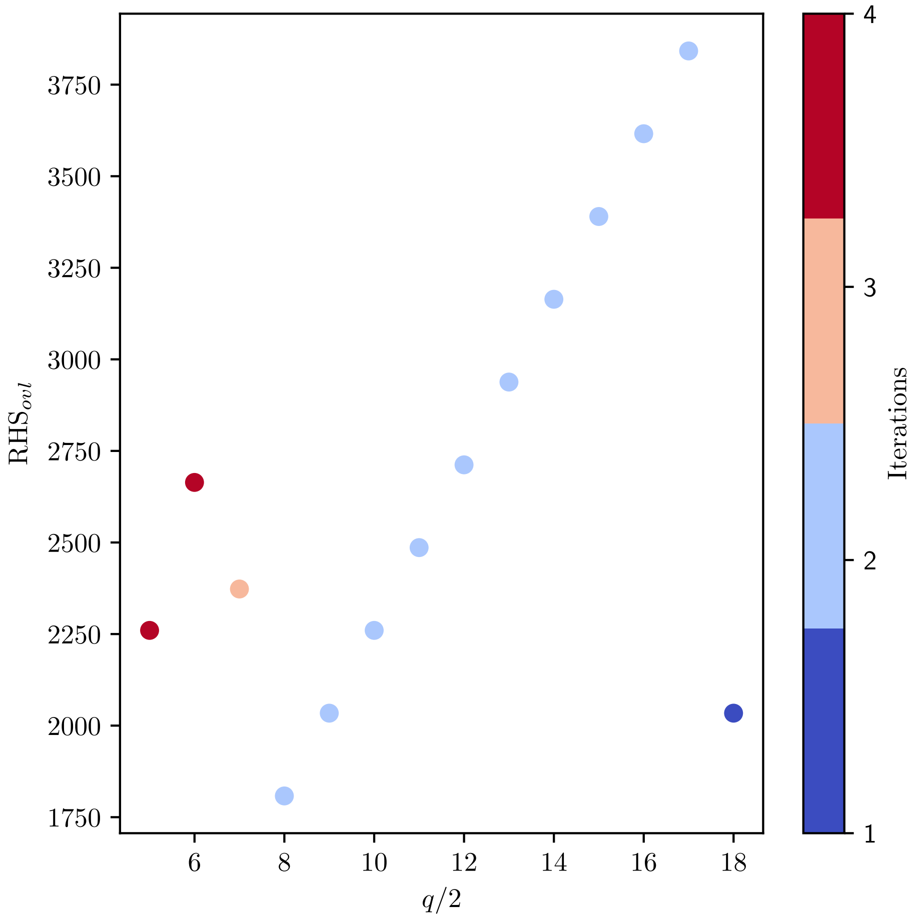


Figure 5.13.: RHS$_{\text{ovl}}$ vs. $q/2$ for `BEAST-M` with problem 25 in Table 5.2.

We note that as the rational filter defined by our quadrature rule is also scaled by I_λ and the same estimate for δ_j is used for all matrices, the value of l_{pred} is relatively constant over problems; indeed, it is the same for all problems seen here. This implies that an estimate for the degree required for a rational filter such that an RFE is expected to converge in a single iteration may be generated a-priori. Obviously, further knowledge about the spectrum would improve the estimation through a better choice of δ_j.

5.5. Conclusion

RFEs with multiple moments are a powerful tool for interior eigenvalue problems, but the repeated solution of large linear systems with many right hand sides is a bottleneck in time and energy. The judicious choice of various parameters has a noticeable effect on the potential overall cost of the solver, particularly those parameters corresponding to the cost of linear system solves. We have explored the heuristic choice of the number of moments, which, while relevant to convergence, may have a less direct effect on convergence than, e.g., the number of quadrature nodes, especially for initial iterations. The choice of the number of moments is a powerful parameter for controlling the cost of linear system solves, and thus the overall eigensolver cost. We explored further heuristics within the context of an iterative eigensolver, including best strategies for choosing the number of moments on a given iteration, and the choice of starting vectors. In a broad comparison between methods, we provide evidence that a multi-moment flexible iterative method may reduce the number of single linear solves over all iterations, and thus the potential overall cost of an eigensolver. Furthermore, the flexibility of the method ensures comparative robustness and accuracy. We provide further evidence for performance capabilities in initial larger experiments.

We have also explored the prediction of the quadrature degree leading to a single RFE iteration with multiple moments, and seen that a good choice here can also reduce the cost of solving linear systems; however, a reliable prediction is not yet assured. Future topics of exploration could include the adaptive choice of quadrature degree in combination with multiple moments, extending the work of [33]. We have seen that a low quadrature degree may not be optimal, but may be less prone to "overshooting" than a high degree. However, this does reduce the scalability of the method, which is improved with a higher degree. Furthermore, the capacity for performance improvement with multiple moments and reduced right hand sides clearly depends on the linear solver used. In these experiments, due to the difficulty of solving these shifted linear systems, we tested with direct solvers. In future work, the capacity for improvement dependant on the properties of the linear solver used may be investigated. Results and exploration for an iterative

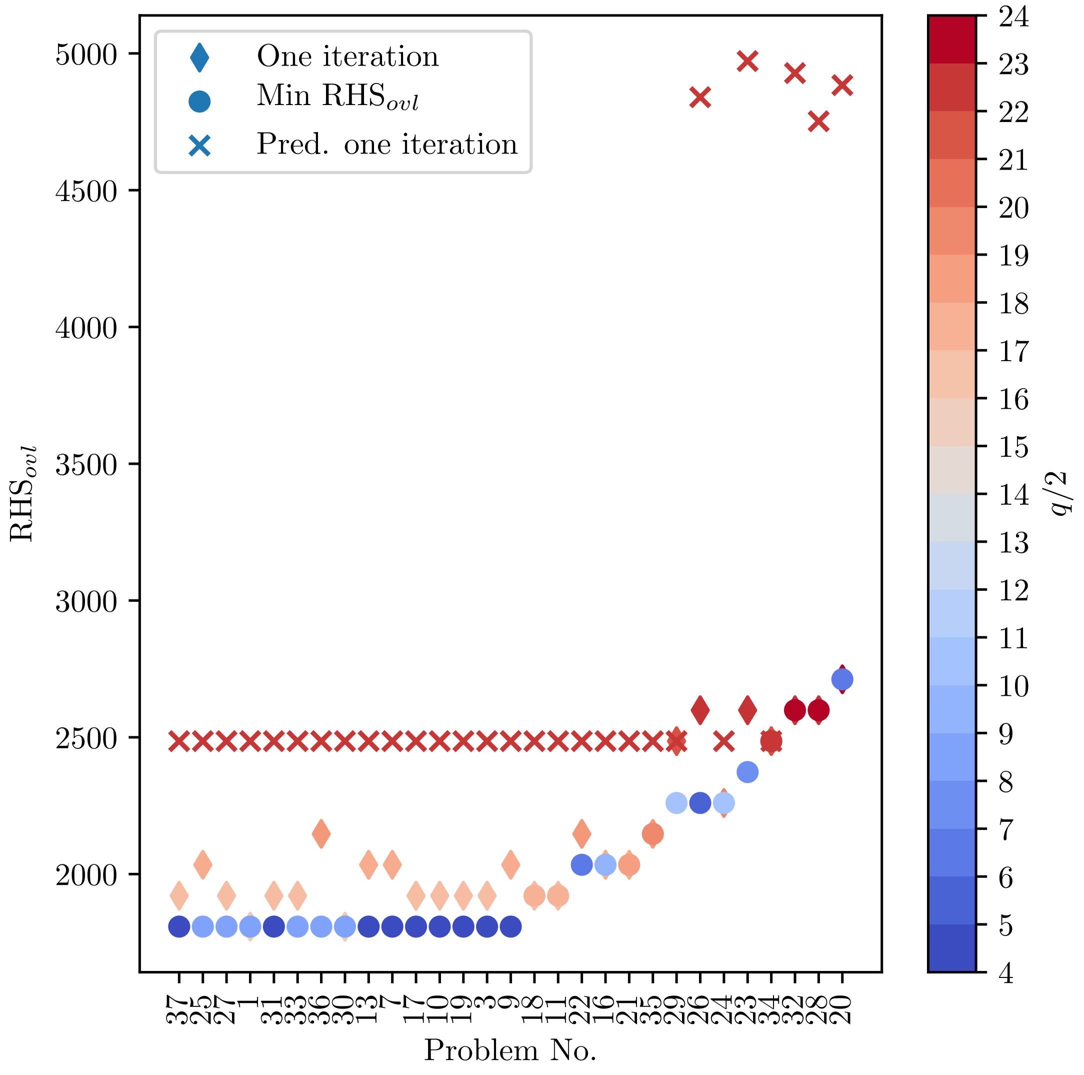


Figure 5.14.: $\mathrm{RHS}_{\mathrm{ovl}}$ of `BEAST-M` with selected values of $q/2$. Shown here are the value of $q/2$ minimizing $\mathrm{RHS}_{\mathrm{ovl}}$ (Min $\mathrm{RHS}_{\mathrm{ovl}}$), the smallest value of $q/2$ such that all desired eigenpairs reach the residual threshold in a single `BEAST-M` iteration (One iteration), and the value of $q/2$ such that convergence is expected in a single `BEAST-M` iteration (Pred. one iteration).

Table 5.5.: $\mathrm{RHS}_{\mathrm{ovl}}$, $q/2$ and iterations corresponding to Figure 5.14. No. corresponds to the problem number listed in Table 5.2. $\mathrm{RHS}_{\mathrm{ovl}}$ is reported for three quadrature degrees: the value of $q/2$ minimizing $\mathrm{RHS}_{\mathrm{ovl}}$ (Min $\mathrm{RHS}_{\mathrm{ovl}}$), the smallest value of $q/2$ such that all desired eigenpairs reach the residual threshold in a single `BEAST-M` iteration (One iteration), and the value of $q/2$ such that convergence is expected in a single `BEAST-M` iteration (Pred. one iteration).

		Min $\mathrm{RHS}_{\mathrm{ovl}}$			One iteration		Pred. one iteration	
Matrix	No.	$q/2$	$\mathrm{RHS}_{\mathrm{ovl}}$	Iter.	$q/2$	$\mathrm{RHS}_{\mathrm{ovl}}$	$q/2$	$\mathrm{RHS}_{\mathrm{ovl}}$
laser	1	8	1,808	2	16	1,808	22	2,486
SiH4	3	4	1,808	4	17	1,921	22	2,486
Pres_Poisson	7	4	1,808	4	18	2,034	22	2,486
Si5H12	9	4	1,808	4	18	2,034	22	2,486
bcsstk37	10	4	1,808	4	17	1,921	22	2,486
bcsstk37	11	17	1,921	1	17	1,921	22	2,486
brainpc2	13	4	1,808	4	18	2,034	22	2,486
SiO	16	9	2,034	2	18	2,034	22	2,486
SiO	17	4	1,808	4	17	1,921	22	2,486
Andrews	18	17	1,921	1	17	1,921	22	2,486
Andrews	19	4	1,808	4	17	1,921	22	2,486
GraI-1k	20	6	2,712	4	24	2,712	22	4,884
GraI-1k	21	18	2,034	1	18	2,034	22	2,486
GraI-11k	22	6	2,034	3	19	2,147	22	2,486
GraI-11k	23	7	2,373	3	23	2,599	22	4,972
GraI-119k	24	10	2,260	2	20	2,260	22	2,486
GraI-119k	25	8	1,808	2	18	2,034	22	2,486
GraII-1k	26	5	2,260	4	23	2,599	22	4,840
GraII-1k	27	8	1,808	2	17	1,921	22	2,486
GraII-11k	28	23	2,599	1	23	2,599	22	4,752
GraII-11k	29	10	2,260	2	22	2,486	22	2,486
GraII-119k	30	8	1,808	2	16	1,808	22	2,486
GraII-119k	31	4	1,808	4	17	1,921	22	2,486
GraIII-1k	32	23	2,599	1	23	2,599	22	4,928
GraIII-1k	33	8	1,808	2	17	1,921	22	2,486
GraIII-11k	34	22	2,486	1	22	2,486	22	2,486
GraIII-11k	35	19	2,147	1	19	2,147	22	2,486
GraIII-119k	36	8	1,808	2	19	2,147	22	2,486
GraIII-119k	37	4	1,808	4	17	1,921	22	2,486

linear solver would be of particular interest as the size of matrices to which these methods are applied continues to grow, and may exceed the capacity of direct solvers.

CONCLUSIONS AND OUTLOOK

This thesis has focused on equipping projected subspace, and in particular rational filter-based eigensolvers for the scalable solution of large and sparse eigenproblems. We have begun by defining the algorithmic framework of RFEs in various forms and described how these methods may be implemented. The rest of the thesis has been concerned with various possibilities for improving the applicability of these methods for efficient computation.

We have shown in Chapter 3 how the precision of computation used may be adjusted without impacting the convergence rate of a subspace iterative scheme. This is a promising option as under the right circumstances, efficiency may be improved without impacting accuracy or robustness. However, the "extra cost" may show up in required user knowledge; a good understanding of the eigenproblems and methods under consideration may be required to avoid stagnation at a lower precision before switching to higher precision in later subspace iterations.

In Chapter 4 we consider how to apply RFEs to larger problems where direct linear solvers become infeasible and we rely on iterative methods for the solution of the linear systems of equations arising from these methods. Though optimization of the rational function for a subspace filtration-based eigensolver has been successfully considered in the past, this is, to the best of our knowledge, a novel result in optimizing the rational function of an RFE with the explicit inclusion of the cost of the iterative linear solver. There is a good deal of potential for further expansion in this area; so far, only two iterative linear solvers have been considered, and only standard eigenvalue problems, but we anticipate that these strategies can be generalized. Improved optimization techniques could also allow for a greater improvement in the overall reduction in cost. Further improvement may be required to make these methods truly applicable for large problems. We anticipate that a certain amount of tuning will continue to be required before using these strategies for a set of eigenproblems.

In Chapter 5, we see that moments are a potential tool for reducing the cost of solving linear systems of equations in an RFE. However, we show in this chapter as well the need for care and flexibility in their use, particularly when the subspace size is reduced to allow this efficiency. We show the potential for flexibility in the number of moments over iterations of an RFE to allow for efficiency and robustness, and highlight the potential for reduction in cost in numerical experiments. We further showcase the scalability of these methods, observing that this improves alongside the number of poles considered in the rational function. Also explored is the choice of quadrature degree with respect to the cost of linear system solves. We see that a high degree that allows the method to converge in a single iteration is often optimal or close to optimal. Using the strategies from Chapter 4, we may be able to predict a reasonably good choice of degree.

When considering future research directions, obvious ideas include exploring the interplay of the different ideas visited in this thesis. Initial explorations with mixed precision and RFEs with multiple moments have not been particularly promising, as especially for larger problems, the convergence threshold for single precision is high enough that stagnation typically sets in too early to make mixed precision calculations efficient. Though we have observed that RFEs with multiple moments are typically more sensitive to a loss in robustness or accuracy, it would be interesting to explore how optimized rational filters could be used with these methods to further reduce the cost of linear solvers.

In general, the largest barrier for the use of RFEs is likely the user knowledge required to use them appropriately and tune them for good performance. Future work to improve "auto–tuning" of these methods would improve the applicability and availability across computational science.

APPENDIX A

SUMMARY OF TEST PROBLEMS

Table A.1.: Sample sparse matrices from graphene modeling [23] and the SuiteSparse Matrix collection [25]. Matrix size, average number of nonzeros per row (nnzr; rounded), and the unscaled spectral range are shown. All problems have been previously considered in [33, 46].

Name	Size	nnzr	$[\lambda_{\min}, \lambda_{\max}]$
laser	3002	3	$[-1.10, 4.25]$
SiH4	5041	34	$[-0.996, 36.8]$
linverse	11999	8	$[-4.70, 15.5]$
Pres_Poisson	14822	48	$[1.28e\text{-}5, 26.0]$
Si5H12	19896	37	$[-0.996, 58.6]$
bcsstk37	25503	45	$[-7.04e\text{-}5, 8.41e7]$
brainpc2	27607	6	$[-2000, 4460]$
rgg_n_2_15_s0	32768	10	$[-5.12, 17.4]$
SiO	33401	39	$[-1.67, 84.3]$
Andrews	60000	13	$[3.64e\text{-}16, 36.5]$
GraI-1k	1152	13	$[-3.43, 2.78]$
GraI-11k	11604	13	$[-3.43, 2.78]$
GraI-119k	119908	13	$[-3.43, 2.78]$
GraII-1k	1152	13	$[-3.43, 2.78]$
GraII-11k	11604	13	$[-3.43, 2.79]$
GraII-119k	119908	13	$[-3.43, 2.79]$
GraIII-1k	1152	12	$[-3.35, 2.73]$
GraIII-11k	11604	12	$[-3.35, 2.73]$
GraIII-119k	119908	13	$[-3.43, 2.78]$

Table A.2.: Test matrices from topological insulator [43] and graphene modelling [23]. The ScaMaC library [7] may be used to generate similar large and sparse matrices, whose spectral properties have been previously studied [76]. Most of these test problems have been previously considered, including in [4, 46]. The size, average nonzeros per row, approximate global spectral bounds, suggested interval with expected number of eigenvalues ($\hat{m}$), and complex status are listed here for each matrix.

Name	Size	nnzr	$[\lambda_{\min}, \lambda_{\max}]$	Interval	$\hat{m}$	Complex?
Graph16M	16000000	4	$[-3.0, 3.0]$	$[-0.0025, 0.0025]$	318	No
Topi1M	1638400	12	$[-4.8, 4.8]$	$[-0.06, 0.06]$	116	Yes
Graph1M	100000	4	$[-3.0, 3.0]$	$[-0.01, 0.01]$	260	No
Graph256	256	4	$[-3.0, 3.0]$	$[-0.5, 0.5]$	16	No

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS & SCRIPTS

LIST OF NOTATIONS

$\mathbb{R}$, $\mathbb{C}$	real and complex numbers
$\overline{a}, A^H$	complex conjugate and Hermitian conjugate
A, B	matrices of eigenvalue problem
n	matrix dimension
I_λ	interval for interior eigenvalue problem
$\underline{\lambda}$, $\overline{\lambda}$	lower and upper bound of I_λ
$\lambda_{\min}$, $\lambda_{\max}$	minimum and maximum eigenvalue of spectrum
h	window function over I_λ
$\tilde{h}$	approximate window function over I_λ
$\mathcal{X}_{I_\lambda}$	eigenspace for eigenvectors with eigenvalues inside I_λ
X	block eigenvectors of (A, B)
X_{I_λ}	block eigenvectors of (A, B) with eigenvalues inside I_λ
$\tilde{X}_{I_\lambda}$	approximate block eigenvectors with eigenvalues inside I_λ
Λ, λ	eigenvalues of (A, B)
I	identity matrix
Y	starting vectors for filtered subspace method
U	constructed basis for $\mathcal{X}_{I_\lambda}$
m	number of eigenvalues inside I_λ
$\tilde{m}$	expected number of eigenvalues inside I_λ
m_U	number of columns of U
m_Y	number of columns of Y
Γ	contour of integration around I_λ
q	degree of rational function
z_j, ω_j	poles and weights of rational function
s	number of moments
$\mathrm{RHS}_{\mathrm{ovl}}$	number of right hand sides in linear systems over all RFE iterations
$\mathrm{BLS}_{\mathrm{ovl}}$	number of block linear systems over all RFE iterations
l	RFE iterations
k	linear solver iterations
Φ	shifted matrix

BIBLIOGRAPHY

[1] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang, *A survey of numerical linear algebra methods utilizing mixed-precision arithmetic*, The International Journal of High Performance Computing Applications, (2021).

[2] L. V. Ahlfors, *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable*, International Series in Pure and Applied Mathematics, McGraw-Hill, New York, 3d ed., 1979.

[3] C. L. Alappat, *Implementation and Performance Engineering of the Kaczmarz Method for Parallel Systems*, Master's thesis, Nov. 2016.

[4] C. L. Alappat, A. Alvermann, A. Basermann, H. Fehske, Y. Futamura, M. Galgon, G. Hager, S. Huber, A. Imakura, M. Kawai, M. Kreutzer, B. Lang, K. Nakajima, M. Röhrig-Zöllner, T. Sakurai, F. Shahzad, J. Thies, and G. Wellein, *ESSEX: Equipping Sparse Solvers For Exascale*, in Software for Exascale Computing - SPPEXA 2016-2019, H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel, eds., vol. 136, Springer International Publishing, Cham, 2020, pp. 143–187.

[5] C. L. Alappat, G. Hager, O. Schenk, J. Thies, A. Basermann, A. R. Bishop, H. Fehske, and G. Wellein, *A Recursive Algebraic Coloring Technique for Hardware-Efficient Symmetric Sparse Matrix-Vector Multiplication*, ACM Transactions on Parallel Computing, 7 (2020), pp. 1–37.

[6] C. L. Alappat and G. Wellein, *SC18:G: RACE - Recursive Algebraic Coloring Engine*, 2019.

[7] A. Alvermann, *A Scalable Matrix Collection.* `https://bitbucket.org/essex/matrixcollection`.

[8] A. Alvermann, A. Basermann, H.-J. Bungartz, C. Carbogno, D. Ernst, H. Fehske, Y. Futamura, M. Galgon, G. Hager, S. Huber, T. Huckle, A. Ida, A. Imakura, M. Kawai, S. Köcher, M. Kreutzer, P. Kus, B. Lang, H. Lederer, V. Manin, A. Marek, K. Nakajima, L. Nemec, K. Reuter, M. Rippl, M. Röhrig-Zöllner, T. Sakurai, M. Scheffler, C. Scheurer, F. Shahzad, D. Simoes Brambila, J. Thies, and G. Wellein, *Benefits from using mixed precision computations in the ELPA-AEO and ESSEX-II eigensolver projects*, Japan Journal of Industrial and Applied Mathematics, 36 (2019), pp. 699–717.

[9] A. Alvermann, A. Basermann, H. Fehske, M. Galgon, G. Hager, M. Kreutzer, L. Krämer, B. Lang, A. Pieper, M. Röhrig-Zöllner, F. Shahzad, J. Thies, and G. Wellein, *ESSEX: Equipping Sparse Solvers for Exascale*, in Euro-Par 2014: Parallel Processing Workshops, L. Lopes, J. Žilinskas, A. Costan, R. G. Cascella, G. Kecskemeti, E. Jeannot, M. Cannataro, L. Ricci, S. Benkner, S. Petit, V. Scarano, J. Gracia, S. Hunold, S. L. Scott, S. Lankes, C. Lengauer, J. Carretero, J. Breitbart, and M. Alexander, eds., Lecture Notes in Computer Science, Cham, 2014, Springer International Publishing, pp. 577–588.

[10] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures*, ACM Transactions on Mathematical Software, 45 (2019), pp. 2:1–2:26.

[11] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[12] J. Asakura, T. Sakurai, H. Tadano, T. Ikegami, and K. Kimura, *A numerical method for nonlinear eigenvalue problems using contour integrals*, JSIAM Letters, 1 (2009), pp. 52–55.

[13] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, *Accelerating scientific computations with mixed precision algorithms*, Computer Physics Communications, 180 (2009), pp. 2526–2533.

[14] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web Page*, 2019. `https://www.mcs.anl.gov/petsc`.

[15] ——, *PETSc Users Manual*, Tech. Rep. ANL-95/11 - Revision 3.14, Argonne National Laboratory, 2020. `https://www.mcs.anl.gov/petsc`.

[16] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries*, in Modern Software Tools for Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhäuser, Boston, MA, 1997, pp. 163–202.

[17] W.-J. Beyn, *An integral method for solving nonlinear eigenvalue problems*, Linear Algebra and its Applications, 436 (2012), pp. 3839–3863.

[18] Å. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[19] J. Brenneck and E. Polizzi, *An Iterative Method for Contour-Based Nonlinear Eigensolvers*, arXiv:2007.03000 [cs, math], (2020). `http://arxiv.org/abs/2007.03000`.

[20] R. L. Burden and D. J. Faires, *Numerical Analysis*, PWS Publishing Company, 1985.

[21] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov, *Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy*, ACM Transactions on Mathematical Software, 34 (2008), pp. 1–22.

[22] A. Buttari, J. Dongarra, J. Langou, J. Langou, P. Luszczek, and J. Kurzak, *Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems*, The International Journal of High Performance Computing Applications, 21 (2007), pp. 457–466.

[23] A. H. Castro Neto, F. Guinea, N. M. R. Peres, K. S. Novoselov, and A. K. Geim, *The electronic properties of graphene*, Reviews of Modern Physics, 81 (2009), pp. 109–162.

[24] W. M. Chan and A. George, *A linear time implementation of the reverse Cuthill-McKee algorithm*, BIT Numerical Mathematics, 20 (1980), pp. 8–14.

[25] T. A. Davis and Y. Hu, *The university of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1:1–1:25.

[26] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal Parallel and Sequential QR and LU Factorizations*, SIAM Journal on Scientific Computing, 34 (2012), pp. A206–A239.

[27] E. Di Napoli, E. Polizzi, and Y. Saad, *Efficient estimation of eigenvalue counts in an interval*, Numerical Linear Algebra with Applications, 23 (2016), pp. 674–692.

[28] R. Feldt, *Robertfeldt/BlackBoxOptim.jl.* `https://github.com/robertfeldt/BlackBoxOptim.jl`, Feb. 2021.

[29] T. Fukaya, Y. Nakatsukasa, Y. Yanagisawa, and Y. Yamamoto, *CholeskyQR2 A simple and communication-avoiding algorithm for computing a tall-skinny QR factorization on a large-scale parallel system*, in https://doi.org/10.1109/ScalA.2014.11, 2014, pp. 31–38.

[30] M. Galgon, *Spectral Projection - Robustness and Orthogonality Considerations*, PhD thesis, Bergische Universität Wuppertal, 2020. `http://elpub.bib.uni-wuppertal.de/servlets/DocumentServlet?id=10670`.

[31] M. Galgon, S. Huber, and B. Lang, *Mixed precision in subspace iteration-based eigensolvers*, PAMM, 18 (2018).

[32] M. Galgon, L. Krämer, and B. Lang, *Counting eigenvalues and improving the integration in the FEAST algorithm*, tech. rep.

[33] ——, *Improving projection-based eigensolvers via adaptive techniques: Adaptive projectors*, Numerical Linear Algebra with Applications, 25 (2018), p. e2124.

[34] M. Galgon, L. Krämer, B. Lang, A. Alvermann, H. Fehske, and A. Pieper, *Improving robustness of the FEAST algorithm and solving eigenvalue problems from graphene nanoribbons*, PAMM, 14 (2014), pp. 821–822.

[35] M. Galgon, L. Krämer, B. Lang, A. Alvermann, H. Fehske, A. Pieper, G. Hager, M. Kreutzer, F. Shahzad, G. Wellein, A. Basermann, M. Röhrig-Zöllner, and J. Thies, *Improved Coefficients for Polynomial Filtering in ESSEX*, in Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, T. Sakurai, S.-L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, and T. Hoshi, eds., vol. 117, Springer International Publishing, Cham, 2017, pp. 63–79.

[36] M. Galgon, L. Krämer, J. Thies, A. Basermann, and B. Lang, *On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues*, Parallel Computing, 49 (2015), pp. 153–163.

[37] B. Gavin, A. Międlar, and E. Polizzi, *FEAST eigensolver for nonlinear eigenvalue problems*, Journal of Computational Science, 27 (2018), pp. 107–117.

[38] B. Gavin and E. Polizzi, *Krylov eigenvalue strategy using the FEAST algorithm with inexact system solves*, Numerical Linear Algebra with Applications, 25 (2018), p. e2188.

[39] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, 3rd ed ed., 1996.

[40] D. Gordon and R. Gordon, *CGMN Revisited: Robust and Efficient Solution of Stiff Linear Systems Derived from Elliptic Partial Differential Equations*, ACM Transactions on Mathematical Software, 35 (2008), pp. 18:1–18:27.

[41] ——, *CARP-CG: A robust and efficient parallel solver for linear systems, applied to strongly convection dominated PDEs*, Parallel Computing, 36 (2010), pp. 495–515.

[42] S. Güttel, E. Polizzi, P. T. P. Tang, and G. Viaud, *Zolotarev Quadrature Rules and Load Balancing for the FEAST Eigensolver*, SIAM Journal on Scientific Computing, 37 (2015), pp. A2100–A2122.

[43] M. Z. Hasan and C. L. Kane, *Colloquium: Topological insulators*, Reviews of Modern Physics, 82 (2010), pp. 3045–3067.

[44] T. Hasegawa, A. Imakura, and T. Sakurai, *Recovering from accuracy deterioration in the contour integral-based eigensolver*, JSIAM Letters, 8 (2016), pp. 1–4.

[45] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Transactions on Mathematical Software, 31 (2005), pp. 397–423.

[46] S. Huber, Y. Futamura, M. Galgon, A. Imakura, B. Lang, and T. Sakurai, *Flexible subspace iteration with moments for an effective contour integration-based eigensolver*, Submitted to Numerical Linear Algebra with Applications, (2020).

[47] S. Huber, Y. Futamura, M. Galgon, B. Lang, and T. Sakurai, *Reducing linear system size with moment based methods in the BEAST framework*, in 31st Advanced Supercomputing Environment (ASE) Seminar, University of Tokyo, Sept. 2017.

[48] ——, *Using the Moment to Reduce Linear System Size*, in 18th SIAM Conference on Parallel Processing for Scientific Computing, Tokyo, Japan, Mar. 2018.

[49] ——, *Contour Integration and Moments for the Solution of Large Eigenproblems*, in 2019 SIAM Conference on Computational Science and Engineering, Spokane, USA, Feb. 24.

[50] S. Huber, M. Galgon, and B. Lang, *Mixed precision in a large iterative parallel eigensolver framework: BEAST*, International Workshop on Eigenvalue Problems: Algorithms; Software and Applications, in Petascale Computing, Mar. 2018.

[51] ——, *Recent developments and results for the BEAST eigensolver*, in R-CCS Cafe, RIKEN, Kobe, Japan, Oct. 2018.

[52] ——, *Using mixed precision in iterative eigensolvers*, in GAMM Annual Meeting, Munich, Germany, Mar. 2018.

[53] S. Huber and B. Lang, *Optimizing rational filters for the scalable solution of large eigenproblems*, in 2021 SIAM Conference on Computational Science and Engineering, Mar. 2021.

[54] IEEE, *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008), (2019), pp. 1–84.

[55] T. Ikegami and T. Sakurai, *Contour Integral Eigensolver for Non-Hermitian Systems: A Rayleigh-Ritz-Type Approach*, Taiwanese Journal of Mathematics, 14 (2010), pp. 825–837.

[56] T. Ikegami, T. Sakurai, and U. Nagashima, *A filter diagonalization for generalized eigenvalue problems based on the Sakurai–Sugiura projection method*, Journal of Computational and Applied Mathematics, 233 (2010), pp. 1927–1936.

[57] A. Imakura, L. Du, and T. Sakurai, *A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems*, Applied Mathematics Letters, 32 (2014), pp. 22–27.

[58] ——, *Error bounds of Rayleigh–Ritz type contour integral-based eigensolver for solving generalized eigenvalue problems*, Numerical Algorithms, 71 (2016), pp. 103–120.

[59] ——, *Relationships among contour integral-based methods for solving generalized eigenvalue problems*, Japan Journal of Industrial and Applied Mathematics, 33 (2016), pp. 721–750.

[60] A. Imakura, Y. Futamura, and T. Sakurai, *An Error Resilience Strategy of a Complex Moment-Based Eigensolver*, in Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, T. Sakurai, S.-L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, and T. Hoshi, eds., vol. 117, Springer International Publishing, Cham, 2017, pp. 1–18.

[61] S. Kaczmarz, *Angenaherte Auflosung von Systemen linearer Gleichungen*, Bulletin International de l'Acad'emie Polonaise des Sciences et des Lettres, (1937).

[62] M. Kawai, A. Ida, and K. Nakajima, *Hierarchical Parallelization of Multicoloring Algorithms for Block IC Preconditioners*, in 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Dec. 2017, pp. 138–145.

[63] J. Kestyn, E. Polizzi, and P. T. Peter Tang, *Feast Eigensolver for Non-Hermitian Problems*, SIAM Journal on Scientific Computing, 38 (2016), pp. S772–S799.

[64] K. Kollnig, P. Bientinesi, and E. Di Napoli, *Rational spectral filters with optimal convergence rate*, arXiv:2001.04184 [cs, math], (2020). `http://arxiv.org/abs/2001.04184`.

[65] L. Krämer, *Integration Based Solvers for Standard and Generalized Hermitian Eigenvalue Problems*, dissertation, Bergische Universität Wuppertal, 2014. `http://elpub.bib.uni-wuppertal.de/servlets/DocumentServlet?id=3982`.

[66] L. Krämer, E. Di Napoli, M. Galgon, B. Lang, and P. Bientinesi, *Dissecting the FEAST algorithm for generalized eigenproblems*, Journal of Computational and Applied Mathematics, 244 (2013), pp. 1–9.

[67] M. Kreutzer, D. Ernst, and C. L. Alappat, *GHOST - General, Hybrid and Optimized Sparse Toolkit*. `https://bitbucket.org/essex/ghost/src/devel/`.

[68] M. Kreutzer, J. Thies, A. Pieper, A. Alvermann, M. Galgon, M. Röhrig-Zöllner, F. Shahzad, A. Basermann, A. R. Bishop, H. Fehske, G. Hager, B. Lang, and G. Wellein, *Performance Engineering and Energy Efficiency of Building Blocks for Large, Sparse Eigenvalue Computations on Heterogeneous Supercomputers*, in Software for Exascale Computing - SPPEXA 2013-2015, H.-J. Bungartz, P. Neumann, and W. E. Nagel, eds., Lecture Notes in Computational Science and Engineering, Cham, 2016, Springer International Publishing, pp. 317–338.

[69] M. Kreutzer, J. Thies, M. Röhrig-Zöllner, A. Pieper, F. Shahzad, M. Galgon, A. Basermann, H. Fehske, G. Hager, and G. Wellein, *GHOST: Building blocks for high performance sparse linear algebra on heterogeneous systems*, International Journal of Parallel Programming, 45 (2017), pp. 1046–1072.

[70] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra, *Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems)*, in ACM/IEEE SC 2006 Conference (SC'06), Tampa, FL, Nov. 2006, IEEE, pp. 50–50.

[71] J. Liesen and P. Tichý, *Convergence analysis of Krylov subspace methods: Convergence analysis of Krylov subspace methods*, GAMM-Mitteilungen, 27 (2004), pp. 153–173.

[72] Y. Maeda, Y. Futamura, A. Imakura, and T. Sakurai, *Filter analysis for the stochastic estimation of eigenvalue counts*, JSIAM Letters, 7 (2015), pp. 53–56.

[73] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, *The ELPA library: Scalable parallel eigenvalue solutions for electronic structure theory and computational science*, Journal of Physics. Condensed Matter: An Institute of Physics Journal, 26 (2014), p. 213201.

[74] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 3.1*, tech. rep., June 2015.

[75] OpenMP Architecture Review Board, *OpenMP Application Programming Interface Version 5.1*. `https://www.openmp.org/specifications/`, Nov. 2020.

[76] A. Pieper, M. Kreutzer, A. Alvermann, M. Galgon, H. Fehske, G. Hager, B. Lang, and G. Wellein, *High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations*, Journal of Computational Physics, 325 (2016), pp. 226–243.

[77] E. Polizzi, *Density-matrix-based algorithm for solving eigenvalue problems*, Physical Review B, 79 (2009), p. 115112.

[78] ——, *FEAST Eigenvalue Solver v4.0 User Guide*, arXiv:2002.04807 [cs], (2020). `http://arxiv.org/abs/2002.04807`.

[79] M. Robbé, M. Sadkane, and A. Spence, *Inexact Inverse Subspace Iteration with Preconditioning Applied to Non-Hermitian Eigenvalue Problems*, SIAM Journal on Matrix Analysis and Applications, 31 (2009), pp. 92–113.

[80] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, *A Distributed-Memory Package for Dense Hierarchically Semi-Separable Matrix Computations Using Randomization*, ACM Transactions on Mathematical Software, 42 (2016), pp. 27:1–27:35.

[81] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, Jan. 2003.

[82] ——, *Numerical Methods for Large Eigenvalue Problems: Revised Edition*, Society for Industrial and Applied Mathematics, Jan. 2011.

[83] T. Sakurai and Y. Futamura, *Personal correspondence.*

[84] T. Sakurai, Y. Futamura, and H. Tadano, *Efficient Parameter Estimation and Implementation of a Contour Integral-Based Eigensolver*, Journal of Algorithms & Computational Technology, 7 (2013), pp. 249–269.

[85] T. Sakurai, P. Kravanja, H. Sugiura, and M. Van Barel, *An error analysis of two related quadrature methods for computing zeros of analytic functions*, Journal of Computational and Applied Mathematics, 152 (2003), pp. 467–480.

[86] T. Sakurai and H. Sugiura, *A projection method for generalized eigenvalue problems using numerical integration*, Journal of Computational and Applied Mathematics, 159 (2003), pp. 119–128.

[87] T. Sakurai and H. Tadano, *CIRR: A Rayleigh-Ritz type method with contour integral for generalized eigenvalue problems*, Hokkaido Mathematical Journal, 36 (2007), pp. 745–757.

[88] A. Stathopoulos and K. Wu, *A Block Orthogonalization Procedure with Constant Synchronization Requirements*, SIAM Journal on Scientific Computing, 23 (2001), pp. 2165–2182.

[89] P. T. P. Tang and E. Polizzi, *FEAST As A Subspace Iteration Eigensolver Accelerated By Approximate Spectral Projection*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 354–390.

[90] J. Thies, M. Galgon, F. Shahzad, A. Alvermann, M. Kreutzer, A. Pieper, M. Röhrig-Zöllner, A. Basermann, H. Fehske, G. Hager, B. Lang, and G. Wellein, *Towards an Exascale Enabled Sparse Solver Repository*, in Software for Exascale Computing - SPPEXA 2013-2015, H.-J. Bungartz, P. Neumann, and W. E. Nagel, eds., Lecture Notes in Computational Science and Engineering, Cham, 2016, Springer International Publishing, pp. 295–316.

[91] J. Thies, M. Röhrig-Zöllner, N. Overmars, A. Basermann, D. Ernst, G. Hager, and G. Wellein, *PHIST: A Pipelined, Hybrid-Parallel Iterative Solver Toolkit*, ACM Transactions on Mathematical Software, 46 (2020), pp. 31:1–31:26.

[92] J. Thies and Röhrig-Zöllner, Melven, *PHIST: A Pipelined, Hybrid-parallel Iterative Solver Toolkit.* `https://bitbucket.org/essex/phist/src/master/`.

[93] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, 1997.

[94] M. Van Barel and P. Kravanja, *Nonlinear eigenvalue problems and contour integrals*, Journal of Computational and Applied Mathematics, 292 (2016), pp. 526–540.

[95] A. Weisse, G. Wellein, A. Alvermann, and H. Fehske, *The Kernel Polynomial Method*, Reviews of Modern Physics, 78 (2006), pp. 275–306.

[96] J. Winkelmann and E. Di Napoli, *Non-linear Least-Squares Optimization of Rational Filters for the Solution of Interior Hermitian Eigenvalue Problems*, Frontiers in Applied Mathematics and Statistics, 5 (2019), p. 5.

[97] Y. Xi and Y. Saad, *A Rational Function Preconditioner For Indefinite Sparse Linear Systems*, SIAM Journal on Scientific Computing, 39 (2017), pp. A1145–A1167.

[98] S. Yokota and T. Sakurai, *A projection method for nonlinear eigenvalue problems using contour integrals*, JSIAM Letters, 5 (2013), pp. 41–44.

[99] E. I. Zolotarev, *Application of elliptic functions to questions of functions deviating least and most from zero*, Zap. Imp. Akad. Nauk. St. Petersburg, 30 (1877), pp. 1–59.