

Computational Challenges in Materials Science

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaft genehmigte Dissertation

vorgelegt von

Jan Winkelmann, Master of Science

aus Darmstadt, Deutschland.

Berichter: Universitätsprofessor Paolo Bientinesi, Ph.D.
Universitätsprofessor Dr. rer. nat. Uwe Naumann
Universitätsprofessor Dr. rer. nat. Bruno Lang

Tag der mündlichen Prüfung: 18.08.2020

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online
verfügbar.

Declaration of Authorship

I, Jan Winkelmann, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I do solemnly swear that:

1. This work was done wholly or mainly while in candidature for the doctoral degree at this faculty and university;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others or myself, this is always clearly attributed;
4. Where I have quoted from the work of others or myself, the source is always given. This thesis is entirely my own work, with the exception of such quotations;
5. I have acknowledged all major sources of assistance;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published before, a list of these publications is listed in [Appendix C](#).

Abstract

This dissertation sets out to improve performance—in terms of runtime as well as accuracy—of Materials Science simulations by means of custom kernels. The approach for each of our use-cases can be summarized as follows: We present some insight into the numerical properties of the simulation method under consideration. Then, we craft a custom numerical kernel that converts our insight into superior performance on high-performance computing systems. Throughout the dissertation we present three numerical kernels:

For the simulation of strongly interacting systems, we derive a new adaptivity criterion for numerical integrals arising within the TUF_{RG} code. We discuss PAID, a high-performance numerical kernel that implements our adaptivity criterion and significantly reduces time-to-solution for the integrals within the TUF_{RG} code.

Some Density Functional Theory simulations require the solution of sequence of Hermitian eigenvalue problems. Subspace Iteration can leverage a kind of correlation present in these sequences into improved convergence. We present CHASE, a high-performance solver for Hermitian eigenvalue problems based on Subspace Iteration equipped with polynomial filtering to further improve convergence. CHASE implements a number of significant improvements to the polynomial filter, resulting in a kernel that excels at solving the Hermitian eigenproblems arising in Density Functional Theory simulations.

Further, we discuss an optimization framework for the rational filters used in the FEAST solver. Our optimization framework, SLiSe, allows for robust and fast generation of rational filters with very specific properties, resulting in convergence improvements to FEAST. What is more, we present an initial concept for the generation of problem-specific rational filters that leverage existing information about the problem at hand.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der in Verbesserung von Materialwissenschaftlichen Simulationen durch spezialisierte Subroutinen, sogenannte Kernel. Für jeden unserer Anwendungsfälle gehen wir folgendermaßen vor. Zuerst präsentieren wir eine relevante Einsicht zu den numerischen Eigenschaften der betrachteten Simulationsmethode. Daraus gestalten wir dann einen speziell angepassten Kernel der unsere Einsicht in die Simulationsmethode nutzt um eine Verbesserung in der Laufzeit oder der Genauigkeit zu realisieren. Wir werden drei solcher Anwendungsfälle sehen:

Im Kontext der Simulation von stark wechselwirkenden Systemen leiten wir ein neues Adaptivitätskriterium für numerische Integrale ab, die im Rahmen des TUfRG Codes entstehen. Wir diskutieren PAID, einen numerischen Kernel zur Nutzung auf Hochleistungsrechnern, der unser Adaptivitätskriterium implementiert und die Lösungszeit für die Integrale des TUfRG Codes deutlich reduziert.

Simulationen in der rechnergestützten Dichtefunktionaltheorie erfordern häufig das Lösen einer Sequenz von Hermiteschen Eigenwertproblemen. Löser auf Basis von Subspace Iteration können eine spezielle Art von Korrelation, die in diesen Sequenzen vorhanden ist, zur Verbesserung ihres Konvergenzverhaltens nutzen. Wir stellen CHASE vor, einen Hochleistungslöser für Hermitesche Eigenwertprobleme, der auf Subspace Iteration basiert und mit einem Polynomfilter ausgestattet ist, um die Konvergenz weiter zu verbessern. CHASE implementiert eine Reihe von signifikanten Verbesserungen des Polynomfilters, was zu einem Kernel führt, der sich zur Lösung der Hermiteschen Eigenwertprobleme, wie sie in Simulationen der Dichtefunktionaltheorie auftreten, anbietet.

Weiterhin diskutieren wir ein Optimierungsframework für die im FEAST Löser verwendeten rationalen Filterfunktionen. Unser Optimierungsframework, SLiSe, erlaubt eine robuste und schnelle Generierung von rationalen Filterfunktionen mit spezifischen Eigenschaften, welche zu Konvergenzverbesserungen in FEAST führen. Darüber hinaus stellen wir ein erstes Konzept zur Gener-

ierung von problemspezifischen rationalen Filterfunktionen vor, welche etwaig vorhandenen Informationen über das vorliegende Eigenwertproblem nutzt.

Acknowledgments

First and foremost, I thank my advisers Edoardo Di Napoli, Ph.D. and Prof. Paolo Bientinesi, Ph.D.; their support made starting—and more importantly finishing—this endeavor possible. Their advice on writing, collaborations, technical matters, and life in general was invaluable throughout my time as a doctoral student, and will stay with me for the rest of my life.

I wish to express my gratitude to a number of people and institutions. First, the members of the HPAC group during my time as a PhD student: Diego Fabregat-Traver; Paul Springer; Elmar Peise; Henrik Barthels; Markus Höhnerbach; Sebastian Achilles; Caterina Fenu; William McDoniel; Mickaël Zehren; and Christos Psarras. Thank you for the good discussions and the great coffee. I would like to acknowledge the Jülich Supercomputing Center (JSC) for hosting me as a member of the SimLab “Quantum Materials”. The JSC sponsored my participation in two “Joint Laboratory for Extreme Scale Computing” (JLESC) Workshops, which were an invaluable experience. I am grateful to the AICES students and the AICES staff, especially Vanessa Schiffers, Nadine Bachem, and Dr. Nicole Faber. Financial support from the Deutsche Forschungsgemeinschaft through grant GSC 111 is gratefully acknowledged.

I would like to extend my thanks to my collaborators: Prof. Blügel and Dr. Daniel Wortmann from the PGI-I at the Jülich Research Centre; Prof. Carsten Honerkamp and Dr. Julian Lichtenstein from the Institute for Theoretical Solid State Physics at the RWTH; and Prof. André Schleife and Xiao Zhang from the University of Illinois at Urbana-Champaign. I would like to thank Prof. Yousef Saad for hosting me during a research stay at the University of Minnesota, and Yuanzhe Xi as well as Vasileos Kalantzis for the discussions during my stay.

Very special thanks goes to Miriam Hinzen for reading my thesis, valuable corrections, even more valuable feedback, and generally putting up with me. Last but not least, I would like to thank my parents, family, and friends for their support; it means more than I can say.

Contents

1	Introduction	1
2	Background and Use-Cases	7
2.1	Quadrature and TUfRG	8
2.1.1	Quadrature	8
2.1.2	Use-Case: TUfRG	12
2.2	Hermitian eigenvalue problems (HEPs) and DFT	15
2.2.1	The HEP	16
2.2.2	Sequences of Correlated Eigenproblems	26
2.2.3	Use-Case: Density Functional Theory	27
3	PAID: An Adaptive Integrator for TUfRG	31
3.1	Adaptive Integration	33
3.1.1	Related Work	37
3.2	Numerical Integration for TUfRG	38
3.3	The PAID Software Package	41
3.4	Numerical Experiments	45
3.5	Summary	50
4	ChASE: An Eigensolver for Sequences of Correlated Hermitian Eigenproblems	51
4.1	Subspace Iteration	53
4.1.1	Filtering via Chebyshev polynomials	57
4.1.2	Related Work	60
4.2	The CHASE Solver	62
4.2.1	Deflation and Locking	65
4.2.2	Filter Degree Optimization	66
4.2.3	Spectral Estimates	74
4.3	Using CHASE	78
4.3.1	Decoupling Kernels From CHASE	78

4.3.2	Available versions of CHASE	82
4.3.3	CHASE Configuration Parameters	84
4.4	Numerical Experiments	86
4.4.1	Sequences	88
4.4.2	Scalability	92
4.5	Summary	97
5	SLiSe: An Optimization Framework for FEAST's Rational Filters	99
5.1	Rational Filters for the HEP	101
5.1.1	Subspace Iteration with Rational Filters	101
5.1.2	Related Work	103
5.1.3	Rational Filters for the Interior HEP	106
5.1.4	Leveraging Symmetries	107
5.1.5	Objective Function	109
5.1.6	Gradient of the Objective Function	112
5.2	Optimizing SLiSe Filters	115
5.2.1	Optimization via Gradient Descent	116
5.2.2	Improving Convergence Speed	119
5.2.3	Systematic Choice of Starting Position	123
5.2.4	The Least-Squares Weights $\mathfrak{G}(t)$	126
5.2.5	Constrained Optimization	131
5.3	Numerical Experiments	137
5.3.1	Replacing the Gauss Filter: γ -SLiSe	138
5.3.2	Replacing the Elliptic Filter: η -SLiSe	143
5.3.3	Breaking Guideline 1: ζ -SLiSe	148
5.3.4	Box Constraint: κ -SLiSe	149
5.4	Optimizing SLiSe Weights	150
5.5	Summary	157
6	Conclusion	159

A Rational Filters	161
B Problem Sets	167
B.1 Obtaining Benchmark Problems From a Matrix	168
C Supercomputers	171
Bibliography	173

*A beginning is the time for taking the most
delicate care that the balances are correct.*

Frank Herbert

1

Introduction

In this dissertation we present numerical kernels that were custom-tailored for our collaborators' simulation methods from the field of Computational Materials Science. On modern supercomputers our kernels consistently out-perform the kernels that were previously used by our collaborators.

Computational Materials Science tackles the discovery of new materials and their properties by means of computer simulations; in this dissertation we specifically focus on quantum physics simulations. Typically, the employed simulation methods are not only very sophisticated but also computationally expensive, making the use of high-performance computing (HPC) resources very common. The computational scientist developing simulation codes must now balance two opposing requirements: On the one hand, there is the continuously rising bar for modern HPC simulation codes: Exascale Readiness, Performance Portability, use of accelerators, and reproducibility of results are just a few of the topics that have become relevant in the last few years. On the other hand, every major code change requires a significant validation effort to ensure the continued correctness of the simulations. We can summarize the Materials Scientist's Challenge as follows: Increase performance and scalability as well as

adopt modern HPC paradigms (e.g., use of GPU accelerators), while touching the simulation code as little as possible in order to reduce validation efforts.

In this thesis, we rise to three of these Challenges, posed to us by our collaborators. Kernels (i.e., numerical routines with a well defined scope) are particularly well suited for our Challenges, as changes to the simulation code are minimal. Accordingly, we consider simulation methods where a large part of the computational cost is spent in repeated calls to a single high-performance kernel. For one of our newly implemented HPC kernels to outperform an existing off-the-shelf kernel, we must identify and leverage the problem structure inherent in the physics of the simulation. Said “problem structure” can be conceptualized as an attribute of the numerical kernel’s input space¹ which we use to enhance performance of the kernel. Rising to these challenges, and the resulting collaboration with Materials Scientists, is the overarching theme of this dissertation.

Contributions

The specific contributions of this dissertation are described in the following:

- **An adaptive quadrature routine for fRG applications:** The functional Renormalization Group (fRG) formalism is used for the simulation of strongly interacting systems, such as superconductors. We consider a particular fRG code, where the most computationally expensive part comprises the element-wise numerical integration of matrices which are then contracted. Our global error criterion yields a more adaptive numerical integration without penalizing scalability, resulting in superior performance.
- **A solver for sequences of correlated Hermitian eigenproblems:** A number of simulation methods require the solution of a sequence of eigenvalue problems. Often, these eigenproblems exhibit a kind of correlation that can be leveraged into increased performance. We improve on a tried-and-true solver that can exploit such correlations: Subspace

¹ For the impatient reader these properties are [Definition 11](#) and [Equations \(3.4\) and \(5.7\)](#).

Iteration with spectral filtering via Chebyshev polynomials. Specifically, we show an improved method for obtaining the spectral estimates required by the polynomial accelerator.

- **Optimized rational filter functions for eigensolvers using spectral projection:** In recent years, so-called “contour (eigen-)solvers” have received a large amount of attention in the community. Typically, one can approach these eigensolvers as employing spectral projection based on rational functions. We present an optimization framework for these rational functions, rooted in an approximation theory ansatz. Our focus is on FEAST, a popular member of the “contour solver” family with support for generalized eigenvalue problems, and our optimized filters enhance the convergence rate of FEAST for a large number of examples. While FEAST can also solve non-Hermitian eigenvalue problems, we focus on rational filter functions for Hermitian ones.

The resulting numerical kernels are:

- PAID, an adaptive quadrature routine with shared memory parallelism. We leverage our global error criterion for fRG codes, which results in PAID’s improved performance by means of enhanced adaptiveness. PAID can be generalized to any element-wise integration of contracted matrices with an infinity norm as error criterion.
- CHASE, a high-performance exterior eigensolver for sequences of correlated Hermitian eigenproblems. CHASE features optimized degrees of the polynomial filter as well as improved methods to obtain spectral estimates required by the filter. Moreover, the CHASE code has a modular design that allows for easy integration into existing codes. Although our focus is on sequences of Hermitian eigenproblems arising from electronic structure calculations, CHASE excels at sequences of correlated eigenvalue problems (see [Definition 11](#)) of any origin, and can even act as a generic Hermitian eigensolver.

1 Introduction

- SLiSe, an optimization framework to obtain rational filters with specific properties for the FEAST solver. Our framework supports constrained optimization for features relevant to Subspace Iteration, for example, box constraints which can improve the condition number of the linear systems solves arising from the application of the rational filter. SLiSe filters significantly out-perform FEAST’s state-of-the-art filters in most usage scenarios. Although focused on optimization of filters for Hermitian eigenproblems, SLiSe can readily be adapted to optimize filters for the non-Hermitian case.

Outline

This dissertation is structured as follows: [Chapter 2](#) lays the groundwork of the subsequent chapters. We introduce numerical integration, or quadrature, for our area of application, the Truncation of Unity functional Renormalization Group (TUfRG) formalism. Then, we introduce the notion of correlated Hermitian eigenvalue problems and continue with a description of Density Functional Theory, our main area of application for the Hermitian eigenproblem.

[Chapter 3](#) treats adaptive quadrature arising in TUfRG, culminating in the PAID code. After some words on adaptive quadrature, we describe our global error criterion and its implications for parallelism. We present PAID’s algorithms and conclude with performance results.

Then we start our discussion of the Hermitian eigenproblems: In [Chapter 4](#) we tackle sequences of Hermitian eigenproblems using Subspace Iteration with polynomial filters. We introduce the Subspace Iteration algorithm, including polynomial filters and parameters. We proceed to improve on the parameters for the filter, first by optimizing the filter degree and then by improving the spectral estimates required by the filter. The chapter concludes with numerical experiments that highlight CHASE’s superior performance and scalability.

[Chapter 5](#) continues with Hermitian eigensolvers, but switches to Subspace Iteration with rational filters. For this chapter, instead of implementing our own solver, we improve on the existing FEAST solver. We present an optimization framework for the rational filters that substantially improves performance in a

number of ways, most importantly the convergence of the Subspace Iteration procedure. The chapter concludes with numerical experiments to substantiate our performance claims and an outlook on our collaborator's work on the optimization framework.

Finally, [Chapter 6](#) concludes this dissertation.

*See, the world is full of things more powerful
than us. But if you know how to catch a
ride, you can go places.*

Neal Stephenson

2

Background and Use-Cases

Materials Science is a large community with various computational challenges. This section elaborates on the specifics of three numerical kernel and use-cases in Materials Science: (1) Adaptive numerical integration for use in the functional Renormalization Group formalism as well as (2) exterior and (3) interior eigenvalue problems for use in Density Functional Theory. These use-cases are born of collaborations with application scientists as part of this thesis, and should not be considered exhaustive. In the following chapters we present substantial optimizations by focusing on problems that are specific enough to allow for such improvements, yet broad enough to benefit a large community of scientists.

Numerical integration is discussed in [Section 2.1](#), as [Chapter 3](#) discusses numerical integration for TUfRG, a kind of functional Renormalization Group formalism. [Section 2.1.1](#) gives a general introduction into the topic, while [Section 2.1.2](#) gives a very quick overview of its use-case, TUfRG.

For the largest part of this work, we discuss Hermitian eigenvalue problems (HEPs) and their application in Density Functional Theory (DFT) applications. HEPs are a particularly rich topic, both with respect to methods as well as applications. We restrict ourselves to two specific variants: The exterior

HEP, where some eigenpairs from one end of the spectrum are sought; and the interior HEP, where eigenpairs in an arbitrary interval (generally inside the spectrum) are sought. An overview of the HEP is given in [Section 2.2.1](#); [Section 2.2.2](#) considers sequences of correlated HEPs. [Section 2.2.3](#) gives a very short overview of DFT. In [Chapters 4](#) and [5](#) we discuss, respectively, exterior and interior HEPs.

Intended audiences of this chapter

This chapter is intended for novices to the field of computational problems; readers familiar with numerical methods may safely skip [Sections 2.1.1](#) and [2.2.1](#). Sequences of correlated eigenproblems, introduced in [Section 2.2.2](#), should be relevant to most readers. The use-cases in [Sections 2.1.2](#) and [2.2.3](#) are meant to be accessible even to people unfamiliar with computational physics. Anyone with basic linear algebra knowledge should find these two sections approachable.

2.1 Quadrature and the Functional Renormalization Group Formalism

We introduce the basics of numerical integration in [Section 2.1.1](#), as well as its use-case in this thesis: the Truncation of Unity functional Renormalization Group (TUfRG) formalism ([Section 2.1.2](#)).

2.1.1 Quadrature

Integration, i.e. determining the area under a curve, is one of the oldest problems in mathematics with innumerable use-cases. We focus on *numerical integration*, or *quadrature*¹, of a definite (Riemann) integral, via a linear combination of

¹ Some people refer to quadrature as numerical integration of exactly two dimensions. Although we focus on the two dimensional case in our use-case, we use the term numerical integration and quadrature interchangeably.

function values of integrand function $f : \mathbb{R} \rightarrow \mathbb{R}$:

$$\int_{-1}^1 f(x)dx \approx \sum_{j=1}^N w_j f(x_j) \quad N \in \mathbb{N}$$

We refer to N as the *order* of the integration rule. In our case the *integration nodes* x_j are constrained to the domain of integration $-1 \leq x_j \leq 1$. The factors $w_j \in \mathbb{R}$ are called the *integration weights*. *Integration rules* $\mathcal{Q}_N$ generate integration nodes and weights for a given order $N \in \mathbb{N}$. To this end we define the rule application:

$$\mathcal{Q}_N(f) = \sum_{j=1}^N w_j f(x_j)$$

For arbitrary domains of integration $\mathcal{D} \subset \mathbb{R}$ we write:

$$\mathcal{Q}_N(f, \mathcal{D}) = \sum_{i=1}^N w_i f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right) \quad \mathcal{D} = [a, b]$$

Occasionally we only write $\mathcal{Q}_N(f)$, when $\mathcal{D}$ is obvious from context. In the following we only consider integration rules that guarantee one or both of the following:

Goal 1 (Convergence)

The integration error converges to zero as N increases in exact arithmetic:

$$\lim_{N \rightarrow \infty} \left| \mathcal{Q}_N(f) - \int_a^b f(x)dx \right| = 0$$

Goal 2 (Polynomial interpolation)

$\mathcal{Q}_N$ integrates all polynomials $\mathbb{P}^m$ up to some degree $m \in \mathbb{N}_+$ without integration error in exact arithmetic:

$$\left| \mathcal{Q}_N(p) - \int_a^b p(x)dx \right| = 0 \quad p \in \mathbb{P}^m$$

Goal 2 is relevant, because we treat integration rules of interpolatory type. “Under the hood” these integration rules interpolate the antiderivative as a

2 Background and Use-Cases

polynomial of degree m . A rule with a high m , with respect to N , approximates smooth functions—those that are sufficiently polynomial-like—particularly well.

We can integrate functions of higher dimension via induction. For example, given a two dimensional integral

$$\int_{-1}^1 \int_{-1}^1 f(x, y) dx dy$$

we set

$$\int_{-1}^1 g(y) dy \quad \text{with: } g(y) = \int_{-1}^1 f(x, y) dx$$

resulting in the double sum

$$\int_{-1}^1 \int_{-1}^1 f(x, y) dx dy \approx \sum_{j=1}^N \sum_{k=1}^N w_j w_k f(x_j, x_k)$$

While this approach applies to arbitrary dimensions, its computational complexity is exponential with respect to the dimension of $\mathcal{D}$. In practice this approach is used for low dimensional integrals, typically 1-5 dimensions. Higher dimensional problems are approached with, for example, Monte Carlo integration. For a very good treatise on all aspects of numerical integration see [23]; in the following we quickly summarize the most relevant aspects.

Newton-Cotes / Trapezoidal

The Newton-Cotes rule $\mathcal{Q}^{NC}$ is our first example of an integration rule. Here, the integration nodes are particularly simple: an equidistant grid along the domain of integration. For a domain of integration of $\mathcal{D} = [-1, 1]$ the integration nodes are given by $x_j = -1 + j \frac{2}{N}, 1 \leq j \leq N$. The formulas for the integration weights are more complex—we omit them in the following for the sake of conciseness. The Newton-Cotes rule of order N , $\mathcal{Q}_N^{NC}$, satisfies [Goal 2](#) as it integrates polynomials up to degree $N - 1$ exactly. However, for large N , the Newton-Cotes rule can become very inaccurate. In some cases the approximation error can grow exponentially in N , due to “Runge’s

phenomenon”: Polynomial interpolation with equidistant nodes x_j causes the interpolant to exhibit oscillations at the interval edges.

One approach to overcome Runge’s phenomenon is the use of a *compound rule*. Instead of integration with a rule $\mathcal{Q}_{NM}$ of order NM , the compound rule $N \times \mathcal{Q}_M$ applies $\mathcal{Q}_M$ to N equally-sized subintervals of the domain of integration. Compound rules do not increase the degree of the polynomial that the integration rule integrates exactly: $N \times \mathcal{Q}_M^{NC}$ is exact for polynomials up to degree $M - 1$. Nevertheless, compound rules are often preferable due to their robustness.

$\mathcal{Q}^{TZ}$, the trapezoidal rule, is based on the Newton-Cotes rule of order 1, $N \times \mathcal{Q}_1^{NC}$. Regarding [Goal 1](#): $\mathcal{Q}^{TZ}$ can be shown to converge geometrically for periodic function on the domain of integration, but offers no such guarantee for non-periodic functions. The trapezoidal rule is a simple, yet popular integration rule for a number of applications [\[42, 43\]](#).

Gauss-Legendre

The Gauss-Legendre rule $\mathcal{Q}^{GL}$, or Gauss rule for short, overcomes some of the shortcomings of the Newton-Cotes rule. The integration nodes are given by the roots of the N^{th} Legendre polynomials². The integration nodes are denser at the ends of the domain of integration than in the middle, and so $\mathcal{Q}^{GL}$ does not suffer from Runge’s phenomenon. The Gauss-Legendre rule, just as the Newton-Cotes one, is an interpolation-based rule, however $\mathcal{Q}_N^{GL}$ is exact for polynomials of degree $2N - 1$, which can be shown to be optimal for the types of integration rules that we consider here.

Regarding [Goal 1](#) the Gauss-Legendre rule exhibits geometric convergence behavior for analytic functions. Due to its good performance with [Goal 1](#) and [Goal 2](#) the Gauss-Legendre rule can be seen as the standard choice for many applications. Generating the nodes and weights for $\mathcal{Q}^{GL}$ is much more complicated than for the Newton-Cotes rule.

² It is possible to derive a separate integration rule with very similar characteristics for each kind of orthogonal polynomial.

Clenshaw-Curtis

In [Chapter 3](#) we encounter a particular problem with the Gauss-Legendre rule: The integration nodes $\{x_j\}_{j=1}^{2N}$ of $\mathcal{Q}_{2N}^{GL}$ are entirely different to the integration nodes of $\mathcal{Q}_N^{GL}$. In [Section 3.1](#) we require a rule where the integration nodes are shared between $\mathcal{Q}_{2N}$ and $\mathcal{Q}_N$, which is the case for the Clenshaw-Curtis rule $\mathcal{Q}^{CC}$.

The integration nodes of $\mathcal{Q}^{CC}$ are given by $\{x_j = \cos \frac{j\pi}{N}\}_{j=1}^N$. Note that the nodes are more densely located at the end of the domain of integration than in the middle, just as with the Gauss-Legendre rule. However, when doubling the order of the Clenshaw-Curtis rule, all N nodes of $\mathcal{Q}_N^{CC}$ reappear in $\mathcal{Q}_{2N}^{CC}$. We omit a discussion of the integration weights for the sake of conciseness.

At first glance the Clenshaw-Curtis rule seems inferior to the Gauss rule. $\mathcal{Q}_N^{CC}$ is exact for polynomials of degrees up to $N - 1$, as opposed to $2N - 1$ for the Gauss rule. The picture is similar with respect to [Goal 1](#): For analytic functions Clenshaw-Curtis converges linearly, in the worst case. From this perspective it would seem that $\mathcal{Q}^{CC}$ is little better than the trapezoidal rule.

In most cases, however, Clenshaw-Curtis performs very similarly to the Gauss rule. Trefethen [\[97\]](#) argues that the two rules often are very comparable in practice. Perhaps surprisingly, the reason for the good performance of the Clenshaw-Curtis rule is related to the polynomial filters of [??](#): $\mathcal{Q}^{CC}$ boils down to an integration of the function's expansion in Chebyshev polynomials. Chebyshev approximations converge quickly for smooth functions and thus integration of the resulting polynomial yields good results.

2.1.2 Use-Case: Truncation of Unity Functional Renormalization Group

This section illustrates the *Truncation of Unity functional Renormalization Group* scheme (TUfRG) [\[64, 63\]](#). We provide a rough overview, with a strong focus on the resulting numerical kernels. Readers interested in the details and a derivation of the method are referred to [\[64\]](#).

The Renormalization Group (RG) is a formidable tool for modeling the

behavior of a physical system at different energy and length scales, such as high-temperature superconductivity or topological states of matter. RG techniques allow the smooth interpolation between well studied models at a given energy scale and complicated emergent phenomena at lower energy scales. At the initial energy scale of our use-case, the physical system is in a well understood metallic state, where weakly correlated electrons interact pairwise through Coulomb repulsion. When lowering the scale, a phase transition may take place at some critical temperature, from which a form of order (e.g., magnetism or superconductivity) can spontaneously emerge.

TUfRG is a kind of RG formalism. From a numerical methods perspective we can approach the TUfRG method as an initial value problem of an ordinary differential equation (ODE). Instead of time we evolve an arbitrary energy scale, the RG *flow*. The flow parameter $\Omega \in \mathbb{R}$ starts at 1 and decreases throughout the flow. We decrease Ω in discrete steps and for each step perform a number of numerical integrations.

For TUfRG, the Quantity of Interest is the two-particle coupling function $\mathbf{V}_\Omega(\mathbf{l})$ at the complex momentum $\mathbf{l} \in \mathbb{C}$ and energy scale Ω . The two-particle coupling function is obtained, in a non-linear fashion, from the following three ordinary differential equations:

$$\dot{\mathbf{P}}(\Omega, \mathbf{l}) = \mathbf{V}^P(\mathbf{l}) \dot{\chi}^{\text{pp}}(\Omega, \mathbf{l}) \mathbf{V}^P(\mathbf{l}) \quad (2.1)$$

$$\dot{\mathbf{C}}(\Omega, \mathbf{l}) = -\mathbf{V}^C(\mathbf{l}) \dot{\chi}^{\text{ph}}(\Omega, \mathbf{l}) \mathbf{V}^C(\mathbf{l}) \quad (2.2)$$

$$\begin{aligned} \dot{\mathbf{D}}(\Omega, \mathbf{l}) = & 2\mathbf{V}^D(\mathbf{l}) \dot{\chi}^{\text{ph}}(\Omega, \mathbf{l}) \mathbf{V}^D(\mathbf{l}) \\ & - \mathbf{V}^C(\mathbf{l}) \dot{\chi}^{\text{ph}}(\Omega, \mathbf{l}) \mathbf{V}^D(\mathbf{l}) \\ & - \mathbf{V}^D(\mathbf{l}) \dot{\chi}^{\text{ph}}(\Omega, \mathbf{l}) \mathbf{V}^C(\mathbf{l}) \end{aligned} \quad (2.3)$$

Here the dot refers to differentiation with respect to Ω . $\mathbf{V}^P, \mathbf{V}^C, \mathbf{V}^D \in \mathbb{C}^{m \times n}$ are the “particle-particle”, “crossed particle-hole”, and “direct particle-hole” couplings. These are assembled from the three left-hand sides at the beginning of each ODE step. The m and n dimensions are given by the “form-factors”, a discretization that is part the TUfRG formalism, with a typical size of

2 Background and Use-Cases

$m \cdot n \approx 350$. Each Ω step in the ODEs requires the solution of each equation with respect to a number of external momenta $\mathbf{l}$, typically between 1 000 and 10 000.

The “susceptibility factor” matrices $\dot{\chi}^{\text{ph}}, \dot{\chi}^{\text{pp}} \in \mathbb{C}^{n \times m}$ are where numerical integration comes into play. For each matrix the $(j, k)^{\text{th}}$ entry $1 \leq j \leq n, 1 \leq k \leq m$ is given by:

$$\dot{\chi}_{j,k}^{\text{pp}}(\Omega, \mathbf{l}) = \int d\mathbf{p} \dot{\psi}_{j,k}^{\text{pp}}(\Omega, \mathbf{l}, \mathbf{p}), \quad (2.4)$$

$$\dot{\chi}_{j,k}^{\text{ph}}(\Omega, \mathbf{l}) = \int d\mathbf{p} \dot{\psi}_{j,k}^{\text{ph}}(\Omega, \mathbf{l}, \mathbf{p}). \quad (2.5)$$

where $\mathbf{p} \in \mathbb{C}$ is an additional complex momentum. Numerical integration of these integrals is the topic of [Chapter 3](#). The real and imaginary parts of $\mathbf{p}$ are integrated between $[-\pi, \pi]$. The integrands are given by:

$$\dot{\psi}_{j,k}^{\text{pp}}(\Omega, \mathbf{l}, \mathbf{p}) = f_j(\mathbf{p})^* f_k(\mathbf{p}) g^{\text{pp}}(\Omega, \mathbf{l}, \mathbf{p}), \quad (2.6)$$

$$\dot{\psi}_{j,k}^{\text{ph}}(\Omega, \mathbf{l}, \mathbf{p}) = f_j(\mathbf{p})^* f_k(\mathbf{p}) g^{\text{ph}}(\Omega, \mathbf{l}, \mathbf{p}). \quad (2.7)$$

Here f are the aforementioned form-factor functions, they are not dependent on Ω . g^{pp} and g^{ph} introduce a dependence on the flow parameter:

$$g^{\text{pp}}(\Omega, \mathbf{l}, \mathbf{p}) = \frac{d}{d\Omega} \int dp_0 G\left(\Omega, p_0, \frac{1}{2} + \mathbf{p}\right) G\left(\Omega, -p_0, \frac{1}{2} - \mathbf{p}\right) \quad (2.8)$$

$$g^{\text{ph}}(\Omega, \mathbf{l}, \mathbf{p}) = \frac{d}{d\Omega} \int dp_0 G\left(\Omega, p_0, \mathbf{p} + \frac{1}{2}\right) G\left(\Omega, p_0, \mathbf{p} - \frac{1}{2}\right) \quad (2.9)$$

Both the derivation with respect to Ω as well as the integral in [Equations \(2.8\)](#) and [\(2.9\)](#) can be computed analytically. For the sake of completeness, G is the “propagator” of the system:

$$G(\Omega, p_0, \mathbf{p}) = \frac{\frac{p_0^2}{p_0^2 + \Omega^2}}{ip_0 - \epsilon(\mathbf{p})}$$

For the purposes of our discussion it suffices to consider the “energy dispersion

function” $\epsilon(\mathbf{p})$ as a computationally cheap, analytical function. Note that poles of $G(\Omega, p_0, \mathbf{p})$ are damped by Ω ; as Ω decreases through the flow of the ODE, we expect G to develop increasingly sharp features.

Equations (2.1) to (2.3) are obtained after a number of simplifying assumptions, including the truncation of unity from which TUF_RG takes its name. Without the truncation of unity, the formula for Equation (2.1) becomes significantly more complex:

$$\mathcal{T}_{pp} = - \int d\mathbf{p} [\partial_{\Omega} \mathbf{G}(\mathbf{p}) \mathbf{G}(k_1 + k_2 - \mathbf{p})] \mathbf{V}(k_1, k_2, \mathbf{p}) \mathbf{V}(k_1 + k_2 - \mathbf{p}, \mathbf{p}, k_3) \quad (2.10)$$

Although the two-particle couplings lose their dependency on $\mathbf{l}$ and $\mathbf{p}$, it is important to realize that in its original form $\mathbf{V}$ is part of the integration.

Example

Using realistic example values, Figure 2.1 shows the integrand $\dot{\psi}_{j,k}^{pp}$ from Equation (2.6) at a high and a low value of Ω . This example illustrates a general characteristic of the integrands: While at high Ω values the $\dot{\psi}$ s are smooth with respect to $\mathbf{p}$, sharp edges and peaks emerge as the energy scale is lowered. Sharp features are much harder to integrate accurately, and as a result the numerical integration becomes significantly more challenging for smaller values of Ω .

Before we proceed to discuss quadrature methods that can efficiently compute integrals with integrands of such varying difficulty in Chapter 3, we provide some background on Hermitian eigenvalue problems as well as its use-case, Density Functional Theory.

2.2 Hermitian eigenvalue problems (HEPs) and Density Functional Theory

This section starts with a discussion of the fundamental properties of the HEP and defines the exterior and interior HEP in Section 2.2.1. In Section 2.2.2

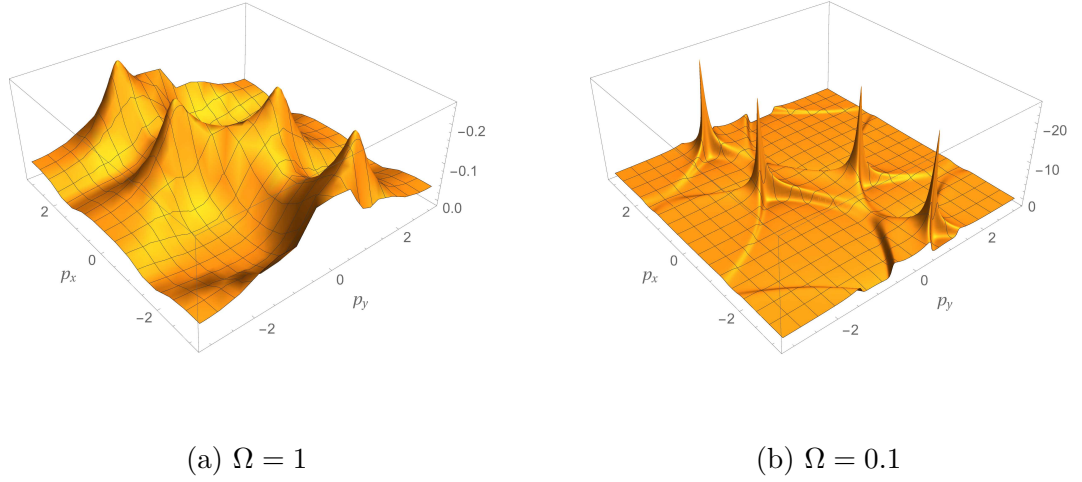


Figure 2.1: An Example Entry of $\dot{\psi}^{\text{pp}}$ Due to the influence of the propagator G , smaller Ω result in much sharper features.
(External momentum $\mathbf{l} = (3.14, 0.78)$)

we continue with sequences of correlated HEPs, while [Section 2.2.3](#) shortly introduces the DFT formalism.

2.2.1 The HEP

In this thesis we exclusively focus on Hermitian eigenvalue problems. A significant number of more general eigenvalue problems do exist, e.g., generalized, non-Hermitian, and non-linear eigenproblems. Solvers for eigenvalue problems are among the most ubiquitous kernels in numerical and computational science, with deep connections to linear algebra, operator theory, and graph theory.

Definition 1 (The Hermitian Eigenvalue Problem, HEP)

Given a Hermitian, or self-adjoint, matrix $M = M^* \in \mathbb{C}^{n \times n}$, i.e. $m_{i,j} = \overline{m_{j,i}}$, the Hermitian eigenvalue problem, or HEP, is the problem of finding pairs $\{(\lambda, v)\}_{i=1}^n$, with $\lambda_i \in \mathbb{C}$, $v_i \in \mathbb{C}^n \setminus \{\mathbf{0}\}$, and $\|v_i\|_2 = 1$ such that:

$$M^*v_i = Mv_i = \lambda_i v_i \quad 1 \leq i \leq n. \quad (2.11)$$

2.2 Hermitian eigenvalue problems (HEPs) and DFT

These pairs (λ, v) are called *eigenpairs* of M , with eigenvalues λ and eigenvectors v .

For each pair (λ, v) , we call v the eigenvector that *corresponds* to λ and vice versa. The set of eigenvalues $\sigma(H) := \{\lambda_i\}_{i=1}^n$ is called the *spectrum*.

HEPs have a number of important properties that we quickly outline in the following:

Remark 1 (Real Eigenvalues [74, Fact 1.2])

All eigenvalues of self-adjoint matrices are real

$$\lambda_i \in \mathbb{R} \quad 1 \leq i \leq n.$$

Remark 2 (Orthogonal Eigenvectors [74, Fact 1.3])

Eigenvectors of M may always be chosen to be pairwise orthogonal

$$\lambda_j \neq \lambda_i \Rightarrow v_j^* v_i = 0 \quad 1 \leq i, j \leq n.$$

Remark 3 (Spectral Theorem [74, Fact 1.4])

M is similar to a diagonal matrix Λ via an orthonormal matrix

$$M = V\Lambda V^* = [v_1, v_2, \dots, v_n] \text{Diag}(\lambda_1, \lambda_2, \dots, \lambda_n) [v_1, v_2, \dots, v_n]^* = \sum_{i=1}^n \lambda_i v_i v_i^*,$$

$$\mathbb{I} = VV^*.$$

As a result of these remarks, we can order the eigenvalues and corresponding eigenvectors

$$\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n. \quad (2.12)$$

The pair (Λ, V) is also called the *eigenvalue decomposition* or *spectral decomposition*. In some cases we refer to collections of eigenpairs, as opposed to a complete spectral decomposition, as a pair matrices: $(\hat{\Lambda}, \hat{V}) := (\text{Diag}(\lambda_1, \dots, \lambda_j), [v_1, \dots, v_j])$ with $j \leq n$.

The spectral theorem already encapsulates much of the theoretical underpinnings that we need to start solving HEPs. Indeed, the eager reader may

2 Background and Use-Cases

already skip to the next section for an algorithm to solve a kind of HEP. We continue this part with additional definitions and remarks that turn out to be useful throughout this thesis.

Definition 2 (Spectral Projector [74, Definition 1.4.1])

A matrix E is a projector if $E^2 = E$. It is an orthogonal projector if $Ey = 0$ for all y orthogonal to $\text{span}\{E\}$. The condition for this is simply

$$E^* = E.$$

For any square matrix M the spectral projector E_λ for an eigenvalue λ satisfies:

$$ME_\lambda = E_\lambda M = \lambda E_\lambda.$$

Any subspace $\tilde{V} \subseteq \mathbb{C}^{n \times n}$ which is invariant under M , i.e. $M\tilde{V} \subseteq \tilde{V}$, is just the span of appropriate eigenvectors. Chapters 4 and 5 rely on the fact that a cheap approximation to the spectral projector can accelerate the computation of the eigenvalues and eigenvectors.

Definition 3 (Normal Matrix)

The matrix $M \in \mathbb{C}^{n \times n}$ is normal if and only if it commutes with its conjugate transpose

$$MM^* = M^*M.$$

All Hermitian matrices are normal. Also *shifted* normal matrices $(M - \alpha\mathbb{I})$, $\alpha \in \mathbb{C}$ remain normal.

Remark 4 (Schur Decomposition)

$M \in \mathbb{C}^{n \times n}$ is normal if and only if there exists a unitary $V \in \mathbb{C}^{n \times n}$ such that $V^*MV = \text{Diag}(\lambda_1, \dots, \lambda_n)$. The following holds for normal matrices: $|\lambda_i| = \sigma_i$, where σ_i is the i^{th} singular value.

Now that we have collected a number of definitions and properties of the HEP we are still no closer to actually *computing* eigenvalues or eigenvectors.

2.2 Hermitian eigenvalue problems (HEPs) and DFT

Algorithms that solve the HEP and compute a (partial) spectral decomposition³ are called *eigenvalue solvers*, or *eigensolvers*.

We define the following goals for an eigendecomposition of a Hermitian matrix $V^* \Lambda V = M \in \mathbb{C}^{n \times n}$, based on the Spectral Theorem.

Goal 3 (Relative Residual)

$$\frac{\|\tilde{\Lambda} - \tilde{V}^* M \tilde{V}\|_{\star}}{\|M\|_{\star}} \stackrel{!}{\approx} n\varepsilon$$

and for the eigenvectors V

Goal 4 (Orthogonality)

$$\|I - \tilde{V}^* \tilde{V}\|_{\star} \stackrel{!}{\approx} n\varepsilon,$$

where ε is the machine epsilon and $\|\cdot\|_{\star}$ is some matrix norm; we use the spectral $\|\cdot\|_2$ norm, unless otherwise specified. A good eigensolver achieves the desired relative residual and orthogonality for most matrices. For the eigensolvers that we consider here, orthogonality of the solution will usually be less of a problem than the residual. In some applications we use the relative residual, normalized by the norm of the matrix, and in other cases the absolute residual $\|\tilde{\Lambda} - \tilde{V}^* H \tilde{V}\|_{\star}$. We can generally distinguish two different kinds of algorithms that solve the HEP.

Direct solvers transform the matrix M into a banded form—most often tridiagonal—via a similarity transform. Then the tridiagonal eigenproblem is solved with a specialized solver, and the resulting eigenvectors are back-transformed via the inverse similarity transform. An example is LAPACK’s ZHEEV, which uses the Method of Multiple Relatively Robust Representations as the tridiagonal solver. Most direct solvers cannot calculate only a part of the eigendecomposition at reduced cost and entirely lack support for reduced accuracy of the solution⁴.

³ We do not consider eigensolvers that exclusively compute eigenvalues although these algorithms exist, see e.g., DQDS [32]

⁴ We consider a relative residual and orthogonality in the order of $n\varepsilon$ to be of “full accuracy”.

2 Background and Use-Cases

Iterative solvers, on the other hand, solve the HEP by multiplying the matrix M with vectors. In all of these methods a candidate solution is refined in each step of the iterative process. Computing the entire spectrum to full accuracy with iterative solvers is usually more computationally expensive than with direct solvers. There are many different approaches to iterative solvers: the family of Krylov and Jacobi-Davidson methods, to name just a few. Particularly relevant for this thesis are Power Iteration and Subspace Iteration, a generalization of Power Iteration. In the following section we discuss Power Iteration, and consider Subspace Iteration in [Section 2.2.1](#).

Exterior HEP Many scientific applications do not require the entire eigendecomposition. To this end, the *exterior HEP* focuses on calculating the eigendecomposition at one extreme of the spectrum, at lower cost than calculating the entire eigendecomposition. For the purpose of this work we define the exterior HEP as the problem of finding the smallest `nev` eigenpairs:

Definition 4 (Exterior HEP (canonical))

Given $M = M^* \in \mathbb{C}^{n \times n}$ and $nev \in \mathbb{N}$ find the first *nev* eigenpairs $\{(\lambda_i, v_i)\}_{i=1}^{nev}$

$$Mv_i = \lambda_i v_i \quad 1 \leq i \leq nev$$

using the ordering of [Equation \(2.12\)](#).

Here—and everywhere else in this thesis, unless explicitly specified—we implicitly use the ordering of [Equation \(2.12\)](#), which ensures that we obtain the smallest eigenvalues. Similarly, we write parameters to problems and their algorithms, such as `nev`, like [this](#). In the following we discuss: (1) Alternative definitions of the exterior HEP and (2) Power Iteration as an example solution method.

Depending on the use-case of the scientific application, there are other approaches to formulating the exterior HEP. Instead of seeking the smallest `nev` eigenvalues, some applications require all eigenvalues smaller than a certain threshold τ :

Definition 5 (Exterior HEP, τ -variant)

Given $M = M^* \in \mathbb{C}^{n \times n}$ and $\tau \in \mathbb{R}$ find all eigenpairs (λ, v) that satisfy

$$Mv = \lambda v \quad \forall \lambda \in (-\infty, \tau].$$

This formulation comes in handy with the definition of the interior HEP, which we consider next. Both exterior HEP variations, **nev** and τ -based, can be reduced to one another by setting $\tau = \lambda_{\text{nev}}$. To perform the reduction in real world settings λ_{nev} must be estimated in some way, see [Section 4.2.3](#) for an example. Further, the **nev** and τ -based definitions can be reformulated to seek the largest eigenvalues, instead of the smallest.

For the sake of completeness, we consider a last problem formulation:

Definition 6 (Exterior HEP, modulus-based)

Given $M = M^* \in \mathbb{C}^{n \times n}$, $\text{nev} \in \mathbb{N}$, and the ordering

$$|\lambda_n| \leq |\lambda_{n-1}| \leq \dots \leq |\lambda_2| \leq |\lambda_1|$$

find:

$$Mv_i = \lambda_i v_i \quad 1 \leq i \leq \text{nev}$$

The first i eigenvalues in this ordering are referred to as the i *dominant* eigenvalues. Note that we do not consider the problem of finding the **nev** eigenvalues of *smallest* modulus as a variation of the exterior HEP.

Example: Power Iteration

One of the simpler iterative methods for tackling the exterior HEP is *Power Iteration*. In its original form Power Iteration solves the modulus-based exterior HEP from [Definition 6](#) for a **nev** = 1.

Power Iteration relies on the fact that successive multiplication of a random independent identically distributed (i.i.d) unit vector q with the matrix M converges q to v_1 (up to sign). Once we obtain v_1 we can obtain λ_1 using the Rayleigh quotient:

2 Background and Use-Cases

```

1: function POWERITERATION(Hermitian Matrix  $M \in \mathbb{C}^{n \times n}$ ,
                           Random i.i.d Unit Vector  $q^{(0)} \in \mathbb{C}^n$ ,
                           Residual Tolerance  $\text{tol}$ )
2:   for  $k = 1, 2, \dots$  do
3:      $z^{(k)} = Mq^{(k-1)}$ 
4:      $q^{(k)} = \frac{z^{(k)}}{\|z^{(k)}\|}$ 
5:      $\lambda^{(k)} = (q^{(k)})^* M q^{(k)}$ 
6:     if  $\|Mq^{(k)} - \lambda^{(k)}q^{(k)}\|/\|M\| < \text{tol}$  then
7:       return  $(\lambda^{(k)}, q^{(k)})$ 

```

Algorithm 1: The Power Iteration Algorithm

Definition 7 (Rayleigh quotient)

For a Hermitian matrix $M \in \mathbb{C}^{n \times n}$ and a vector $\hat{v} \in \mathbb{C}^n \setminus \{0\}$ the Rayleigh quotient is defined as:

$$r_M(\hat{v}) := \frac{\hat{v}^* M \hat{v}}{\hat{v}^* \hat{v}}$$

For an eigenvector v_j the Rayleigh quotient $r_M(v_j)$ results in the corresponding eigenvalue λ_j .

The full Power Iteration algorithm is provided in [Algorithm 1](#). [Line 3](#) and [Line 4](#) perform the aforementioned multiplication with M and normalize the vector q . [Line 5](#) forms the Rayleigh quotient in order to obtain the approximate eigenvalue corresponding to q . Finally, we check the relative residual of (λ, q) and return if the residual passes the comparison; otherwise we continue to the top of the loop.

To understand the workings of the Power Iteration method we expand q in the eigenvector basis $V = [v_1, \dots, v_n]$ of M , with expansion coefficients c_i :

$$q^{(0)} = \sum_{i=1}^n c_i v_i$$

then successive multiplication with M yields:

$$M^k q^{(0)} = M^k \sum_{i=1}^n c_i v_i = \sum_{i=1}^n \lambda_i^k c_i v_i = c_1 \lambda_1^k \left(v_1 + \sum_{j=2}^n \frac{c_j}{c_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k v_j \right) \quad (2.13)$$

2.2 Hermitian eigenvalue problems (HEPs) and DFT

With $q^{(0)}$ chosen as a i.i.d random vector, the probability of $c_1 \neq 0$ is 1. The situation in [Line 3](#) is very similar. As $q^{(k)} \in \text{span}\{M^k q^{(0)}\}$ we conclude

$$\text{dist}\left(\text{span}\{q^{(k)}\}, \text{span}\{v_1\}\right) = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

It follows that the convergence of the largest eigenvector depends on the distance between the largest and second-largest eigenvalues. This dependence of the convergence on the spectral gap is central to our discussion in [Chapters 4 and 5](#). Creating and leveraging spectral gaps by means of appropriate projection methods is a central part of this thesis.

We will delve into more detail on *Subspace Iteration*, a generalization to the Power Iteration for multiple vectors, in [Section 4.1](#). For now we will focus on the interior HEP, a variant of the HEP that is generally more involved as well as computationally more demanding.

Interior HEP Consider the more general case of finding all eigenvalues inside an interval:

Definition 8 (Interior HEP)

Given $M = M^* \in \mathbb{C}^{n \times n}$ and $\mathcal{I} \subset \mathbb{R}$ with $\mathcal{I} \cap \sigma(M) \neq \emptyset$ find all $\mathbf{m}$ eigenpairs (λ, v) :

$$Mv = \lambda v \quad \forall \lambda \in \mathcal{I}$$

In the context of the interior HEP we denote the—a priori unknown—number of eigenvalues inside $\mathcal{I}$ by $\mathbf{m}$. In contrast to the exterior HEP we do not consider the problem of finding eigenvalues by index for the interior HEP.

Our motivations for posing the problem are as follows: First, some of the methods that we discuss can be extended to non-Hermitian and non-linear eigensolvers, where the only feasible definition of exterior is based on the modulus. With such a strong restriction on the definition of the exterior eigenproblem, many applications must fall back to the more general interior formulation. Second, it is possible to decompose an interior HEP into multiple, independent interior HEPs by partitioning the interval: $\mathcal{I} = \bigcup \mathcal{I}_j$. This

2 Background and Use-Cases

technique is called *spectrum slicing*, and each interval $\mathcal{I}_j$ is then called a *slice* of the spectrum of M . The resulting additional parallelism is particularly important for High-Performance Computing scenarios. Finally, the interior HEP is more general than the exterior HEP, by setting $\mathcal{I} = (-\infty, \tau)$ we obtain an exterior HEP.

From a computational perspective, the interior HEP tends to be much more expensive than the exterior HEP. Simple methods, such as Power Iteration, cannot compute interior eigenvalues without significant modifications to the algorithm. We can distinguish between two common approaches for interior HEP solvers: shift-and-invert and projection methods. Shift-and-invert methods rely on the fact that eigenvalues of M closest to some shift $\psi \in \mathbb{R}$ become the largest eigenvalues of the shifted and inverted $(M - \psi\mathbb{I})^{-1}$. Example methods include Inverse Iteration, a variation of the Power Iteration method that operates on a shifted and inverted M and the Jacobi-Davidson method. For the sake of numerical stability the shifted M is not formed explicitly but rather substituted with a linear system solve. Projection methods usually operate on a collection of approximate eigenvectors $\tilde{V}$ and aim to “filter out” components of eigenvectors corresponding to unwanted eigenvalues via some (more or less cheaply computable) projection operator. The spectral projector E_λ , see [Definition 2](#), is the perfect projector for the eigenvector that corresponds to λ ⁵.

In [Chapter 5](#) we focus on a family of eigensolvers based on approximations of the spectral projector. If Γ is a simple closed curve enclosing only one eigenvalue λ in the complex plane, then the spectral projector is given by [\[55, p. 558\]](#):

$$E_\lambda = \frac{1}{2\pi i} \int_{\Gamma} (z\mathbb{I} - M)^{-1} dz \quad (2.14)$$

More generally, when Γ encloses more eigenvalues we obtain a projector onto the space spanned by all eigenvectors corresponding to eigenvalues enclosed by Γ . The function $R(z; M) = (z\mathbb{I} - M)^{-1}$ is called the matrix resolvent, or

⁵ This distinction is somewhat arbitrary. FEAST with iterative solvers for the linear systems is both: a projection method as well as a Krylov-type method [\[36\]](#).

2.2 Hermitian eigenvalue problems (HEPs) and DFT

resolvent operator. There is a convenient connection between [Equation \(2.14\)](#) and matrix functions in general. We can define a matrix function in terms of a scalar valued function and the resolvent operator:

Definition 9 (Matrix Function [44, Definition 1.11])

For $M \in \mathbb{C}^{n \times n}$,

$$f(M) := \frac{1}{2\pi i} \int_{\Gamma} f(z)(z\mathbb{I} - M)^{-1} dz \quad (2.15)$$

where f is analytic on and inside a closed contour Γ that encloses $\sigma(M)$.

A maybe more intuitive way to define matrix functions is via its effect on the eigenvalues of the matrix:

Definition 10 (Matrix Function [44, Definition 1.2])

Let f be defined on the spectrum of $M \in \mathbb{C}^{n \times n}$ and let M have the spectral decomposition $V \Lambda V^*$. Then

$$f(M) := V \text{Diag}(f(\lambda_1), f(\lambda_2), \dots, f(\lambda_n)) V^* \quad (2.16)$$

As a result of the two definitions we can alter the eigenvalues of a matrix without changing the eigenvectors by means of an integration procedure.

[Chapter 5](#) aims to construct matrix functions f that (cheaply) approximate spectral projectors with a special focus on Subspace Iteration. [Section 2.1](#) discusses numerical integration, a straight-forward approach to obtain such an approximate spectral projector from [Equation \(2.14\)](#).

[Chapters 4](#) and [5](#) consider polynomial and rational approximations of the spectral projector that act as spectral filters, increasing convergence speed for certain algorithms to be discussed later on. We do not choose polynomials and rational functions because they are particularly fit to act as basis functions for spectral filters; our choice is motivated by what we can cheaply apply to matrices. Matrix polynomials can be evaluated via matrix-matrix multiplication, and rational matrix functions can be evaluated by means of linear system solves. On the other hand, matrix exponentials, for example, which could be used to create Gaussians are non-trivial to apply as a matrix function at low cost.

2.2.2 Sequences of Correlated Eigenproblems

Consider the following definition.

Definition 11 (Sequence of **nev-vector correlated eigenproblems [37])**

A sequence of Hermitian matrices $(M^{(\ell)})_{\ell \in \mathbb{N}}$ is **nev**-vector correlated if and only if $\mathbf{nev} \leq n$ and for all ℓ there exists a non-negative constant $\delta^{(\ell)}$, with $\delta^{(\ell)} \leq \delta^{(\ell-1)}$ and $\delta^{(1)} < 1$, such that the following holds for the smallest **nev** eigenvectors $Y^{(\ell)} = [v_1^{(\ell)}, \dots, v_{\mathbf{nev}}^{(\ell)}] \subset V^{(\ell)}$ of $(M^{(\ell)})_{\ell \in \mathbb{N}}$

$$1 - \cos \angle \left(v_{i_k}^{(\ell)}, v_{j_k}^{(\ell-1)} \right) < \delta^{(\ell)} \quad k = 1, \dots, \mathbf{nev}$$

where the indices i_k and j_k are given by

$$\cos \angle \left(v_{i_k}^{(\ell)}, v_{j_k}^{(\ell-1)} \right) := \max_{\substack{a \neq i_1, \dots, i_{k-1} \\ b \neq j_1, \dots, j_{k-1}}} \left[\cos \angle \left(v_a^{(\ell)}, v_b^{(\ell-1)} \right) \right]$$

For the sake of simplicity, we drop the “**nev**-vector” prefix and write *sequences of correlated eigenproblems*. Our definition of correlation is based on the definition of canonical correlations which are also known as the cosine of principal angles between subspaces [46] of the space spanned by $V^{(\ell)}$. When dealing with a sequence of correlated eigenproblems the angles get smaller as one travels along the sequence, implying that the corresponding subspaces become increasingly aligned.

Simply put, we consider sequences where the invariant subspaces $Y^{(\ell-1)}$ and $Y^{(\ell)}$ spanned by the smallest **nev** eigenvectors of adjacent matrices $M^{(\ell-1)}$ and $M^{(\ell)}$ are very similar. One of the reasons for the definition’s complexity is due to the fact that the definition does not make assumptions about the eigenvalues of the matrices $(M^{(\ell)})_{\ell \in \mathbb{N}}$. For example, the eigenvectors $[v_1^{(\ell)}, \dots, v_{\mathbf{nev}}^{(\ell)}]$ may change their order throughout the sequence as their corresponding eigenvalues change. It follows that there is not necessarily an easily discernable similarity in the matrix entries of correlated eigenproblems. Note $1 - \cos \angle \left(v_{i_{\mathbf{nev}}}^{(\ell)}, v_{j_{\mathbf{nev}}}^{(\ell-1)} \right)$ does not have to be non-increasing as long as it can be bound by a monotonically

non-increasing $\delta^{(\ell)}$. Indeed, the angles can fluctuate quite a bit depending on the process generating the sequence.

These kinds of correlations occur in many scientific computing applications, e.g., condensed matter [56], thermoacustics [90], and optoelectronics [99]. What is more, correlated sequences also emerge from numerical methods, such as the solution of non-linear over-damped Hermitian eigenvalue problems $T(\Lambda, Y)Y = 0$ where $T(z)$ is a non-polynomial function of z . These problems are guaranteed to have n real eigenvalues [82], and, when solved by successive linearization [83] or safeguarded iteration [100], can result in long sequences of correlated HEPs.

Throughout Chapters 4 and 5 we focus on sequences of correlated eigenvalue problems. Indeed, both chapters deal with variations of the Subspace Iteration method, which can easily leverage sequences of correlated eigenvalue problems to improve time-to-solution.

2.2.3 Use-Case: Density Functional Theory

Our main application for the HEP is Density Functional Theory (DFT), the de-facto standard in atomistic condensed matter computations [56]. Without going too much into the physics, electronic structure is famously described by the Schrödinger partial differential equation. From a computational point of view the many-body problem resulting from a direct solution of Schrödinger's Equation is intractable. In its stead computational scientists have developed a number of tractable alternatives, including the Hartree-Fock and DFT formalism.

DFT is based on the theorem of Hohenberg and Kohn [45] around the density function $\mathbf{n}(x, y, z)$ of the spacial coordinates x, y, z . Underlying the theory are a number of simplifying assumptions. The crucial aspect of DFT is, that it casts the many-body problem as a computationally more tractable single-body problem. What is more, DFT employs the Born-Oppenheimer approximation, assuming that atom nuclei move on a slower, separable time scale than the electrons.

The approach results in a non-linear eigenvalue problem $[T(\Lambda, Y) - \Lambda \mathbb{I}] Y = \mathbf{0}$, where T is a (non-linear) matrix function. A popular approach for this class of

2 Background and Use-Cases

problems is the use of a self-consistency scheme: An initial guess for the eigenpairs $(\Lambda^{(0)}, Y^{(0)})$ results in a *Hamiltonian* $H^{(1)} := T(\Lambda^{(0)}, Y^{(0)})$, that gives rise to a HEP: The eigenpairs $(\Lambda^{(1)}, Y^{(1)})$ of $H^{(1)}$ are then used to obtain a new Hamiltonian $H^{(2)}$ in the hope of eventually converging to (Λ, Y) , the solution of the non-linear problem. At each iteration ℓ a HEP $T(\Lambda^{(\ell-1)}, Y^{(\ell-1)})Y^{(\ell)} = \Lambda^{(\ell)}Y^{(\ell)}$ is solved. This sequence $(T(\Lambda^{(\ell)}, Y^{(\ell)}))_{\ell \in \mathbb{N}}$ is a classic example of a sequence of correlated eigenproblems.

Eigenstates of the non-linear operator $T(\Lambda, Y)$ describe molecular orbitals, that may be occupied by an electron. In the typical case of non-excited—so called ground-state—calculations, only the eigenstates corresponding to the lowest eigenvalues are occupied. The charge density, our Quantity of Interest, is obtained by summing over the eigenvectors corresponding to the lowest eigenvalues. As a result, DFT applications typically result in sequences of correlated *exterior* eigenvalue problems⁶.

There is a plethora of different DFT methods, too many to discuss in this work. From the perspective of the resulting HEP, we are interested in the size and structure of the Hamiltonians, and whether the resulting sequence of eigenproblems is correlated or not. As part of [Chapter 4](#) we specifically consider a representative DFT method, namely the FLEUR [\[106\]](#) code, based on a variation of the plane-wave method, called the Full Potential Linearized Augmented Plane Wave (FLAPW) method. The individual Hamiltonians are dense, resulting in sequences of correlated exterior HEPs with `nev` typically between $\frac{n}{20}$ and $\frac{n}{4}$. For [Chapter 5](#) we implicitly consider the NESSIE [\[110\]](#) DFT code, as it is authored by the same group as FEAST, the solver that we improve in the chapter. NESSIE uses a mesh-based approach, resulting in correlated sequences of sparse HEPs. Contrary to the eigenproblems arising from FLEUR, NESSIE typically slices the exterior HEP into multiple interior HEPs, due to the size of the Hamiltonians.

The properties of the Hamiltonians depend not only on the specific DFT

⁶ Most DFT methods produce generalized Hermitian eigenvalue problems, $HV = \Lambda BV$ with B Hermitian positive definite, but we ignore this detail as these problems can be reduced to (standard) HEPs.

2.2 Hermitian eigenvalue problems (HEPs) and DFT

method but also the physical system under investigation. For our purposes, a very relevant feature of the Hamiltonian is the existence of a spectral gap around the n th eigenvalue: Methods based on Subspace Iteration can be shown to converge much faster in the presence of such a gap than in its absence. In the electronic structure community these gaps are called HOMO-LUMO gaps—**h**ighest **o**ccupied **m**olecular **o**rbital and **l**owest **u**noccupied **m**olecular **o**rbital—and are typically present in insulating materials. Conversely, conducting materials lack a HOMO-LUMO gap and tend to converge more slowly with Subspace Iteration methods.

*Wo rohe Kräfte sinnlos walten,
Da kann sich kein Gebild gestalten.*

Friedrich Schiller

3

PAID:

An Adaptive Integrator for TUfRG

In this chapter we examine the computation of the susceptibility factors $\dot{\chi}^{\text{ph}}$ and $\dot{\chi}^{\text{pp}}$, see [Equations \(2.4\)](#) and [\(2.5\)](#), via numerical integration. We present the synthesis of TUfRG with a modern numerical integration method, in order to create a tailor-made numerical integration kernel for the susceptibility factors. The crux lies in adapting the numerical integration to an error criterion based on not a single integral, but on the entirety of $\dot{\chi}^{\text{pp}}$ and $\dot{\chi}^{\text{ph}}$. Our approach requires fewer operations to compute the susceptibility factors at the cost of a more complex implementation, as compared to the naïve approach of treating each integral separately.

Before we delve into the details, let us take a step back and summarize the problem in a structured manner. For the sake of simplicity, we focus on numerical integration and disregard details born from the TUfRG use-case wherever possible. We summarize the problem of computing the susceptibility

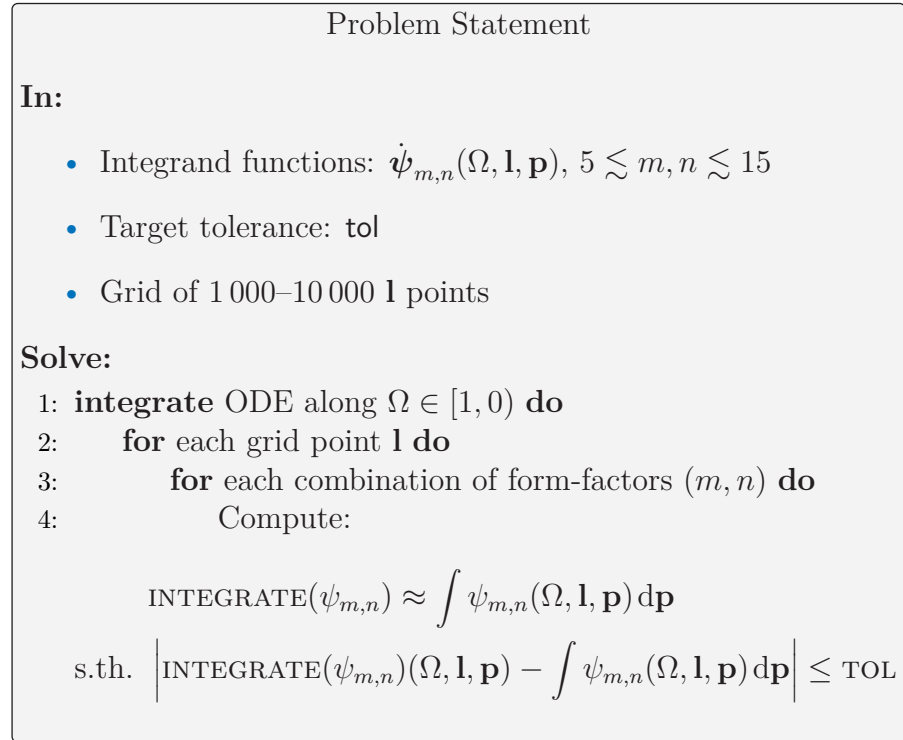


Figure 3.1: Original Problem Statement: We Focus on the Calculation for Equation (2.1)

factors in Figure 3.1, including an overview of inputs, outputs, and their expected sizes. The numerical integration occurs within a triple loop of: the flow parameter Ω ; the external momentum grid $\mathbf{l}$; and the combination of form-factors (m, n) . Our goal is the computation of each integral while keeping the integration error below a specified tolerance. Naturally, we aim to minimize the time-to-solution of the integration, while guaranteeing an integration error less than **tol**.

Accuracy, always our primary concern, is not entirely straight-forward in this application. In order to achieve small integration errors—even on very challenging integrals—we introduce the technique of *adaptive integration* in Section 3.1. Section 3.2 reexamines the integration error of TUF_{RG}’s Quantity of Interest.

Our other goal—the speed of integration—is addressed in [Section 3.3](#), where we introduce our high-performance adaptive quadrature routine: **Parallel Adaptive Integration** in two **D**imensions, PAID for short. [Section 3.4](#) provides performance results for PAID and [Section 3.5](#) concludes this chapter.

Publication

The work presented in this chapter is in parts based on research previously published in:

- [4] J. Lichtenstein, J. Winkelmann, D. Sanchez de la Pena, T. Vidovic, and E. Di Napoli. “Parallel Adaptive Integration in High-Performance functional Renormalization Group Computations”. In: *Jülich Aachen Research Alliance High-Performance Computing Symposium 2016*. Lecture Notes in Computer Science. Springer-Verlag, 2017. DOI: [10.1007/978-3-319-53862-4_15](https://doi.org/10.1007/978-3-319-53862-4_15).

3.1 Adaptive Integration

The (numerical) integrals which form the entries of the susceptibility factors vary strongly in difficulty, see example in [Section 2.1.2](#). Quadrature rules with a fixed order do not perform well for such varying difficulties: rules of small order are too inaccurate for difficult integrals, and rules of larger order are too expensive for the easier integrals. Adaptive integration methods aim to solve this problem by ‘adapting’ to the difficulty of the integration task. Before we discuss adaptive integration methods we must, as a prerequisite, first consider methods to estimate the integration error.

Bounding the integration error is quite difficult, depending on the assumptions about the integrand. For arbitrary integrals a bound is impossible, as there is no guarantee on the integrand’s behavior between two evaluation points. Therefore, bounds on the error, as well as a-priori error estimates, are beyond the scope of this work, even when assuming some smoothness of the integrand between any two evaluation points. Rather, we *estimate* the error by performing

3 PAID: An Adaptive Integrator for TUG

two numerical integrations with different degrees N and M :

$$\text{ERR}[f, \mathcal{D}] := |\mathcal{Q}_M(f, \mathcal{D}) - \mathcal{Q}_N(f, \mathcal{D})|. \quad (3.1)$$

Equation (3.1) has some parameters that require discussion; namely the values for M and N , as well as, to which result the error estimate is assigned: $\mathcal{Q}_M(f)$ or $\mathcal{Q}_N(f)$. We exclusively consider error estimates based on the Clenshaw-Curtis integration rule $\mathcal{Q}^{CC}$. It turns out to be convenient to use $M = 2N$ for $\mathcal{Q}_M^{CC}(f)$, and then consider $\mathcal{Q}_M^{CC}(f)$ to be the estimated value, as the integration nodes of $\mathcal{Q}_N^{CC}$ are a subset of the nodes for $\mathcal{Q}_{2N}^{CC}$. Thus, we can obtain the error estimate without additional function evaluations.

The underlying assumption in Equation (3.1) can be stated as follows: If the integral changes little when interpolating with N and $2N$ points we estimate it as being close to the true solution. Note that Equation (3.1) is an estimate and not a bound on the error. A low value does not necessarily indicate a correct solution, and a large error does not necessarily indicate a large absolute error. Nevertheless, this error estimation scheme works well for our—and many other—applications. There are a few alternatives to this particular approach, see [38] for a discussion. Moreover, Equation (3.1) provides us with an a-posteriori estimate, only. We need to perform two numerical integrations and, only then, can we estimate the resulting error. A-posteriori estimates result in trial-and-error approaches: first we pick parameters; integrate; check the error and, if the tolerance is not reached; retry with a more accurate method. In its essence, adaptive integration is such a trial-and-error scheme.

Let us consider the case of $\text{ERR}[f, \mathcal{D}] > \text{tol}$. That is, we have performed quadrature on an integrand and a region, estimated the error, and it turns out the error is larger than our desired tolerance. The first, and maybe the most obvious, solution is to increase the order of the integration rule. The more preferable approach is related to the idea behind compound rules: Split the domain of integration into multiple parts, while keeping the integration order fixed.

There is a critical difference between using compound rules and splitting the

3.1 Adaptive Integration

domain of integration: Splitting yields a new, smaller integration problem that can be solved recursively. The recursion allows for the integration routine to adaptively refine those parts of the domain of integration where the integrand is particularly challenging. For example, the recursion would be deeper at poles of the integrand. Algorithm 2 shows a simple adaptive integration scheme based on recursion. Whenever the integration error is larger than **tol**, the problem is split into d parts. For each new subdomain $\mathcal{D}^{s_d}$ we perform a recursive call and adjust the **tol** parameter accordingly.

```

1: function ADAPTIVE_RECURSIVE(Integrand  $\phi$ , Domain  $\mathcal{D}$ ,
                                Target Tolerance tol)
2:    $\text{ERR}[\phi] = |\mathcal{Q}_N^{CC}(\phi, \mathcal{D}) - \mathcal{Q}_{2N}^{CC}(\phi, \mathcal{D})|$ 
3:   if  $\text{ERR}[\phi] < \text{tol}$  then
4:     return  $\mathcal{Q}_{2N}^{CC}(\phi, \mathcal{D})$ 
5:   else
6:     Partition  $\mathcal{D}$  into  $d$  parts  $\mathcal{D}^{s_1}, \dots, \mathcal{D}^{s_d}$ 
7:     return  $\sum_{i=1}^d \text{ADAPTIVE\_RECURSIVE}(\phi, \mathcal{D}^{s_i}, \frac{\text{tol}}{d})$ 

```

Algorithm 2: Adaptive Integration on Domain $\mathcal{D}$ via Recursion

Adaptive integration based on recursion implicitly relies on a *local error* criterion: Consider the example where a one dimensional domain is split into two equally sized subdomains which are solved recursively. In order for the sum over the two domains to be below the tolerance, the errors of each domain must be below half of the original tolerance. More generally, the target tolerance must be proportional to the size of the domain of integration. Isolated features of an integrand which are hard to integrate, such as sharp peaks, can produce recursive problem instances that are very difficult. To continue our example, resolving a sharp peak results in very deep recursion and very small target tolerances.

For many applications it is not necessary to fully resolve the entire domain of integration to the desired tolerance; instead the average error over the entire domain is of relevance. For these applications it is often more efficient to adopt a *global error* criterion. When employing a global error criterion new problem

3 PAID: An Adaptive Integrator for TUfRG

instances are not solved independently, instead subdomains with large error estimate are refined until the sum over all errors falls below the tolerance. With a global error criterion any challenging subdomains of the integrand are refined just enough for the overall error to be below the tolerance.

Algorithm 3 outlines an adaptive integration routine that takes as input: an integrand ϕ ; a domain of integration D ; and a desired tolerance tol . The algorithm returns the integral of ϕ over $\mathcal{D}$ with an estimated error lower than tol . We require a data structure that allows us to easily retrieve the subdomain of largest error, even for a large number of stored subdomains as the number of subdomains grows by $d - 1$ elements for each loop iteration. A global error criterion is implemented via a heap data structure to ensure the tuple with largest error is treated first. We use a heap $\text{heap} = \{(\mathcal{D}^s, \text{ERR}[\phi, \mathcal{D}^s])\}_{n=1}^N$, with the error estimate as the key for the heap property and the additional constraint that the domains stored in the heap partition $\mathcal{D}$, i.e. $\dot{\bigcup}_{(\mathcal{D}^s, \text{ERR}[\phi, \mathcal{D}^s]) \in \text{heap}} \mathcal{D}^s = \mathcal{D}$. For each loop iteration in **Algorithm 3** the subdomain with largest error is taken from the *heap*. The subdomain is then partitioned into d new parts, the error for each new subdomain is computed, and the new subdomains and corresponding error is stored in the *heap*. Eventually the overall error $\text{ERR}[\phi, \mathcal{D}]$ falls below tol and we return the integration result.

```

1: function ADAPTIVE(Integrand  $\phi$ , Domain  $\mathcal{D}$ , Target Tolerance  $\text{tol}$ )
2:    $\text{ERR}[\phi, \mathcal{D}] = |\mathcal{Q}_N^{CC}(\phi, \mathcal{D}) - \mathcal{Q}_{2N}^{CC}(\phi, \mathcal{D})|$ 
3:   Initialize heap with  $(\mathcal{D}, \text{ERR}[\phi, \mathcal{D}])$ 
4:   while  $\text{ERR}[\phi, \mathcal{D}] > \text{tol}$  do
5:      $(\mathcal{D}^s, \text{ERR}[\phi, \mathcal{D}^s]) \leftarrow \text{heap.pop}()$ 
6:     Partition  $\mathcal{D}^s$  into parts  $\mathcal{D}^{s_1}, \dots, \mathcal{D}^{s_d}$ 
7:     for  $a = 1 : d$  do
8:        $\text{ERR}[\phi, \mathcal{D}^{s_a}] = |\mathcal{Q}_N^{CC}(\phi, \mathcal{D}^{s_a}) - \mathcal{Q}_{2N}^{CC}(\phi, \mathcal{D}^{s_a})|$ 
9:        $\text{heap.push}(\mathcal{D}^{s_a}, \text{ERR}[\phi, \mathcal{D}^{s_a}])$ 
10:     $\text{ERR}[\phi, \mathcal{D}] = \sum_{(\mathcal{D}^s, \text{ERR}[\phi, \mathcal{D}^s]) \in \text{heap}} \text{ERR}[\phi, \mathcal{D}^s]$ 
11:  return  $\sum_{(\mathcal{D}^s, \text{ERR}[\phi, \mathcal{D}^s]) \in \text{heap}} \mathcal{Q}_{2N}^{CC}(\phi, \mathcal{D}^s)$ 

```

Algorithm 3: Adaptive Integration on Domain $\mathcal{D}$ with Global Error Criterion

3.1.1 Related Work

Numerical integration is one of the oldest mathematical problems, with work going all the way back to Pythagoras ca. 500 BC. In addition to work related to quadrature rules, we also discuss adaptive integration: methods that aim to “automatically” satisfy some error tolerance.

Two very good treatises on the topic are, e.g., [57, 23]. Very good work in the recent past is Trefethen’s work comparing the Clenshaw-Curtis to the Gauss rule [97] as well as Trefethen and Weideman’s work on the trapezoidal rule [98]. The following very short history of quadrature is largely taken from these two works. Integration rules, as defined in this chapter, go back to the 1810-1820s with the trapezoidal [78] and Gauss [35] rules. Clenshaw and Curtis published their rule in 1960 [20] with a focus on use with the “automatic computer”. A major theoretical breakthrough was achieved by Stieltjes in 1884, who proved that the integration weights for the Gauss rule are always positive. Positive weights in integration rules avoid Runge’s phenomenon; the weights of the Clenshaw-Curtis rules can also be shown to be positive. Lastly, there is a riveting story in the complexity of computing the integration rules for Gauss and Clenshaw-Curtis. We do not go into detail in this work, but it is highly recommended to the interested reader.

Adaptive quadrature as a topic is comparatively new, when compared to Pythagoras, although an old topic on the time frame of computer science. A very good treatment on a meta-algorithm was published by Rice in 1975 [81]. The topic is rounded out with a thesis with a focus on parallel computers by Bull in 1988 [19]. A good overview on recent advances in quadrature is available in [22].

There are some venerable quadrature codes: QUADPACK [77] (1983) is one of the classic non-adaptive numerical integration codes. DCUHRE [15] (1991) remains one of the state-of-the-art algorithms for adaptive numerical integration. Indeed, it was used in the TUFrg code before our own work superseded it. CUBATURE [108] is a reimplementation of DCUHRE with more explicit licensing.

Lastly, some words on the use of quadrature in the fRG community. Although

High-Performance fRG is still a young field, all of the fRG codes that we have collaborated on already use adaptive integration schemes, such as the DCUHRE library, or routines taken from Numerical Recipes [52]. Nevertheless, with the exception of this work the fRG community limits the adaptivity to the integral at hand instead of considering adaptivity over the entire linear operator.

3.2 Numerical Integration for TUF_{RG}

Adaptive integration is an apt tool to calculate the susceptibility factors: The routine ensures a given estimated integration error, whether the current integration problem is easy (for large Ω), or difficult (for small Ω). For TUF_{RG} a global error criterion over the susceptibility factors is acceptable, as we are interested in limiting the (estimated) average error. While adaptive integration over $\dot{\chi}^{\text{pp}}(\Omega, \mathbf{l})$ and $\dot{\chi}^{\text{ph}}(\Omega, \mathbf{l})$ is a workable approach, we can do better: In this section we derive an adaptivity scheme over $\dot{\mathbf{P}}(\Omega, \mathbf{l})$ instead of just over the elements of the susceptibility factors. The result is a significant increase in performance, which we present in Section 3.4. This section focuses on calculating $\dot{\mathbf{P}}(\Omega, \mathbf{l})$ for simplicity, but the same methodology applies, mutadis mutandis, to $\dot{\mathbf{C}}(\Omega, \mathbf{l})$ and $\dot{\mathbf{D}}(\Omega, \mathbf{l})$.

Let us consider a single element of $\dot{\mathbf{P}}$

$$\dot{\mathbf{P}}_{m,n} = \sum_{j,k} \mathbf{V}_{m,j}^P \dot{\chi}_{j,k}^{\text{pp}} \mathbf{V}_{k,n}^P = \sum_{j,k} \mathbf{V}_{m,j}^P \mathcal{Q}_N(\dot{\psi}_{j,k}^{\text{pp}}) \mathbf{V}_{k,n}^P. \quad (3.2)$$

Here, and in the following, we keep the external momentum $\mathbf{l}$ and the flow parameter Ω implicit in favor of a more readable notation. When considering the elements of $\dot{\mathbf{P}}$ we must consider the error of all $\dot{\chi}_{j,k}^{\text{pp}}$ involved in the right-hand side of Equation (3.2), obtained from Equation (2.10) via discretization and truncation of unity. In its original form, the integral encompasses not only the susceptibility but also the two particle couplings $\mathbf{V}$. Hence we must also

include $\mathbf{V}$ in the global error, resulting in

$$\begin{aligned}
 & \left| \sum_{j,k} [\mathbf{V}_{m,j}^P \mathcal{Q}_N(\dot{\psi}_{j,k}^{\text{pp}}) \mathbf{V}_{k,n}^P - \mathbf{V}_{m,j}^P \mathcal{Q}_{2N}(\dot{\psi}_{j,k}^{\text{pp}}) \mathbf{V}_{k,n}^P] \right| \\
 & \leq \left\| \mathbf{V}_{m,:}^P \right\|_{\infty} \left\| \mathbf{V}_{:,n}^P \right\|_{\infty} \sum_{j,k} \left| \mathcal{Q}_N(\dot{\psi}_{j,k}^{\text{pp}}) - \mathcal{Q}_{2N}(\dot{\psi}_{j,k}^{\text{pp}}) \right| \\
 & = \left\| \mathbf{V}_{m,:}^P \right\|_{\infty} \left\| \mathbf{V}_{:,n}^P \right\|_{\infty} \sum_{j,k} \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}] \\
 & =: \widehat{\text{ERR}}[\dot{\mathbf{P}}_{m,n}]
 \end{aligned} \tag{3.3}$$

where $\mathbf{V}_{m,:}^P$ is the m^{th} row, and $\mathbf{V}_{:,n}^P$ is the n^{th} column of $\mathbf{V}$.

$\widehat{\text{ERR}}[\dot{\mathbf{P}}_{m,n}]$ is a modified error estimate, and does not conform to the definition for ERR given in Equation (3.1). Instead of estimating the integration error, we estimate the integration error's influence on $\dot{\mathbf{P}}_{m,n}$ directly. By generalizing the error to involve our Quantity of Interest, the entire $\dot{\mathbf{P}}$ operator, we obtain an approach that allows for even more adaptability in the integration of the functions $\dot{\psi}_{j,k}^{\text{pp}}$, $1 \leq k \leq m, 1 \leq j \leq n$. For the maximum norm of $\dot{\mathbf{P}}$ we obtain:

$$\begin{aligned}
 \widehat{\text{ERR}}[\dot{\mathbf{P}}] &= \max_{m,n} \left\{ \text{ERR}[\dot{\mathbf{P}}_{m,n}] \right\} \\
 &= \max_{m,n} \left\{ \left\| \mathbf{V}_{m,:}^P \right\|_{\infty} \left\| \mathbf{V}_{:,n}^P \right\|_{\infty} \sum_{j,k} \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}] \right\} \\
 &= \left\| \mathbf{V}^P \right\|_{\max}^2 \sum_{j,k} \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}]
 \end{aligned} \tag{3.4}$$

We must integrate all $\dot{\psi}_{j,k}$ at the same time. The error criterion of Equation (3.4) allows the integration of each $\dot{\psi}_{j,k}$ to be adaptive with respect to the Quantity of Interest of the use-case. A global error over $\dot{\mathbf{P}}$ is not only a mathematically sound approach, the resulting adaptivity also decreases the required number of operations. What is more, estimating the error of $\dot{\mathbf{P}}$ via a matrix norm can be motivated by the physical application, as the coupling function is constructed with $\dot{\mathbf{P}}$ as a linear operator.

Combining Algorithm 3, with our novel error estimate (Equation (3.4)) results

3 PAID: An Adaptive Integrator for TUF_RG

```

1: function QUAD_TUFRG(Integrand  $\dot{\psi}^{\text{pp}} : \mathbb{C}^{n \times m} \rightarrow \mathbb{C}^{n \times m}$ ,
    Domain of Integration  $\mathcal{D} \subseteq \mathbb{C}$ ,
    Target Tolerance  $\text{tol}$ , Coupling Matrix  $\mathbf{V}^P$ )
2:   Initialize empty heap  $\text{heap}$ 
3:   for all elements  $(j, k)$  of  $\dot{\psi}^{\text{pp}}$  do
4:      $\text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}] = |\mathcal{Q}_N^{\text{CC}}(\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}) - \mathcal{Q}_{2N}^{\text{CC}}(\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D})|$ 
5:      $\text{heap.push}((\mathcal{D}, \dot{\psi}_{j,k}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}]))$ 
6:      $\widehat{\text{ERR}}[\dot{\mathbf{P}}, \mathcal{D}] = \|\mathbf{V}^P\|_{\max}^2 \sum_{(\mathcal{D}^s, \dot{\psi}_{j,k}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^s]) \in \text{heap}} \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^s]$ 
7:     while  $\widehat{\text{ERR}}[\dot{\mathbf{P}}, \mathcal{D}] > \text{tol}$  do
8:        $(\mathcal{D}^s, \dot{\psi}_{j,k}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^s]) \leftarrow \text{heap.pop}()$ 
9:       Partition  $\mathcal{D}^s$  into parts  $\mathcal{D}^{s_1}, \dots, \mathcal{D}^{s_4}$ 
10:      for  $a = 1 : 4$  do
11:         $\text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^{s_a}] = |\mathcal{Q}_N^{\text{CC}}(\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^{s_a}) - \mathcal{Q}_{2N}^{\text{CC}}(\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^{s_a})|$ 
12:         $\text{heap.push}((\mathcal{D}^{s_a}, \dot{\psi}_{j,k}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^{s_a}] ))$ 
13:         $\widehat{\text{ERR}}[\dot{\mathbf{P}}, \mathcal{D}] = \|\mathbf{V}^P\|_{\max}^2 \sum_{(\mathcal{D}^s, \dot{\psi}_{j,k}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^s]) \in \text{heap}} \text{ERR}[\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^s]$ 
14:      For all  $(j, k)$  return  $\dot{\chi}_{j,k}^{\text{pp}} = \sum_{(\mathcal{D}^s, \dot{\psi}_{j',k'}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j',k'}^{\text{pp}}, \mathcal{D}^s]) \in \text{heap}} \mathcal{Q}_{2N}^{\text{CC}}(\dot{\psi}_{j,k}^{\text{pp}}, \mathcal{D}^s)$ 
     $(j', k') = (j, k)$ 

```

Algorithm 4: Parallel Adaptive Integration for TUF_RG with Global Error Criterion over $\dot{\mathbf{P}}$

in Algorithm 4. The QUAD_TUF_RG routine requires the following inputs: the matrix-valued integrand $\dot{\psi}^{\text{pp}}$; the domain of integration $\mathcal{D}$; a target tolerance tol ; and the coupling matrix $\mathbf{V}^P$. The routine returns the susceptibility matrix $\dot{\chi}^{\text{pp}}$. For an algorithm we require a heap that stores multiple integrands. We adapt the previous heap as follows: $\text{heap} = \{(\mathcal{D}^s, \dot{\psi}_{j,k}, \text{ERR}[\dot{\psi}_{j,k}, \mathcal{D}^s])_{n=1}^N\}$, with the error $\text{ERR}[\dot{\psi}_{j,k}, \mathcal{D}^s]$ as the key for the heap property, independent of which $\dot{\psi}_{j,k}$ the error is computed for. We require the additional constraint that the subdomains stored in the heap partition the domain of integration, $\mathcal{D}$, for each $\dot{\psi}_{j,k}$, i.e

$$\forall(j, k) : \bigcup_{(\mathcal{D}^s, \dot{\psi}_{j',k'}^{\text{pp}}, \text{ERR}[\dot{\psi}_{j',k'}^{\text{pp}}, \mathcal{D}^s]) \in \text{heap}} \mathcal{D}^s = \mathcal{D}.$$

Line 3 to Line 5 of Algorithm 4 initialize *heap* with an item for each element of the operator $\dot{\psi}^{\text{pp}}$. Line 6 and Line 13 compute the new global error, see Equation (3.4). As long as the global error is above the tolerance `tol` we proceed with the **while** loop: First, we retrieve the Domain $\mathcal{D}^s$ and the integrand $\dot{\psi}_{j,k}^{\text{pp}}$ of largest error. Recall that for TUfRG the domain of integration $\mathcal{D}$ is a square from $(-\pi, -\pi i)$ to $(\pi, \pi i)$. Accordingly, we split each domain $\mathcal{D}^s$ into four square subdomains $\mathcal{D}^{s_1}$ to $\mathcal{D}^{s_4}$. For each of the resulting subdomains, we proceed to calculate the error (Line 11) and then store the resulting subdomain, $\dot{\psi}_{j,k}^{\text{pp}}$ indices, and error. At the end of the loop we update the global error—here the inner sum iterates over all subdomains $d \subset \mathcal{D}$ of the same (j, k) index stored in the heap. Once the **while** loop is completed, we return $\dot{\chi}^{\text{pp}}$ as the result in Line 14.

3.3 The PAID Software Package

In this section we present the PAID—**P**arallel **A**daptive **I**ntegration in two **D**imensions—adaptive numerical integration routine. PAID is developed specifically as a high-performance solver for the numerical integrals arising from TUfRG. PAID is a shared memory solver; within the TUfRG code the **I** grid as well as the different matrix-valued integrands arising from Equations (2.1) to (2.3) are solved using distributed memory parallelism.

First, let us consider a parallelization scheme for Algorithm 4. The central work-item in the algorithm is a tuple of integrand, domain of integration, and integration error estimate. These work-items are managed in a central data structure, the heap. In every loop iteration a work-item is retrieved from the heap, processed, and the resulting new work-items are stored in the heap. For TUfRG we obtain four such new work-items per retrieved work-item.

Algorithms with very well defined work-items storable in a data structure are good candidates for parallelization via the *task parallelism* paradigm: Each work-item is a task that may be executed in parallel. Task-parallel applications can scale very well and are often easy to implement. Multiple

3 PAID: An Adaptive Integrator for TUfRG

threads simultaneously retrieve items for the heap, execute the work-items, and then store the new items.

There is a fundamental problem with a parallel implementation of [Algorithm 4](#). In order to reduce the error most effectively, the task with highest integration error is retrieved from the heap. Within the **while** loop, new tasks are added to the heap before a new item (with largest error) is retrieved again. We say, a single task is *active*, that is, not in the queue and actively being worked on. Now, when multiple threads retrieve tasks from the heap there are more active tasks, one per thread. Some threads necessarily retrieve tasks with errors smaller than the largest error. For a large number of active tasks some of the retrieved tasks would not be retrieved in the case of a serial execution. Not only do many active tasks additional work—and thus overhead that degrades the efficiency of the parallel implementation—the parallel algorithm also becomes non-deterministic: The amount of work as well as the obtained result changes with the number of threads, possibly even when rerunning with the same number of threads.

In the extreme case of a function with a single pole there would be very few domains of large error. Partitioning such a domain typically results in three subdomains of small error and the domain that contains the pole of large error. In that case working on many items simultaneously results in overhead, in the sense that a sequential algorithm would not execute these tasks. Although a potentially serious problem, this is not a concern for the integrals in TUfRG for a number of reasons: Each integrand has a number of poles, confer [Figure 2.1](#); The integrands have difficult to integrate features that are not as isolated as a pole, such as plateaus; and, finally, we integrate over multiple integrands at the same time, effectively increasing the number of poles to be integrated. [Section 3.4](#) reports performance results that support our claims.

The main design challenge for task-parallel algorithms is the scheduling of tasks. Correct access to the heap requires a *critical section*, typically implemented via a mutual exclusion (*mutex*) mechanism. In order to acquire new work a thread must wait to enter the critical section, resulting in overhead.

Therefore, our main concern in the implementation design of PAID must be lowering the amount of time each thread spends within the critical section.

There are three aspects [Algorithm 4](#) that we investigate for potential to lower the amount of time spent in the critical section. First, storing and retrieving items from the heap ([Lines 8 and 12](#)) requires $\mathcal{O}(\log n)$ operations, which becomes expensive very quickly as the size of the heap grows continuously. Second, recomputation of the error via summing over the entire heap ([Line 13](#)) is even more expensive ($\mathcal{O}(n)$ operations). Finally, entering the critical section itself is not a free operation either. Acquiring a contested mutex requires thousands of operations.

The most effective method to reduce the time within the critical section revolves around the computation of the error in [Line 13](#). There is no need for every thread to recompute the error at every iteration of the **while** loop. In many cases it is possible to entirely avoid costly recomputation of the error estimate. Instead, PAID caches the error by adding the error estimates of tasks added to the heap and subtracting the error estimates of the tasks retrieved from the heap. This approach reduces the $\mathcal{O}(n)$ summing operation to a constant time approach. For challenging integrals, resulting in many **while** loop iterations, the cached error estimate may become inaccurate due to inexact arithmetic. In order to mitigate this problem, PAID optionally recomputes the error via summing after a (configurable) number of iterations.

Another method to reduce time within the critical section is to retrieve multiple tasks from the heap at the same time. This approach effectively lengthens the intervals between accesses to the critical section, at the cost of a larger number of active tasks. For PAID the number of active tasks per thread is configured via the `MaxTasks` parameter; a typical choice of `MaxTasks` for TUF_{RG} is between 5 and 20.

PAID's critical section is described in [Algorithm 5](#). For the sake of conciseness, we make a number of simplifying assumptions in the algorithm; e.g., we assume that the heap always contains at least `MaxTasks` work-items. The critical section is located at the end of the while loop that corresponds to [Line 7](#) in [Algorithm 4](#). At this point, `in_queue` contains work-items taken from the

```

    ▷ in_queue contains MaxTasks work-items from the heap
    ▷ out_queue contains the 4 MaxTasks resulting work-items
1: begin critical
2:   for i = 1 : 4MaxTasks do
3:     err += out_queue[i].error
4:     if out_queue[i].area < thresh_area
5:     or out_queue[i].error < thresh_err then
6:       archive.push( out_queue[i] )
7:     else
8:       heap.push( out_queue[i] )
9:   for i = 1 : MaxTasks do
10:    err -= in_queue[i].error
11:  out_queue.clear()
12:  in_queue.clear()
13:  for i = 1 : MaxTasks do
14:    in_queue.push( heap.back() )
15:    heap.pop()
16: end critical

```

Algorithm 5: PAID’s Critical Section. Now shown: Occasional Recomputation of the Error from All Items in the Heap

heap, partitioned, and the error estimate for the resulting work-items already computed. The resulting work-items are stored in `out_queue`. In order to reduce the size of the heap, tasks that are guaranteed to never become active are stored separately in the `archive` data structure.

The critical section itself involves updating the error estimate as well as storing in and retrieving from the queue. Updating the error involves summing over the errors for work-items in the `out_queue` and adding them to the total error. Then we can sum over the items in the `in_queue` and subtract the result from the global error estimate. The work-items in `out_queue` are added to the heap, only then are `MaxTasks` new work-items taken from the heap.

PAID has three parameters that concern access to the critical section: the number of active tasks for each thread `MaxTasks`; the order of the Clenshaw-Curtis integration rule N ; and archival thresholds for the error estimate and

area of the domain of integration, respectively, `thresh_err` and `thresh_area`. For the TUF_{RG} code, `thresh_area` = 10^{-9} , and `thresh_err` is not used.

3.4 Numerical Experiments

In the following we discuss PAID's performance on integrals arising from the TUF_{RG} formalism. We compare against DCUHRE, the state-of-the-art adaptive integrator previously used within the TUF_{RG} code. To show-case the performance and efficacy we discuss PAID's and DCUHRE's performance in three separate experiments.

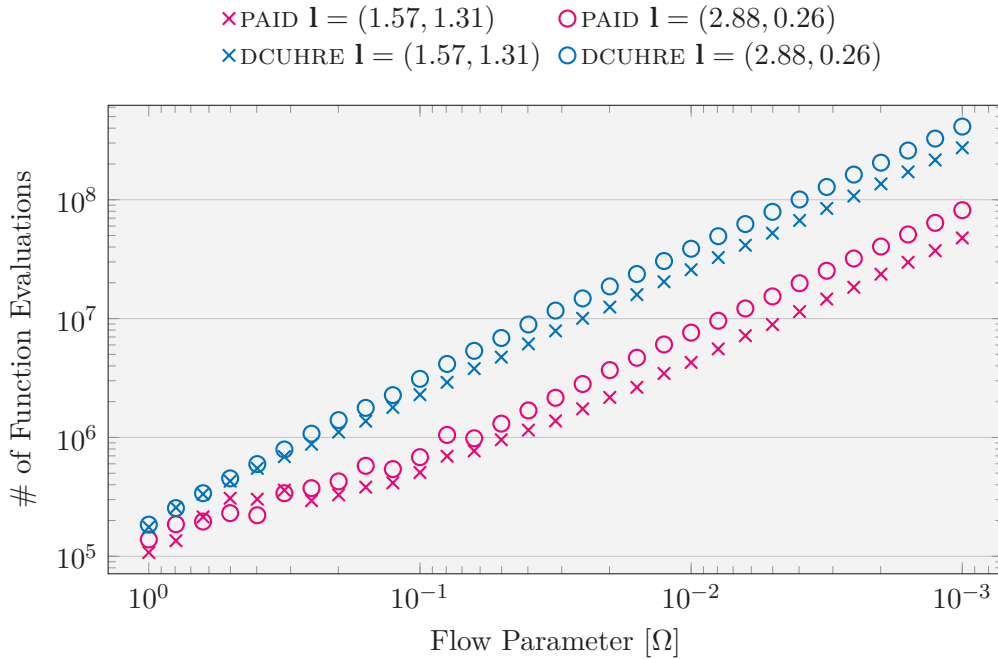


Figure 3.2: The Number of Function Evaluations Needed for Calculating all Integrals of a Fixed l Value is Plotted Against Ω . N chosen to yield the same number of function evaluations per subdomain as in DCUHRE.

(MaxTasks = 10, $N = 4$, $mn = 45$ form-factor combinations. Run on a [JURECA](#) node.)

Figure 3.2 plots the number of function evaluations required to compute all

entries of the $\dot{\mathbf{P}}$, $\dot{\mathbf{C}}$, and $\dot{\mathbf{D}}$ operators against the flow parameter Ω . For the simulation, we use a form-factor expansion that results in $mn = 45$ integrals $\dot{\mathbf{x}}_{j,k}^{\text{pp}}$ and $\dot{\mathbf{x}}_{j,k}^{\text{ph}}$ for $1 \leq j \leq n, 1 \leq k \leq m$. DCUHRE performs adaptive integration on each of the resulting $2 \cdot 45$ integrals separately and independently. PAID implements the improved adaptivity scheme of Equation (3.4)¹, resulting in two adaptive integration procedures with 45 operator entries each. We consider two external momenta $\mathbf{l}$ as representative examples: $1.57 + 1.31i$ (shown as $\times/\times$) and $2.88 + 0.26i$ (shown as $\circ/\circ$). For this experiment PAID and DCUHRE perform the same number of function evaluations per subdomain; as DCUHRE performs 65 evaluations per subdomain, for PAID we choose $N = 4$ resulting in $(2N)^2 = 64$ function evaluations per subdomain. `MaxTasks` is set to 10. This experiment uses an entire JURECA node without HyperThreading.

Figure 3.2 shows (1) the integrals become more computationally expensive throughout the flow (decreasing Ω), and (2) that the different external momenta vary in difficulty. For both solvers the external momentum plotted as $\circ/\circ$ is the more challenging one. This is expected, as the $\mathbf{l}$ dependency may cause some of the integrands to develop more or sharper peaks. The figure further indicates that PAID requires significantly fewer function evaluations for both external momenta, especially for lower values of the flow parameter, where the integrals become more computationally expensive. For $\circ/\circ$, PAID requires 5 times fewer function evaluations than DCUHRE at most values of the flow parameter Ω ; E.g., for the integrals arising from $\Omega = 10^{-3}$ PAID requires only 81 675 650 function evaluations while DCUHRE requires 412 524 645, a 5.05 factor reduction. The situation is similar for the easier external momentum $\times/\times$ at $\Omega = 10^{-3}$. Here, PAID requires 5.7 times fewer function evaluations than DCUHRE (47 711 450 versus 273 521 885). The difference between PAID and DCUHRE must be attributed to the new adaptivity scheme of Equation (3.4), and not to the difference in the integration rules between two solvers. Indeed, when PAID uses the same adaptivity criterion as DCUHRE the performance gap between the two solvers closes.

¹ The error criterion must be modified to form the maximum over not only $\mathbf{V}^P$ but also $\mathbf{V}^C$ and $\mathbf{V}^D$, where appropriate.

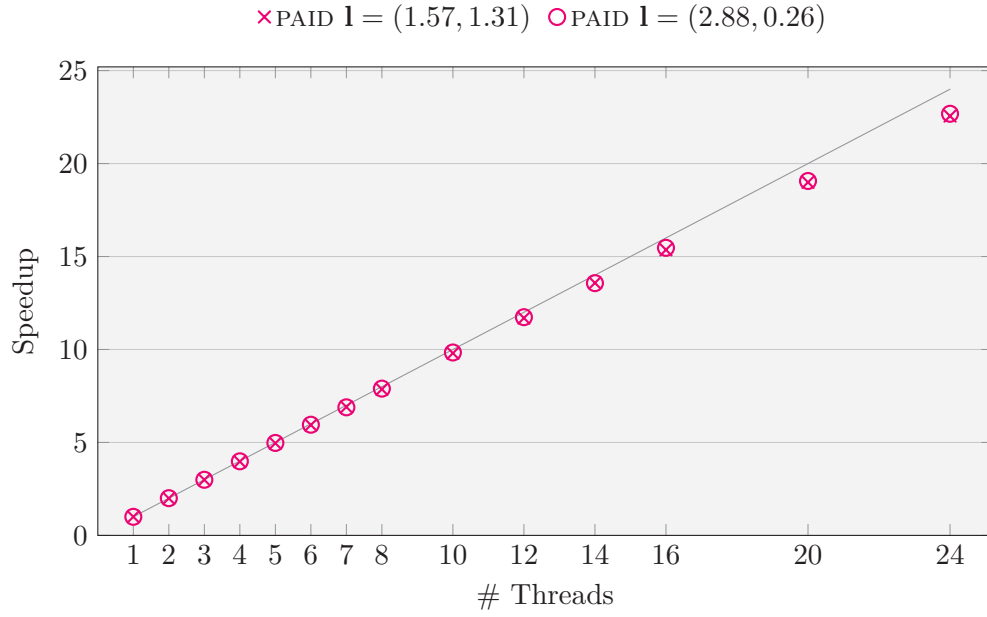


Figure 3.3: Speedup Plot of PAID: This Analysis Uses the PAID Parameters that Result in the Best Performance. The High Values for MaxTasks and N Indicate that Many Active Tasks do not Cause Excessive Overhead.

(MaxTasks = 18, $N = 6$, $nm = 325$ form-factor combinations. Run on a [JURECA](#) node.)

We can conclude that our newly developed adaptivity criterion drastically reduces the number of required function evaluations. However, we must also consider the parallel overhead of PAID's implementation, as discussed in [Section 3.3](#). A parallel implementation of [Algorithm 4](#) may incur overhead by either: (1) too many active tasks, resulting in the execution of unnecessary tasks; or (2) the locking cost of, as well as contention for, the mutex.

[Figure 3.3](#) shows speedup plots for PAID and shows that there is actually very little overhead incurred by either too many active tasks or overhead due to the mutex. As we do not compare with DCUHRE, we are free to choose parameters in a way that stresses PAID with many active tasks. Scenario (2), overhead due to locking of, and contention for, the mutex only becomes a problem when each task is very cheap. This is not the case for TUF_{RG}, largely due to the fact that our domain of integration is two dimensional, necessitating tens or

3 PAID: An Adaptive Integrator for TUF_{RG}

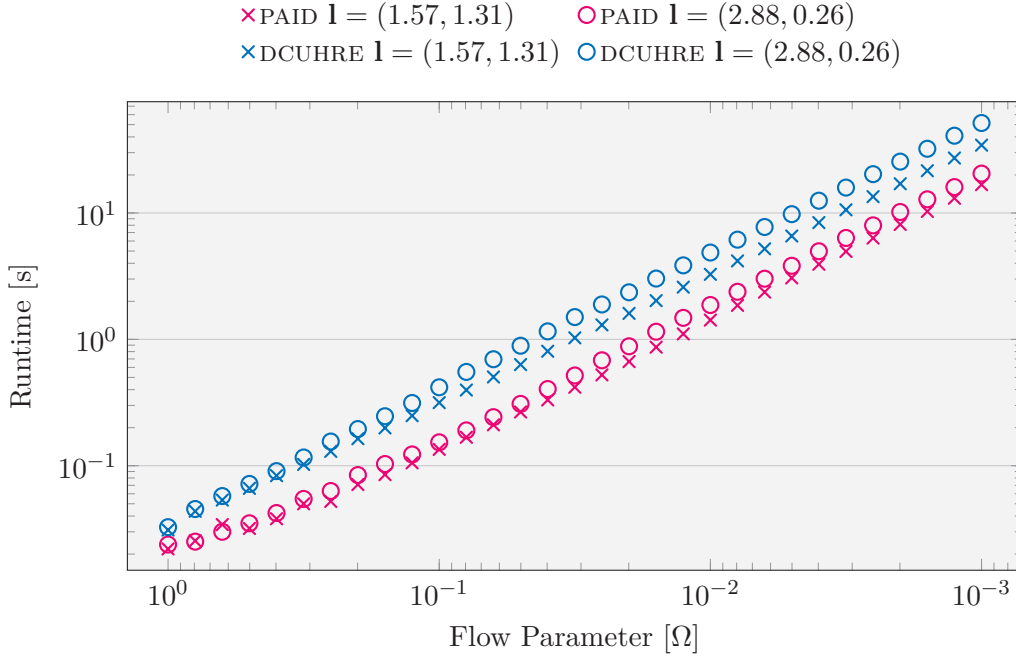
hundreds of function evaluations per integration task. Accordingly, we focus on scenario (1) and choose a larger **MaxTasks** of 18, $N = 6$, as well as a form-factor expansion that results in $mn = 325$ integrals. We present the speedup for the most difficult integrals arising at $\Omega = 10^{-3}$.

The speedup of [Figure 3.3](#) are close to ideal all the way up to JURECA's core count of 24; perfect speedup is indicated by the gray line in the plot. Although we have previously established that the difficulty of the integrals at different external momenta $\mathbf{l}$ varies, the resulting parallel speedup does not: PAID performs practically identical for both external momenta. We can conclude that for the TUF_{RG} integrals a large number of active tasks does not incur significant overhead, neither due to a large **MaxTasks** nor due to a large number of threads.

Although a shared memory parallelization with DCUHRE is much simpler, we find a speedup which is as high as the one we achieve using PAID (not shown). This can be explained by the fact that the runtimes of the different integrals within a fixed value of $\mathbf{l}$ do not vary much in serial execution. Thus, the parallelization over the j and k values does not suffer from load imbalances and results in a close to ideal speedup.

Let us now see if PAID can transport the increased adaptivity and good parallelism into faster runtimes. To this end [Figure 3.4](#) compares runtimes of DCUHRE and PAID. The experiment setup is similar to that in [Figure 3.2](#): We plot the runtimes required to compute all entries of the $\dot{\mathbf{P}}$, $\dot{\mathbf{C}}$, and $\dot{\mathbf{D}}$ operators against the flow parameter Ω . However, we choose **MaxTasks** = 7 instead of 10, as PAID performs slightly better with this particular **MaxTasks**. Identically to the experiment in [Figure 3.2](#) we compare against the same two external momenta $\times/\times$ and $\circ/\circ$, set $N = 4$, and use the same form-factor expansion with $mn = 45$. This experiment uses an entire JURECA node with 24 threads, one per core.

There are many similarities to the experiment in [Figure 3.2](#). Both PAID and DCUHRE require less runtime to solve the integrals for the external momentum $\times/\times$. What is more, PAID is significantly faster than DCUHRE. For $\times/\times$ DCUHRE requires 245 seconds for the entire flow, with most time spent on the integrals

Figure 3.4: Runtime for Calculation of all Integrals for a Fixed $\mathbf{l}$ (MaxTasks = 7, $N = 4$, $mn = 45$ form-factor combinations. Run on a [JURECA](#) node.)

of $\Omega = 10^{-3}$ (51.2 seconds). On the other hand, PAID requires only 77 seconds, with 16.7s spent on the integrals for $\Omega = 10^{-3}$. PAID cannot leverage the entire factor 5 improvement over DCUHRE in the number of function evaluations (Figure 3.2) into a $5\times$ runtime advantage. Instead, PAID performs a respectable 3 times faster than DCUHRE throughout the flow. The reduction in performance is due to the choice of MaxTasks and N : Recall that for this experiment we provide a fair comparison against DCUHRE by choosing $N = 4$, a parameter that is not optimal for PAID's parallel performance.

To summarize, we have proposed a new adaptivity scheme for the integrals arising in TUF_{RG}. The PAID software implements this adaptivity scheme and greatly reduces the time-to-solution for TUF_{RG}'s integrals without sacrificing parallelism. The end result is roughly a factor 3 speedup in runtime on a [JURECA](#) node.

3.5 Summary

In this chapter we introduced adaptive integration and derived an adaptivity scheme for the TUFfRG formalism that encompasses the entire susceptibility factor matrix. Our approach must integrate all entries of the susceptibility factor matrices at the same time, requiring an explicitly parallel approach. The PAID code implements our proposed adaptivity scheme, and we presented numerical experiments that our approach performs significantly better than integrating each entry of the susceptibility factor separately.

The improved adaptivity scheme leverages the knowledge that the integrals themselves are not the Quantity of Interest for the simulation. Rather, we are interested in the two particle couplings, which are obtained from the susceptibility matrices via a (matrix) contraction. By choosing an appropriate approximation scheme for the contraction, we obtained an application specific adaptivity criterion reducing time-to-solution by a factor of $5\times$ without suffering in scalability.

This chapter closely followed the course of action outlined in [Chapter 1](#): We identified [Equation \(3.4\)](#) as the relevant adaptivity criterion for TUFfRG and developed PAID to leverage our improved criterion. As a result, PAID significantly outperforms a black-box numerical kernel. Initial experiments with other fRG methods yields promising results, indicating PAID’s potential for wider usage.

*What one programmer can do in one month,
two programmers can do in two months.*

Fred Brooks

4

ChASE: An Eigensolver for Sequences of Correlated Hermitian Eigenvalue Problems

This chapter treats HEPs with a focus on problems arising from DFT applications. We consider Subspace Iteration with polynomial acceleration and our own implementation, CHASE, in particular. CHASE’s focus is sequences of dense correlated eigenvalue problems (in the sense of [Definition 11](#)) arising from DFT and related applications, where a significant part of the lower spectrum is sought ($\text{nev} \leq \frac{n}{5}$ of eigenvalues), i.e., exterior eigenvalue problems in the sense of [Definition 4](#).

Sequences (of correlated) exterior HEPs are very common in Materials Science. For many applications, including DFT, the solution of the HEP is one of the most expensive parts of each simulation. CHASE addresses a number of pain points for the DFT community: (1) CHASE leverages the correlation of the

HEPs in a way that few simulation codes currently do. Particularly, in the case of dense HEPs the community strongly relies in direct solvers, which do not exploit correlation between HEPs at all. We present numerical experiments that indicate that CHASE out-performs a state-of-the-art direct solver for a number of problems. (2) CHASE addresses the need for high-performance eigensolvers with GPU support. Our solver supports distributed-memory (MPI) as well as shared-memory (via threaded BLAS) parallelism. What is more, CHASE can make use of GPU accelerators and thus leverage all available computing resources on a typical supercomputer node. (3) CHASE is easy to integrate into existing simulation codes. The CHASE algorithm is separated from the low-level numerical kernels via an interface that abstracts away the location of the data in the case of distributed-memory parallelism. As a result, CHASE supports a number of different parallelization approaches *and* can often easily be integrated into existing simulation codes.

The rest of this chapter is organized as follows. [Section 4.1](#) presents the basics of Subspace Iteration, including an introduction into the Chebyshev polynomial filter as well as a discussion of related work. We outline the novelties of CHASE in [Section 4.2](#). Significant effort went into making CHASE an easy-to-use library, its design and usage is outlined in [Section 4.3](#). Finally, performance results are presented in [Section 4.4](#) and we provide a short summary in [Section 4.5](#).

Publication

The work presented in this chapter is in parts based on research previously published in:

- [3] J. Winkelmann, P. Springer, and E. Di Napoli. “ChASE: Chebyshev Accelerated Subspace Iteration Eigensolver for Sequences of Hermitian Eigenvalue Problems”. In: *ACM Trans. Math. Softw.* 45.2 (Apr. 2019), 21:1–21:34. DOI: [10.1145/3313828](https://doi.org/10.1145/3313828).

The results of [Section 4.2.2](#) are not my own work and have been previously published without my involvement. They are included for context only.

4.1 Subspace Iteration

We start out with a short overview of a modern Subspace Iteration (SI) algorithm, including most of the basic subroutines. CHASE builds on the modern SI algorithm, and those subroutines that are dramatically changed in CHASE are discussed in the next section.

Subspace Iteration is a generalization of the Power Iteration Method, presented earlier in [Section 2.2.1](#). A very simple Subspace Iteration scheme begins with a matrix $\tilde{V}^{(0)}$ of element-wise independent and identically distributed vectors, and performs the following two steps until convergence:

1. $\tilde{V}^k := M\tilde{V}^{(k-1)}$
2. Orthonormalize $\tilde{V}^{(k)}$

We write $V = [v_1, \dots, v_{\text{nev}}] \in \mathbb{C}^{n \times \text{nev}}$ as the desired eigenspace, and denote the matrix involved in the Subspace Iteration procedure by $\tilde{V}$, typically $\tilde{V} = [\tilde{v}_1, \dots, \tilde{v}_{\text{nev}+\text{nex}}] \in \mathbb{C}^{n \times (\text{nev}+\text{nex})}$ for some $\text{nex} > 0$. We use $\tilde{V}$ to denote both the matrix and the subspace spanned by it; a natural correspondance in the context of Subspace Iteration, as $\tilde{V}$ is guaranteed to be of full rank due to the orthonormalization procedure. A straight-forward optimization of the Subspace Iteration procedure involves multiplications with higher powers of M , which is possible since the subspace (spanned by) $\tilde{V}^{(k)}$ is the same as the subspace (spanned by) $M^k \tilde{V}^{(0)}$. Under mild assumptions we can show that $\tilde{V}$ converges to the eigenvectors corresponding to the eigenvalues of largest modulus:

Theorem 1 (Convergence of Subspace Iteration [87])

Let $\lambda_1, \dots, \lambda_m$ be the m dominant eigenvalues of M labeled in decreasing order of magnitude and assume that $|\lambda_j| > |\lambda_{j+1}|, 1 \leq j \leq m$. Let $V = [v_1, v_2, \dots, v_m]$ be the Schur vectors associated with $\lambda_j, 1 \leq j \leq m$ and E_j be the spectral projector associated with the eigenvalues $\lambda_1, \dots, \lambda_i$. Assume that for $\tilde{V}^{(k)} = M^k \tilde{V}^{(0)}$:

$$\text{rank}(E_j[\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_j]) = j \quad 1 \leq j \leq m.$$

Then the j^{th} column of $\tilde{V}^{(k)}$ converges essentially to $v_j, 1 \leq j \leq m$, in the sense

that there exists a sequence of signs $e^{i\theta_k}$ such that the sequence $e^{i\theta_k}\tilde{v}_j$ converges to v_j as k tends to infinity.

We refer to the columns of $\tilde{V}$ as vectors since these vectors converge to eigenvectors.

It can be shown [87] that in SI the eigenvectors converge linearly with a convergence ratio of $\frac{\lambda_{j+1}}{\lambda_j}$. The addition of a Rayleigh-Ritz procedure enhances the convergence of eigenvalues and eigenvectors in the case of Hermitian eigenproblems [93]: Eigenvectors converge with the slightly better ratio of $\frac{\lambda_{m+1}}{\lambda_j}$, while the eigenvalues converge quadratically. As a significant advantage, SI can accept approximate solutions $\tilde{V}$ instead of a random subspace, which can greatly reduce the number of required iterations. Many iterative eigensolvers, such as Krylov-type solvers, can not fully leverage approximate solutions to the same extent that SI can.

In the following we aim to further improve convergence of SI by preconditioning M to increase the distance between λ_{m+1} and the λ_j , $1 \leq j \leq m$. This chapter focuses on polynomial filtering methods, while the next chapter focusses on filtering via rational functions. Details of the polynomial filter are discussed in [Section 4.1.1](#).

We build on work by Rutishauser from 1969 and would like to spend a few words on the influence of the algorithm originally proposed by Rutishauser on the present work. The major aspects of SI can already be found in his paper [85], including insights on when SI should be used and how to determine a good set of initial input parameters. Moreover, all the key algorithmic ingredients are already present: polynomial filtering; QR factorization; search space projection; and a Ritz iteration. In addition, Rutishauser suggests:

1. the use of an initial Ritz iteration in order to estimate the bounds of the eigenvalue interval to be filtered;
2. the use of an initial iteration with a low polynomial degree to avoid large error in computing the extremal eigenvalues on the interval;

3. a small number of filtering iterations before performing the orthonormalization, as a trade-off between computational cost and excessive alignment of the vectors to the dominant eigenvector;
4. to carefully select the initial set of vectors since an unlucky choice can substantially influence convergence;
5. to limit the maximum value of the polynomial degree used in the filter to mitigate the effect of floating point errors;
6. a locking mechanism that stores converged eigenpairs and excludes them from further filtering.

Notable improvements to the method after Rutishauser include: (1) the Ritz procedure suggested by Rutishauser, applicable only to positive definite M , has been superceeded by the aforementioned Rayleigh-Ritz procedure; and (2) Rutishauser suggests to omit the computation of eigenpair residuals until the eigenvalues start to stagnate. In the following we present SI with computation of the residuals at each iteration. However, modern variations of this scheme do exist: for example, some solvers consider the trace of the reduced problem during the Rayleigh-Ritz procedure, see the discussion of [Algorithm 7](#).

The resulting modern SI algorithm is presented in [Algorithm 6](#) and includes Rutishauser's insights as well as the more modern improvements discussed above. We focus on the canonical exterior HEP as defined in [Definition 4](#). As input the SI procedure requires: the Hermitian matrix M ; the number of desired eigenpairs $\mathbf{nev}$; a search space increment $\mathbf{nex}$, which we discuss in the following; the tolerance for the eigenpair residuals $\mathbf{tol}$; a matrix of approximate eigenvectors $\tilde{V}$; and spectral estimates $\mu_1, \mu_{\mathbf{nev}+\mathbf{nex}}$, and b_{sup} . As output the SI procedure returns the $\mathbf{nev}$ extremal eigenpairs, corresponding to the smallest eigenvalues as well as the residuals of the eigenpairs. We use the $\text{RES}(y, \lambda) := \|My - \lambda y\|$ to denote the residual of the eigenpair (λ, y) .

The rationale behind the extended subspace $\tilde{V}$ is as follows. The convergence rate of SI is governed by $\frac{\lambda_{\mathbf{nev}+\mathbf{nex}+1}}{\lambda_j}$ for all $1 \leq j \leq \mathbf{nev} + \mathbf{nex}$. Note that increasing the number of vectors in $\tilde{V}$ increases the convergence of *all* vectors. Therefore,

Input: Hermitian matrix M , number of desired eigenpairs nev , search space increment nex ; threshold tolerance for residuals tol ; polynomial degree m ; matrix $\tilde{V} = [\tilde{v}_1, \dots, \tilde{v}_{\text{nev}+\text{nex}}]$; and spectral estimates $\mu_1, \mu_{\text{nev}+\text{nex}}$, and b_{sup} .

Output: nev extremal eigenpairs (Λ, V) with $\Lambda = [\lambda_1, \dots, \lambda_{\text{nev}}]$; $V = [y_1, \dots, y_{\text{nev}}]$; and residuals $[\text{RES}(y_1, \lambda_1), \dots, \text{RES}(y_{\text{nev}}, \lambda_{\text{nev}})]$

```

1:  $V \leftarrow []$ 
2: while  $\text{size}(V) < \text{nev}$  do
3:    $\tilde{V} \leftarrow \text{FILTER}(M, b_{\text{sup}}, \mu_1, \mu_{\text{nev}+\text{nex}}, \tilde{V}, m)$ 
4:    $Q \leftarrow \text{ORTHOGONALIZE}([V, \tilde{V}])$ 
    $\triangleright$  REDUCE TO ACTIVE SUBSPACE
5:    $Q \leftarrow [Q_{:, \text{size}(V)+1}, \dots, Q_{:, \text{nev}+\text{nex}}]$ 
6:    $(\tilde{V}, \tilde{\Lambda}) \leftarrow \text{RAYLEIGH-RITZ}(M, Q)$ 
7:   Compute the residuals  $\text{RES}(\tilde{V}, \tilde{\Lambda})$ 
8:    $(\tilde{V}, \Lambda, V) \leftarrow \text{DEFLATION \& LOCKING}(\tilde{V}, \tilde{\Lambda}, \text{RES}(\tilde{V}, \tilde{\Lambda}), V)$ 
9:    $\mu_1 \leftarrow \min([\Lambda, \tilde{\Lambda}])$ 
10:   $\mu_{\text{nev}+\text{nex}} \leftarrow \max([\Lambda, \tilde{\Lambda}])$ 

```

Algorithm 6: A Modern SI Algorithm

the search space increment, nex , can positively affect the convergence behavior of the SI procedure, even when we stop the procedure once the first nev eigenpairs have converged. This scheme improves convergence of the SI procedure at the cost of additional work in each iteration.

The spectral estimates are required for the `FILTER` subroutine, the preconditioner that uses spectral filtering to enhance the convergence of the SI procedure. Note that without such a filter we could not obtain the eigenpairs corresponding to the *smallest* eigenvalues, see [Theorem 1](#). Here $\text{size}(V)$ denotes the number of vectors in V .

The `FILTER` subroutine returns vectors that can easily become linearly dependent and require orthogonalization. We perform the orthogonalization of all vectors, including the already locked ones (we discuss locking later on). A typical implementation of the orthogonalization procedure is via a QR

1:	procedure RAYLEIGH-RITZ(M, Q)	
2:	$G = Q^* M Q$	▷ RAYLEIGH QUOTIENT
3:	$(G, \tilde{\Lambda}) \leftarrow \text{TSOLVE}(G)$	
4:	return $(Q, G, \tilde{\Lambda})$.	

Algorithm 7: Rayleigh-Ritz

factorization based on Householder reflectors (Algorithm 6, Line 4). Afterwards the locked vectors are discarded from Q and we continue with the active subspace of unconverged vectors.

We then invoke the RAYLEIGH-RITZ subroutine on the resulting orthonormal vectors Q , described in Algorithm 7. The Rayleigh-Ritz quotient G represents a projection of the eigenproblem M onto an active subspace approximating the sought after eigenspace. The reduced eigenproblem G is diagonalized, indicated as TSOLVE (Algorithm 7, Line 3). The computed eigenvectors $\tilde{W}$ together with the eigenvalues $\tilde{\Lambda}$ constitute the computed eigenpairs.

After the Rayleigh-Ritz step the tentative pairs $(\tilde{V}, \tilde{\Lambda})$ are then used to compute residuals. If the residuals are below the chosen threshold **tol**, they are accepted, locked in V and deflated from $\tilde{V}$ (see Section 4.2.1), otherwise they are used as new starting vectors in the **while** loop. The procedure terminates when at least **nev** eigenpairs have been locked.

4.1.1 Filtering via Chebyshev polynomials

We proceed to discuss the mathematical foundation of the Chebyshev filter: Chebyshev polynomials; a minimax theorem based on Chebyshev polynomials; and, finally, the FILTER routine itself.

Definition 12 (Chebyshev Polynomials)

A Chebyshev polynomial $C_m(t)$ of degree m is defined on the real axis as

$$C_m(t) = \cosh \left(m \cosh^{-1}(t) \right) \text{ for } t \in \mathbb{R} \quad (4.1)$$

For $[-1, 1]$ the definition reduces to $C_m(t) = \cos(m \cos^{-1}(t))$.

On the interval $[-1, 1]$ the Chebyshev polynomial of degree m is an oscillating function with $m - 1$ extrema; for $|t| > 1$ the function diverges rapidly, even for modest values of m . The polynomials $C_m(t)$ can also be defined through a three-terms recurrence relation

$$C_{m+1}(t) = 2tC_m(t) - C_{m-1}(t), \quad (4.2)$$

where the first two polynomials are given by $C_0(t) = 1$ and $C_1(t) = t$.

The asymptotic behavior of the polynomials for $|t| > 1$ can be quantified by expressing $C_m(t)$ as an explicit function of t . By defining

$$y := \cosh^{-1}(t) \quad \text{and} \quad \rho := \exp(y)$$

it is straightforward to show that t and ρ are related through a quadratic equation

$$\rho^2 - 2t\rho + 1 = 0$$

admitting ρ and ρ^{-1} as solutions. By convention we choose for a fixed t

$$|\rho| = \max \left| t \pm \sqrt{t^2 - 1} \right| \quad \text{and} \quad |\rho|^{-1} = \min \left| t \pm \sqrt{t^2 - 1} \right|$$

and plug them back in the definition of y so that the polynomial can be rewritten as

$$C_m(t) = \cosh(m \ln |\rho|) = \frac{|\rho|^m + |\rho|^{-m}}{2}. \quad (4.3)$$

For $|t| > 1$, $|\rho|$ is always larger than 1 and the leading asymptotic behavior of the polynomial for increasing $|t|$ is given by $C_m(t) \sim \frac{|\rho|^m}{2}$. In other words, outside of the interval $[-1, 1]$ $C_m(t)$ diverges as a polynomial of degree m .

The Chebyshev filter is based on the following theorem from function approximation theory:

Theorem 2 (Minimax Theorem [87, p. 109])

Let $|s| > 1$ and $\mathbb{P}_m$ denote the set of polynomials of degree smaller or equal to

m . Then the extremum

$$\min_{p \in \mathbb{P}_m, p(s)=1} \max_{t \in [-1,1]} |p(t)|$$

is reached by

$$p_m(t) := \frac{C_m(t)}{C_m(s)}.$$

This theorem states that $p_m(t)$ is the smallest polynomial of degree m inside the interval $[-1, 1]$ for any chosen point s outside of it. In the limit of large degrees these polynomials have an asymptotic limit

$$p_m(t) = \frac{C_m(t)}{C_m(s)} \approx \frac{1}{|\rho_s|^m}, \text{ for large } m$$

with $|\rho_s| = \max_{\{\pm: |s|>1\}} |s \pm \sqrt{s^2 - 1}|,$

where only the leading term of Equation (4.3) is kept together with $\sup_{t \in [-1,1]} C_m(t) = 1$. For a generic interval $[\alpha, \beta] \subset \mathbb{R}$, one defines the center of the interval $c = \frac{\alpha+\beta}{2}$ and the half-width of the interval $e = \frac{\beta-\alpha}{2}$, and builds a linear transformation $x(t) = c + e \cdot t$ mapping the interval $[-1, 1]$ onto $[\alpha, \beta]$. Using this transformation for t and s , we obtain Chebyshev polynomials for a general interval

$$p_m(x) := \frac{C_m\left(\frac{x-c}{e}\right)}{C_m\left(\frac{\gamma-c}{e}\right)} \approx \frac{1}{|\rho_\gamma|^m}, \text{ for large } m \quad (4.4)$$

with $|\rho_\gamma| = \max_{\{\pm: \gamma \notin [\alpha, \beta]\}} \left| \frac{\gamma - c}{e} \pm \sqrt{\left(\frac{\gamma - c}{e}\right)^2 - 1} \right|.$

Theorem 2, and recent numerical analysis results [25], suggest that the polynomial $p_m(t)$ can be exploited to enhance the components of the vectors $\tilde{V}$ along the eigenvectors corresponding to the sought after spectrum. In practice, by setting $\mu_1 := \gamma$, $\mu_{\text{nev}+\text{nex}} := \alpha$, and $b_{\text{sup}} := \beta$, one can show that

$$\min_{\tilde{v} \in \mathcal{V}} \|\tilde{v} - v_a\| \leq \eta_a |\rho_a|^{-m}, \quad (4.5)$$

where $\mathcal{V} = \text{span}\{p_m(M) \cdot \tilde{V}\}$ and η_a is a constant. The subspace spanned by

the filtered vectors always contains a vector which converges to the eigenvector v_a , corresponding to an eigenvalue $\lambda_a \in [\mu_1, \mu_{\text{nev}+\text{nex}}]$ with a convergence ratio of $|\rho_a|^{-1}$. Thus, it is possible to use an opportunely adapted 3-terms recurrence relation [87] that defines the action of the p_m polynomial onto the $\tilde{V}$ vectors,

$$p_m(\tilde{V}) = 2 \frac{\sigma_m}{e} (M - c\mathbb{I}) p_{m-1}(\tilde{V}) - \sigma_m \sigma_{m-1} p_{m-2}(\tilde{V}), \quad (4.6)$$

with $p_m(\tilde{V}) = p_m(M) \cdot \tilde{V}$, $\sigma_m = \left(\frac{2}{\sigma_1} - \sigma_{m-1}\right)^{-1}$, and $\sigma_1 = \left(\frac{\gamma-c}{e}\right)^{-1}$. The relation above is used in Algorithm 8. The output has the same number of vectors as the input, which are progressively aligned with the active search space, and have components orthogonal to the space converging to zero with ratios proportional to the ones reported in Equation (4.5).

```

1: procedure FILTER( $M, b_{\text{sup}}, \mu_1, \mu_{\text{nev}+\text{nex}}, \tilde{V}, m$ )
2:    $c = \frac{b_{\text{sup}} + \mu_{\text{nev}+\text{nex}}}{2}$ 
3:    $e = \frac{b_{\text{sup}} - \mu_{\text{nev}+\text{nex}}}{2}$ 
4:    $\sigma_1 = \frac{e}{\mu_1 - c}$ 
5:    $\tilde{W} \leftarrow \frac{\sigma_1}{e} (M - c\mathbb{I}_n) \tilde{V}$ 
6:    $\sigma \leftarrow \sigma_1$ 
7:   for  $i = 1, \dots, m$  do
8:      $\tau \leftarrow \frac{1}{\frac{2}{\sigma_1} - \sigma}$ 
9:      $\tilde{V} \leftarrow 2 \frac{\tau}{e} (M - c\mathbb{I}_n) \tilde{W} - \tau \sigma \tilde{V}$ 
10:    SWAP( $\tilde{V}, \tilde{W}$ )
11:     $\sigma \leftarrow \tau$ 
12:  return  $\tilde{W}$ 
    
```

Algorithm 8: Chebyshev Filter

4.1.2 Related Work

In the following, we outline related work. First, we start off with some historical context on the Subspace Iteration method itself. Then we proceed with work related to polynomial acceleration methods; work related to rational acceleration methods is discussed in Section 5.1.2.

Subspace Iteration

One of the earliest modern mentions of Subspace Iteration as a numerical solver was in 1957 by Bauer [11]. In 1969 and 1970 Rutishauser [85, 86] introduced many of the key ideas that are used even to this day. He presented RITZIT a fairly robust implementation of Subspace Iteration. Stewart introduced the Rayleigh-Ritz step in 1969 [93] improving convergence and obviating the need for special handling of non-positive definite matrices. Simultaneously, Subspace Iteration was generalized to non-Hermitian and non-symmetric eigenproblems [94]. Clint and Jennings (1970) [21] proposed a very similar algorithm, in the context of civil engineering. A review of the improved RITZIT algorithm can be found in Parlett's excellent book [74].

The development of iterative eigensolvers for the HEP shifted away from Subspace Iteration after the late 1970s, due to the ascent of the Lanczos method and its variations. In the middle of the 2000s Subspace Iteration methods saw a resurgence in popularity, particularly for problems arising in electronic structure theory: first, in the context of sparse eigenproblems [105, 103], and later on also for sequences of dense eigenproblems [14, 60, 8]. The comeback of Subspace Iteration can be attributed to (1) the emerging need to solve for the subspace without the need to resolve the single eigenpairs [105], and (2) the good performance of Subspace Iteration on sequences of correlated HEPs.

Polynomial Approximations of the Spectral Projector

For our purposes it suffices to consider *polynomial filtering* or *polynomial acceleration* to be an approximation of some spectral projector with polynomials, see Section 4.2.2. The use of polynomial filters for exterior HEPs is an old technique and very little has changed over the years thanks to the sturdy mathematical framework of polynomial approximation theory. Most of the polynomial filters are based on the min-max theorem, see Theorem 2. Alternatively, a Chebyshev approximation of the step functions can be used, but usually with inferior performance.

RITZIT from 1970 already contains a polynomial filter based on Chebyshev polynomials. The same filter persists in methods from the early 2000s, e.g.,

[105, 60, 9]. Indeed, we use the very same technique as RITZIT, with some improvements, in [Chapter 4](#). Recently, electronic structure codes use polynomials with much higher degrees to construct filters for the interior HEP, for example as part of the ESSEX project [76, 34].

Polynomial filtering is not restricted to Subspace Iteration. FILTLAN [31, 12] (2008) is an example of a Lanczos procedure with polynomial filtering. EVSL [62] (2016) is a very recent example of a Lanczos procedure with filtering capabilities; originally a thick-restart Lanczos procedure with polynomial filtering [62], the code now supports polynomial and rational filters [61].

4.2 The ChASE Solver

We proceed by introducing CHASE, our solver that improves on the modern SI algorithm described in [Algorithm 6](#). Starting off with an overview of the CHASE algorithm, we proceed to the subroutines not already described in the previous section. [Section 4.2.1](#) describes our approach to (1) vectors that have already converged as well as (2) ensuring that the `nev` converged vectors are indeed the smallest ones. Our degree optimization for the polynomial filter is described in [Section 4.2.2](#), and our approach to obtaining the spectral estimates is given in [Section 4.2.3](#).

The main routine of CHASE ([Algorithm 9](#)) requires a number of mandatory parameters as well as some optional ones. Besides the matrix M , standard inputs are: the number of required extremal eigenpairs `nev`; the minimum required tolerance for the eigenpair residuals `tol`; and the generic polynomial degree `deg`. CHASE uses `nev` to set up the minimal size of the subspace which is incremented by the value of an additional required parameter `nex`. The combined value `nev + nex` denotes the total size of the search subspace. A good choice of `nex` represents a trade-off between extra computation and an enhanced eigenpair convergence ratio. A bigger `nex` increases the distance $|\lambda_{\text{nev}+\text{nex}+1} - \lambda_{\text{nev}}|$ which in turn implies higher values for $|\rho_a|$ (see [Section 4.2.2](#)), but also enlarges the size of $\tilde{V}$ with a consequent increase of required FLOPs.

Input: Hermitian matrix M ; number of desired eigenpairs nev ; threshold tolerance for residuals tol ; initial polynomial degree deg ; search space increment nex ; **approx** and **optim** flags; matrix $\tilde{V} = [\tilde{v}_1, \dots, \tilde{v}_{\text{nev}+\text{nex}}]$; and estimates μ_1 and $\mu_{\text{nev}+\text{nex}}$.

Output: nev extremal eigenpairs (Λ, V) with $\Lambda = [\lambda_1, \dots, \lambda_{\text{nev}}]$; $V = [y_1, \dots, y_{\text{nev}}]$; and residuals $[\text{RES}(y_1, \lambda_1), \dots, \text{RES}(y_{\text{nev}}, \lambda_{\text{nev}})]$

▷ INITIAL CONSTANT DEGREE

1: $m_{1:\text{nev}+\text{nex}} \leftarrow \text{deg}$

▷ BOUND AND OPTIONAL INPUT

2: $(b_{\text{sup}}, \mu_1, \mu_{\text{nev}+\text{nex}}, \tilde{V}) \leftarrow \text{LANCZOS}(M, \text{approx})$

3: $V \leftarrow []$

4: **while** $\text{size}(V) < \text{nev}$ **do**

 ▷ USE ARRAY OF DEGREES m

5: $\tilde{V} \leftarrow \text{FILTER}(M, b_{\text{sup}}, \mu_1, \mu_{\text{nev}+\text{nex}}, \tilde{V}, m, \text{optim})$

6: $Q \leftarrow \text{ORTHOGONALIZE}([V, \tilde{V}])$

 ▷ REDUCE TO ACTIVE SUBSPACE

7: $Q \leftarrow [Q_{:, \text{size}(V)+1}, \dots, Q_{:, \text{nev}+\text{nex}}]$

8: $(\tilde{V}, \tilde{\Lambda}) \leftarrow \text{RAYLEIGH-RITZ}(M, Q)$

9: Compute the residuals $\text{RES}(\tilde{V}, \tilde{\Lambda})$

10: **Sort** $\text{RES}(\tilde{V}, \tilde{\Lambda}), \tilde{V}, \tilde{\Lambda}$ according to $\tilde{\Lambda}$

11: $(\tilde{V}, \Lambda, V) \leftarrow \text{DEFLATION \& LOCKING}(\tilde{V}, \tilde{\Lambda}, \text{RES}(\tilde{V}, \tilde{\Lambda}), V, \text{tol})$

12: $\mu_1 \leftarrow \min([\Lambda, \tilde{\Lambda}])$

13: $\mu_{\text{nev}+\text{nex}} \leftarrow \max([\Lambda, \tilde{\Lambda}])$

14: $c \leftarrow \frac{b_{\text{sup}} + \mu_{\text{nev}+\text{nex}}}{2}$

15: $e \leftarrow \frac{b_{\text{sup}} - \mu_{\text{nev}+\text{nex}}}{2}$

16: **for** $a = 1 \rightarrow \text{size}(\tilde{V})$ **do**

 ▷ COMPUTE POLYNOMIAL DEGREE

17: $m_a \leftarrow \text{DEGREES}(\text{tol}, \text{RES}(\tilde{V}_{:,a}, \tilde{\lambda}_a), \tilde{\lambda}_a, c, e)$

18: **end for**

19: **Sort** $\text{RES}(\tilde{V}, \tilde{\Lambda}), \tilde{V}, \tilde{\Lambda}, m$ according to m

Algorithm 9: The CHASE Algorithm; Lines in Blue Differ from Algorithm 6.

Ideally, the choice of `nex` guarantees a large spectral gap between the eigenvalues λ_{nev} and $\lambda_{\text{nev}+\text{nex}}$.

Two additional optional flags are `approx` and `optim`. The first is a flag to indicate whether the user provides CHASE with information about the approximate solution of the eigenproblem (e.g., when dealing with a sequence of correlated eigenproblems) or uses CHASE in isolation as a traditional black-box solver. When the `approx` flag is set to `true`, CHASE expects the arrays $\tilde{V}$ and Λ to hold approximate vectors and values respectively. The smallest and largest values in Λ are used as estimates μ_1 and $\mu_{\text{nev}+\text{nex}}$ for the lower and upper end of the sought after eigenspectrum so only these two values need to be populated. On the other hand, the entire set of vectors in $\tilde{V}$ are used in the Chebyshev filter as a pre-conditioner to accelerate convergence. In the output, both of these arrays are overwritten with the computed solution. If `approx` is set to `false`, the vectors used by the Chebyshev filter are overwritten by the LANCZOS routine with i.i.d. random vectors. Likewise, the LANCZOS routine computes the estimates μ_1 and $\mu_{\text{nev}+\text{nex}}$, see [Section 4.2.3](#).

The `optim` flag turns on the filter optimization for the Chebyshev filter. When `optim` is set to `false` the value stored in `deg` is passed to the FILTER routine each time it is called in [Algorithm 9](#). In case CHASE is called with the `optim` flag set to `true`, the value stored in `deg` is passed to the FILTER only during the first call, while for all successive calls in the `while` loop, the FILTER routine receives an array of computed optimized degrees ([Algorithm 9, Line 17](#)).

The array of initial vectors $\tilde{V}$ play a special role in SI. Already in the *ritzit* program [\[86\]](#) Rutishauser allows the user to provide a parameter to input “starting values to the iteration vectors”. He also recognizes that a poor choice of vectors orthogonal to the sought after eigenspace can negatively influence convergence. Consequently, a substitution is enforced in *ritzit*: after few subspace iterations the vector with inner-most Rayleigh-Ritz projection is replaced by a random vector. This expedient is motivated by the following argument, see [\[95, page 60\]](#): since any starting vector can be represented as $\tilde{v} = c_1 v_1 + c_\perp V_\perp$ then a measure of the distance of $\tilde{v}$ from the dominant eigenvector $\tilde{x}_1$ is given by the ratio $\frac{c_\perp}{c_1}$. One can also reverse the argument in

```

1: procedure DEFLATION & LOCKING( $\tilde{V}$ ,  $\tilde{\Lambda}$ ,  $\text{RES}(\tilde{V}, \tilde{\Lambda})$ ,  $\text{tol}$ )
    $\triangleright$  FOR EVERY UNCONVERGED EIGENPAIR
2:   for  $a = 1, \dots, (\text{nev} + \text{nex} - \text{size}(\Lambda))$  do
3:     if  $\text{RES}(\tilde{V}_{:,a}, \tilde{\lambda}_a) > \text{tol}$  then
4:       return ( $\tilde{V}$ ,  $\Lambda$ ,  $V$ )
5:      $\Lambda.\text{PUSH}(\tilde{\lambda}_a)$  and  $V.\text{PUSH}(\tilde{V}_{:,a})$ 
6:      $\tilde{V}.\text{REMOVE}(\tilde{V}_{:,a})$  and  $\tilde{\Lambda}.\text{REMOVE}(\tilde{\lambda}_a)$ 
7:   return ( $\tilde{V}$ ,  $\Lambda$ ,  $V$ )

```

Algorithm 10: Deflation and Locking

order to understand the importance of using an approximate solution: Having a starting vector, which is already a good approximation to the desired eigenvector, can substantially accelerate the solution. This specific observation is at the heart of the CHASE algorithm—it targets exactly those sequences of correlated eigenproblems where the solution of the ℓ^{th} problem provides a good set of starting vectors for the $\ell + 1^{\text{st}}$ problem. The end result is a magnification of CHASE’s convergence as a function of the sequence index [26].

4.2.1 Deflation and Locking

Filtering eigenpairs $(\tilde{V}_{:,a}, \tilde{\lambda}_a)$ that already satisfy the $\text{RES}(\tilde{V}_{:,a}, \tilde{\lambda}_a) < \text{tol}$ condition would unnecessarily decrease the value of their residual and substantially increase the number of FLOPs performed by CHASE. For this simple reason it is good practice to remove the converged vectors and values from the arrays $\tilde{V}$ and $\tilde{\Lambda}$ (Algorithm 10, Line 6) and store them in the respective output arrays V and Λ (Algorithm 10, Line 5). These two operations go under the conventional names of deflation and/or locking, they are included in the CHASE algorithm (Algorithm 9, Line 11). Their overall effect is to reduce the FLOPs-count by converging the whole subspace in cumulative chunks.

While the advantage is evident, this procedure can introduce convergence issues when the tol parameter is several orders of magnitude larger than the machine precision. A rather high threshold tolerance for the residual may cause locking eigenpairs quite early on in the execution of CHASE. The locked

vectors may include “directions” along the sought after eigenspace which are no longer accessible by the remaining search space spanned by the vectors left in $\tilde{V}$. This happens when $\tilde{V}$ is reorthogonalized against V (Algorithm 9, Line 6) at the next execution of the **while** loop. This phenomenon, known as *early locking* can, in rare cases, cause the stagnation of successive eigenpairs and the consequent lack of convergence for CHASE. A fail-safe mechanism for the early locking problem was proposed by Stathopoulos [92]. In the CHASE algorithm we have not introduced such a mechanism but suggest, as a rule of thumb, to avoid selecting $\text{tol} > 10^{-8}$ in double precision and $\text{tol} > 10^{-4}$ in single precision. Alternatively, the selection of a lower accuracy for the QR procedure can help to avoid stagnation in most practical cases.

It may sometimes happen—especially in the case of tight clusters of eigenvalues—that some non-consecutive eigenpairs are declared converged, i.e. that an eigenpair (λ_i, v_i) converges before all eigenpairs (λ_j, v_j) with $j < i$ converge. This is not a problem if the eigenpairs are within the desired eigenspace of size `nev`. Because the size of the active subspace is larger than `nev`, it is not uncommon that some of these converged non-consecutive eigenpairs are outside the sought after spectrum but end up locked in the `nev` required eigenpairs. For this reason, as part of the DEFLATION & LOCKING procedure, we sort the approximate eigenpairs and their corresponding residuals according to the computed approximate eigenvalues $\tilde{\Lambda}$ (Algorithm 10, Line 10). Thanks to the fact that eigenvalues converge much faster than the eigenvectors (quadratically vs. linearly), in all practical cases sorting the approximate pairs guarantees that only consecutive converged eigenpairs are locked and deflated.

4.2.2 Filter Degree Optimization

In the following we give a description of how to optimize the filtering degree, the corresponding optimizing algorithm, and its implementation in the Chebyshev filter. We conclude with some numerical results supporting our claim of increased performance.

Conventional polynomial filtering uses a fixed degree of the Chebyshev

```

1: procedure DEGREES(tol, RES( $\tilde{V}_{:,a}, \tilde{\lambda}_a$ ),  $\tilde{\lambda}_a$ ,  $c$ ,  $e$  )
2:    $t = \frac{\tilde{\lambda}_a - c}{e}$ 
3:    $|\rho| = \max \left\{ \left| t - \sqrt{t^2 - 1} \right|, \left| t + \sqrt{t^2 - 1} \right| \right\}$ 
    $\triangleright$  DEGREE REQUIRED FOR CONVERGENCE
4:    $\text{deg} = \left\lceil \left| \log \frac{\text{Res}(\tilde{V}_{:,a}, \tilde{\lambda}_a)}{\text{tol}} / \log |\rho| \right| \right\rceil$ 
    $\triangleright$   $\text{degExtra} \approx 2$ 
5:    $\text{deg} = \min \{ \text{deg} + \text{degExtra}, \text{degMax} \}$ 
    $\triangleright$  ENSURE DEG IS EVEN
6:   return  $\text{deg} + \text{mod}(\text{deg}, 2)$ 

```

Algorithm 11: Degree Optimization

filter, as in [Algorithm 6](#). Any fixed choice of the degree causes two separate problems: First, eigenvectors close to convergence are filtered more than necessary, resulting in unnecessary work. On the other hand, eigenvectors far from convergence are not filtered enough requiring more iterations of the **while** loop in [Algorithm 6](#). Choosing the degree of the filter becomes a trade-off between the number of required iterations and excessive filtering of nearly converged vectors. This trade-off can be avoided entirely by *optimizing the degree* of the filter separately for each vector, see [Algorithm 12](#). This section briefly discusses the derivation of the minimal polynomial degree for each vector, and the necessary algorithmic changes. CHASE's degree optimization can save up to 20% of required FLOPs.

[Equation \(4.5\)](#) suggests that eigenvectors having residuals with a given level of tolerance can be obtained using a Chebyshev filter of minimal degree $m_{\min}$ ¹ for each of the vectors $\tilde{V}_{:,a}$. This minimal value for m can be computed using a simple strategy. One executes the very first instance of the **while** loop in [Algorithm 9](#) using an initial low degree m_0 (between 8 and 15) for filtering all the $\tilde{V}$ ([Algorithm 9, Line 1](#)). At the end of the first loop, all residuals of the approximate eigenpairs, $\text{RES}(\tilde{V}_{:,a}, \tilde{\lambda}_a)$, are computed ([Algorithm 9, Line 9](#)). In the following discussion of the filter's influence on the residuals, we refer

¹ Mention of a tailored polynomial degree was already suggested in [85] and [95, p.395]. A rigorous analysis is provided in [25].

to $\tilde{V}$ *before* filter application as $\tilde{V}^{(0)}$ and *after* filter application as $\tilde{V}^{(1)} := p_{m_a}(M) \cdot \tilde{V}_{:,a}^{(0)}$; $\tilde{\lambda}_a$ remains fixed during the filter application. Since the curve describing the residuals of the approximate solutions as a function of m is a function of the convergence ratio for each single eigenvector [25], the next filtering step within the **while** loop produces a residual equal to

$$\text{RES}(\tilde{V}_{:,a}^{(1)}, \tilde{\lambda}_a) = \frac{\text{RES}(\tilde{V}_{:,a}^{(0)}, \tilde{\lambda}_a)}{|\rho_a|^{m_a}}.$$

The requirement that $\text{RES}(\tilde{V}_{:,a}^{(1)}, \tilde{\lambda}_a) \leq \text{tol}$ immediately translates into the condition

$$m_a^{(1)} \geq \frac{\log \left| \frac{\text{RES}(\tilde{V}_{:,a}^{(0)}, \tilde{\lambda}_a)}{\text{tol}} \right|}{\log |\rho_a|}. \quad (4.7)$$

Because $m_{\min}$ depends on $|\rho_a|$, to each approximate eigenvector requires a different minimal degree value m_a . When using a filter with a polynomial degree tailored to the specific approximate eigenpair, some eigenpairs may already converge at the second filtering step. Moreover the filtering step will perform (close to) the minimal amount of matrix-vector operations necessary for the eigenpair to converge. In other words, the computational cost of the filtering step has been optimized.

4.2.2.1 Implementation notes

The procedure for obtaining the optimal degree for a single approximate eigenpair is outlined in Algorithm 11. In addition to the center c and half-width e of the interval $[\mu_{\text{nev}+\text{nex}}, b_{\text{sup}}]$, the algorithm requires the tolerance threshold tol below which eigenpairs are declared converged, the eigenvalue $\tilde{\lambda}_a$, and the residual for the unconverged eigenpair $\text{RES}(\tilde{V}_{:,a}, \tilde{\lambda}_a)$. The degree estimate satisfying the inequality in Equation (4.7) is incremented by a small amount, $\text{degExtra} \approx 2$, to account for rounding errors in the degree computation. In addition, we introduce an upper threshold degMax for the largest admissible value of the minimal degree m_a , as too large of a degree causes the filtered subspace $\tilde{V}$ to become rank deficient. In such cases orthogonalization via a

```

1: procedure FILTER( $M, b_{\text{sup}}, \mu_1, \mu_{\text{nev}+\text{nex}}, \tilde{V},$ 
   sorted  $m = [m_1, \dots, m_{\text{size}(\tilde{V})}]$ )
2:    $c = \frac{b_{\text{sup}} + \mu_{\text{nev}+\text{nex}}}{2}$ 
3:    $e = \frac{b_{\text{sup}} - \mu_{\text{nev}+\text{nex}}}{2}$ 
4:    $\sigma_1 = \frac{e}{\mu_1 - c}$ 
5:    $\tilde{W} \leftarrow \frac{\sigma_1}{e} (M - c \mathbb{I}_n) \tilde{V}$ 
6:    $\sigma \leftarrow \sigma_1$ 
7:    $s \leftarrow 1$ 
8:   for  $i = 2, \dots, m_{\text{size}(\tilde{V})}$  do
9:      $\tau \leftarrow \frac{1}{\frac{2}{\sigma_1} - \sigma}$ 
10:     $\tilde{V}_{:,s:\text{end}} \leftarrow 2 \frac{\tau}{e} (M - c \mathbb{I}_n) \tilde{W}_{:,s:\text{end}} - \tau \sigma \tilde{V}_{:,s:\text{end}}$ 
11:    SWAP( $\tilde{V}, \tilde{W}$ )
12:     $\sigma \leftarrow \tau$ 
13:    while  $m_s \leq i$  do
14:       $s \leftarrow s + 1$ 
15:    end while
16: return  $\tilde{W}$ 

```

Algorithm 12: Chebyshev Filter for Optimized Even Degrees

QR factorization introduces numerical instabilities, resulting in a systematic increase of the residuals of the non-converged vectors [25]. In our experience a maximum degree of 36 (for double precision) effectively prevents the appearance of divergent residuals (Algorithm 11, Line 5).

Including the optimization of the filter degree in the CHASE algorithm requires some changes and additions to the Subspace Iteration algorithm which are color coded in Algorithm 9. Degree optimization calls for knowledge of the residuals (Algorithm 9, Line 9), and as such is only possible from the second iteration of the **while** loop onwards. Hence, the initial filtering is carried out with a low polynomial degree (Algorithm 9, Line 1). A high performance implementation of the Chebyshev filter involves additional changes to Algorithm 12. In the matrix-matrix multiplication (expressed as calls to the BLAS-3 HEMM kernel) in Lines 5 and 10, each vector of $\tilde{V}$ is filtered with an optimized and generally distinct polynomial degree m_a . Consequently, the

multiplication has to be implemented to omit vectors already filtered up to the desired degree from $\tilde{V}$ and $\tilde{W}$. To maximize performance, it is preferable to keep a single HEMM call regardless of which vectors have to be omitted. Sorting $\tilde{V}$ according to the polynomial degree (Algorithm 9, Line 19) forces all matrix-matrix multiplications to be on contiguous vectors and thus can be achieved with a single call to the HEMM routine. Each time the smallest entry m_s in the array of degrees becomes larger than the iterator i of the **for** loop (Algorithm 12, Line 8), the relative vectors are removed from the next HEMM call and so on and so forth, until the iterator equals $m_{\text{size}(\tilde{V})}$ (Algorithm 12, Line 13 to Line 15).

We conclude with a minor but important remark: the double buffering in Line 10 of Algorithm 11 (used to implement Equation (4.6)) causes any vector filtered with an odd degree to be saved to $\tilde{V}$. However, since the FILTER routine returns the $\tilde{W}$ array, any such vector would need to be copied from $\tilde{V}$ to $\tilde{W}$. We obviate the need for data movement by constraining the returned array of degrees to contain only even values (Algorithm 11, Line 6).

4.2.2.2 Optimization and performance

Given the residuals $\text{RES}(\tilde{V}_{:,a}, \tilde{\Lambda}_a)$ for each eigenpair, the calculation of the array of optimal degrees is quite inexpensive (see Algorithm 11), and has an overall positive impact on CHASE’s performance. Minimizing the FLOPs executed by the FILTER routine accounts for a reduced time-to-solution. How much the minimization influences CHASE’s performance depends on two distinct factors: (1) the choice of the initial polynomial degree—which, in the absence of degree optimization, is the polynomial degree used in each call of the FILTER procedure for all vectors $\tilde{V}$ —and (2) the context in which CHASE is employed. For the latter, we show that the impact on its performance is maximized when CHASE is used to tackle sequences of eigenvalue problems.

For the experiments in this section we use a sequence of eigenproblems obtained from a DFT simulation using the FLEUR code [106], and labeled NaCl. The matrix size of the eigenvalue problems is $n = 3893$ and we seek the

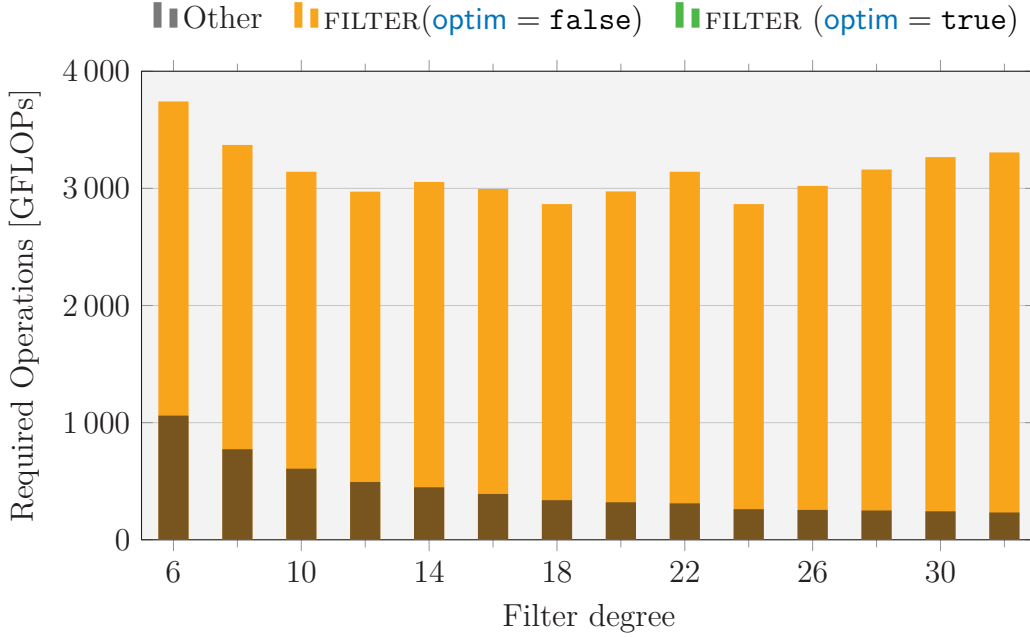


Figure 4.1: *Non* Degree-optimized CHASE: GFLOPs for CHASE without (`optim = false`) degree optimization

smallest `nev` = 256 eigenpairs, corresponding to about 7% of the spectrum. For CHASE’s `tol` and `nex` parameters we select values of 10^{-10} and 26, respectively. The presence of a spectral gap between the first `nev` and the next `nex` eigenvalues causes CHASE to converge quite well for each problem in the sequence.

An appropriate choice of the initial/constant degree is particularly important in order to obtain high performance. As discussed at the beginning of the section, choosing a large degree results in “over-filtering” eigenvectors that are already converged. Conversely, a small polynomial degree avoids over-filtering at the cost of additional iterations. This trade-off is visualized in Figure 4.1, which plots the (constant) filter degree against the GigaFLOPs (GFLOPs) required by CHASE to converge. These results are obtained with the first problem in the `NaCl` sequence and a random i.i.d. collection of vectors $\tilde{V}$ as input. The GFLOPs not spent in the Chebyshev filter are indicated as a darker shade in the bar plot; this includes, for example, the Rayleigh-Ritz procedure,

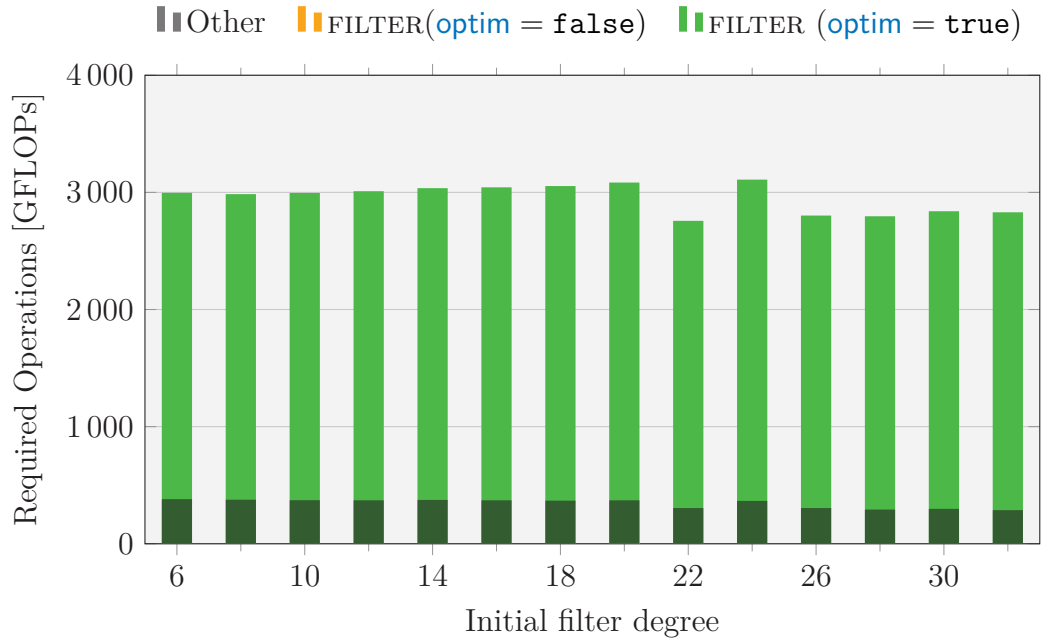


Figure 4.2: Degree-optimized CHASE: GFLOPs for CHASE with (`optim = true`) degree optimization

the QR factorization, and the computation of the residuals. For low polynomial degrees more GFLOPs are spent outside of the filter, indicating a larger number of iterations of the **while** loop in [Algorithm 9](#). The total number of FLOPs forms a rough “bathtub” shape, with the optimum filter degree between 18 and 24: a high polynomial degree causes over-filtering and increases the total number of required GFLOPs.

[Figure 4.2](#) shows the same experiment setup as [Figure 4.1](#), but with an optimized array of degrees m . Here the degree specified on the x-axis is the initial degree, i.e. the polynomial degree `deg` used in the first CHASE iteration. As a result, the number of GFLOPs required for CHASE to converge is practically independent of the initial degree. The degree optimization acts as a sort of stabilizer of the FLOPs count against variations of `deg`: the choice of the initial degree has only a minimal impact on the eigensolver’s performance. As expected, CHASE needs fewer GFLOPs to converge for the majority of the

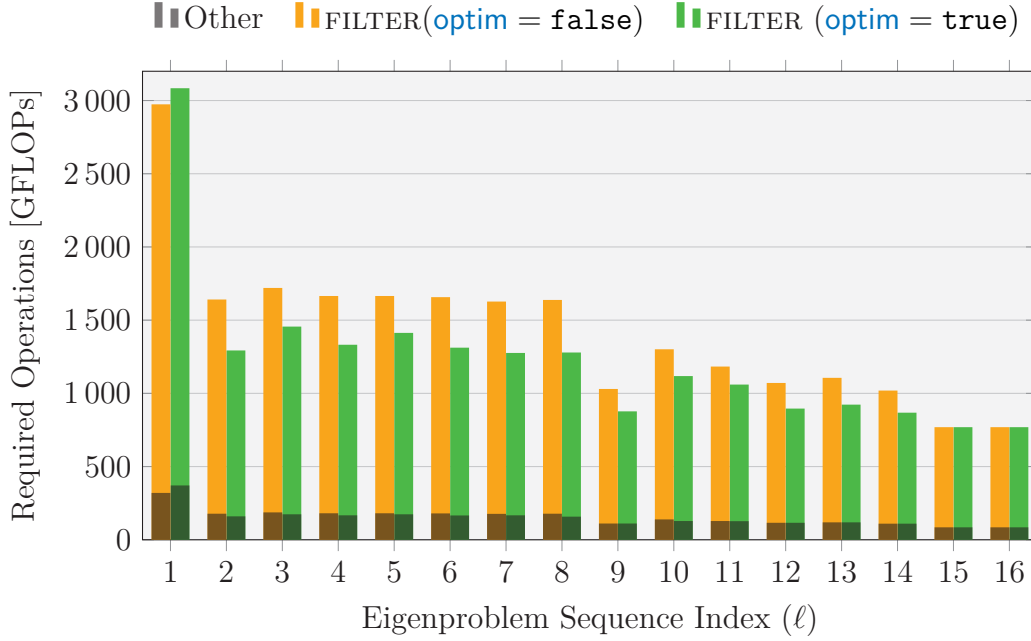


Figure 4.3: Optimized ($\text{optim}=\text{true}$) vs. Non-optimized ($\text{optim}=\text{false}$) FILTER
(NaCl sequence, $n = 3893$, $\text{nev} = 256$, $\text{deg} = 20$)

deg values, implying that degree optimization yields, in most cases, a faster solver.

On sequences of eigenvalue problems the impact of degree optimization is even more dramatic. Figure 4.3 compares the GFLOPs required for CHASE to converge with, and without, degree optimization. The data in the plot refers to the NaCl sequence, where the first matrix $M^{(1)}$ is started with a random i.i.d. set of vectors $\tilde{V}$. For the rest of the sequence, the matrix $M^{(\ell)}$ is solved by providing $V^{(\ell-1)}$ (the solution of the $(\ell - 1)^{\text{st}}$ problem) to CHASE. We select an initial value of $\text{deg} = 20$, a choice that favors the use of CHASE with a constant degree. Accordingly, CHASE with degree optimization performs slightly worse than CHASE with constant degree for the first problem of the sequence. For most of the remaining problems, the degree optimization saves FLOPs as compared to a constant degree, in some cases up to 25%. Notable exceptions are the last two problems with $\ell = 15, 16$; these problems converge

within a single iteration and thus there is no practical difference in the execution of CHASE with optimized or with constant degrees. As in the previous figure, the darker area indicates GFLOPs spent in CHASE routines other than the polynomial filter: for this area there is very little difference between the two versions of CHASE, as the degree optimization does not directly influence the number of **while** loop iterations in this sequence. A more in-depth analysis of CHASE's performance for sequences is provided in [Section 4.4.1](#).

4.2.3 Spectral Estimates

```

1: procedure LANCZOS( $M$ , approx)
    $\triangleright k$  Lanczos steps
2:    $MU = UT_k + f_k e_k^\top$ ,  $T_k = Z^* \tilde{\Lambda}_k Z$ ,  $\tilde{\Lambda}_k = \text{Diag}[\tilde{\lambda}_1, \dots, \tilde{\lambda}_k]$ 
3:    $b_{\text{sup}} = \|f_k\|_2 + \max[\tilde{\lambda}_1, \dots, \tilde{\lambda}_k]$ 
4:   if approx then
5:     return  $b_{\text{sup}}$ 
6:   for  $j = 1, \dots, \text{nvec}$  do
    $\triangleright k$  Lanczos steps
7:      $MU^{[j]} = U^{[j]} T_k^{[j]} + f_k^{[j]} e_k^\top$ ,  $T_k^{[j]} = (Z^{[j]})^H \tilde{\Lambda}_k^{[j]} Z^{[j]}$ 
    $\triangleright$  See [65, Section 3.2]
8:     Construct DoS:  $\tilde{\phi}(t) = \frac{1}{\text{nvec}} \sum_{j=1}^{\text{nvec}} \sum_{i=0}^k |Z_{1,i}^{[j]}|^2 g_\sigma(t - \tilde{\lambda}_i^{[j]})$ 
9:     Find  $\bar{t}$  such that  $\int_{-\infty}^{\bar{t}} \tilde{\phi}(t) dt \approx \frac{\text{nev} + \text{nex}}{N}$ 
10:     $\mu_{\text{nev} + \text{nex}} = \bar{t}$   $\mu_1 = \min_{j=1} \tilde{\lambda}_1^{[j]}$ 
11:    Intersperse  $U^{[1]} Z^{[1]}$  corresponding to  $\tilde{\lambda}^{[1]} < \mu_{\text{nev} + \text{nex}}$  into  $\tilde{V}$ 
12:   return  $\mu_1$ ,  $\mu_{\text{nev} + \text{nex}}$ ,  $b_{\text{sup}}$ , and  $\tilde{V}$ .

```

Algorithm 13: Lanczos Procedure for Parameter Estimates. The calculated Density of States $\tilde{\phi}(t)$ yields a sufficiently good approximation of $\lambda_{\text{nev} + \text{nex}}$.

(k and nvec are CHASE parameters.)

The spectral estimates μ_1 , $\mu_{\text{nev} + \text{nex}}$, and b_{sup} are of critical importance for both correctness and performance of CHASE. The estimates are used in two separate but related procedures: the Chebyshev filter ([Algorithm 12](#)) and the degree optimization ([Algorithm 11](#)). Given the value of μ_1 , the filter

is constructed to suppress the spectrum between $\mu_{\text{nev}+\text{nex}}$ and b_{sup} . In the degree optimization the spectral estimates are used to compute the ratios of convergence $|\rho_a|^{-1}$. In this section we illustrate how to obtain the spectral estimates, both with and without the availability of an approximate solution. We discuss the performance implications in the choice of the bounds, as well as the cost of obtaining accurate estimates.

μ_1 and $\mu_{\text{nev}+\text{nex}}$ are approximations of the first and $(\text{nev} + \text{nex})^{\text{th}}$ eigenvalue, respectively. After the first iteration of the **while** loop, new approximations for these values can be obtained from the Ritz values computed in [Algorithm 7](#). Even when initial values for μ_1 and $\mu_{\text{nev}+\text{nex}}$ are inaccurate, CHASE capitalizes on the quadratic convergence of the Ritz values and self-corrects μ_1 and $\mu_{\text{nev}+\text{nex}}$ after only a few iterations. In spite of its robustness, this self-correcting behavior is sometimes detrimental of the overall algorithm efficiency. For instance, over-correcting the estimate for $\mu_{\text{nev}+\text{nex}}$ makes the filter less effective, and similarly the degree optimization does not produce accurate results. Contrary to μ_1 and $\mu_{\text{nev}+\text{nex}}$, b_{sup} is always computed at the beginning of the CHASE algorithm and must be an upper bound of the largest eigenvalue, otherwise CHASE will fail because some of the largest eigenvalues are not filtered. For all the reasons above, we wish to obtain an approximation of μ_1 , $\mu_{\text{nev}+\text{nex}}$, and b_{sup} that is accurate and yet inexpensive.

In the case of sequences of correlated eigenvalue problems, μ_1 and $\mu_{\text{nev}+\text{nex}}$ need to be approximated only for the first problem of the sequence. Successive problems can use the approximate spectrum of $M^{(\ell-1)}$ to obtain the estimates for the ℓ^{th} problem. However, in order to ensure that b_{sup} is an upper bound of the spectrum it must be recomputed for each new problem. Estimates for b_{sup} with a varying degree of accuracy can be inexpensively computed using the method in [\[104\]](#). Estimating an interior eigenvalue is quite more complicated and expensive than estimating the extrema of the spectrum. To obtain an estimate for $\mu_{\text{nev}+\text{nex}}$, which is usually in the first quarter of the spectrum, we propose to employ an approximation of the spectral density, sometimes referred to as Density of States (DoS) [\[65, Section 3.2.1\]](#). The DoS function can be interpreted as a spectral probability density function. In its regularized form it

is given by

$$\phi(t) = \frac{1}{n} \sum_{j=1}^n g_{\sigma}(t - \lambda_j), \quad (4.8)$$

where g_{σ} is the Gaussian kernel of standard deviation σ , and controls the smoothness of the resulting function.

Naturally, we want to approximate ϕ without requiring all eigenvalues of the matrix. To this end, let $MU = UT_k + f_k e_k^{\top}$ be the result of k steps of the Lanczos method with a starting vector ν_0 . Let us further write the tridiagonal symmetric matrix T_k as $Z^H \tilde{\Lambda}_k Z$, with $\tilde{\Lambda}_k = \text{Diag}[\tilde{\lambda}_1, \dots, \tilde{\lambda}_k]$. It can be shown that $\sum_{j=1}^k |Z_{1,j}|^2 g_{\sigma}(t - \tilde{\lambda}_j)$ approximates a weighted DoS, where the weighting is related to the expansion coefficients of ν_0 in the basis of M . By repeating the Lanczos method `nvec` times with different random vectors ν_0 and taking the average of them, we reduce the effect of the weighting

$$\tilde{\phi}(t) = \frac{1}{\text{nvec}} \sum_{i=1}^{\text{nvec}} \sum_{j=1}^k |Z_{1,j}^{[i]}|^2 g_{\sigma}(t - \tilde{\lambda}_j^{[i]}). \quad (4.9)$$

Our approximation of $\mu_{\text{nev}+\text{nex}}$ is a value $\bar{t}$, such that $\int_{-\infty}^{\bar{t}} \tilde{\phi}(t) dt \approx \frac{\text{nev}+\text{nex}}{n}$. [Algorithm 13](#) illustrates the complete procedure. b_{sup} is computed via k initial Lanczos steps and the formula given in [104, Equation (2.5)] ([Algorithm 13, Line 3](#)). If the other two spectral approximations are required, we perform `nvec` additional Lanczos procedures and form the DoS from [Equation \(4.9\)](#) ([Algorithm 13, Line 8](#)). Finally, we obtain $\mu_{\text{nev}+\text{nex}}$ by integrating over the spectral density. For the sake of simplicity, we obtain μ_1 not via the DoS, but rather by using the smallest Ritz value of the Lanczos process. Since UZ are the approximate eigenvectors of M , we use those vectors corresponding to the Ritz values smaller than $\mu_{\text{nev}+\text{nex}}$ and intersperse them with random i.i.d. vectors in $\tilde{V}$.

The [Algorithm 13](#) has three parameters: the number of steps k for each Lanczos procedure; the number of different starting vectors `nvec`; and the regularization parameter σ for the DoS. They regulate a trade-off between accuracy and computational cost. When $\mu_{\text{nev}+\text{nex}}$ is required the LANCZOS subroutine

may account for about 10% of CHASE total execution time. Nevertheless, less accurate spectral estimates degrade the effectiveness of the filter by far more than 10%. A good parameter choice must trade-off a better convergence rate of the SI against the additional time spent in the LANCZOS routine. This is complicated by the fact that the relationship between the quality of the bounds and the subspace convergence is not straightforward. For small k , the computational cost is dominated by the $k \cdot \text{nvec}$ matrix-vector multiplications inside the Lanczos method. The cost for solving the tridiagonal eigenproblem, as well as the construction of $\tilde{\phi}$ and its integration, are negligible. A larger choice of k would have a number of advantages, namely better accuracy of the DoS inside of the spectrum and more approximate eigenvectors UZ (Algorithm 13, Line 11). However, in most cases the increased orthogonalization cost of large k does not pay off in terms of better convergence of the SI. In double precision, we suggest the default value $k = 25$. The work by Lin, Saad, and Yang [65] uses values between 40 and 100, however their goal is an accurate approximation of the entire spectral density. Since the Lanczos method tends to approximate the extrema of the spectrum rather quickly, a smaller choice of k is quite reasonable to estimate $\mu_{\text{nev}+\text{nex}}$.

The regularization parameter σ controls the width of the Gaussians g_σ and thus the smoothness of the spectral density. Since k essentially determines the number of Gaussians, there is a strong connection between the two parameters. We propose a standard value of $\sigma = 0.025 \cdot |b_{\text{sup}} - \mu_1|$. Note that σ needs to be scaled to the spectral radius of M . Too small a value for nvec results in a poor average and thus a distorted spectral density. Conversely, too large of a value for nvec is needlessly expensive without a noticeable benefit for CHASE. We recommend a value between 4 and 10 for nvec , with the default set to 4. Naturally, for CHASE's implementation the nvec matrix-vector multiplications are coalesced into a single HEMM call, significantly increasing performance.

Fine tuning the method we described in this section took considerable effort. We omit the complicated and tedious details due to space constraints. In addition the CHASE code implements a number of additional heuristics that result in values for $\mu_{\text{nev}+\text{nex}}$ that work well on the vast majority of cases. To

sum up, our implementation almost always performs better than a naïve choice of $\mu_{\text{nev}+\text{nex}}$.

4.3 Using ChASE

The linear algebra routines within CHASE, such as the matrix-matrix multiplication and orthogonalization procedures, are decoupled from the algorithm proper. [Section 4.3.1](#) explains the underlying modularity concept. This separation of concerns allows for different parallelization approaches. In [Section 4.3.2](#) we present two such approaches, CHASE-MPI and CHASE-Elemental, that ship with the CHASE library. [Section 4.3.3](#) introduces the configuration parameters of CHASE, as well as a code snippet that illustrates CHASE’s usage in application codes².

4.3.1 Decoupling Kernels From ChASE

CHASE relies on a modest number of numerical kernels whose calls execute almost all floating-point operations.³ Because of its simple structure there are clear advantages in the separation of the numerical kernels from the algorithm implementation. Reverse Communication Interfaces (RCI) [\[30\]](#) are a time-honored method to decouple some operation(s), like the matrix-vector multiplication, from a given algorithm. A popular and established example is the ARPACK library [\[58\]](#), but RCIs are also used in more recent software packages such as FEAST [\[79\]](#). RCIs tend to produce complicated code because information must be passed via return values. There are, however, modern alternatives to RCIs. In CHASE the linear algebra kernels are separated from the main algorithm through the use of an object-oriented software interface. This approach is similar to the one realized in ANASAZI [\[7\]](#), although our implementation is simpler and leads to a number of immediate benefits:

² To align our notation with the CHASE code snippets, we refer to the rank of matrix M with the letter N in this section only.

³ The only notable exception is the calculation of the Chebyshev coefficients and the polynomial degree.

Listing 4.1: Class Interface that Abstracts the CHASE Algorithm from the Numerical Kernels.

```

1 using T = std::complex<double>;
2 class Chase {
3 public:
4     virtual void Start() = 0; // Algorithm 9 Line 1
5     virtual void End() = 0; // Algorithm 9 Line 19
6     virtual void Resd(double *ritzv, // Algorithm 9 Line 9
7         double *resd, size_t fixednev) = 0;
8     virtual void Lock(size_t new_converged) = 0; // Algorithm 10 Line 5
9     virtual void QR(size_t fixednev) = 0; // Algorithm 9 Line 6
10    virtual void RR(double *ritzv, // Algorithm 7 Line 1
11        size_t block) = 0;
12    virtual void HEMM(size_t nev, T alpha, // Algorithm 12 Line 10
13        T beta, size_t s) = 0;
14    virtual void Lanczos(size_t k, // Algorithm 13 Line 3
15        double *upperb) = 0;
16    virtual void Lanczos(size_t M, size_t j, // Algorithm 13 Line 7
17        double *upperb, double *ritzv,
18        double *Tau, double *ritzV) = 0;
19    virtual void LanczosDos(size_t idx, // Algorithm 13 Line 11
20        size_t m, T *ritzVc) = 0;
21    virtual void Shift(T c, bool isunshift = false) = 0; //  $M - cI_n$ 
22    virtual void Swap(size_t i, size_t j) = 0; // Swap  $\tilde{V}_{:,i}$  and  $\tilde{V}_{:,j}$ 
23
24    /* omitted Getters */
25 };

```

First, CHASE can be easily integrated into existing codes thanks to the relative simplicity of the software interface. For instance, the low-level kernels can be implemented according to an existing distribution of the matrix elements of M so as to avoid the need to redistribute data. Second, CHASE can easily exploit existing linear algebra libraries such as BLAS and LAPACK as well as GPU-based kernels, and even complex distributed-memory dense linear algebra frameworks such as Elemental [80]. Indeed, the CHASE library includes a version supporting MPI+GPUs (CHASE-MPI) as well as one that uses Elemental for the numerical kernels (CHASE-Elemental).

As a C++ program, the decoupling between the CHASE algorithm and the implementation of the numerical linear algebra kernels is accomplished

via a *pure abstract class* which defines the interface for the C++ numerical kernels. A slightly simplified version of the interface is given in [Listing 4.1](#). The actual implementation is templated to allow for real-valued matrices and single precision data types. A derived class of `Chase` must implement these listed kernels, including an Hermitian matrix-matrix multiplication, or `HEMM`. Other kernels are more complex, such as the k -step Lanczos and the Rayleigh-Ritz procedure. All parallelism and handling of data is performed by classes derived from `Chase`. In addition to the advantages already mentioned, the abstraction allows the CHASE algorithm itself to be short and easily readable, with the corresponding C++ source code being very similar to [Algorithm 9](#). As an illustrative example, we illustrate a derived class of `Chase` based on the BLAS and LAPACK kernels.

4.3.1.1 ChASE With BLAS+LAPACK

`ChaseBLAS` is an implementation of the interface given in [Listing 4.1](#) using BLAS and LAPACK kernels. The Level 3 Basic Linear Algebra Subprograms (BLAS) [29] and the Linear Algebra PACKage [5] are staples of (single-node) dense linear algebra. `ChaseBLAS` is a relatively simple example and yet it is rich enough to highlight the power of the interface-based approach used by CHASE; in fact, it constitutes a high-performance implementation for shared-memory computers.

[Listing 4.2](#) shows only a partial view of the `ChaseBLAS` class where, for the sake of brevity, only the constructor, data members, the `HEMM`, and `Lock` functions are displayed. `ChaseBLAS` is constructed with the necessary sizes `N`, `nev`, and `nex`, as well as the required buffers for M , $\tilde{V}$, etc. The heart of any `Chase` implementation is the `HEMM` function ([Listing 4.2](#), [Section 4.3.1.1](#)), which is typically implemented as a wrapper to an optimized BLAS library (e.g., MKL [107] or cuBLAS).

The three-terms recurrence relation of the Chebyshev filter is implemented via the `HEMM` function. `approxV_` is a column major array that contains the approximate eigenvectors sorted according to the increasing entries of the array

Listing 4.2: ChaseBLAS: CHASE's Kernels with BLAS+LAPACK

```

1 using T = complex<double>;
2 class ChaseBLAS : public chase::Chase {
3 public:
4     ChaseBLAS(size_t N, size_t nev, size_t nex, T *H, T *V, T *W,
5               double *ritzv, double *resid)
6         : N_(N), nev_(nev), nex_(nex), locked_(0), H_(H), approxV_(V),
7           workspace_(W), ritzv_(ritzv), resid_(resid),
8           config_(N_, nev_, nex_) {}
9
10    void HEMM(size_t block, T alpha, T beta, size_t s) override {
11        zhemm('L', 'L', N_, block, &alpha, H_, N_,
12             approxV_ + N_ * (locked_ + s), N_, &beta,
13             workspace_ + N_ * (locked_ + s), N_);
14        swap(approxV_, workspace_);
15    };
16    void Lock(size_t new_converged) override {
17        memcpy(workspace_ + locked_ * N_, approxV_ + locked_ * N_,
18             N_ * (new_converged) * sizeof(T));
19        locked_ += new_converged;
20    };
21    /* Additional functions from Listing 4.1 omitted for conciseness */
22 private:
23     int N_, nev_, nex_, locked_;
24     T *M_, *approxV_, *workspace_;
25     double *ritzv_, resid_;
26     ChaseConfig<T> config_;
27 };

```

of degrees m (see [Section 4.2.2.1](#)). A natural way to leverage the recurrence relation is via double-buffering, implemented here by swapping the pointers for `approxV_` and `workspace_`. The `HEMM` function is called hundreds of times during a typical solve and performs around 80% of the total FLOPs. There are two reasons why the first `locked_+s` vectors stored in `approxV_` are excluded from the multiplication, and so they are not filtered any further. First, because they are locked vectors; in the language of [Algorithm 9](#) such vectors are not part of $\tilde{V}$ but rather part of V . In the C++ implementation, CHASE does not maintain a separate V buffer. More precisely, it moves converged vectors to the beginning of the `approxV_` buffer and locks them by calling the `Lock` function.

Within `ChaseBLAS` the `locked_` variable indicates how many vectors at the beginning of the `approxV_` buffer have been locked (Listing 4.2, Section 4.3.1.1). Second, when `optim = true` vectors may be excluded from the multiplication with M if they have already been sufficiently filtered. To this end, the first vector of $\tilde{V}$ to be filtered and passed to the `HEMM` function is the one indicated with s in Line 10 of Algorithm 12. We omit the rest of the implementation details. Casting the other methods from Listing 4.1 into appropriate BLAS and LAPACK calls yields a complete version of CHASE.

4.3.2 Available versions of ChASE

In the software package we present two HPC versions of CHASE, which differ both in the parallelization approach and the underlying linear algebra kernels. First, we describe CHASE-MPI with a custom implementation of a distributed matrix-matrix product at its core. Then we shortly describe CHASE-Elemental, where all the linear algebra kernels are implemented in terms of the corresponding Elemental library routines.

4.3.2.1 ChASE-MPI

In CHASE the matrix operator M is invoked only via matrix-matrix multiplications with (a subset of) $\tilde{V}$. These `HEMMs` occur in the `FILTER` (Algorithm 12, Line 10), the Rayleigh-Ritz procedure (Algorithm 7, Line 2), and the computation of the residuals. We illustrate a version of CHASE that implements an MPI-parallel `HEMM`. The rest of the CHASE-MPI derived class is similar to the `ChaseBLAS` implementation⁴; all non-`HEMM` operations on $\tilde{V}$ are performed redundantly on each node using threaded BLAS. This approach is simple, flexible, and surprisingly effective. We discuss performance aspects of CHASE-MPI in Section 4.4.

The custom implementation of the parallel `HEMM` is of particular interest. Each MPI rank is assigned a block of M , see Figure 4.4a. Such an approach requires

⁴ We can interpret `ChaseBLAS` as a special case of `ChASE-MPI` for exactly one MPI rank.

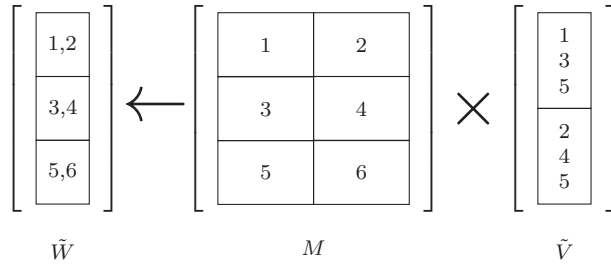
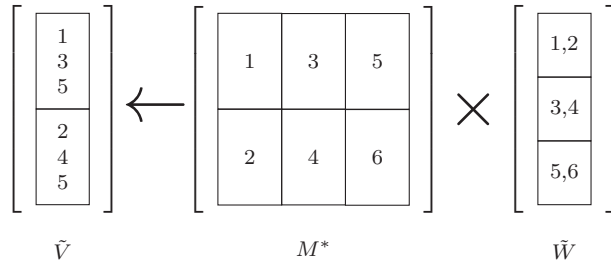
(a) Multiplication of M with $\tilde{V}$ into $\tilde{W}$ (b) Multiplication of M^* with $\tilde{W}$ into $\tilde{V}$

Figure 4.4: Example Data Distribution for 6 MPI Ranks

(Numbers inside blocks indicate owning MPI processes. Alternating multiplications of $\tilde{V}$ and $\tilde{W}$ with M do not require redistribution of data.)

only communication along the “rows of blocks”, e.g., between MPI ranks 1 and 2, 3 and 4, etc. In general, the data distribution of $\tilde{V}$ and $\tilde{W}$ are different. However, the Chebyshev filter requires the alternating multiplication of $\tilde{V}$ and $\tilde{W}$ with M . Consequently the approach of Figure 4.4a requires a redistribution of data after each multiplication. We can avoid the redistribution by multiplying $\tilde{W}$ with the conjugate transpose of M , which is possible as M is Hermitian, see Figure 4.4b. Since all other operations are performed redundantly on each node, the full vectors have to be reassembled via a “gather” operation. In addition to avoiding communication, the resulting matrix-matrix multiplications on each node are large and contiguous, often resulting in performance close to the theoretical peak. This data distribution easily allows the offloading of the multiplication to accelerators. CHASE currently supports a single

GPU per MPI rank to accelerate the matrix-matrix multiplication. The main disadvantage of this approach is that it implements a general matrix-matrix multiplication GEMM on each MPI rank, not a Hermitian one.

4.3.2.2 ChASE-Elemental

Elemental [80] is an excellent library for distributed-memory dense linear algebra. From an implementation point of view CHASE-Elemental is straightforward as the Elemental's kernels match CHASE's requirements very closely. We omit further details on CHASE-Elemental and point the interested reader to the source code. CHASE-Elemental has a key advantage over CHASE-MPI: it distributes $\tilde{V}$ and $\tilde{W}$ over the MPI processes and accordingly also parallelizes other operations, such as the reorthogonalization, over MPI. We discuss the performance aspect of this implementation together with CHASE-MPI in [Section 4.4](#). The CHASE-Elemental version serves as an example to showcase that CHASE can easily accommodate kernels from external numerical linear algebra libraries. For example, a CHASE version based on ScaLAPACK routines would be just as straightforward.

4.3.3 ChASE Configuration Parameters

The specifics of CHASE's parameters have already been discussed in previous sections as part of the description for each algorithm. This section summarizes all of them and provides an example of how to configure and call CHASE in a code snippet.

There are three categories of configuration parameters: general parameters, parameters concerning the Chebyshev filter, and parameters for the spectral estimates. Most of the *general* parameters were introduced in [Algorithm 9](#). The parameters for the *Chebyshev filter* are discussed as part of [Algorithm 11](#) and [Algorithm 12](#) in [Section 4.2.2](#), whereas the *spectral estimate* parameters concern [Algorithm 13](#) and are illustrated in [Section 4.2.3](#). All parameters are summarized in [Table 4.1](#). Within the source code the configuration is encapsulated by the `ChaseConfig` class. In addition to the parameter name, we

Category	Parameter	ChaseConfig member routine	Default value
General (Algorithm 9)	N	<i>constructor</i>	N/A
	nev	<i>constructor</i>	N/A
	nex	<i>constructor</i>	N/A
	approx	<code>void SetApprox(bool);</code>	false
	tol	<code>void SetTol(double);</code>	1e-10 (1e-5)
	MaxTasks	<code>void SetMaxIter(size_t);</code>	25
Chebyshev Filter (Algorithms 11 and 12)	optim	<code>void SetOpt(bool);</code>	false
	degExtra	<code>void SetDegExtra(size_t);</code>	2
	degMax	<code>void SetMaxDeg(size_t);</code>	36 (18)
	deg	<code>void SetDeg(size_t);</code>	20 (10)
Spectral Estimates (Algorithm 13)	k	<code>void SetLanczosIter(size_t);</code>	25(12)
	nvec	<code>void SetNumLanczos(size_t);</code>	4

Table 4.1: Overview of CHASE Configuration Parameters. Values in parentheses indicate default values for single precision

specify the member function of `ChaseConfig` that sets each parameter. Further, we list the default value for each parameter; wherever applicable the values in round brackets refer to the default values for single precision data types.

Table 4.1 lists two extra parameters that we have not discussed so far: `MaxTasks` and `nex`. To ensure the termination of CHASE, the number of `while` loop iterations are limited to `MaxTasks` (with a default of 25) after which CHASE returns the current approximate subspace. `nex` specifies the size by which the search space $\tilde{V}$ is incremented. In many cases increasing `nex` improves convergence of the SI at the cost of more expensive iteration steps. Unfortunately, the impact of the `nex` parameter on CHASE’s performance is difficult to predict. Therefore, instead of providing a default value, we recommend a `nex` value between 10% and 30% of `nev`.

4.3.3.1 Example: Using ChASE

Usage of CHASE amounts to constructing a valid instance of a class derived from `chase` and then calling the `chase::Solve()` function on it. How each

Listing 4.3: Calling ChaseBLAS

```

1 using T = std::complex<double>;
2 void chase_driver(size_t N, size_t nev, size_t nex, T *M, T *V, T *W,
3                 double *EVs, double *resid) {
4     ChaseBlas<T> chase(N, nev, nex, M, V, W, EVs, resid);
5     single.GetConfig().SetTol(1e-10); // Set residual tolerance
6     chase::Solve(&single); // Solve for the nev smallest eigenpairs
7     cout << "Smallest Eigenvalue: " << EVs[0]
8          << " with residual: " << resid[0] << '\n';
9 }

```

subclass of `chase` is constructed depends on the class implementation. In some cases the implementation is complicated by the data distribution; for example CHASE-MPI distributes the matrix M among MPI ranks in a highly customized fashion. Usage of ChaseBLAS, the CHASE variant that we sketched out earlier in this section, is shown in Listing 4.3. We begin by setting up a ChaseBLAS object via the constructor defined in Listing 4.1. Section 4.3.3.1 sets the desired residual tolerance to 10^{-10} . Finally, we invoke the CHASE algorithm on the constructed object.

4.4 Numerical Experiments

In this section we discuss performance aspects of CHASE and highlight its potential as a high-performance solver. Section 4.4.1 deals with sequences of eigenvalue problems. CHASE achieves significant performance improvements by using the solution $V^{(\ell-1)}$ of the previous problem in the sequence as input $\tilde{V}$. When compared to direct solvers, CHASE further improves performance through the ability to compute eigenpairs to a lower residual at reduced cost. Orthogonality between computed eigenvectors is inherited by the direct solver used in the Rayleigh-Ritz step (see Algorithm 7, Line 3). Strong- and weak-scaling behavior of CHASE-MPI and CHASE-Elemental are illustrated in Section 4.4.2.

For sequences of eigenvalue problems a natural comparison [26] is the Locally

Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method. Among the publicly available LOBPCG implementations that support dense complex-valued matrices we opt for ANASAZI’s [7] LOBPCG implementation. Due to reasons we discuss in Section 4.4.1, ANASAZI is not well suited for runtime comparisons. Instead we compare runtimes against Elemental’s state-of-the-art direct solver which uses PMRRR [75] for the tridiagonal solve. While many other distributed memory direct solvers for dense matrices exist [17, 67, 51], the performance differences between them do not significantly contribute to our discussion. Consequently, we use Elemental’s direct solver as a representative for all of them.

In our numerical tests we exclusively consider complex-valued double precision Hermitian matrices. Experiments were run on the JURECA commodity cluster. CHASE-MPI is configured to use 48 threads per node (using HyperThreading) for JURECA. Currently, CHASE supports the use of a single GPU device per MPI rank, and thus only a fourth of the two available Nvidia K80’s (4 visible CUDA devices). On the BLUEWATERS cluster, CHASE makes use of the Nvidia K20 of the Cray XK compute nodes. Both Elemental’s direct solver and CHASE-Elemental employ the Elemental library version 0.84-p1, with 24 MPI ranks per node. Elemental distributes its matrices according to a “grid”, which is essentially a two-dimensional Cartesian MPI communicator. The shape and size of the grid is critical in order to obtain good performance. As CHASE-Elemental and the Elemental direct solver usually obtain the best performance on grids with different configurations, we report the timings for the grid shape that is optimal for each solver. The reduction to tridiagonal form uses the default algorithmic block size of 64. CHASE uses the standard setting as discussed in Section 4.3.3, including a convergence criterion of $\text{tol} = 10^{-10}$. The increase of the subspace (`nex` parameter) is chosen as 20% of `nev` for every problem.

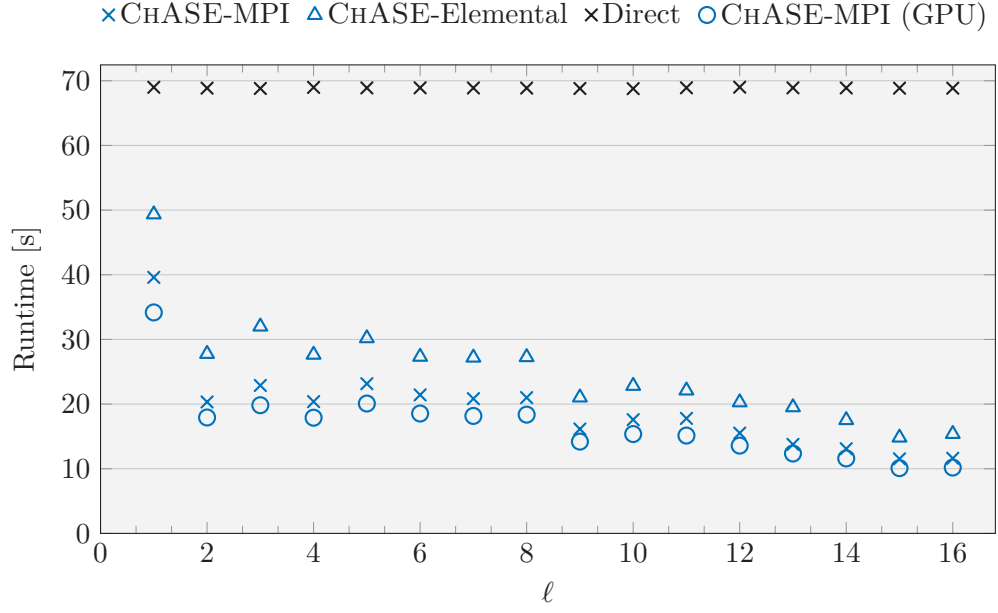


Figure 4.5: Runtimes for Sequences of Eigenvalue Problems (Single Node)

(NaCl: $n = 9\,273$, $\text{nev} = 256$)

4.4.1 Sequences

The pivotal advantage of algorithms based on SI is the ability to use the matrix of vectors $\tilde{V}$ as an approximate solution, thus effectively reducing runtime. CHASE performs particularly well because the Chebyshev accelerator reduces the number of required iterations and can be implemented in terms of HEMM calls.

Figures 4.5 and 4.6 shows runtimes for two sequences of eigenvalue problems solved on a single node using CHASE-MPI (with and without GPU), CHASE-Elemental, and Elemental’s direct solver. Figure 4.5 shows results for the NaCl sequence of length 16 with matrices of rank 9 273, a larger version of the eigenproblem matrix seen in Section 4.2.2.2. Figure 4.6 illustrates a sequence from the new AuAg data set. As for NaCl, the AuAg sequence originates from simulations based on a DFT self-consistent field iteration based on the FLAPW method [101, 53]. The eigenproblems in AuAg have a larger matrix size $n = 13\,379$, a larger sought after spectrum $\text{nev} = 972$, and are part of a

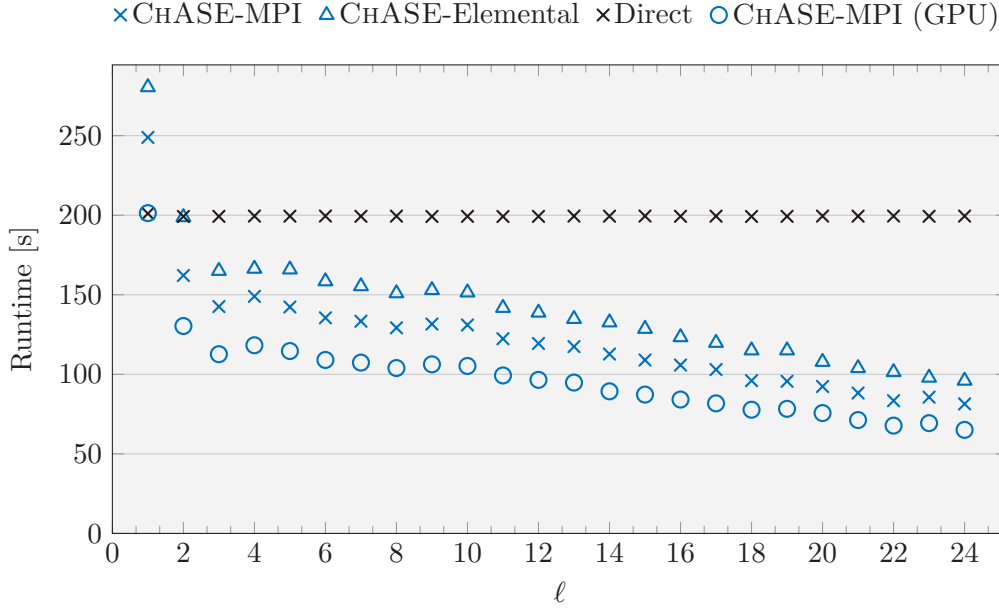


Figure 4.6: Runtimes for Sequences of Eigenvalue Problems (Single Node)

(AuAg: $n = 13\,379$, $\text{nev} = 972$)

sequence of length $N = 24$. In contrast to the NaCl data set, these matrices do not have a spectral gap, resulting in a smaller convergence ratio $|\rho_a|^{-1}$ for the eigenvalues closer to the upper end of the search space.

CHASE exhibits a similar pattern for both sequences: The time-to-solution for both CHASE variants is much higher for the first eigenproblem of the sequence than for later ones. In the NaCl sequence, CHASE-MPI (CHASE-Elemental) requires 39 (49) seconds to converge for $\ell = 1$ and only 20 (27) seconds for $\ell = 2$, which corresponds to a speedup of 2.0 (1.8). A small part of this speedup is due to cheaper calculation of the spectral bounds (see [Section 4.2.3](#)), but the vast majority is caused by better starting vectors $\tilde{V}$. What is more, the runtime for CHASE reduces further for increasing ℓ indices. On the last problem of the NaCl sequence, CHASE-MPI (CHASE-Elemental) requires only 11.6 (15.4) seconds, a total speedup of 3.3 (3.2) over the first problem in the sequence. The AuAg displays similar speedups. This phenomenon is due to the convergence of the underlying non-linear eigenvalue

problem, see [Definition 11](#): Some dips in runtime occur within a sequence for both versions, e.g., at $\ell = 9$ in [Figure 4.5](#), due to natural oscillations of the angle between subspaces of adjacent DFT iterations. For all ℓ indices, CHASE-MPI is consistently faster than CHASE-Elemental. Profiling indicates that for this particular set of problems the matrix-matrix multiplication routine of Elemental exhibits inferior performance, probably caused by the inferior node-level parallelism of its pure MPI distribution.

The GPU version of CHASE-MPI uses the GPU for the matrix-matrix multiplication with M . In this manner the Chebyshev filter, the computation of the residuals, and the Rayleigh-Ritz procedure are accelerated via the GPU. M is copied to the device at the first **while** loop iteration and then reused throughout the following iterations. Only the vectors V and W are copied to and from the device. Using the GPU version of CHASE-MPI outperforms the non-GPU on all problems. Leveraging the GPU results in speedup of ≈ 1.24 for the entire [AuAg](#) sequence. The results are slightly worse for [NaCl](#) because the smaller value for `nev` = 256 results in less efficient HEMMs. The GPU version achieves a speedup of respectively 1.2 and 1.1 for $\ell = 1$ and $\ell = 16$ over CHASE-MPI without GPU. The uncharacteristic speedups are due to the fact that the GK210 chip on the [JURECA](#) nodes have a peak performance ranging from 932 to 1456 GFLOPs, which is only slightly faster than the 960 GFLOPs of the CPU.

As expected, the direct solver requires the same amount of time for all eigenproblems in a sequence. However, the performance difference between CHASE and the direct solver is sequence dependent. CHASE is faster than the direct solver for all eigenproblems in the [NaCl](#) sequence. For [AuAg](#) on the other hand, CHASE is significantly slower than the direct solver for the first problem, but is faster from the second iteration onwards. This behavior is in part due to the spectral differences between the two problems (presence vs. absence of a spectral gap located between `nev` and `nev+nex` for the [NaCl](#) and [AuAg](#) respectively), and in part to the different ratio $\frac{\text{nev}}{n}$ (2.8 % vs. 7.3%). CHASE can only compete against a direct solver when `nev` is a relatively small fraction of the size of the matrix n , see [\[14\]](#). A choice of `nev` close to n is

ℓ index		CHASE			LOBPCG		
		Matvecs	Iterations	Time	Matvecs	Iterations	Time
<u>NaCl</u>	1	32 062	7	39.6 s	34 153	76	232.4 s
	16	9 636	2	11.6 s	20 090	35	134.3 s
<u>AuAg</u>	1	97 766	7	248.9 s	99 110	50	2 493.0 s
	24	31 052	2	81.4 s	65 296	27	1 563.9 s

Table 4.2: Comparison of ANASAZI’s LOBPCG with CHASE. LOBPCG timings are not representative.

inadvisable since the Rayleigh-Ritz subroutines contains a direct solve of rank `nev`.

As shown in [26], another iterative solver that can take advantage of an approximate solution is LOBPCG. We compare against ANASAZI’s LOBPCG implementation based on Trilinos version 12.12.1, with the same block size (`nev` + `nex`) and convergence tolerance of 10^{-10} used for CHASE. In order to match CHASE’ configuration, the only non-standard configuration setting is the usage of an absolute convergence criterion, instead of a relative one. As part of the default configuration ANASAZI fully orthogonalizes the approximate eigenvectors, and locks with a tolerance of 10^{-11} , a factor 10 lower than convergence.

ANASAZI’s LOPBCG can not easily make use of all 24 cores of a JURECA node, since many of the operations on the approximate eigenvectors are bandwidth-bound. For these reasons, a runtime comparison is not appropriate; instead we compare the number of matvecs⁵. Considering only the matvecs disregards operations on the approximate eigenvectors, such as orthogonalization, which strongly favors LOBPCG since the algorithm performs many more iterations and thus more reorthogonalizations. In Table 4.2 we present the results of our numerical experiments The table compares CHASE against LOBPCG and contains: the number of matvecs; the number of iterations needed to achieve convergence; and the required runtime on a single JURECA node. We present

⁵ Matvec, i.e. Matrix-vector products, is a misnomer: Both in CHASE and LOBPCG the matrix M is applied to a block of vectors, resulting in a HEMM call.

	n	nev	nex	# nodes	JURECA		BLUEWATERS	
					# cores	$\frac{n^2}{\# \text{ cores}}$	# cores	$\frac{n^2}{\# \text{ cores}}$
<u>NaCl</u>	3 893	256	51	4	96	N/A	N/A	N/A
	9 273	256	51	25	600	N/A	N/A	N/A
<u>AuAg</u>	13 379	972	194	25	600	N/A	N/A	N/A
<u>BSE</u>	22 360	100	20	9	216	2 314 674	64	7 812 025
	32 976	100	20	16	384	2 831 814	128	8 495 442
	47 349	100	20	36	864	2 594 823	288	7 784 471
	62 681	100	20	64	1 536	2 557 882	512	7 673 648
	76 674	100	20	100	2 400	2 449 542	800	7 348 628

Table 4.3: Overview of Matrices Used in Scaling Experiments

data for the first and the last problem of the NaCl and the AuAg problems. The table shows that both CHASE and LOBPCG employ fewer iterations and matvecs for the last problem in the sequence than for the first one. For the first index $\ell = 1$ LOBPCG performs only slightly more matvecs than CHASE. However, as the sequence progresses, CHASE executes fewer and fewer matvecs than LOBPCG (data not shown), ending with roughly half the number of matvecs of LOBPCG for both problems.

4.4.2 Scalability

In this section we analyze CHASE’s behavior both in strong- and weak-scalability regimes and compare it to Elemental’s direct solver. Since the parallel efficiency of CHASE is not influenced by the use of a good approximate solution⁶, in this section we focus only on “single” eigenvalue problems instead of sequences. To this end, we introduce the BSE data set with considerably larger matrices that only contains sequences of length one. Note that this puts CHASE at a disadvantage over direct solvers, since there are not spectral bounds or approximate solutions available. The BSE data set contains five

⁶ The experiments in Section 4.4.1 already use parallelism, since they were performed on an entire node with 24 cores.

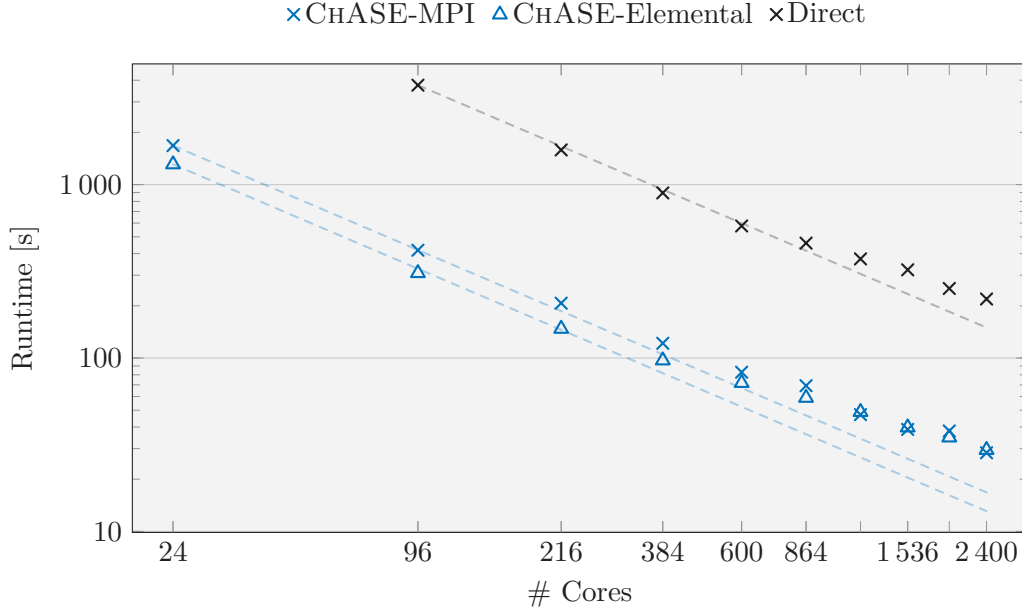


Figure 4.7: Runtimes of a Strong Scaling Experiment

(BSE: $n = 62681$, $\text{nev} = 100$)

matrices of increasing size from $n = 22\,360$ to $n = 76\,674$, corresponding to the same physical system, but with an increasing number of representative points in momentum space. The exact sizes are available in Table 4.3. These matrices are the result of the discretization of the Bethe-Salpeter equation used in optoelectronics simulations.⁷ For each matrix we choose $\text{nev} = 100$, and we increment the subspace size by $\text{nex} = 20$.

4.4.2.1 Strong Scaling

Figures 4.7 and 4.8 illustrate the results of a strong scaling experiment using the BSE matrix of size $n = 62681$. We select node counts corresponding to square numbers $9, 16, \dots, 100$, as these tend to yield efficient matrix distributions, but in the plots we report the resulting number of cores (each node has 24 cores). Figure 4.7 reports the runtimes in a log-log plot, while parallel efficiency is

⁷ Graciously provided by André Schleife (UIUC).

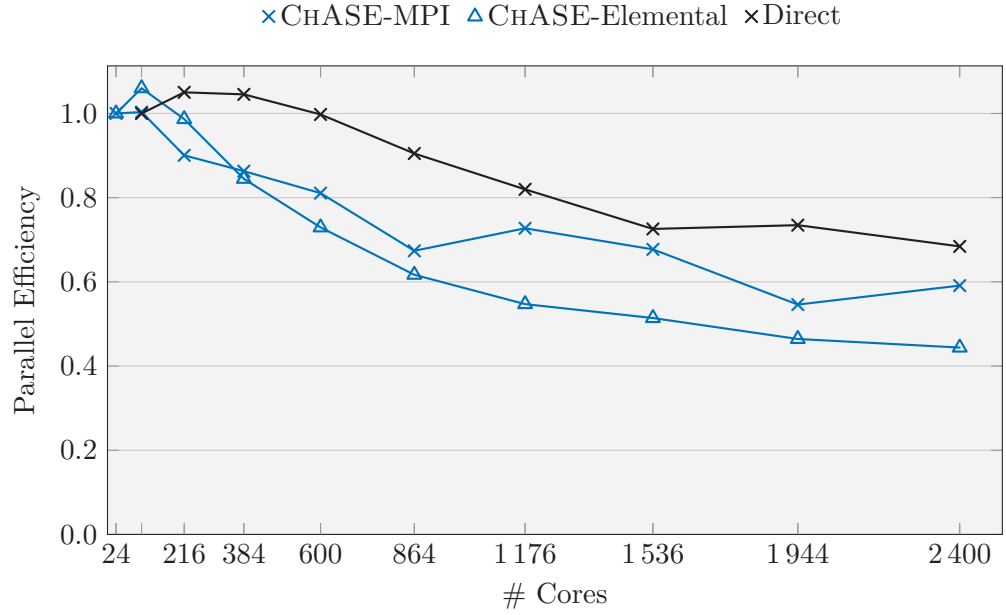


Figure 4.8: Parallel Efficiency of Runtimes in Figure 4.7

(BSE: $n = 62\,681$, $\text{nev} = 100$)

shown in Figure 4.8. The parallel efficiencies are calculated with respect to the timing of the corresponding solver on a single node using all available cores, as opposed to the (fastest available) sequential version.

The following remarks are in order: (1) The direct solver has a significantly longer runtime than CHASE. Both CHASE versions require roughly the same amount of time, with CHASE-Elemental being marginally faster. CHASE-MPI and CHASE-Elemental respectively solve the problem on 2400 cores in 28 and 29 seconds. Elemental’s direct solver on the other hand requires 219 seconds on 2400 cores. (2) The direct solver is missing the data point for a single node, because the solver requires more than the available 128 GiB of memory. The parallel efficiency is adjusted to account for the missing data. Note that CHASE does not have this problem—it needs only $n(\text{nev} + \text{nex})$ elements of working memory. (3) All three solvers scale well, with the direct solver being the most efficient. For 2400 cores CHASE-MPI and CHASE-Elemental show a parallel efficiency of, respectively, 59% and 44%, performing only slightly

# cores	Iterations		Matvecs		Runtime		
	CHASE-MPI	CHASE-Elemental	CHASE-MPI	CHASE-Elemental	CHASE-MPI	CHASE-Elemental	Direct
216	11	11	19 990	20 192	25.1 s	26.0 s	81.5 s
384	10	9	16 778	16 100	23.7 s	24.0 s	141.2 s
864	17	11	23 424	27 506	39.8 s	45.2 s	211.1 s
1 536	13	12	23 268	21 940	36.4 s	41.4 s	367.8 s
2 400	10	13	22 614	21 720	38.4 s	40.8 s	380.1 s

Table 4.4: Weak Scaling Experiment Results on [JURECA](#)

worse than the direct solver with 64%. (4) Both the Elemental-based solvers (direct and CHASE-Elemental) show super-scalar speedups for smaller number of nodes, which may occur due to the large dependence of Elemental on the computing grid size: an increased number of MPI ranks may yield a more efficient grid and thus a super-scalar speedup.

4.4.2.2 Weak Scaling

Weak scaling experiments are particularly relevant to application scientists, who require the simulation of increasingly larger physical systems, which in turn corresponds to bigger matrices. Unfortunately, in the case of an iterative eigensolver the notion of a weak scaling experiment is non-trivial. Therefore, we need to address how to set up weak scaling experiments with CHASE, before we discuss the results of the numerical tests.

Despite being generated essentially from the same physical system, CHASE performs very differently for matrices in the [BSE](#) data set, resulting in widely varying numbers of required iterations. We are also restricted to certain matrix sizes, making it difficult to have the same number of matrix elements per core. Nevertheless, in our experiment we strive for a large and roughly equal number of matrix elements per node. We collect timings for 5 different matrices from the [BSE](#) set, using between 9 and 100 nodes. The matrix sizes, number of cores, as well as the resulting number of matrix elements per core are presented in [Table 4.3](#). We cannot provide GPU timings for these problems on [JURECA](#), as it

# cores	Iterations	Matvecs	Filter Runtime			Total Runtime		
			CPU	GPU	Speedup	CPU	GPU	Speedup
64	11	20 106	176.4 s	22.8 s	7.7	228.0 s	43.5 s	5.2
128	9	16 856	175.9 s	27.5 s	6.4	236.1 s	52.6 s	4.5
288	12	23 610	231.5 s	30.2 s	7.7	306.8 s	70.3 s	4.4
512	14	23 080	225.5 s	30.1 s	7.5	316.5 s	87.3 s	3.6
800	12	22 868	209.1 s	30.8 s	6.8	299.2 s	89.9 s	3.3

Table 4.5: Weak Scaling Experiment Results for CHASE-MPI on BLUEWATERS

only has 75 nodes equipped with GPUs. Instead we provide runtimes obtained from BLUEWATERS for both CPU and GPU. Table 4.4 reports the runtimes and the matvecs on JURECA, while Table 4.5 shows them for BLUEWATERS.

Before we discuss the runtimes of Table 4.4 in detail, note that CHASE-MPI and CHASE-Elemental yield a somewhat different number of iterations and matvecs. The difference is due to variations in the randomization of the initial subspace V , and to variance in the numerical kernels, largely due to rounding errors in floating-point arithmetic. Table 4.4 shows that the direct solver needs much larger runtimes than CHASE. Moreover, the direct solver execution increases drastically for larger ranks, as the reduction to tridiagonal form is of cubic complexity. The runtimes for the two CHASE versions are roughly similar, and surprisingly do not strictly increase with the problem size. We observe that the number of required iterations changes just as much from one CHASE implementation to next next, as it does for the separate BSE problems. In addition, the number of matvecs and iterations does not necessarily increase with problem size. We conclude that for a realistic scaling scenario the spectral differences between the increasingly larger problems have a bigger influence on performance than any adverse weak scaling behavior of CHASE. Finally, for CHASE-MPI the number of matvecs is roughly similar and so are the runtimes, implying that CHASE-MPI scales well in a weak scaling regime.

Table 4.5 contains runtimes, matvecs, and iterations for CHASE-MPI with and without GPU. In addition to the total runtime and speedup achieved via the

GPU, we also report these metrics for the Chebyshev filter. The BLUEWATERS “XK” node has a peak performance of 156 GFLOPs on the CPU and 1310 GFLOPs on the available K20 GPU. Aside from the performance difference, the CPU runtimes behavior looks similar to the one observed on JURECA: the number of iterations changes from problem to problem, and the runtimes fluctuate accordingly. The GPU timings, however, provide new insight. On the Chebyshev filter CHASE leverages the GPU for a 6.4–7.7 speedup over the CPU. The rather small `nev` = 100 value influences the latency of the transfer from GPU to RAM—required for the parallel matrix-matrix multiplication—and is a significant source of overhead. For larger values of `nev` the incurred latency amortizes over more computation, resulting in better speedups.

The speedup from the GPU over the total runtime decreases for the experiments involving larger matrix sizes. While the speedup for the smallest problem is 5.2, it decreases to 3.3 for the largest problem. Because CHASE-MPI parallelizes only the matrix-matrix multiplication of V with M over MPI ranks, operations such as the QR are not parallelized but executed redundantly on the node. Thus, when the matrix size becomes bigger, the costs for the orthogonalization increase accordingly. For the largest matrix, the Chebyshev filter accounts for only a third of the runtime (30.8 s vs. 89.9 s).

4.5 Summary

This chapter introduced CHASE, a high-performance solver for sequences of correlated exterior HEPs with a focus on problems arising from Density Functional Theory simulations. CHASE is based on a modern version Subspace Iteration method and employs polynomial acceleration via a Chebyshev filter. We presented an approach to adaptively choose the polynomial degree of the filter as well as a method to cheaply obtain the required spectral estimates. What is more, CHASE touts a modern and modular software architecture, resulting in minimal integration effort. Our numerical experiments show that CHASE can outperform direct solvers both on single as well as multiple nodes.

CHASE has a plethora of uses outside of DFT simulations: we presented experiments in which CHASE is competitive even for large HEPs that are not sequences arising from Bethe-Salpeter simulations.

The sooner we can forget that FORTRAN has ever existed, the better, for as a vehicle of thought it is no longer adequate: it wastes our brainpower, is too risky and therefore too expensive to use.

Edsger W. Dijkstra (1972)

5

SLiSe:

An Optimization Framework for FEAST’s Rational Filters

The goals of this chapter are very similar to those in [Chapter 4](#): We aim to increase performance of HEP solves arising from DFT simulations. However, this chapter outlines a slightly different process, as testified by the software artifact resulting from this chapter. Instead of a solver library, as was the case with CHASE in [Chapter 4](#), we present an optimization framework. In a manner of speaking we optimize a parameter, the rational filter function, of an existing eigensolver. Our method is based on a non-linear Least-Squares optimization approach, which leads to custom filters we term **S**ymmetric non-**L**inear **O**ptimized Least-**S**quares (SLiSe), read “slice”.

Although the work we present in this chapter focuses on the Hermitian version of FEAST [\[109, 79\]](#), it can be extended to other resolved-based eigensolver methods. A prominent use-case of FEAST is the NESSIE nanoelectronics DFT simulation code. FEAST’s user base is quite large as it has been integrated into

MKL [107] as the “Extended Eigensolver”. As an interior HEP eigensolver, FEAST can slice the exterior HEP into multiple independent parts, increasing the available parallelism in a HPC setting. We give a short introduction into FEAST’s algorithms in the next section.

Our goal in this chapter is not the generation of one or two very good filters to be distributed with FEAST. Instead, the long term goal is to enable a near-optimal choice of rational filter on a per problem basis. To this end, this chapter works towards the goal of such *problem specific filters* by examining performance characteristics of rational filters within FEAST and providing a fast optimization framework.

The rest of this chapter is structured as follows. Section 5.1 considers the formulation of the non-linear Least-Squares problem. We discuss the use of symmetries of the filters, as well as the objective function and its gradients, which also make use of these symmetries. In Section 5.2 we provide a full framework for the generation of SLiSe filters, including a discussion of: more advanced optimization methods; choice of starting position; and constrained optimization. Numerical experiments are presented in Section 5.3, and we provide an outlook of further work on our optimization framework in Section 5.4. We conclude this chapter with a small summary in Section 5.5.

Publication

The work presented in this chapter is in parts based on research previously published in:

- [2] J. Winkelmann and E. Di Napoli. “Non-linear Least-Squares Optimization of Rational Filters for the Solution of Interior Hermitian Eigenvalue Problems”. In: *Frontiers in Applied Mathematics and Statistics* 5 (2019). DOI: [10.3389/fams.2019.00005](https://doi.org/10.3389/fams.2019.00005).
- [1] K. A. S. Kollnig, J. Winkelmann, P. Bientinesi, and E. Di Napoli. “Rational Filters with Faster Convergence and Constraints for Contour-based Eigensolvers”. In: *Preparation* (2020).

Edoardo Di Napoli, Ph.D. has provided the derivation presented in Sections 5.1.4 to 5.1.6.

5.1 Rational Filters for the HEP

We start off this section by considering rational filter functions for Subspace Iteration, an extension of our discussion in the previous chapter. [Section 5.1.3](#) defines our objection function for our optimization, and we restrict the input parameter space by leveraging symmetries in [Section 5.1.4](#). We derive matrix formulations, including the symmetries, for the object function and its gradient in, respectively, [Section 5.1.5](#) and [Section 5.1.6](#).

5.1.1 Subspace Iteration with Rational Filters

We aim to find all eigenvalues of M in the interval $\mathcal{I} = (a, b)$, see [Section 2.2.1](#). Recall that for the interior HEP we do not know the number of eigenvalues inside $\mathcal{I}$. We denote the (unknown) number of eigenvalues inside $\mathcal{I}$ by $\mathbf{m}$ and the size of the subspace used in the SI procedure by $\mathbf{p}$, $\mathbf{p} \geq \mathbf{m}$. Notably, this deviates from the standard notation used by the FEAST authors that use, respectively, M_0 and M .

In this chapter we focus on rational filter functions $f(M)$ of the matrix M of degree $(n-1, n)$, that can be expressed in partial fraction decomposition as:

$$f(M) := \sum_{i=1}^n \alpha_i (M - \mathbb{I}z_i)^{-1} \quad (5.1)$$

where $z_i \in \mathbb{C} \setminus \mathbb{R}$ and $\alpha_i \in \mathbb{C}$, $1 \leq i \leq n$. An arbitrary choice of z s and α s does not result in a good filter. However, this formulation is practically identical to a numerical integration of the spectral projector ([Equation \(2.14\)](#)), up to a constant factor. While numerical integration is one way to obtain z s and α s that result in good filters, there are alternative approaches: Güttel et al. [\[40\]](#) showed a correspondence between numerical integration of the spectral projector and classical filters from signals processing. In this work we consider a more general problem: We aim to optimize z s and α s such that $f(M)$ maps eigenvalues of M inside $[a, b]$ to roughly one, and the eigenvalues outside to roughly zero.

The filters of Equation (5.1) suppress eigenvalues outside of $[a, b]$, while the eigenvectors remain constant. As a result the convergence of the Subspace Iteration procedure for the desired eigenvalues and eigenvectors is accelerated. In this context f , as a scalar function, can be seen as a rational approximation to the indicator function $\chi_{[a,b]}$, with unit value inside $[a, b]$ and zero everywhere else. Filters are usually generated for a search interval of $[-1, 1]$ and then mapped to $[a, b]$ via an appropriate transformation on the α_i s and z_i s. Without loss of generality we always consider the search interval to be $[-1, 1]$. Forming $f(M)$ from Equation (5.1) explicitly involves the calculation of the matrix inverse. Using $f(M)$ within a Subspace Iteration procedure requires M only implicitly, i.e. $X := f(M)Y$, which can be rewritten as n independent linear system solves:

$$(M - \mathbb{I}z_i)X = \alpha_i Y, \quad 1 \leq i \leq n. \quad (5.2)$$

If we assume that $f(M)$ outputs a good approximation to the search subspace, a common form of subspace iteration is based on the use of the Rayleigh-Ritz (RR) projection at each iteration. Provided that the approximating subspace has a dimension $\mathbf{p}$ equal or larger than $\mathbf{m}$, such a projection reduces the size of the eigenproblem to be solved, and is guaranteed to output eigenpairs of the reduced problem that are in a one-to-one correspondence with the eigenpairs inside the search interval.

FEAST also uses RR projection and derives the filter function via n -point quadrature of the contour integral:

$$\frac{1}{2\pi i} \int_{\Gamma} \frac{dt}{M - \mathbb{I}t} \approx \frac{1}{2\pi i} \sum_{i=1}^n w_i (M - \mathbb{I}x_i)^{-1} \quad (5.3)$$

where Γ is a contour in the complex plane enclosing the search interval $[a, b]$. Quadrature along Γ yields integration nodes $x_i \in \mathbb{C}$ and weights $w_i \in \mathbb{C}$. While any quadrature rule can filter the eigenvalues to some extent, FEAST defaults to the Gauss-Legendre quadrature rule.

Similar to CHASE, FEAST is a subspace iteration procedure with a spectral

filter. Accordingly, FEAST's convergence rate depends strongly on the properties of the filter.

Theorem 3 (Convergence rate of FEAST [96])

Let the Hermitian matrix M has $\mathbf{m}$ eigenvalues inside the interval $\mathcal{I}$ and $\tilde{V} \in \mathbb{C}^{n \times \mathbf{p}}$ be the vectors passed to the FEAST algorithm. Further, let the rational filter $f(M)$ (as defined in Equation (5.1)) used by FEAST admit the following ordering

$$|f(\lambda_1)| \geq \dots \geq |f(\lambda_{\mathbf{m}})| \geq \dots \geq \frac{1}{2} \geq |f(\lambda_{\mathbf{p}})| \geq \dots \geq |f(\lambda_{\mathbf{p}+1})| \geq \dots \geq |f(\lambda_n)|.$$

Then, the convergence rate of FEAST is given by

$$\tau(\lambda_i) = \frac{|f(\lambda_{\mathbf{p}+1})|}{|f(\lambda_i)|}, \quad 1 \leq i \leq \mathbf{m}.$$

Our main goal for this chapter is the acceleration of FEAST's convergence by optimizing the rational filter function $f(M)$. We accomplish the desired acceleration by means of filter functions that provide favorable rates of convergence, per Theorem 3. In simple words, we want filters with function values that are large within $\mathcal{I}$, and small outside of it. An optimal filter function would be the (discontinuous) indicator function of $\mathcal{I}$, with zeros for all function values outside of $\mathcal{I}$, and one inside of $\mathcal{I}$. In the following, we generate rational functions that approximate the indicator function.

5.1.2 Related Work

In the following, we outline work related to rational approximations of the spectral projector in the context of Subspace Iteration. Related work for Subspace Iteration was discussed back in Section 4.1.2.

As compared to polynomial filters, rational approximations to the spectral projector are a more recent development as compared to the polynomial approximations seen in Section 4.1.2. While Chebyshev-based filters have worked very well for polynomial filters, development of rational filters is a much more

colorful story: In the following, we start off with an overview of solvers that make use of rational approximations of the spectral projector, and then proceed to work related to rational filters for eigensolvers.

Rational Krylov methods [84] (1984) can be seen as a predecessor to rational filtering methods. As a generalization of the shift-and-invert Arnoldi algorithm, the rational Krylov method allows the computation of eigenpairs near multiple shifts. The change of the factorization from one shift to another does not require operations on long vectors, which dramatically reduces computational cost. There have been several extensions to the rational Krylov methods, including parallelization [13] and generalizations to non-linear eigenvalue problems and matrix functions [39, 41].

The resolvent operator is the standard method for building rational filters. Due to its connection to matrix function and operator theory, there is a large mathematical framework to build upon. The last ten years have witnessed a plethora of papers on methods that use the matrix resolvent for the solution of interior HEPs. One of the main advantages of these methods is that they can naturally be extended to non-Hermitian generalized eigenvalue problems. A comparison of these solvers is available in [49].

Two well-known resolved-based eigensolvers are the “Sakurai and Sugiura” (SS) family of solvers [89], and FEAST due to Polizzi [79, 96]. Sakurai and Sugiura initially proposed the SS-H method, a non-iterative method based on complex moment (Hankel) matrices in 2003. Polizzi’s FEAST, originally published in 2009 is an iterative solver based on the matrix resolvent; in 2014 Tang and Polizzi established that FEAST is a Subspace Iteration procedure with a rational filter. Work by Sakurai et al. in the 2010s resulted in a Rayleigh-Ritz type method (SS-R) [47, 48] and various block methods [50]. Beyn [16] proposed a similar method for non-linear problems in 2012. Despite the rich variety of recent works, eigensolvers with rational filters are still a very active area of research where most of the focus is on improving the performance and robustness of the algorithms as well as their implementations.

Galgon et al. [33] transport the notion of an adaptive degree choice, similar

to [Section 4.2.2](#), to FEAST. Their approach results in an adaptive choice of the number of integration points, closely related to adaptive integration.

We require some discussion on the nomenclature before we can proceed with the discussion. Initially, eigensolvers based on the matrix resolvent were called “contour solvers”, because they approximate the spectral projector via integration along a contour, see [Equation \(2.14\)](#). With the name comes a focus on the numerical integration aspect of these methods. Considering contour solvers as fundamentally based on rational approximation and not numerical integration is arguably the largest paradigm shift in the development of these solvers since their inception: It allows “contour methods” to leverage rational function approximation. We refer to these solvers as *resolvent-based eigensolvers* in order to highlight that these solvers employ rational filters.

The correspondence between contour methods and rational filters was discussed for FEAST in 2014 [\[96\]](#), for SS-H in 2010 [\[48\]](#), and for SS-R also in 2010 [\[47\]](#). Murakami proposed the use of classic rational filters from signal processing as early as 2008 [\[73, 72, 69, 71, 70\]](#) in the “filter diagonalization method”, for a discussion of this work we refer to [\[6\]](#). For FEAST the Elliptic filters were proposed by Güttel et al. in 2015 [\[40\]](#), resulting in improved load balancing and (in some cases) better convergence behavior. Moreover, the authors establish an equivalence between filters obtained via numerical integration and well-known filters from signal processing. The rational function point of view also enables the development of specialized algorithms: for instance, for real-symmetric matrices it is possible to avoid complex arithmetic entirely [\[6\]](#).

More recently the rational filter itself has been treated as a parameter that can be designed via optimization methods. Van Barel [\[10\]](#) proposed a non-linear Least-Squares approach for non-Hermitian filters within the SS-H framework, while Xi and Saad [\[102\]](#) described linear Least-Squares optimized filters for the Hermitian FEAST solver. Since Van Barel’s approach is geared towards non-Hermitian eigenproblems, it is not possible to make use of symmetries in the rational function. The resulting degrees of freedom make for a more difficult optimization problem. By requiring a parameter-space search his approach results in filters that are not as robust as the approach we describe in [Chapter 5](#).

What is more, Van Barel's Least-Squares optimization is based on the discrete ℓ_2 norm, not a function approximation approach. Even in the case of Hermitian problems, optimizing the squared distances in only a sample of points must be done with great care. Choosing too small a number of sample points can have undesirable effects: it is possible to obtain poles near the real axis which are not detected by the sample points. On the other hand having a large number of sample points is very expensive. Finally, Van Barel's approach does not support constraints.

Xi and Saad's approach focuses on FEAST. Their work exclusively optimizes the coefficients of the rational function. The result is a robust process that is much easier to solve, at the cost of being less expressive. Without optimization of the poles, the resulting functions are worse approximations of the spectral projector. Xi and Saad's approach also does not support constraints, as the poles of the filter must be chosen manually. On the other hand, they do discuss a penalty term to influence features of the resulting rational function.

5.1.3 Rational Filters for the Interior HEP

Before discussing the actual optimization, we need to reformulate the mathematical problem. Rational filters can be written as a sum of simple rational functions ϕ_i

$$f(t, z, \alpha) = \sum_{i=1}^n \phi_i = \sum_{i=1}^n \frac{\alpha_i}{(t - z_i)} \quad (5.4)$$

with $z_i, \alpha_i \in \mathbb{C}$. We restrict ourselves to the case where the z_i are pair-wise distinct, the α_i are pair-wise distinct, and the z_i have non-zero real and imaginary parts. In the Hermitian case the ideal filter corresponds to the indicator function defined as

$$h(t) = \chi_{[-1,1]} = \begin{cases} 1 & \text{if } -1 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases} . \quad (5.5)$$

As $h(t)$ is discontinuous we have to specify in what sense a given filter $f(t, z, \alpha)$ is close to the target function $h(t)$. To this end, we focus on the

squared difference of $h(t) - f(t, z, \alpha)$ with respect to the L_2 norm which is equivalent to defining a Least-Squares error function, or *objective function*

$$\mathcal{F}(z, \alpha) = \|h - f(z, \alpha)\|_2^2. \quad (5.6)$$

An optimal filter, in this sense, is the minimizer of the objective function

$$\min_{\alpha_i, z_i} \int_{-\infty}^{\infty} \mathfrak{G}(t) \left| h(t) - \sum_{i=1}^n \frac{\alpha_i}{(t - z_i)} \right|^2 dt, \quad (5.7)$$

resulting in a weighted Least-Squares optimization problem with the weight function $\mathfrak{G}(t)$.

In order to simplify the optimization process we impose the following restrictions on the weight function: (1) $\mathfrak{G}(t)$ is piece-wise constant with bounded interval endpoints; (2) the number of piece-wise constant parts is finite; (3) the weight function is even, i.e. $\mathfrak{G}(t) = \mathfrak{G}(-t)$. The implications for this decision become apparent in [Section 5.1.5](#): The restrictions result in a formulation for the objective function entirely made up of cheaply computable integrals.

5.1.4 Leveraging Symmetries

$h(t)$ has two properties which we want to be present in $f(t, z, \alpha)$ as well:

1. Realness: $h(t) \in \mathbb{R}$
 2. Evenness: $h(-t) = h(t)$.
- (5.8)

Both properties are not ensured based on our formulation of f in [Equation \(5.4\)](#). The following rational function, on the other hand, is real-valued and even:

$$f(t, \alpha, z) = \sum_{i=1}^{\frac{n}{4}} \left[\frac{\alpha_i}{t - z_i} - \frac{\bar{\alpha}_i}{t + \bar{z}_i} + \frac{\bar{\alpha}_i}{t - \bar{z}_i} - \frac{\alpha_i}{t + z_i} \right]. \quad (5.9)$$

In the following it is useful to express our desired properties as symmetry operations. For the simple rational functions ϕ_i of [Equation \(5.4\)](#) these operations

are:

$$\begin{aligned} 1. \text{ Conjugation (C): } & \overline{\phi_i}(t, z, \alpha) = \frac{\overline{\alpha_i}}{t - \overline{z_i}}, \\ 2. \text{ Reflection (R): } & \phi_i^\pi(t, z, \alpha) = \frac{-\alpha_i}{t + z_i}. \end{aligned} \quad (5.10)$$

Note that the R symmetry results in even functions $f(t)$. Now, we can rewrite Equation (5.9) as

$$f(t, \alpha, z) = \sum_{i=1}^{\frac{n}{4}} \left(\psi_i + \overline{\psi_i^\pi} + \overline{\psi_i} + \psi_i^\pi \right).$$

By convention we refer to functions f that are CR invariant, i.e. invariant under the conjugation and reflection operations, as:

$$\begin{aligned} f(t, w, \gamma) &= \sum_{k=1}^q \left(\chi_k + \overline{\chi_k^\pi} + \overline{\chi_k} + \chi_k^\pi \right) \\ &= \sum_{k=1}^q \left[\frac{\gamma_k}{t - w_k} - \frac{\overline{\gamma_k}}{t + \overline{w_k}} + \frac{\overline{\gamma_k}}{t - \overline{w_k}} - \frac{\gamma_k}{t + w_k} \right]. \end{aligned} \quad (5.11)$$

Here $q = \frac{n}{4}$, so $f(t, w, \gamma)$ has a fourth the number of variables as compared to $f(t, z, \alpha)$ resulting in faster optimization methods. One way to look at Equation (5.11) is, that it suffices to consider f on a single quadrant; the other three quadrants can be obtained via the symmetry operations. From that perspective, q can be understood as the number of poles (and coefficients) *per quadrant*.

For the formulation of the objective function and its gradient in the next sections, it is useful to have a formulation of f that is invariant under the C symmetry but not the R symmetry:

$$f(t, v, \beta) = \sum_{j=1}^p \left(\psi_j + \overline{\psi_j} \right) = \sum_{j=1}^p \left[\frac{\beta_j}{(t - v_j)} + \frac{\overline{\beta_j}}{(t - \overline{v_j})} \right], \quad (5.12)$$

with $p = \frac{n}{2}$.

5.1.5 Objective Function

Let us expand Equation (5.6) by temporarily disregarding the symmetries and expressing f as in Equation (5.4):

$$\begin{aligned}
 \mathcal{F}(z, \alpha) &= \|h - \sum_i \alpha_i \phi_i\|_2^2 \\
 &= \langle h - \sum_i \alpha_i \phi_i, h - \sum_j \alpha_j \phi_j \rangle \\
 &= \langle h, h \rangle - \langle h, \sum_j \alpha_j \phi_j \rangle - \langle \sum_i \alpha_i \phi_i, h \rangle + \langle \sum_i \alpha_i \phi_i, \sum_j \alpha_j \phi_j \rangle \\
 &= \langle h, h \rangle - 2\Re[\sum_i \langle \phi_i, h \rangle \alpha_i] + \sum_i \sum_j \bar{\alpha}_i \alpha_j \langle \phi_i, \phi_j \rangle.
 \end{aligned}$$

The objective function is expressible as

$$\mathcal{F}(z, \alpha) = \alpha^* \mathcal{G} \alpha - 2\Re[\eta^* \alpha] + \|h\|^2 \quad (5.13)$$

with

$$\begin{aligned}
 \mathcal{G}_{ij} &= \langle \phi_i, \phi_j \rangle, \\
 \eta_i &= \langle \phi_i, h \rangle.
 \end{aligned}$$

The inner products $\langle \cdot, \cdot \rangle$ are defined through a weighted L_2 norm with weight function $\mathfrak{G}(t)$:

$$\langle \phi_i, \phi_j \rangle = \int_{-\infty}^{+\infty} \mathfrak{G}(t) \bar{\phi}_i(t) \phi_j(t) dt = \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \bar{z}_i)(t - z_j)} dt, \quad (5.14)$$

and

$$\begin{aligned}
 \eta_i &= \langle \phi_i, h \rangle = \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t) h(t)}{t - \bar{z}_i} dt \\
 \|h\|^2 &= \int_{-\infty}^{+\infty} \mathfrak{G}(t) h(t)^2 dt.
 \end{aligned}$$

Note the definition of the complex scalar product although h and f are

real-valued functions. We require a complex scalar product for the gradient of f , as its domain is complex-valued.

C Symmetry

Before we present the objecte function that leverages the CR symmetries, let us consider a version of the objective function that only leverages the conjugation symmetry. One could start directly with the CR invariant formulation from Equation (5.11) of the rational function f . However, we prefer to show how each symmetry requirement has direct consequences on the structure of the matrix G and vector η . Recall from Equation (5.12) how the z_i s and α_i s were mapped to the v_j s and β_j s so as to make explicit the C symmetry. The objective function has the following block structure:

$$\mathcal{F}(v, \beta) = (\beta^* \quad \beta^\top) \begin{pmatrix} \mathcal{A} & \mathcal{B} \\ \overline{\mathcal{B}} & \overline{\mathcal{A}} \end{pmatrix} \begin{pmatrix} \beta \\ \overline{\beta} \end{pmatrix} - 2\Re \left[(\zeta^* \quad \zeta^\top) \begin{pmatrix} \beta \\ \overline{\beta} \end{pmatrix} \right] + \|h\|^2 \quad (5.15)$$

with

$$\begin{aligned} \mathcal{A}_{i,j} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{v}_i)(t - v_j)} dt \\ \zeta_i &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)h(t)}{t - \overline{v}_i} dt \\ \mathcal{B}_{i,j} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{v}_i)(t - \overline{v}_j)} dt \\ \mathcal{B}_{i,i} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{v}_i)^2} dt \end{aligned}$$

for $1 \leq i, j \leq p$. The matrix A is Hermitian while B is complex symmetric, so that $\overline{\mathcal{B}} = \mathcal{B}^*$.

Equation (5.15) can be reduced since it contains only half of the unknowns as the initial objective function. For example $\beta^\top \overline{\mathcal{A}} \overline{\beta} = \beta^* \mathcal{A} \beta$ and $\beta^* \mathcal{B} \overline{\beta} = \overline{(\beta^\top \overline{\mathcal{B}} \beta)}$, so that

$$(\beta^* \quad \beta^\top) \begin{pmatrix} \mathcal{A} & \mathcal{B} \\ \overline{\mathcal{B}} & \overline{\mathcal{A}} \end{pmatrix} \begin{pmatrix} \beta \\ \overline{\beta} \end{pmatrix} = 2 \left[\beta^* \mathcal{A} \beta + \Re (\beta^* \mathcal{B} \overline{\beta}) \right]$$

and similarly $\zeta^* \beta = \overline{\zeta^\top \beta}$. With the simplifications the objective function above reduces to

$$\mathcal{F}(v, \beta) = \beta^* \mathcal{A} \beta + \Re \left(\beta^* \mathcal{B} \overline{\beta} \right) - 2 \Re \left(\zeta^* \beta \right) + \frac{1}{2} \|h\|^2. \quad (5.16)$$

CR Symmetry

Now we can require [Equation \(5.16\)](#) to also satisfy the reflection symmetry R. The objective function now takes on a new block form

$$\begin{aligned} \mathcal{F}(w, \gamma) = & \begin{pmatrix} \gamma^* & (\gamma^\pi)^\top \end{pmatrix} \begin{pmatrix} \mathcal{X} & \mathcal{Y} \\ \mathcal{Y}^* & \overline{\mathcal{X}}^\pi \end{pmatrix} \begin{pmatrix} \gamma \\ \overline{\gamma}^\pi \end{pmatrix} + \Re \left[\begin{pmatrix} \gamma^* & (\gamma^\pi)^\top \end{pmatrix} \begin{pmatrix} \mathcal{W} & \mathcal{Z} \\ \mathcal{Z}^* & \overline{\mathcal{W}}^\pi \end{pmatrix} \begin{pmatrix} \overline{\gamma} \\ \gamma^\pi \end{pmatrix} \right] \\ & - 2 \Re \left[\begin{pmatrix} \theta^* & (\theta^\pi)^\top \end{pmatrix} \begin{pmatrix} \gamma \\ \overline{\gamma}^\pi \end{pmatrix} \right] + \frac{1}{2} \|h\|^2 \end{aligned} \quad (5.17)$$

where the matrix blocks are defined by

$$\begin{aligned} \mathcal{W}_{k,\ell} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{w}_k)(t - \overline{w}_\ell)} dt \\ \overline{\mathcal{W}}_{k,\ell}^\pi &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t + w_k)(t + w_\ell)} dt \\ \mathcal{X}_{k,\ell} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{w}_k)(t - w_\ell)} dt \\ \overline{\mathcal{X}}_{k,\ell}^\pi &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t + w_k)(t + \overline{w}_\ell)} dt \\ \mathcal{Y}_{k,\ell} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{w}_k)(t + \overline{w}_\ell)} dt \\ \mathcal{Z}_{k,\ell} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{w}_k)(t + w_\ell)} dt \\ \theta_k &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)h(t)}{t - \overline{w}_k} dt \\ \overline{\theta}_k^\pi &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)h(t)}{t + w_k} dt \end{aligned}$$

for $1 \leq k, \ell \leq q$. These submatrix blocks preserve some of the properties of

the matrix they are part of. For instance, $\mathcal{X}^* = \mathcal{X}$ and $\overline{\mathcal{W}} = \mathcal{W}^*$ while the R symmetry imposes new relations $\mathcal{Y}^* = \overline{\mathcal{Y}}^\top$ and $\mathcal{Z}^* = \overline{\mathcal{Z}}^\top$. In addition to these, the R symmetry allows for additional equalities thanks to the symmetric integration boundaries of the inner product $\langle \cdot, \cdot \rangle$. In other words

$$\begin{aligned} \int_0^a g(t, w_k) dt &= \int_{-a}^0 g^\top(t, w_k) dt && \text{for } g \text{ an even function of } t \\ \int_0^a g(t, w_k) dt &= - \int_{-a}^0 g^\top(t, w_k) dt && \text{for } g \text{ an odd function of } t. \end{aligned}$$

The direct implication of this observation is that, for instance, $\mathcal{X} = \mathcal{X}^\top$, $\mathcal{W} = \mathcal{W}^\top$.

If we expand the matrix expression for the objective function and exploit all the symmetries the final expression for $\mathcal{F}$ becomes

$$\begin{aligned} \mathcal{F}(w, \gamma) &= \gamma^* \mathcal{X} \gamma + (\gamma^* \mathcal{X} \gamma)^\top - 2 \Re(\gamma^* \mathcal{Y} \overline{\gamma}) \\ &\quad + \Re(\gamma^* \mathcal{W} \overline{\gamma} + (\gamma^* \mathcal{W} \overline{\gamma})^* - 2 \gamma^* \mathcal{Z} \gamma) \\ &\quad - 2 \Re(\theta^* \gamma - (\theta^\top)^\top \overline{\gamma}) + \frac{1}{2} \|h\|^2. \end{aligned} \tag{5.18}$$

Despite the apparent complexity of the expression above, it would have been quite more complex if we started to compute the objective function directly from [Equation \(5.11\)](#); the quadratic term in γ_k s alone would have accounted for 16 terms. Moreover the expression is a function of only γ_k s and their conjugates. $\mathcal{Z}$ and $\mathcal{Y}$ matrices appear only once, while $\mathcal{X}$ and $\mathcal{W}$ can be transformed after their computation.

5.1.6 Gradient of the Objective Function

Most optimization methods—including all the ones that we consider—require the gradient of the objective function with respect to the w_i s and γ_i s. One way to compute ∇f from [Equation \(5.4\)](#) in conjunction with the formulation of $\mathcal{F}$

in terms on inner products $\langle \cdot, \cdot \rangle$ is given by:

$$\begin{aligned}\nabla \mathcal{F} &= \nabla \langle h - f, h - f \rangle = -\langle \nabla f, h - f \rangle - \langle h - f, \nabla f \rangle \\ &= 2 [\langle f, \nabla f \rangle - \langle h, \nabla f \rangle].\end{aligned}\quad (5.19)$$

The quantities $\langle \nabla f, f \rangle$ and $\langle f, \nabla f \rangle$, while formally different, are actually the same as $\bar{f}(t) = f(t)$ and the linearity of the ∇ operator. In addition, the gradient of a real function with respect to the conjugate of a complex variable is necessarily the conjugate of the gradient with respect to variable itself: $\nabla_{\bar{z}} f = \frac{\partial f}{\partial \bar{z}} = \frac{\partial \bar{f}}{\partial z} = \overline{\nabla_z f}$. Consequently we do not need to explicitly compute the gradient with respect the conjugate of the poles and the coefficients. Equation (5.19) can be written making explicit use of the C-symmetric or the full CR-symmetric formulation of f , which is Equations (5.11) and (5.12) respectively.

C Symmetry

The derivative of f with respect to v_k and β_k is

$$\begin{aligned}\frac{\partial f}{\partial v_k} &= \frac{\beta_k}{(t - v_k)^2}, \\ \frac{\partial f}{\partial \beta_k} &= \frac{1}{(t - v_k)}.\end{aligned}$$

The expressions above are the k^{th} -component of the gradients $\nabla_v f$ and $\nabla_{\beta} f$, respectively. Plugging them in Equation (5.19) and separating terms, we arrive at the matrix expressions

$$\nabla_v \mathcal{F} = 2 [\beta^* \nabla \mathcal{A} + \beta^\top \nabla \bar{\mathcal{B}} - \nabla \zeta^*] \beta \mathbb{I} \quad (5.20)$$

$$\nabla_{\beta} \mathcal{F} = 2 [\beta^* \mathcal{A} + \beta^\top \bar{\mathcal{B}} - \zeta^*] \quad (5.21)$$

where $\mathcal{A}$, $\bar{\mathcal{B}}$ and ζ are the same quantities defined in Section 5.1.5, while

remaining matrices are defined for $1 \leq i, j \leq p$:

$$\begin{aligned}\nabla \mathcal{A}_{i,j} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \bar{v}_i)(t - v_j)^2} dt \\ \nabla \zeta_i &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)h(t)}{(t - \bar{v}_i)^2} dt \\ \nabla \bar{\mathcal{B}}_{i,j} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - v_i)(t - v_j)^2} dt\end{aligned}$$

CR Symmetry

Analogously to the C-symmetric case, we first compute the derivatives of f with respect to the poles and the coefficients

$$\begin{aligned}\frac{\partial f}{\partial w_k} &= \gamma_k \left[\frac{1}{(t - w_k)^2} + \frac{1}{(t + w_k)^2} \right], \\ \frac{\partial f}{\partial \gamma_k} &= \frac{1}{(t - w_k)} - \frac{1}{(t + w_k)}.\end{aligned}$$

After entry-wise substitution of the components above for $\nabla_w \mathcal{F}$ and $\nabla_\gamma \mathcal{F}$, some tedious rearrangement, and using the reflection symmetry with respect to the integration limits, we arrive at the following matrix equations

$$\nabla_w \mathcal{F} = 4 \left[\gamma^* (\nabla \mathcal{X} - \nabla \mathcal{Z}) + \gamma^\top (\nabla \bar{\mathcal{W}} - \nabla \bar{\mathcal{Y}}) - \nabla \theta^* \right] \gamma \mathbb{I} \quad (5.22)$$

$$\nabla_\gamma \mathcal{F} = 4 \left[\gamma^* (\mathcal{X} - \mathcal{Z}) + \gamma^\top (\bar{\mathcal{W}} - \bar{\mathcal{Y}}) - \theta^* \right]. \quad (5.23)$$

The matrices not previously introduced are defined as follows. For $1 \leq k, \ell \leq q$:

$$\begin{aligned}\nabla \mathcal{X}_{k,\ell} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \bar{w}_k)(t - w_\ell)^2} dt \\ \nabla \mathcal{Z}_{k,\ell} &= - \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \bar{w}_k)(t + w_\ell)^2} dt \\ \nabla \bar{\mathcal{W}}_{k,\ell} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - w_k)(t - w_\ell)^2} dt \\ \nabla \bar{\mathcal{Y}}_{k,\ell} &= - \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - w_k)(t + w_\ell)^2} dt\end{aligned}$$

$$\nabla \bar{\theta}_k = \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)h(t)}{(t - w_k)^2} dt$$

Notice that, in both C-symmetric and CR-symmetric cases, the gradients of $\mathcal{F}$ are row vectors. While this is an arbitrary choice, it is a quite natural one to make. Moreover, we decided to maintain the overall multiplicative factor in front of them, this a somewhat arbitrary choice as rescaling the gradient of $\mathcal{F}$ does not influence the minimization process.

5.2 Optimizing SLiSe Filters

With a formulation of the objective function and its gradient that leverage the symmetries, we can now proceed to optimize SLiSe filters. Using the CR symmetries our minimization problem can now be stated as

$$\min_x \mathcal{F}(x) = \min_{x=[w, \gamma]} \|h - f(w, \gamma)\|_2^2, \quad (5.24)$$

In the following, we minimize $\mathcal{F}(x)$ via *descent methods*; this popular family of minimization approaches starts out at a given *starting position* $x^{(0)}$ and iteratively refining this position, such that $\mathcal{F}(x^{(k+1)}) \leq \mathcal{F}(x^{(k)})$. We use the following notation

$$x^{(0)} = [w^{(0)}, \gamma^{(0)}] = [w_1^{(0)}, \dots, w_q^{(0)}, \gamma_1^{(0)}, \dots, \gamma_q^{(0)}].$$

In [Section 5.2.1](#) we present a first method to minimize [Equation \(5.24\)](#). [Section 5.2.2](#) introduces the Levenberg-Marquardt method, a more sophisticated solver for Least-Squares problems. $\mathcal{F}(x)$ is non-convex in the poles w of the filter function and thus a local minimum x^* is not guaranteed to be a global minimum. [Section 5.2.3](#) deals with the choice of the starting position in a way that mitigates this problem. We address the choice of weight function for the Least-Squares optimization in [Section 5.2.4](#). In some cases it is advantageous to optimize more than just the squared differences between the filter and the

indicator function $h(t)$; [Section 5.2.5](#) concludes with a discussion of constrained optimization.

There are alternative approaches to the optimization problem. A very different method is Branch and Bound (B&B), which has a much greater chance to find the global minimum of our objective function. However, B&B requires a convex underestimation of the objective function instead of the gradient. We will not consider this method for two reasons. First, its success highly depends on the ability of finding a suitable underestimation, which is not an easy task. Second, B&B usually is computationally more expensive than the methods we present. Since our goal is to generate filters on the fly, we focus on cheaper methods.

5.2.1 Optimization via Gradient Descent

In this section we remark, through an illustrative example, on a number of issues of the optimization process, which we use to formulate some significant guidelines that are illustrated in the following sections.

The most basic descent method is *gradient descent*. A gradient descent step drives down the objective function, at the current position, along a direction collinear with its negative gradient. Given some mild assumptions, one eventually arrives at a local minimum. For a given starting position $x^{(0)}$ the update step of the gradient descent method is

$$x^{(k+1)} = x^{(k)} + s \cdot \Delta x^{(k)} := x^{(k)} - s \cdot \nabla_x \mathcal{F}(x) \Big|_{x=x^{(k)}}, \quad s \geq 0. \quad (5.25)$$

There are number of ways to select the *step-length* s in [Equation \(5.25\)](#). While it is possible to set s as constant, we can improve the convergence of the method by approximating $s^{(k)} = \operatorname{argmin}_{1 \geq u \geq 0} \mathcal{F}(x^{(k)} + u \cdot \Delta x^{(k)})$ at each iteration. In order to cheaply approximate this minimum, we use a backtracking line search [18]: s is initialized at each step with $s = 1$ and then s is halved until $\mathcal{F}(x + s\Delta x) < \mathcal{F}(x) + \frac{s}{2} \nabla_x \mathcal{F}(x)^* \Delta x$.

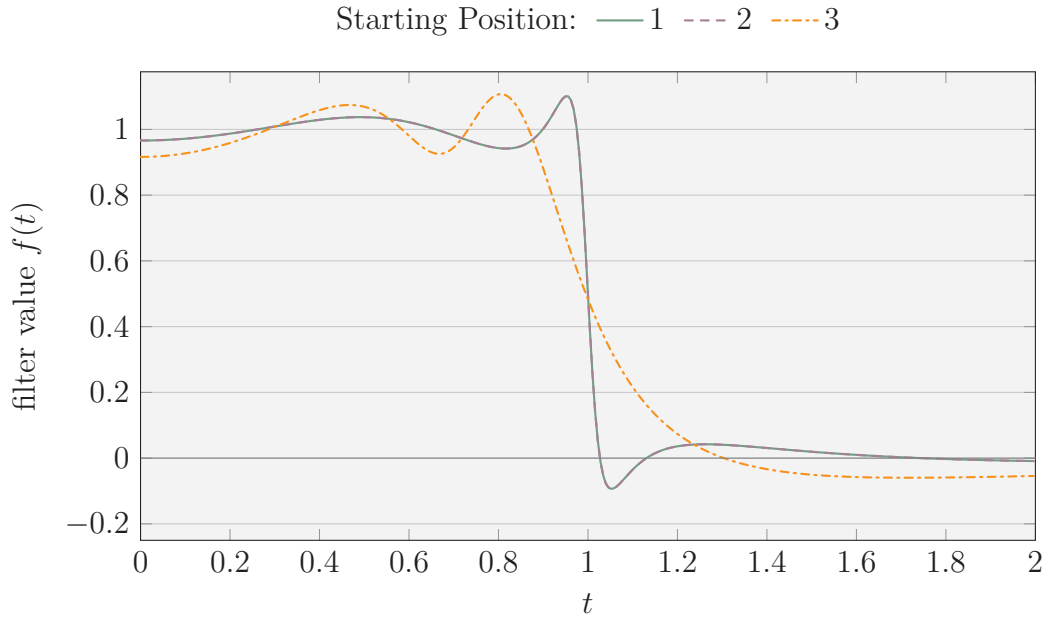


Figure 5.1: Three SLiSe Filters Obtained from Gradient Descent with Different Starting Positions

(See [Table 5.1](#))

	Starting Position		
	1	2	3
w_1	$+0.997360 + 0.044537i$	$+0.721876 + 0.550662i$	$-0.742834 + 0.246735i$
w_2	$+0.721876 + 0.550662i$	$+0.997360 + 0.044537i$	$+0.744336 + 0.212182i$
γ_1	$-0.024019 - 0.002516i$	$-0.135117 - 0.148326i$	$+0.791429 - 0.331455i$
γ_2	$-0.135117 - 0.148326i$	$-0.024019 - 0.002516i$	$+0.639334 + 0.196414i$
$F([w \ \gamma])$	0.005846	0.005846	0.033640

Table 5.1: Poles, Coefficients, and Values of the Objective Function for the Filter Functions Show in [Figures 5.1](#) and [5.2](#)

5.2.1.1 Example: Using Gradient Descent

In the following, we present three filters obtained using the gradient descent method. The real and imaginary parts of the poles w and coefficients γ of the

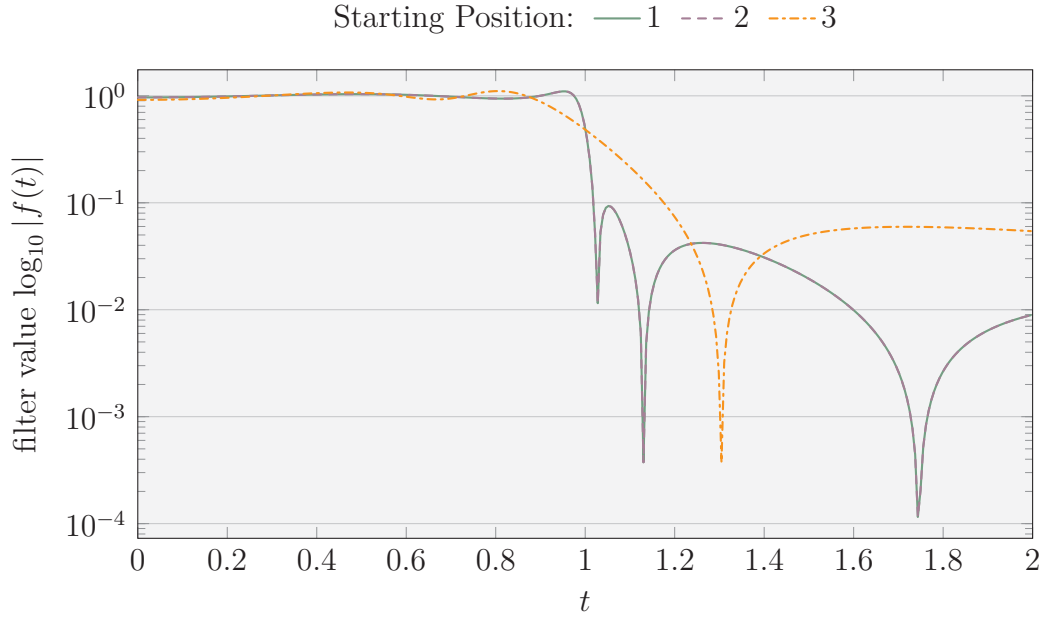


Figure 5.2: Semi-log Plot with Absolute Value of the Filter Functions Given in [Figure 5.1](#)
(See [Table 5.1](#))

starting positions were chosen uniformly random between $(-1, 1)$ ¹. For each filter function $q = 2$, and so the filters have a total of 8 poles and coefficients. Using the symmetries described in [Section 5.1.4](#), we need to optimize only w_1, w_2, γ_1 , and γ_2 . The three filters are plotted on semi-log scale ([Figure 5.2](#)) and on a normal scale ([Figure 5.1](#)). [Table 5.1](#) shows the poles, coefficients, and values of the objective function of the resulting filter functions. Starting Position 1 and 2 result in the same filter. Starting Position 3 generates a different filter with an objective that is one order of magnitude higher than the others. As the semi-log plot shows, this last filter will likely perform significantly worse than the first two. This can be understood by noticing that the filters obtained from Starting Positions 1 and 2 have a smaller absolute value in $[1, 1.21]$ and—not fully visible in the plot— $[1.4, \infty]$. From this simple

¹ The Least-Squares weights used to obtain these results are 1.0 inside $[-1000, 1000]$ and 0.0 everywhere else.

example we can comment on a number of undesirable properties of the gradient descent method:

Remark 5 (Slow convergence)

Gradient descent is a fairly slow method, with linear convergence behavior at best. Even when equipped with a line-search the number of iterations required to reach a minimum is often in the millions. We see in [Section 5.2.2](#) how convergence can be substantially improved by switching to a more advanced Least-Squares optimization method.

Remark 6 (Dependence on starting position)

The filter obtained from the optimization depends on the starting position $x^{(0)}$. Different starting positions can result in very different filters with different values of the objective function. Our experiments indicate that when increasing q the optimization finds more and more local minima, and their values of the objective function compared to the best known solutions get worse. Ideally, we want a method allowing us to choose the starting position in a manner that consistently yields good results. In [Section 5.2.3](#) we discuss an approach to choosing the starting position which mitigates this problem.

Remark 7 (Lack of flexibility)

In order to obtain filters with distinct properties we need to modify the optimization process accordingly. [Section 5.2.4](#) outlines a number of criteria for choosing the Least-Squares weight function $\mathfrak{G}(t)$. Furthermore, out-of-the-box gradient descent does not support constrained optimization. In [Section 5.2.5](#) we illustrate a number of ways to use optimization constraints on filters.

5.2.2 Improving Convergence Speed

We address [Remark 5](#) by introducing the Levenberg-Marquardt method [[68](#), [59](#)] (LM). This is a solver for non-linear Least-Squares problems with a better convergence rate than gradient descent. LM is a hybrid of the gradient descent and the Gauss-Newton method (a good description of both methods can be found in [[66](#)]). We present the Gauss-Newton method first, then introduce the

LM method. We give a formulation based on the inner product presented in Equation (5.14).

For the sake of clarity, we rewrite the objective function in terms of $\xi([z \ \alpha], t) = h(t) - f(t, z, \alpha)$

$$\begin{aligned}\mathcal{F}(x) &= \|\xi(x)\|_2^2 = \langle \xi(x), \xi(x) \rangle = \int_{-\infty}^{\infty} \mathfrak{G}(t) |\xi(x, t)|^2 dt \\ &= \int_{-\infty}^{\infty} \mathfrak{G}(t) |h(t) - f(t, z, \alpha)|^2 dt,\end{aligned}$$

where we indicate the collection of parameters $[z \ \alpha]$ with x . The basis of the Gauss-Newton method is a linear approximation of ξ

$$\xi(x + \Delta x) = \xi(x) + \nabla \xi(x) \cdot \Delta x \doteq \xi + \sum_i \frac{\partial \xi}{\partial x_i} \Delta x_i \quad (5.26)$$

which, in the following, we refer to with the shortcut notation $\xi + \nabla \xi \cdot \Delta x$. The Levenberg-Marquardt method aims at minimizing a linear approximation of the objective function by Gauss-Newton iterates. By using the linear approximation of ξ , such a requirement can be formulated as the minimization of the following function with respect to Δx

$$\begin{aligned}\mathcal{F}(x + \Delta x) &= \|\xi(x + \Delta x)\|_2^2 = \langle \xi, \xi \rangle + 2\langle \xi, \nabla \xi \cdot \Delta x \rangle + \langle \nabla \xi \cdot \Delta x, \nabla \xi \cdot \Delta x \rangle \\ &= \mathcal{F}(x) + \nabla \mathcal{F}(x) \cdot \Delta x + (\Delta x)^* \cdot \langle \nabla \xi, \nabla \xi \rangle \cdot \Delta x.\end{aligned}$$

We mildly abuse the notation of the scalar product $\langle \cdot, \cdot \rangle$, as it does not necessarily result in a scalar; for example, $\langle \nabla \xi, \nabla \xi \rangle$ results in a matrix. Notice that the condition $\bar{\xi}(x + \Delta x) = \xi(x + \Delta x)$ implies that $(\Delta x)^* \cdot \bar{\nabla} \xi = \nabla \xi \cdot \Delta x$ with the consequence that $\langle \nabla \xi \cdot \Delta x, \xi \rangle = \langle \xi, \nabla \xi \cdot \Delta x \rangle$. Taking the partial derivative of $\mathcal{F}(x + \Delta x)$ with respect to Δx and equating it to zero one gets

$$\partial_{\Delta x} \mathcal{F}(x + \Delta x) = 2 [\langle \xi, \nabla \xi \rangle + \langle \nabla \xi, \nabla \xi \rangle \cdot \Delta x] = 0$$

Using the formulation above, the Gauss-Newton method iterates over k in

the following way:

1. Set: $\mathcal{H} := \langle \nabla \xi(x^{(k)}), \nabla \xi(x^{(k)}) \rangle$
2. Solve: $\mathcal{H} \cdot \Delta x_{GN}^{(k)} = -\frac{1}{2} \nabla \mathcal{F}(x^{(k)})$
3. Update: $x^{(k+1)} = x^{(k)} + \Delta x_{GN}^{(k)}$.

Here $\mathcal{H}$ acts as a linear approximation of the Hessian of $\mathcal{F}$. Consequently, if $\mathcal{H}$ is well-conditioned, the Gauss-Newton method can have quadratic convergence. However, there are still scenarios where Gauss-Newton steps provide only a small improvement of the objective function. In those cases it is beneficial to temporarily switch to gradient descent, even though it only has linear convergence at best. The Levenberg-Marquardt method employs a *dampening parameter* $\mu \geq 0$ to switch between the Gauss-Newton and gradient descent methods. By adding a dampening term, the second step is substituted by

$$2. \text{ Solve: } (\mathcal{H} + \mu \mathbb{I}) \Delta x_{LM}^{(k)} = -\frac{1}{2} \nabla \mathcal{F}(x^{(k)}), \quad (5.27)$$

where the gradient of $\mathcal{F}$ is intended over both the poles and coefficients. Such an addition is equivalent to a constrained Gauss-Newton with the parameter μ working as a Lagrange multiplier.

In practice, the dampening parameter μ is readjusted after every iteration. For a small μ , the solution to Equation (5.27) is similar to the Gauss-Newton update step $\Delta x_{LM} \simeq \Delta x_{GN}$. On the other hand, when μ is large then $\Delta x_{LM} \simeq -\frac{1}{\mu} \nabla \mathcal{F}(x)$, which is similar to the gradient descent update step. Often, instead of $\mathcal{H} + \mu \mathbb{I}$ one uses $\mathcal{H} + \mu \cdot \text{Diag}(\mathcal{H})$. Generally, when a LM-step reduces the value of the objective function by a large amount, then μ is decreased. When a step does not decrease the Least-Squared error, or not by enough, μ is increased.

Since the formulation of the LM method given so far is independent from the symmetry of f , it is valid for any formulation of the rational filter. Let us look now more in detail at $\mathcal{H}$. This is a matrix whose entries are $\mathcal{H}_{i,j} = \langle \nabla_{x_i} f, \nabla_{x_j} f \rangle$

which, in the case of CR symmetry, can be represented in block form as

$$\mathcal{H} = \begin{pmatrix} \mathcal{H}_2 & -\overline{\mathcal{H}}_1 & \overline{\mathcal{H}}_1 & -\mathcal{H}_2 \\ -\mathcal{H}_1 & \mathcal{H}_2^* & -\mathcal{H}_2^* & \mathcal{H}_1 \\ \mathcal{H}_1 & -\mathcal{H}_2^* & \mathcal{H}_2^* & -\mathcal{H}_1 \\ -\mathcal{H}_2 & \overline{\mathcal{H}}_1 & -\overline{\mathcal{H}}_1 & \mathcal{H}_2 \end{pmatrix}. \quad (5.28)$$

Only two of the matrix blocks composing $\mathcal{H}$ are independent and are made of the following sub-blocks

$$\begin{aligned} \mathcal{H}_1 &= \begin{pmatrix} \langle \nabla_w f, \nabla_w f \rangle & \langle \nabla_w f, \nabla_\gamma f \rangle \\ \langle \nabla_\gamma f, \nabla_w f \rangle & \langle \nabla_\gamma f, \nabla_\gamma f \rangle \end{pmatrix}, \\ &= 2 \begin{pmatrix} I_\gamma [\nabla \nabla \overline{\mathcal{W}} - \nabla \nabla \overline{\mathcal{Y}}] I_\gamma & I_\gamma [\nabla \overline{\mathcal{W}} - \nabla \overline{\mathcal{Y}}]^\top \\ [\nabla \overline{\mathcal{W}} - \nabla \overline{\mathcal{Y}}] I_\gamma & \overline{\mathcal{W}} + \overline{\mathcal{Y}} \end{pmatrix} \end{aligned} \quad (5.29)$$

$$\begin{aligned} \mathcal{H}_2 &= \begin{pmatrix} \langle \nabla_{\overline{w}} f, \nabla_w f \rangle & \langle \nabla_{\overline{w}} f, \nabla_\gamma f \rangle \\ \langle \nabla_{\overline{\gamma}} f, \nabla_w f \rangle & \langle \nabla_{\overline{\gamma}} f, \nabla_\gamma f \rangle \end{pmatrix} \\ &= 2 \begin{pmatrix} I_\gamma^* [\overline{\nabla} \nabla \mathcal{X} - \overline{\nabla} \nabla \mathcal{Z}] I_\gamma & I_\gamma^* [\nabla \mathcal{X} - \nabla \mathcal{Z}]^* \\ [\nabla \mathcal{X} - \nabla \mathcal{Z}] I_\gamma & \mathcal{X} - \mathcal{Z} \end{pmatrix} \end{aligned} \quad (5.30)$$

where the only additional matrices that need to be defined are

$$\begin{aligned} \nabla \nabla \overline{\mathcal{W}}_{i,j} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - w_i)^2 (t - w_j)^2} dt \\ \nabla \nabla \overline{\mathcal{Y}}_{i,j} &= - \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - w_i)^2 (t + w_j)^2} dt \\ \overline{\nabla} \nabla \mathcal{X}_{i,j} &= \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{w}_i)^2 (t - w_j)^2} dt \\ \overline{\nabla} \nabla \mathcal{Z}_{i,j} &= - \int_{-\infty}^{+\infty} \frac{\mathfrak{G}(t)}{(t - \overline{w}_i)^2 (t + w_j)^2} dt \\ I_\gamma &= \gamma \mathbb{I}. \end{aligned}$$

The factor of 2 in front of the block matrices $\mathcal{H}_1$ and $\mathcal{H}_2$ comes from the R symmetry in combination with the evenness of the integral boundaries (i.e., $\mathcal{X} = \mathcal{X}^\top$, $\nabla \mathcal{X} = \nabla \mathcal{X}^\top$, etc.). Notice that block rows 3 and 4 of Equation (5.28) are exactly equivalent, up to a sign, to block rows 2 and 1, respectively. Similarly, block columns 3 and 4 are linearly dependent with respect to block columns 2 and 1. This is expected due to the CR symmetry. In addition, $\mathcal{H}_1$ is complex symmetric ($\mathcal{H}_1^* = \overline{\mathcal{H}_1}$), and $\mathcal{H}_2$ is complex Hermitian ($\mathcal{H}_2^* = \mathcal{H}_2$) which, implicitly, verifies that $\mathcal{H}$ is complex Hermitian itself.

Going back to the Equation (5.27), one can also write the vector Δx_{LM} in block notation,

$$\Delta x_{LM} = \begin{bmatrix} \Delta y, \\ -\overline{\Delta y}, \\ (\overline{\Delta y}), \\ -\Delta y \end{bmatrix}$$

with $(\Delta y)^\top = [\Delta w^\top \Delta \gamma^\top]$. Writing Equation (5.27) in terms of Δy with $\mu \cdot \text{Diag}(\mathcal{H})$ in place of $\mu \mathbb{I}$ one obtains 4 separate equations. Thanks to the symmetries of $\mathcal{H}$, it is possible to reduce them to a single equation solving for Δy . There are several different ways to achieve this result. Starting from the third equation, one can extract $(\mathcal{H}_1 \Delta y)$ as a function of $\mathcal{H}_2^*$, $\Delta \overline{y}$, and $\nabla_{\overline{y}} F$, then take the conjugate of it, and substitute it in the fourth equation. The final result is a linear system whose dimension is one fourth of the original size of $\mathcal{H}$

$$[i \Im(\mathcal{H}_2) - \mu \text{Diag}(\mathcal{H}_2)] \Delta y = \frac{1}{2} \nabla_y \mathcal{F} \quad (5.31)$$

5.2.3 Systematic Choice of Starting Position

In this section we address the dependence of the optimization process on the initial conditions. In Remark 6 we pointed out that the optimized filter crucially depends on the starting position. Usually, the choice of the starting position for non-linear optimization is a difficult problem. Many heuristics exist, some methods rely on solving a convex variant of the problem [18]. Generally, it is preferable to use domain knowledge in the initial choice of starting position.

Recall that the parameters of our optimization, the poles and coefficients, define a rational filter function. In general, random choices for w and γ do not produce good filters. We can reinterpret the problem of finding good starting positions for the poles and coefficients as finding a filter that serves as a good initial guess for an optimized filter. As previously discussed, there are multiple perspectives which once can use to generate filters: Numerical integration of the spectral projector, rational filters from signal processing, and function approximation. The Gauss-Legendre filter is a filter obtained from numerical integration, and is FEAST's default filter. Recommended in the presence of spectral clusters, the Elliptic or Cauchy filter [40] is a classic signal processing filter.

To realize why the starting positions are crucial for the optimization method let us consider what happens when increasing q , the number of poles and coefficients in one quadrant of $\mathbb{C}$. For some spectrum slices of a given eigenvalue problem, larger values for q , say $q = 10$ or higher, may be required to obtain the desired convergence of the solver. For instance, some methods, such as the DD-PP projection method [54] that do not iterate the subspace, have little choice but to choose large q . A given value of q results in $4q$ real degrees of freedom in the optimization: q poles and q coefficients, each one with a real and imaginary part. As a result, optimization with large q becomes more and more expensive. In Figure 5.2 we have seen an example of multiple local minima of the objective function $\mathcal{F}$ with $q = 2$. Our extensive numerical experiments indicate that higher q values not only increase the number of local minima in $\mathcal{F}$, but have the effect of worsening of the objective function of most of these additional local minima with respect to the best known local minimum. In one of our numerical experiments, using a random starting position with $q = 8$ yields a filter with a (Least-Squares) error of 5.4×10^{-4} , while with $q = 4$ we achieved a objective of 2.4×10^{-4} . Choosing $q = 8$ requires 18 linear system solves for each filter application, whereas $q = 4$ requires only 8 solves. To offset the additional cost of the linear system solves we want a filter that has a lower objective function value, thus likely requiring fewer subspace iterations. Without a good starting position we obtain a filter with $q = 8$ that requires *more* linear system solves and *more* iterations. What is more, the optimization

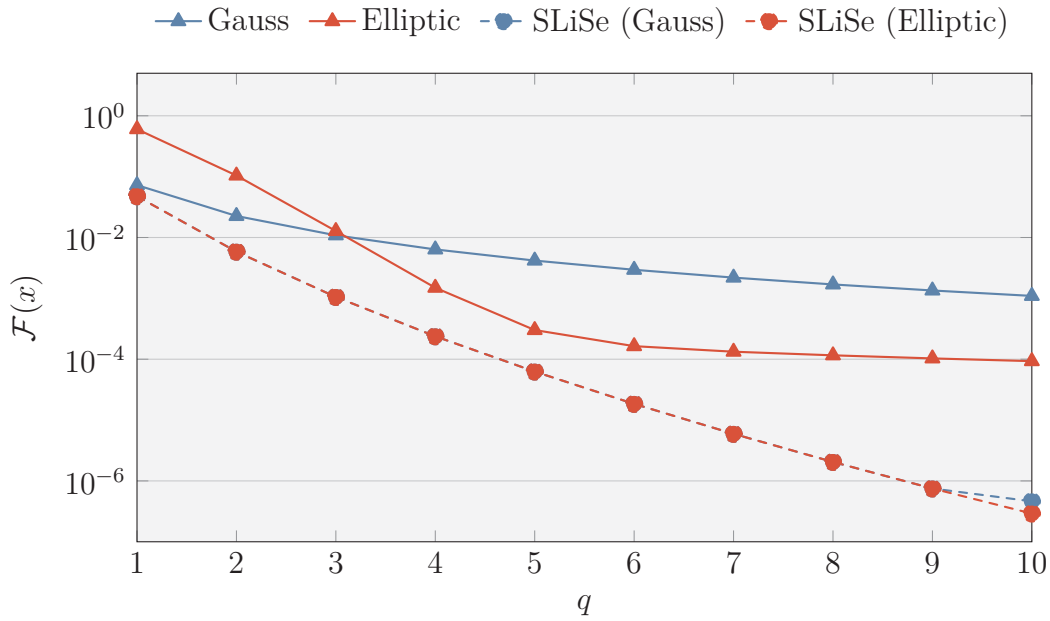


Figure 5.3: Values of the Objective Function for SLiSe Filters Obtained from Gauss and Elliptic Filters as Starting Positions

for a filter with $q = 8$ is significantly more expensive than for $q = 4$. Simply put, random starting positions yield worse and worse filters for larger q .

While choosing random starting positions does not work well with larger q , choosing existing filters produces very good results. Figure 5.3 shows the values of the objective function² of different filters for different q . The values of the objective function of the Gauss filter are indicated with blue triangles, and those of the Elliptic filter in red triangles. Shown with blue circles is the objective of the filter that is obtained when the initial position of the SLiSe optimization is started with the Gauss filter poles and coefficients. Conversely, the red circles indicate the values of the objective function achieved by the filter obtained by starting SLiSe optimization with the Elliptic filter. The SLiSe filters have significantly lower values of the objective function than the Gauss and Elliptic filters. Using poles and coefficients from Gauss and Elliptic filters as starting positions for $q < 10$ results in the same unconstrained SLiSe

² The weight function used has unit weight between -1000 and 1000

$ t \in [0, .95)$	$[.95, .995)$	$ [.995, 1.005)$	$[1.005, 1.05)$	$[1.05, 1.2)$	$[1.2, 2)$	$[2, 5)$	$[5, \infty)$
$\mathfrak{G}_1(t)$	1	4	2	4	1	0	0
$\mathfrak{G}_2(t)$	1	4	2	4	1	.05	.001
$\mathfrak{G}_3(t)$	1	4	0	0	1	.2	.2

Table 5.2: Three Separate Weight Functions $\mathfrak{G}_1$, $\mathfrak{G}_2$, and $\mathfrak{G}_3$. The resulting filter functions have undesirable properties that make them poor rational filters.

filter. The number of iterations in the LM method differs for both starting positions, when starting with the Elliptic filter fewer iterations are required. In our experience existing filters, and the Elliptic filter in particular, make for excellent starting positions practically eliminating the problem of finding an appropriate starting position.

5.2.4 The Least-Squares Weights $\mathfrak{G}(t)$

A major influence on the optimization procedure is the Least-Squares weight function $\mathfrak{G}(t)$ —“weights”, for short—used in the optimization process. As discussed in [Remark 7](#) the optimization process requires some flexibility, and the weight function supplies this flexibility.

The result of the optimization is invariant under scaling of the weight function, so we consider scaled weight functions with $\mathfrak{G}(0) = 1$, for the sake of consistency. With the flexibility provided by the weight function comes the problem of specifying a weight function that results in a filter with the desired properties. We will see that setting weights without a specific strategy almost certainly yields unexpected results.

We aim for a fire-and-forget approach for the optimization process. In this sense, our target is to obtain the desired results without parameter space exploration. This section contains three guidelines for the choice of $\mathfrak{G}(t)$. The guidelines help the user with the choice of the weights outside the search interval $[-1, 1]$, near its endpoints, and inside it. Each guideline is a heuristic that can be

easily implemented as part of an optimization algorithm, or checked manually. These guidelines are an important ingredient for a usable optimization, as following them virtually eliminates the problem of filters with unexpected properties. We provide examples for unexpected properties that occur when violating Guidelines 1 and 2. However, if a filter with very specific properties is desired, some trial-and-error is unavoidable unless some form of hyperparameter optimization is used, see [Section 5.4](#).

The weight function must be non-zero in an interval larger than just $[-1, 1]$. Otherwise the optimization may result in filters that have large absolute values outside $[-1, 1]$. However, care must be taken when choosing the weights outside the search interval. By tapering the weights too quickly we obtain a steep filter that will nevertheless exhibit insufficient convergence rates. In this context the quality of a filter can be expressed as a function of the filter's local extrema outside of the search interval. Specifically, non-increasing local maxima of the absolute function value away from the search interval is a very desirable property. Existing filters already have this property. The Gauss filter, for example, has local extrema that decay particularly quickly.

Guideline 1 (Outside)

Slowly taper off the weights in a large enough neighborhood outside the search interval. In order to avoid trial-and-error, we suggest to detect increasing local maxima of the absolute value of the rational function during the optimization process and increase the weights where required.

By construction, existing filters have a value of 0.5 at the endpoints of the search interval, $f(1) = f(-1) = 0.5$. While not strictly required for most contour solvers, other methods that use rational filters do require a value of 0.5 at the end of the search interval. One example of such a method is the estimation of eigenvalues inside an interval [27]. SLiSe filters are not guaranteed to have a value of 0.5 at the endpoints, unless an appropriate constraint is added to the optimization. In practice SLiSe filters with $f(\pm 1) \approx 0.5$ are easily achievable via appropriate weight functions. Such an approximate value is enough for most applications. By choosing the weight function as symmetric

around some region around the endpoint of the search interval we achieve $f(\pm 1) \approx 0.5$. If an exact value is required the filter can be scaled accordingly.

Guideline 2 (Endpoints)

The weight function around the endpoints of the search interval should be chosen symmetrically. Accordingly, during the optimization process, choose $\mathfrak{G}(\pm 1 - \epsilon) = \mathfrak{G}(\pm 1 + \epsilon)$ for $\epsilon \leq 0.2$. Additionally, the optimization procedure should check the weight function for symmetry around the search interval endpoints and, if desired, scale the resulting filter to $f(\pm 1) = 0.5$.

When the weights inside $[-1, 1]$ are too small the filter function will oscillate inside the search interval. Oscillations cause the filter value to dip below 1.0, hurting the convergence rate of the entire subspace. We can effectively solve this problem by adjusting the weight function during the optimization process. Previously, we suggested to increase the weights outside the search interval to produce filters with non-increasing local maxima. By the same token we can detect any oscillation inside $[-1, 1]$ and increase the assigned weights accordingly.

Guideline 3 (Inside)

The weights in the entire search interval should be chosen large enough to prevent large oscillations. During the optimization process, monitor the difference between minima and maxima of the filter inside $[-1, 1]$ and adapt weights accordingly.

Example: Very Small Weights Outside the Search Interval

Let us look what happens when the resulting filters were obtained without adhering to one of the three guidelines. In the process of obtaining a replacement candidate for the Elliptic filter the weight function in [Table 5.2](#) could be considered. We will see that $\mathfrak{G}_1$ and $\mathfrak{G}_2$ do not taper off the weights outside $[-1, 1]$ in accordance with Guideline 1.

Starting from the assumption that only the weights near the endpoints are relevant, so $\mathfrak{G}_1$ is chosen with non-zero values only for $|t| < 1.2$. [Figures 5.4](#)

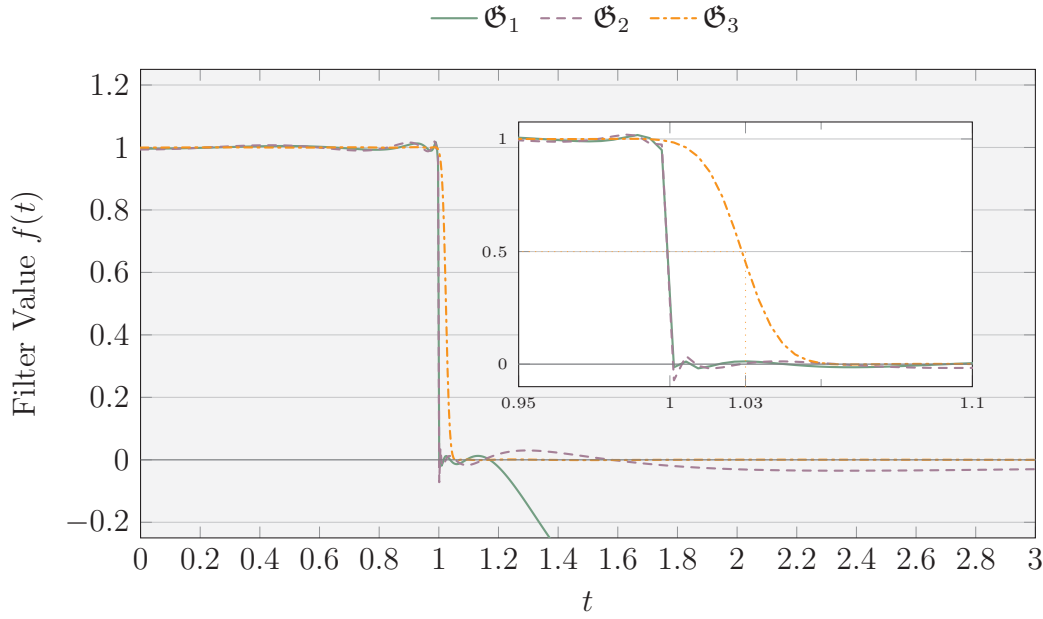


Figure 5.4: Plots of Filters Obtained with Three Separate Weight Functions $\mathfrak{G}_1$, $\mathfrak{G}_2$, and $\mathfrak{G}_3$. The resulting functions act as poor rational filters. (Linear-scale plot. Smaller plot: Abscissa limited to $[0.9, 1.1]$.)

and 5.5 show the filter function obtained with $\mathfrak{G}_1$. The filter is a good approximation within $[-1.2, 1.2]$. However, for $|t| > 1.2$, the rational function poorly approximates zero, and would hardly work as a filter at all. In fact, a value of $f(1.8) \approx -0.5$ for the filter would result in severely degraded convergence rates, see Theorem 3. To obtain a better filter it is enough to adjust the weight function to have non-zero values in a larger interval around $[-1, 1]$, for instance $[-5, 5]$. The weight $\mathfrak{G}_2$ satisfy our requirement, but the weights taper off very quickly for $t > |1.2|$. The resulting filter function has values close to zero outside the search interval. Despite this improved behavior, the filter has another more subtle shortcoming. Figure 5.5 shows in a semi-log plot that the filter drops steeply to low values near $[1, 1.2]$, just as desired. However, the filter's values inside $[1.2, 3]$ are *larger* than the values inside $[1, 1.2]$. In other words, some of the local maxima outside of the search interval increase away

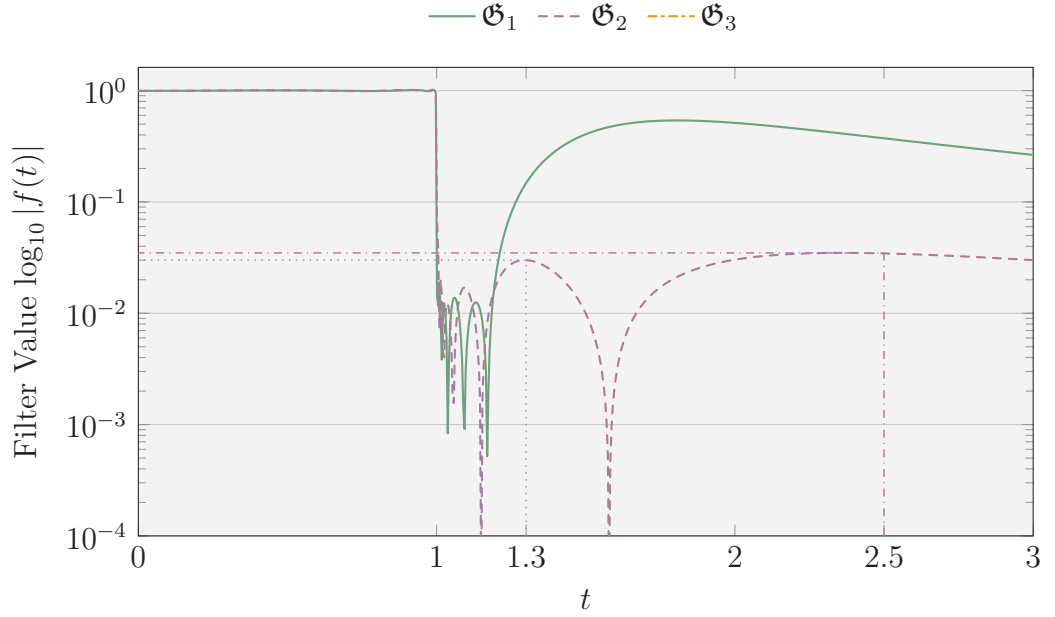


Figure 5.5: Plots of Filters Obtained with Weight Functions $\mathfrak{G}_1, \mathfrak{G}_2$
(Semi-log plot of Figure 5.4. $\mathfrak{G}_3$ omitted for better readability.)

from it, once again resulting in deteriorated convergence rates. Sufficiently large weights in $1.2 < |t| < 5$ result in non-increasing local minima.

Example: Asymmetric Weights Near Endpoints

The weights $\mathfrak{G}_3$ in Figures 5.4 and 5.5 illustrate the results of a weight function that is not symmetric around the endpoint of the search interval, violating the second guideline. In particular, $\mathfrak{G}_3$ has a larger weight on the inside of the search interval than on the outside. Figure 5.4 plots the resulting filter function as an inlay inside the figure. Assigning a weight of zero between 0.995 and 1.05 results in a filter that fails to drop to zero at the end of the search interval: its value at the endpoint is almost one! Because of the delayed drop the filter crosses the value 0.5 at a later point $f_{\mathfrak{G}_3}(1.025) \approx 0.5$. This behavior does not appear in the filter produced with $\mathfrak{G}_1$ and $\mathfrak{G}_2$ as they respect the second guideline: the weights are symmetric around the endpoints $|t| = 1$ and $f(\pm 1) \approx 0.5$ without the need to rescale the filter. This effect can be

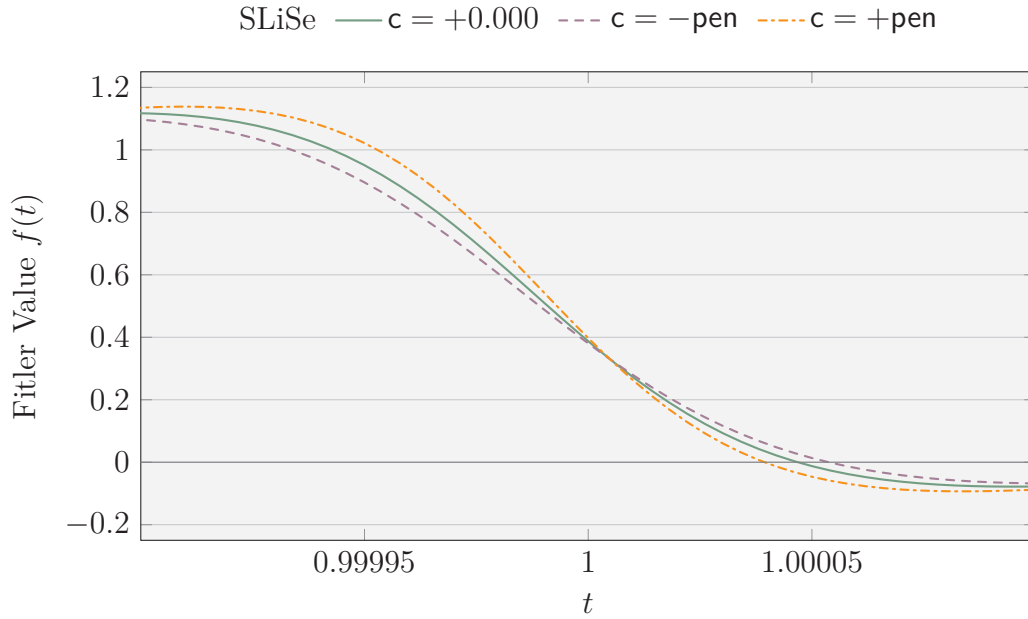


Figure 5.6: Examples of Filters Obtained via Penalty Terms: An Appropriately Sized Penalty Parameter Yields Filters of Different Steepness
($\text{pen} = 2 \cdot 10^{-10}$)

understood in terms of the minimization of the squared errors during the function approximation. The continuous drop of the filter function at the endpoints, no matter how steep, incurs in a large squared error. When the weight function lacks symmetry around the endpoint of the search interval, the optimization shifts the drop towards the part of the neighborhood of $[-1, 1]$ with lesser weight. The result is a filter value $f(\pm 1) < 0.5$ if the weight is smaller just inside the search interval, and a value $f(\pm 1) > 0.5$ if the weight is smaller just outside the search interval. Choosing symmetric weights around the endpoints results in $f(\pm 1) \approx 0.5$.

5.2.5 Constrained Optimization

We have seen that the weight function $\mathfrak{G}$ provides some flexibility to the optimization process. However, some properties we want in a filter cannot be directly expressed by an appropriate choice of $\mathfrak{G}$. In this section we discuss

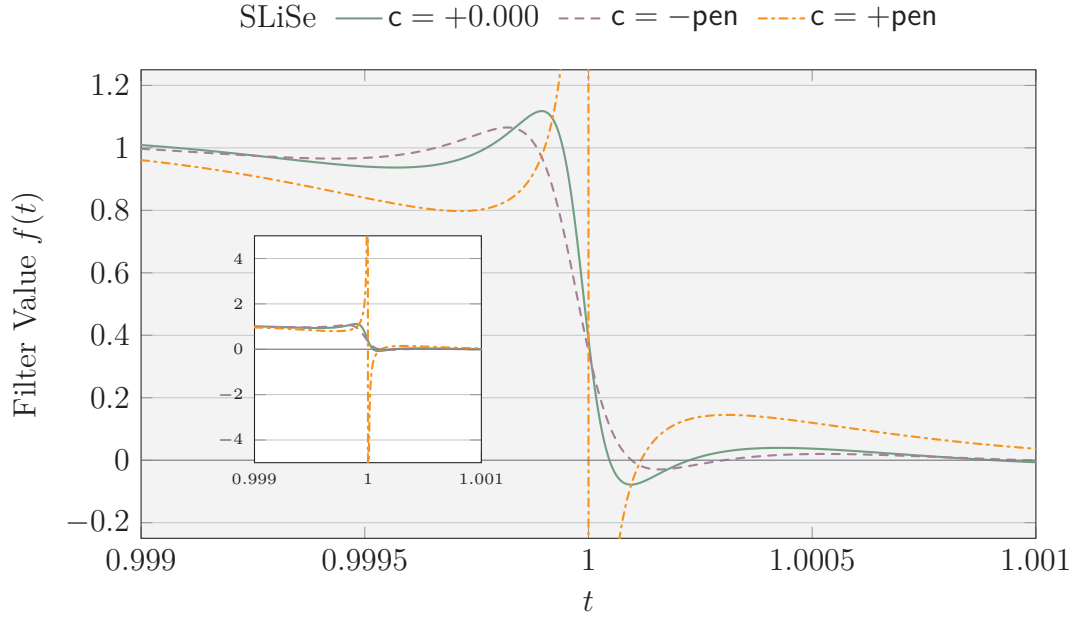


Figure 5.7: Examples of Filters Obtained via Penalty Terms: Too Large a (Positive) Penalty Parameter Results in a Pole on the Real Axis ($\text{pen} = 2 \cdot 10^{-9}$)

two filter properties that can be achieved by implementing constraints on the optimization. First, we discuss a penalty term on the gradient of the filter function, which results in filters of varying “steepness” and with varying amounts of “overshooting”. Given our standard search interval, $(-1, 1)$, we call the derivative of the filter $f(t)$ at the endpoints of the interval $\nabla_t f(t)|_{t=1} = -\nabla_t f(t)|_{t=-1}$ the *steepness* of the filter. The steepness—or separation factor—of a filter has previously been suggested as a quality measure in the context of optimizing exclusively the coefficients of the rational function [102]. A steep filter is considered desirable because it dampens eigenvalues outside the search interval even when the eigenvalues are close to the interval endpoints and, in doing so, improves convergence. When optimizing the poles of the rational functions together with its coefficients the steepness alone does not serve as a good measure of quality anymore. It is possible for a filter to be steep, and yet not to dampen eigenvalues outside of the search interval sufficiently well. The lack

of dampening is due to an effect called overshooting, corresponding to a filter that greatly exceeds its target value. Overshooting is a known behavior when approximating discontinuous functions with continuous ones.

Usually a very steep filter tends to overshoot at the endpoint of $(-1, 1)$, and conversely filters that do not overshoot tend to not be particularly steep.

In [Section 5.2.5.1](#) we discuss how to adjust the optimization to generate filters of varying steepness, and thus also with varying amounts of overshooting. A *penalty term* can modify the optimization procedure to influence the steepness. By adding a term to the objective function we penalize steepness by a factor of `pen`, the penalty parameter.

$$\min_{w_i, \gamma_i} \|h - f(w, \gamma)\|_2^2 + \frac{\text{pen}}{2} \cdot (\nabla_t f(w, \gamma, t)|_{t=1} - \nabla_t f(w, \gamma, t)|_{t=-1}). \quad (5.32)$$

Note that since $\nabla_t f(1)$ is supposed to be negative, a positive `c` will make for a steeper filter when minimizing the error. Solving the optimization problem requires straight-forward changes to the gradient.

[Section 5.2.5.2](#) describes a different kind of constraint, namely box constraints on the imaginary part of the poles. Such a constraint may be advantageous when Krylov solvers are used for the linear system solves as they appear in [Equation \(5.2\)](#). The condition number of the shifted matrix $A - w_i \mathbb{I}$ may become large when $|\Im m(w_i)|$ is small, which can occur for large degrees of the Elliptic and SLiSe filters. One possible way to circumvent this problem is to ensure that the absolute value of the imaginary part for each pole is large enough. For SLiSe filters this translates into a box constraint d on the parameters:

$$\min_{w_i, \gamma_i} \|h(t) - f(t, w, \gamma)\|_2^2 \quad \text{s.t.} \quad \forall i : |\Im m\{w_i\}| \geq d \quad (5.33)$$

Both constrained optimization approaches are not fire-and-forget; some trial and error is always required for the parameter selection. This lack of robustness for the parameter selection is the exception in our work. Constrained

pen	Least-Squares Error	Steepness
0	2.265829×10^{-5}	-1.211081×10^4
-2×10^{-10}	2.279018×10^{-5}	-1.063877×10^4
$+2 \times 10^{-10}$	2.288353×10^{-5}	-1.429542×10^4

Table 5.3: Values of the Objective Function and Steepness of the Filter Functions Shown in Figure 5.6

optimization without the need for extensive parameter search is an area of future work.

5.2.5.1 Penalty Term

Before we look at some examples of filters obtained with penalty terms, let us consider how the steepness and overshooting influence the convergence behavior of the filter by comparing γ -SLiSe and the Gauss filter from Figure 5.11. By starting to drop significantly earlier than the endpoint of the search interval at $t = 1$, the Gauss filter is significantly less steep than the SLiSe filter. Conversely, the SLiSe filter maintains a value close to one inside the search interval closer to the endpoint, and it takes small values just outside the search interval. The increased steepness of the γ -SLiSe filter comes at the cost of overshooting. Figure 5.1 clearly shows that the SLiSe filter shoot up above 1 and drops below 0 just inside and outside the search interval respectively. In terms of the convergence rate, the surge above 1 is not a problem, however the overshooting outside $(-1, 1)$ is an undesirable effect. An overshooting filter maps eigenvalues to relatively large negative—and thus large in magnitude—values outside the endpoints, hurting convergence.

While it is possible to increase the steepness of SLiSe filters via the penalty term, there is little benefit in even steeper filters with more overshooting. Figure 5.6 shows an illustrative example of filters obtained with an appropriately sized penalty parameter and number of poles per quadrant $q = 4$. Table 5.3 shows the Least-Squares errors without penalty term and the steepness of the filter $\nabla_t f(1)$ for the resulting filters. The figure makes plain how a negative

penalty term tends to decrease steepness and overshooting while a positive **pen** increases them. Since the penalty parameter is chosen significantly smaller than the error, the change to the filter is not very pronounced. Choosing the penalty parameter to be larger is not without hazards, especially when **pen** is chosen to be positive. Figure 5.7 shows a filter generated with a larger value for **pen**, again with $q = 4$. Such a filter has a pole on the real axis which results in a very steep slope, but not an effective filter. For one thing, the pole causes significant overshooting which substantially hurts convergence. Additionally, a real pole affects the corresponding linear system, which can become seriously ill-conditioned causing failure or slow converge of the iterative solver. While a large and positive parameter is problematic because of the risk of a real pole, this is usually not a concern in practical applications. Often, it is more important to limit the overshooting than it is to make the already steep SLiSe filters even steeper. Figure 5.7 shows that a negative penalty parameter **pen** reduces overshooting³. η -SLiSe was obtained using a negative penalty parameter, which we discuss further in Section 5.4.

5.2.5.2 Large Imaginary Parts via Box Constraints

Implementing box constraints in our framework is straightforward: Instead of the old update step in Equation (5.25) we use gradient projection by updating as follows

$$x^{(k+1)} = \mathcal{P} \left(x^{(k)} - s \cdot \nabla_x F(x) \Big|_{x=x^{(k)}} \right), \quad s \geq 0, \quad (5.34)$$

where $\mathcal{P}(x)$ projects x into the constraints. In practice we project onto the constrained value whenever the constraint is violated. We implement this scheme by forcing the absolute value of the imaginary part of every pole to be equal or larger than some value **lb**. For a single pole w_l the projection would be

$$\mathcal{P}(w_l) = \begin{cases} \Re(w_l) + i \cdot \text{sgn}(\Im(w_l)) \cdot \text{lb} & |\Im(w_l)| < \text{lb} \\ \Re(w_l) + i \cdot \Im(w_l) & \text{otherwise} \end{cases}$$

³ Some trial and error may be required to obtain a penalty parameter of appropriate size.

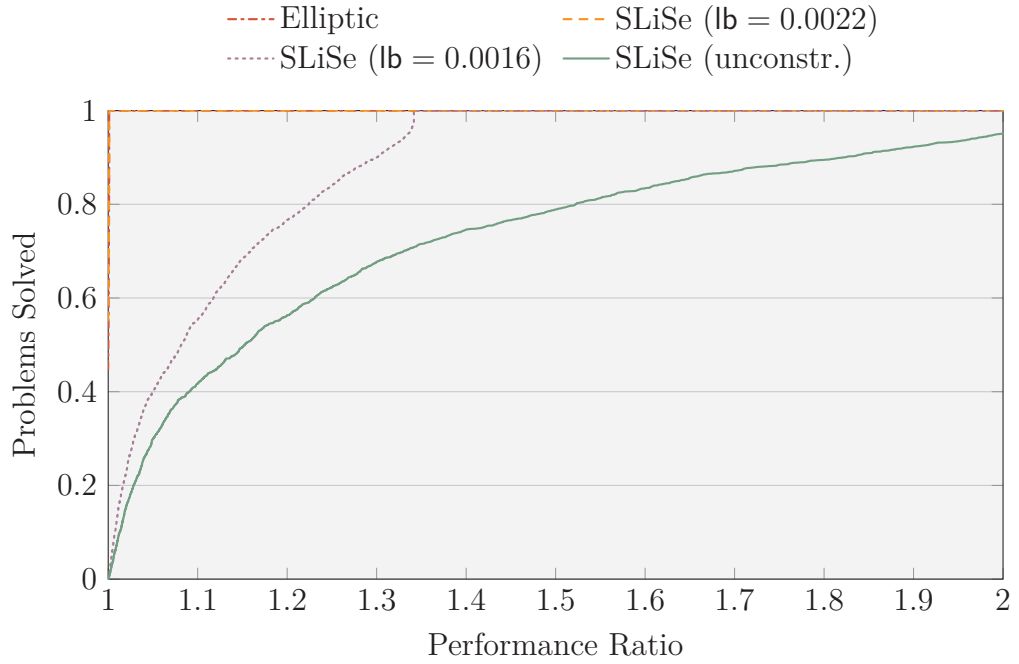


Figure 5.8: Performance Profile Comparing the Worst Condition Number for the Linear Systems Arising from Filter Application. The Performance Ratio for the Elliptic and SLiSe ($\text{lb} = 0.0022$) Filters are One over the Entire Abscissa Range. Performance Profiles are introduced in [Section 5.3.1.1](#).

(2117 problems obtained from the [Si2](#) matrix.)

In the following, we compare filters with different box constraints. [Figure 5.8](#) shows four filters each with $q = 4$. Shown are the Elliptic filter, a SLiSe filter obtained via unconstrained optimization⁴, and two additional SLiSe filters obtained with different lb constraints. For the Elliptic filter $\min_{1 \leq i \leq q} |\Im m(w_i)|$ is about 0.0022, while it is 0.001 for the unconstrained SLiSe filter. In the case of the constrained SLiSe filters, we consider “SLiSe ($\text{lb} = 0.0022$)” filter, with a lb corresponding the Elliptic filter. Additionally, we also examine “SLiSe ($\text{lb} = 0.0016$)” generated with a constraint of 0.0016, about half a way between the Elliptic and the unconstrained filter. We calculate the condition number for each of the shifted linear systems that result from filter application on

⁴ The weights are available in [Appendix A](#) under “Box-SLiSe”.

the 2117 problems extracted from the Si2 matrix as illustrated in [Section 5.3](#). [Figure 5.8](#) is a performance profile of the largest condition number for each of the benchmark problems. The Elliptic filter and the SLiSe ($\text{lb}=0.0022$) perform identically in the performance profile. Both filters have a value of one over the entire abscissa range, which indicates that the condition numbers are the same. The figure implies that the largest condition number is influenced solely by the smallest absolute imaginary value of the poles. Accordingly, the constrained filter with lb of 0.0016 performs worse, and the unconstrained SLiSe filter is the worst. This effect can be understood by realizing that the shifted matrices $A - \mathbb{I}w_l$ become nearly singular only if w_l is near an eigenvalue of A .

An increase in the lb constraint has a large influence on the filter. As a result, caution is required when optimizing with box constraints. For example, for a choice of weight function that follow Guideline 1 from [Section 5.2.4](#) in the unconstrained case, the same is not necessarily true when optimizing a filter with box constraints. We discuss an example filter with box constraints in the next section.

5.3 Numerical Experiments

In this section we consider the performance of selected SLiSe filters. First we discuss a replacement for the Gauss filter, as presented in the previous section. An improvement for this filter is particularly usefull, since the Gauss filter and its variations are the default choice for many contour solvers [\[88, 79\]](#). We continue our discussion with a replacement for the Elliptic filter, intended for use on problems where an endpoint of the search interval coincides with, or is near, a large spectral cluster. Such a scenario can occur within parallel solvers when multiple adjacent search intervals are selected. Next, we consider another replacement for the Elliptic filter which performs well in our tests, but violates one of our Guidelines. Finally, we present performance results for a filter that makes use of box constraints to decrease the condition number of the arising linear systems. For the sake of brevity we limit the examples to filters with 16

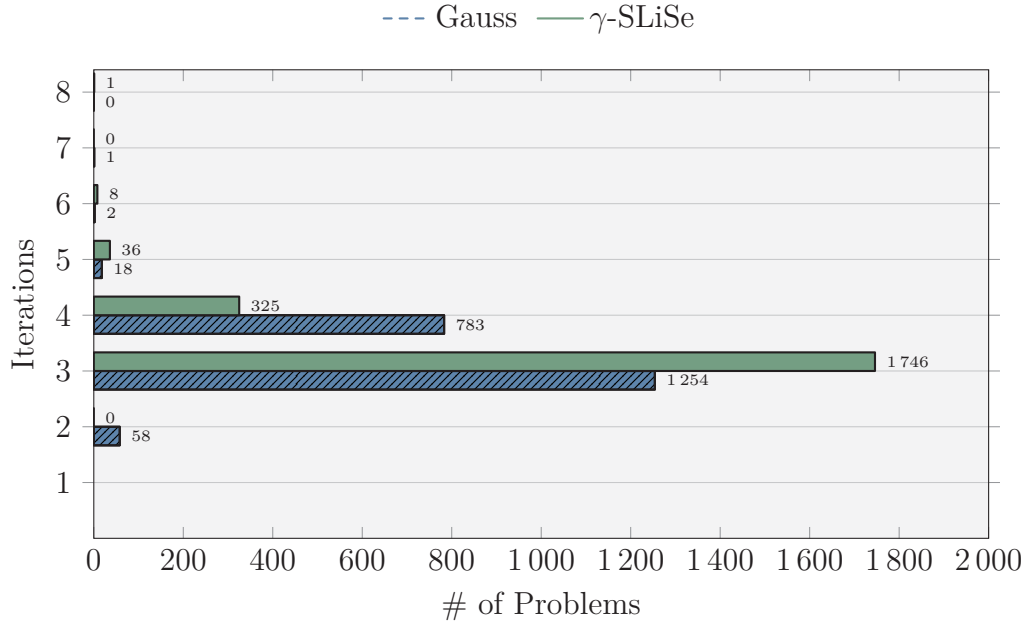


Figure 5.9: Number of FEAST Iterations Saved by γ -SLiSe over the Gauss Filter. The filters are plotted in Figure 5.10.

($p = 1.5m$, Si2 problem set, 16 poles.)

poles and coefficients total. Naturally, the optimization methods presented in this work are applicable to filters with an arbitrary number of poles.

5.3.1 Replacing the Gauss Filter: γ -SLiSe

In order to have a fair comparison between an existing filter and our candidate as filter replacement, we need to establish a testing environment. Since our focus is the use of rational filters within Subspace Iteration methods, we use FEAST as our test environment for comparing filters. A simple filter comparison could use the number of FEAST iterations required to converge the entire subspace for a given benchmark problem; the filter that requires fewer iterations is the better one. While rather basic, we show that this simple criterion already provides useful insights. Then, we introduce a more sophisticated evaluation criterion based on the convergence ratio of the benchmark problem.

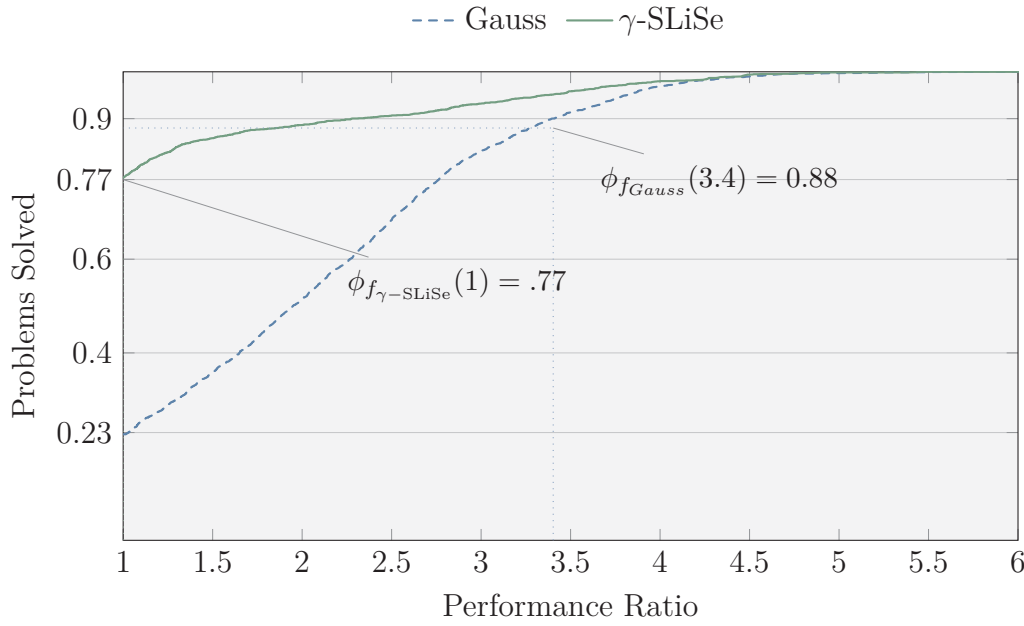


Figure 5.10: Performance Profiles of Convergence Rates for γ -SLiSe and Gauss Filters

($p = 1.5m$, Si2 problem set, 16 poles.)

Once the comparison criterion has been selected, a natural way to decide which filter is superior is to select many, representative benchmark problems and see which filter performs better. Comparing filters on only a few, hand-selected, problems can introduce a strong bias. In [Appendix B.1](#) we propose a method to obtain many benchmark problems from a given matrix which we will use throughout this section. The resulting comparison is based on a large number of benchmark problems and provides a good statistical measure of our filters' quality.

We construct a set of benchmark problems as follows: We fix the matrix M , construct a large number of distinct search intervals (a, b) , compute the exact number of eigenvalues m each interval contains, and then set p to be a multiple of this number. We use the Si2 problem set throughout this section. It was obtained by selecting 2116 search intervals, each containing between 5% and 20% of the spectrum of the Hermitian “Si2” matrix in the University

of Florida matrix collection [24]. The Gauss and γ -SLiSe filter were used as filtering method in FEAST's version 3.0 `screvx` routine, a driver routine that uses a sparse direct solver for the linear system solves⁵. For a problem with $\mathbf{m}$ eigenpairs inside the search interval we select a size of the Subspace Iteration of $\mathbf{p} = 1.5\mathbf{m}$, the value that FEAST recommends for the Gauss filter.

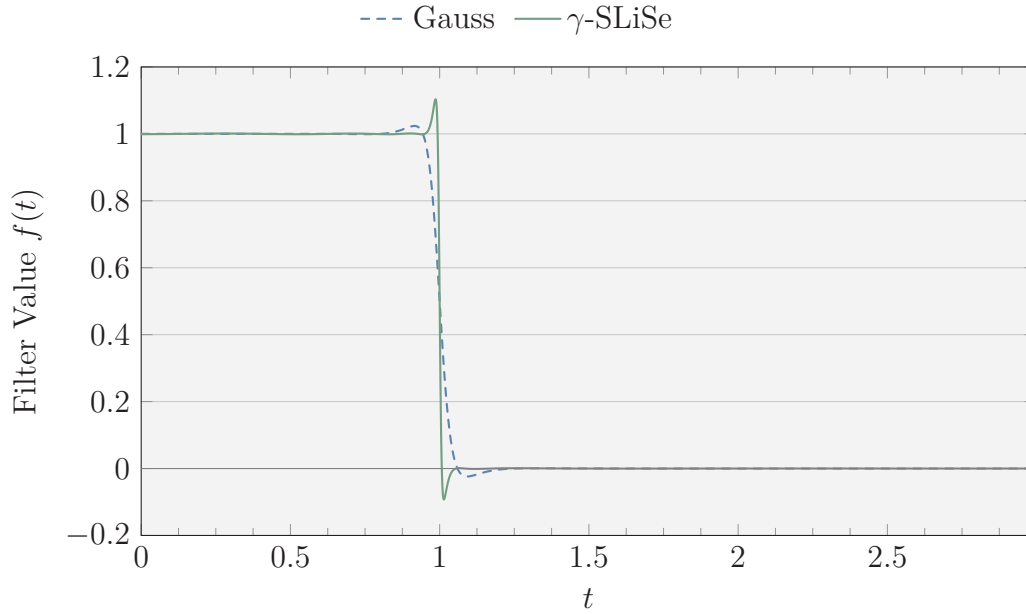
Figure 5.9 shows a histogram of the iterations required for the 2116 benchmark problems by the γ -SLiSe and the Gauss filter. Independently of which filter is used the vast majority of problems of the set requires either 3 or 4 FEAST iterations. Fast convergence of FEAST's Subspace Iteration is a known feature for the Gauss filter when $\mathbf{p}$ is chosen large enough. When FEAST uses the γ -SLiSe filter, most benchmark problems require 3 iterations, with only some problems requiring 4 or more iterations. In contrast, when the Gauss filter is used, a larger number of problems require 4 iterations. Summing the iterations for all of the benchmark problems the γ -SLiSe filter requires 6774 iterations, while the Gauss filter requires 7119 iterations. Since every iteration requires 8 linear system solves and additional overhead, such as the computation of the residuals, saving even a single iteration is a substantial performance improvement.

The increased performance of the γ -SLiSe filter comes with some drawbacks. On the one hand, the Gauss filter is more versatile than the γ -SLiSe. For instance, outside of the selected interval, the Gauss filter decays very quickly to low values (see Figure 5.12), which yields better convergence when increasing $\mathbf{p}$, the number of vectors in the Subspace Iteration procedure, to a larger multiple of $\mathbf{m}$. The γ -SLiSe does not have this property, and so does worse when a very large spectral gap is present. On the other hand, it is always possible to generate a different SLiSe filter that performs well also for large spectral gaps.

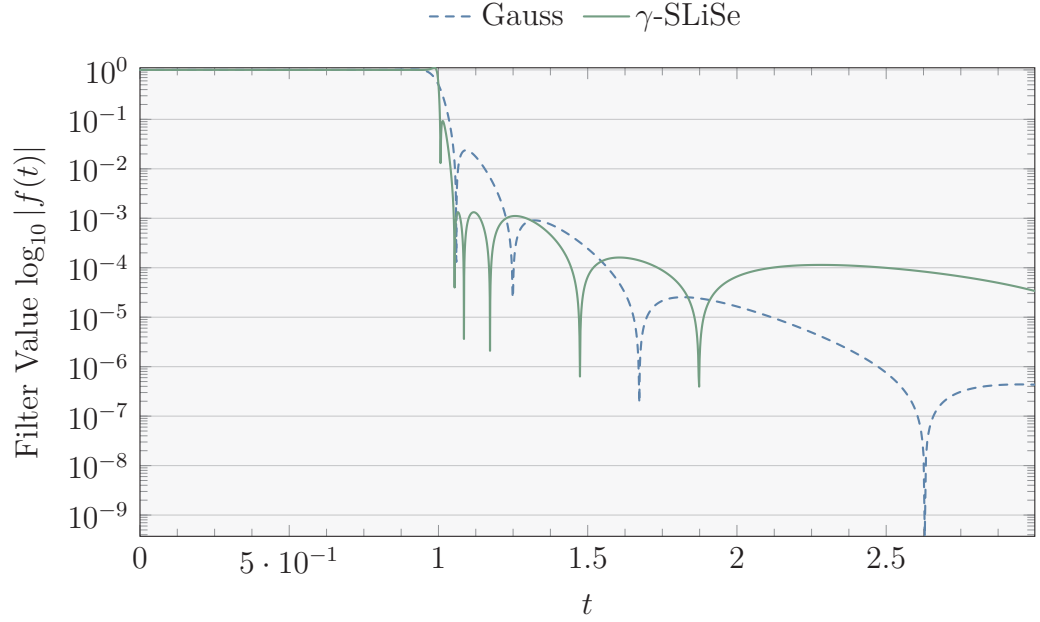
5.3.1.1 Convergence Ratio as a Means of Comparison

While Figure 5.9 shows some promise as a tool for comparing filters, the approach has a number of problems. Subspace Iteration counts provide a very

⁵ Feast was compiled with the Intel Compiler v16.0.2 and executed with a single thread; the target residual was adjusted to 10^{-13}

Figure 5.11: Linear-Scale Plot of the Gauss and γ -SLiSe Filters(Both filters have 16 poles and coefficients, i.e. $q = 4$.)

coarse look at the performance of a filter, underlined by the fact that most benchmark problems require a very similar number of iterations for both filters. Further, the evaluation requires the use of a linear system solver, and thus the outcome is dependent on the algorithm and the parameters used (such as the desired accuracy, the type of solver, etc.). A more reliable comparison should be based on a more fine-grained metric which does not depend on a specific solver, its specific parameters, and the computational kernels used. One such metric is the rate of convergence $\tau(\lambda)$, defined in [Theorem 3](#), exhibited by the filters on each problem of the benchmark set. For small to medium sized problems, we can obtain all eigenvalues of the matrix and calculate the convergence rate analytically. The rate of convergence has a number of advantages over comparisons based on iteration counts. First, it does not depend on a specific solver, and it is cheaper to compute, as $\tau(\lambda)$ is a function of only the eigenvalues and not the entire matrix. Then for testing purposes, we can compute all

Figure 5.12: Semi-log Plot of the Gauss and γ -SLiSe Filters(Both filters have 16 poles and coefficients, i.e. $q = 4$.)

eigenvalues up front, and then compute inexpensively $\tau(\lambda)$ for every benchmark problem.

Figure 5.10 shows the convergence rates for the same 2116 benchmark problems as in Figure 5.9 in a form called *performance profiles* [28]. For a filter f , and a given point x on the abscissa, the corresponding value $\phi_f(x)$ of the graph indicates that for $100 \cdot \phi_f(x)$ percent of the benchmark problems the filter f is at most a factor of x worse than the fastest of all methods in question. For example, $\phi_{f_{\text{Gauss}}} (3.4) \approx 0.88$ indicates that for 88% of the benchmark problems the Gauss filter yields either the best convergence rate or a rate that is within a factor of 3.4 from the best. So the performance profiles not only report how often a filter performs best, but also how badly the filter performs when it is not the best. We will use performance profiles multiples times in this section to compare the filters. From the value of $\phi_{f_{\gamma\text{-SLiSe}}} (1)$ in Figure 5.10 we can infer that γ -SLiSe achieves the best convergence rate for 77% of the benchmark problems, while the Gauss filter does best on the remaining 23% the problems.

Moreover, the Gauss filter achieves a convergence rate that is worse than 3.4 times that of the γ -SLiSe filter for 12% of the problems $\phi_{f_{\text{Gauss}}}(3.4) \approx 0.88$. In conclusion, the SLiSe filter performs better than the Gauss filter in both metrics, iterations counts and convergence rate. However, the performance profile gives a much more detailed comparison.

Figure 5.11 plots both filter functions; only the positive part of the abscissa is shown as the functions are even. Figure 5.12 shows the absolute value of the same filter functions as a logplot. The γ -SLiSe filter performs better than the Gauss filter due to a smaller absolute value between 1.1 and 1.6 than the Gauss filter. A smaller absolute value outside the search interval—specifically for $|f(\lambda_{\mathbf{p}+1})|$ —improves the convergence rate $\tau(\lambda)$. We set $\mathbf{p}$ to the recommended value of 1.5m. A roughly equidistant spacing of the eigenvalues around the canonical search interval $[-1, 1]$ yields $\lambda_{\mathbf{p}+1} \approx \pm 1.25$. The γ -SLiSe filter outperforms the state-of-the-art Gauss filter for $\lambda_{\mathbf{p}+1} \in [1.1, 1.6]$. If either end of the search interval is near a large spectral cluster then $\lambda_{\mathbf{p}+1} \ll 1.1$, and neither of the two filters will do particularly well. In such a case a larger $\mathbf{p}$ is chosen, or Elliptic filters are used. We present the Elliptic filter—and a SLiSe filter meant to replace it—in the next subsection.

5.3.2 Replacing the Elliptic Filter: η -SLiSe

The Elliptic filter, also called Cauey or Zolotarev filter, has been proposed for use in the context of the interior eigenvalue problem in a number of publications [73, 72, 69, 71, 70, 40]⁶. This filter is used specifically when large spectral clusters are at, or near, the endpoints of a search interval. In these cases the Gauss filter exhibits very slow convergence unless $\mathbf{p}$ is chosen much larger than 1.5m. We propose η -SLiSe as a replacement for the Elliptic filter currently used in FEAST.

The Elliptic filter is an optimal filter, in the sense that the filter function is the best uniform rational approximation of the indicator function. Such a

⁶ Technically the Elliptic filter is a class of filters depending on a number of parameters, we consider the specific filter discussed in [40]

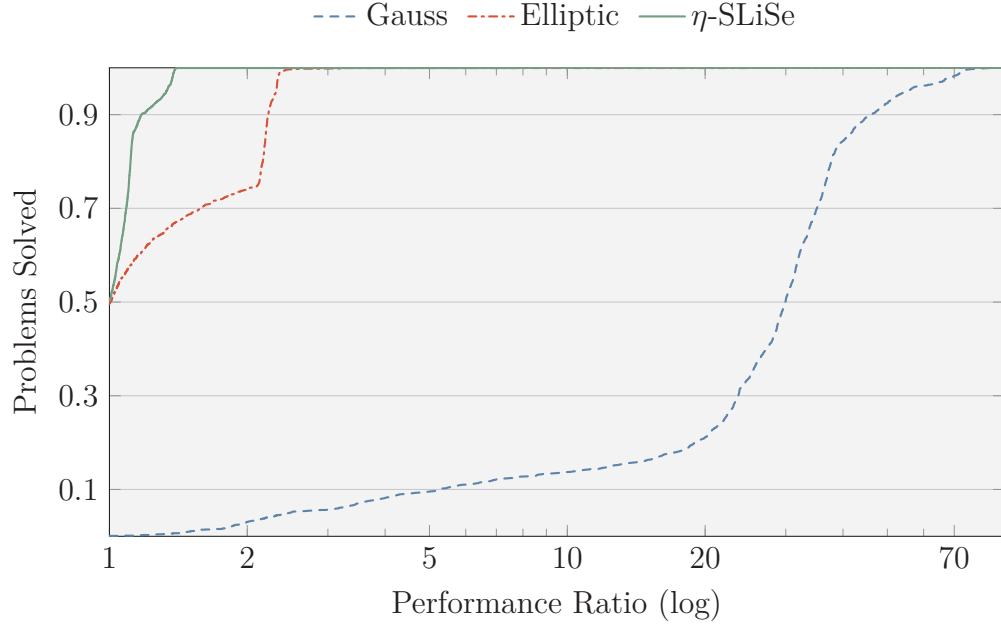


Figure 5.13: Semi-log performance profile of the convergence rate for the Gauss, η -SLiSe, and Elliptic filters. η -SLiSe is meant to replace the Elliptic filter.

(All filters have 16 poles and $\mathbf{p} = \mathbf{m}$.)

filter is worst-case optimal with respect to the convergence rate of [Theorem 3](#), for a subspace size of $\mathbf{p} = \mathbf{m}$. In simple terms, the Elliptic filter works better than the Gauss filter in the presence of spectral clusters close to the interval boundary because it drops from 1 to 0 more quickly. The η -SLiSe filter trades off a slightly larger absolute values inside $[1, 1.01]$ for smaller absolute values in $[1.01, \infty]$ ($[-1.01, 1]$ and $[-\infty, 1.01]$, respectively).

[Figures 5.13](#) and [5.14](#) show a performance profile of the convergence rate for the Gauss, η -SLiSe, and Elliptic filters. [Figure 5.13](#) is a semi-log plot of the performance profile, while [Figure 5.14](#) is a linear-scale plot, zoomed into the interval $[1, 3]$. The comparison between the two filters needs to be adjusted to represent the specific use case of spectral clusters near the endpoints. We use the [Si2](#) benchmark set from the previous comparison, however, instead of target subspace size of $\mathbf{p} = 1.5\mathbf{m}$, we choose $\mathbf{p} = \mathbf{m}$. Thus, the eigenvalue of largest

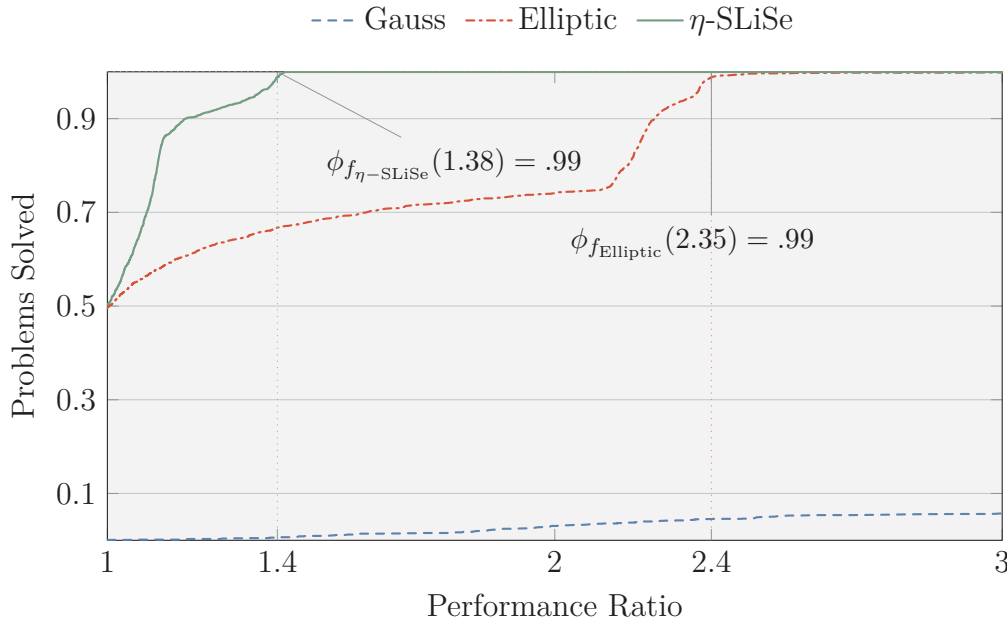


Figure 5.14: The Same Performance Profile as Figure 5.13 Zoomed into $[1, 3]$ with a Linear Axis

(All filters have 16 poles and $\mathbf{p} = \mathbf{m}$.)

absolute function value outside the search interval determines convergence for all benchmark problems. For large spectral clusters $|f(\lambda_{\mathbf{p}+1})|$ is likely to be close to $|f(\lambda_{\mathbf{m}})|$, so this metric serves as an estimator of the filter's behaviour with spectral clusters near the endpoints of the search interval.

The figure indicates that, for our representative set of benchmark problems, the η -SLiSe filter displays better convergence rates than the Elliptic filter. The Elliptic filter achieves the best convergence ratio for half of the benchmark problems, and η -SLiSe does it for the other half. Further, the figure indicates that for 99% of the problems, the Elliptic filter exhibits a convergence ratio within a larger factor from the convergence ratio of η -SLiSe with $\phi_{f_{\text{Elliptic}}}(2.35) = .99$, and $\phi_{f_{\eta\text{-SLiSe}}}(1.38) = .99$ respectively. That is, for 99% of the problems η -SLiSe performs within a factor of 1.38 of the convergence ratio of the Elliptic filter. Simply put, η -SLiSe performs better than the Elliptic filter for half of the problems, and for the other half it performs very similar to the Elliptic filter.

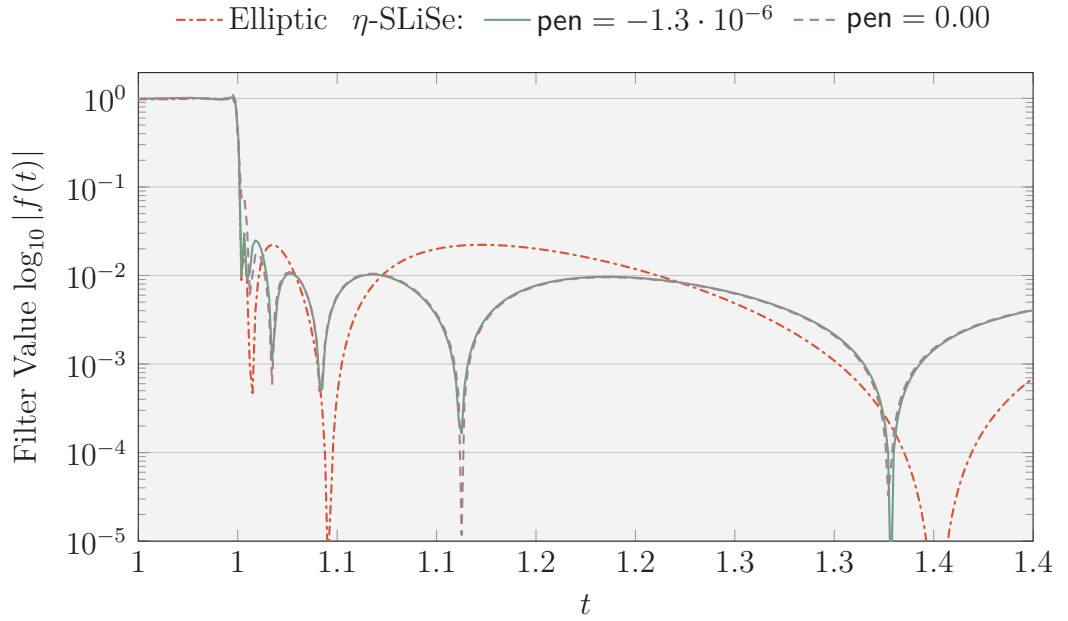


Figure 5.15: Plots of the Elliptic filter, η -SLiSe, and the η -SLiSe Filter without Penalty Term

(All Filters have 16 Poles and Coefficients.)

Conversely, the Elliptic filter exhibits convergence rates for some problems that are twice as bad as the convergence rate of η -SLiSe. The Gauss filter exhibits the worst rates of convergence by far, but this is to be expected as the Gauss filter is not meant for this use-case.

Figures 5.15 and 5.16 show the Elliptic and η -SLiSe filter. Additionally, we include the unconstrained version of η -SLiSe, being the filter obtained with the same weights as η -SLiSe but without penalty term. Figure 5.15 shows an abscissa range of $[0.95, 1.4]$. At this scale both SLiSe filters look identical. Both filters oscillate less than the Elliptic filter, which we would expect from the rate of convergence for η -SLiSe. Figure 5.16 shows the same filters for an abscissa range of $[1, 1.01]$, just outside the search interval. At this scale, the two SLiSe filters are very different. The SLiSe filter without penalty term is steeper and overshoots more than the Elliptic filter, which results in larger absolute function values inside $[1.001, 1.006]$. The penalty parameter for η -SLiSe was

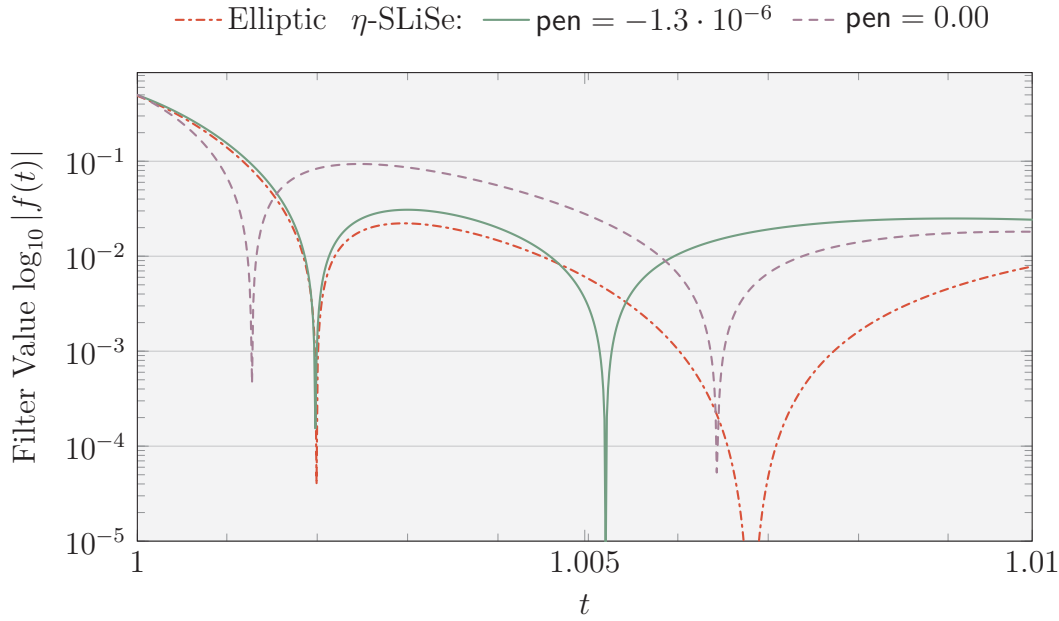


Figure 5.16: Semi-Log Plot: Plots of the Elliptic filter, η -SLiSe, and the η -SLiSe Filter without Penalty Term. Abscissa limited to $[1, 1.01]$.

(All Filters have 16 Poles and Coefficients.)

chosen large enough to make the filter about as steep as the Elliptic filter. As a result, η -SLiSe overshoots less than the unconstrained filter and oscillates at a magnitude only slightly larger than the Elliptic filter. Compared to the Elliptic filter, η -SLiSe trades-off slightly larger (absolute) function values near the end of the search interval for much smaller values farther away.

In this example, the penalty parameter is chosen to be of large absolute value, as compared to the values in [Section 5.2.5.1](#). Such a large penalty parameter results in a significant reduction in the steepness and overshooting of the filter. The difference between the SLiSe filters with and without penalty term is large near the end of the search interval, but negligible in $[1.05, \infty]$. The “design procedure” for such a filter is straight-forward: first, we chose the weights such that the filter has the desired behavior for most of the t axis, e.g., $[0, 1] \cup [1.05, \infty]$ and such that the filter overshoots slightly more than desired.

Then, we used a negative penalty term to lessen the overshooting; this step usually requires only very few iterations and so can be done very quickly.

5.3.3 Breaking Guideline 1: ζ -SLiSe

We specifically advocate 3 guidelines for the choice of weight function in [Section 5.2.4](#). However, when done judiciously breaking these guidelines can be advantageous, as we illustrate in this section. We discuss ζ -SLiSe a filter that violates Guideline 1; this filter also aims to replace the Elliptic filter.

[Figure 5.17](#) shows a performance profile of the Gauss, Elliptic, and ζ -SLiSe filter that compares the rates of convergence for these filters. The setup is the same as in [Section 5.3.2](#): we use the [Si2](#) benchmark set; $\mathbf{p} = \mathbf{m}$; and all filters have 16 poles. The Gauss filter performs similar to the performance profile in [Section 5.2.4](#). ζ -SLiSe filter not only performs better than the Elliptic filter, it appears to perform better than η -SLiSe! η -SLiSe achieves a better rate of convergence than the Elliptic filter for only 50% of the problems, whereas ζ -SLiSe does so for 70% of the problems. The performance profile also encodes a more subtle difference: In [Figure 5.17](#) $f_{\text{Elliptic}}(2.35) \approx 0.92$, which indicates that ζ -SLiSe exhibits rates of convergence better than 2.4 times the rate of the Elliptic filter for 8% of the benchmark problems. This is not the case for η -SLiSe, where $f_{\text{Elliptic}}(2.35) \approx 0.999$.

It appears that ζ -SLiSe would be a better replacement for the Elliptic filter. However, the filter does not conform to Guideline 1. The filter has a single local extremum that has a significantly larger absolute value than both of its neighboring extrema. This extremum is still smaller than the equi-oscillation of the Elliptic filter, which is the reason the why ζ -SLiSe performs better than the Elliptic filter. The 8% of benchmark problems where ζ -SLiSe performs significantly better than the Elliptic filter are exactly those where no eigenvalue coincides with this larger local extremum of ζ -SLiSe. It is possible to obtain filters that violate our guidelines and still yield good convergence rates, as illustrated by ζ -SLiSe. However, adhering to our guidelines vastly simplifies

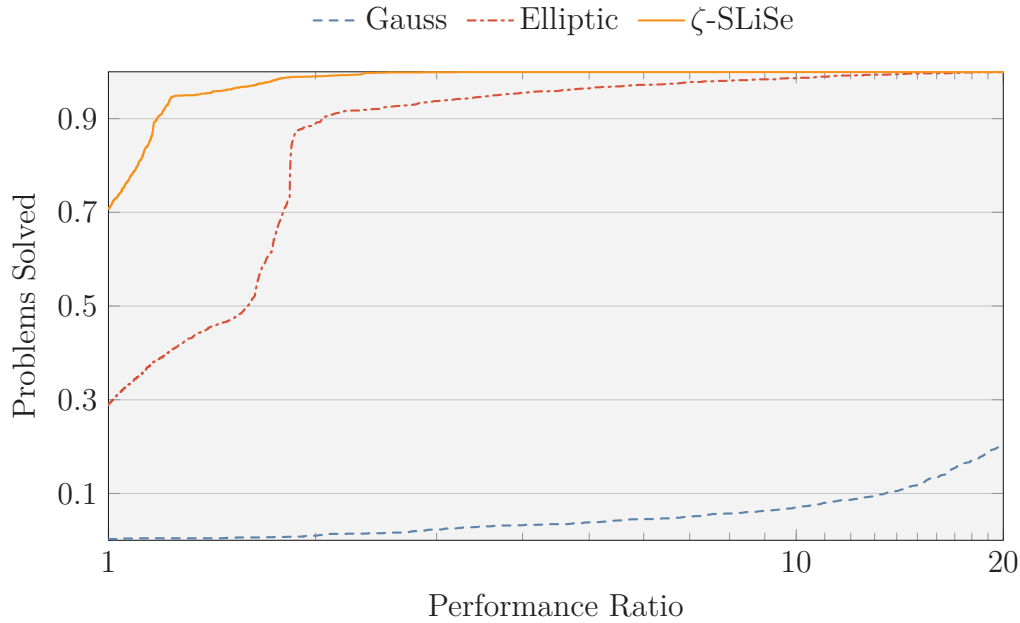


Figure 5.17: Semi-log Performance Profile of the Convergence Rate for the Gauss, ζ -SLiSe, and Elliptic Filters. Zoomed into the Interval $[1, 20]$. Both filters violate Guideline 1.
(All filters have 16 poles and $p = m$.)

the creation of filters. For that reason, we propose η -SLiSe, and not ζ -SLiSe, as a replacement for the Elliptic filter.

5.3.4 Box Constraint: κ -SLiSe

There is another problem with ζ -SLiSe. The filter has a pole with an absolute imaginary value of $\min_{1 \leq i \leq q} |\Im m(w_i)| \approx 0.001$. Analogously to the filters discussed in [Section 5.2.5.2](#) this deteriorates the convergence of Krylov-based linear system solvers. To mitigate this problem we can use box constraints to obtain a filter where each pole has an absolute imaginary value of at least 0.0022, the corresponding value for the Elliptic filter. We discuss a filter called κ -SLiSe, obtained with a box constraint of $lb = 0.0022$ and the same weight function used to obtain ζ -SLiSe. κ -SLiSe violates Guideline 1, just as ζ -SLiSe does.

Figure 5.18 shows the performance profile for κ -SLiSe. The box constraint results in two changes of κ -SLiSe as compared to the ζ -SLiSe. First, the box constraint contribute to lessen the overshooting of the filter. As a result κ -SLiSe attains a better rate of convergence than the Elliptic filter for 86% of the benchmark problems, as compared to the 70% of ζ -SLiSe. Second, κ -SLiSe has larger (as compared to ζ -SLiSe) absolute function values for most of the t axis, which results in rates of convergence that are closer to the rates of the Elliptic filter. The larger absolute function values cause Figure 5.18 to look roughly like a version of Figure 5.17 compressed along the abscissa. From the performance profile we can infer that the convergence rates of κ -SLiSe are only slightly better than those of the Elliptic filter.

Section 5.2.5.2 already illustrates that box constraints can effectively decrease the condition number of the linear systems that arise in the application of the filter. The condition numbers of the linear systems for κ - and ζ -SLiSe are very similar to the ones shown in Figure 5.8. κ -SLiSe illustrates that box constraints can be used to obtain SLiSe filters with good convergence rates. Nevertheless, larger box constraints usually tend to result in worse rates of convergence, as compared to an unconstrained filter. While κ -SLiSe is still marginally better than the Elliptic filter, it yields rates of convergence that are not nearly as good as ζ -SLiSe or η -SLiSe.

5.4 Optimizing SLiSe Weights

In this section we present an outlook on joint work performed with Konrad Kollnig. We present a way to perform hyperparameter optimization on the Least-Squares weights $\mathfrak{G}(t)$, and thus render any manual selection of the weights unnecessary. Our hyperparameter optimization is primarily parametrized by the spectral gap around the search interval $\mathcal{I}$ of the HEP. The spectral gap is a measureable—and often readily available—information about the problem at hand. For sequences of correlated HEPs the spectral gap can be estimated by considering the solution of the previous problem.

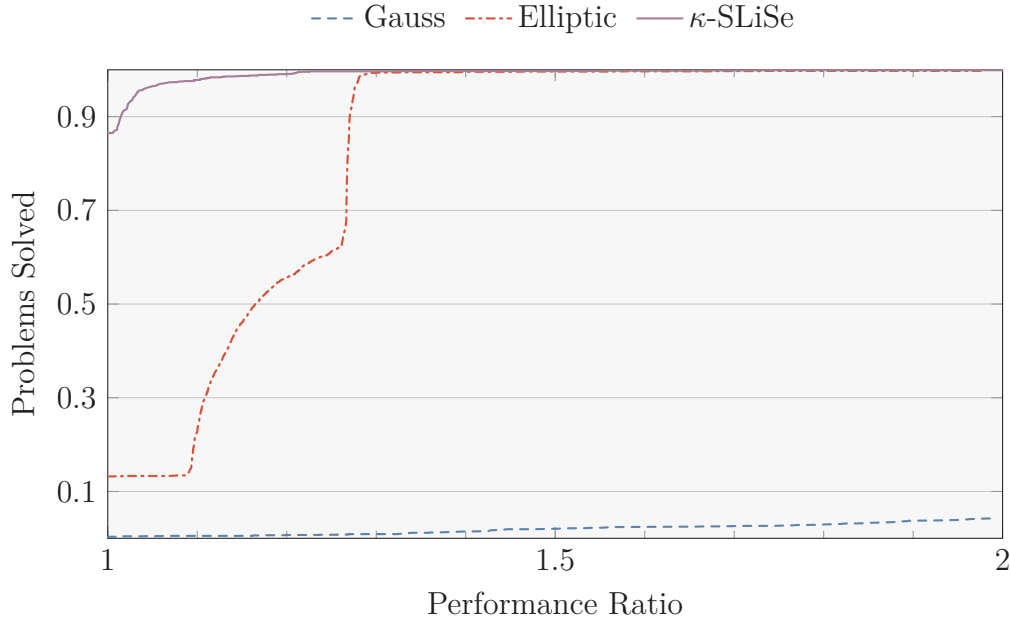


Figure 5.18: [Figure 5.17](#) Zoomed into $[1, 2]$. Both filters violate Guideline 1.
(All filters have 16 poles and $p = m$.)

We start off the rest of this section with the insight that we can use existing, real-valued optimization methods for the generation of SLiSe filters. Then, we introduce our new hyperparameter optimization approach, based on the definition of the worst-case convergence rate of FEAST for some spectral gap G . We close out this chapter with numerical results for the filters generated via our hyperparameter optimization approach.

Real-valued Optimization

In [Section 5.2](#) we derived specialized optimization algorithms based on gradient descent and Levenberg-Marquard; both optimization algorithms treat the objective function as a complex-valued function $\mathcal{F} : \mathbb{C}^q \rightarrow \mathbb{R}$. However, we can express also the optimization problem of [Equation \(5.24\)](#) as a real-valued one. Real-valued optimization problem can be tackled by a plethora of existing, high-quality optimization routines.

For the objective function obtaining a real-valued function $\hat{\mathcal{F}} : \mathbb{R}^{2q} \rightarrow \mathbb{R}$ is

trivial: By splitting the argument vectors $[w, \gamma]$ into real and imaginary parts we obtain

$$\begin{aligned}\hat{\mathcal{F}}([x, y]) &= \mathcal{F}([x + iy]) \\ &\text{with} \\ x &= \Re([w, \gamma]), \\ y &= \Im([w, \gamma]).\end{aligned}$$

The more interesting case concerns the gradient of $\hat{\mathcal{F}}$; we can show [91]:

$$\begin{aligned}\nabla_x \hat{\mathcal{F}}([x, y]) &= \Re(\nabla \mathcal{F}([x + iy])) \\ \nabla_y \hat{\mathcal{F}}([x, y]) &= \Im(\nabla \mathcal{F}([x + iy])).\end{aligned}$$

Equipped with a real-valued objective and gradient functions, we can use many well-known optimization algorithms and packages, for example the **B**royden-**F**letcher-**G**oldfarb-**S**hanno (BFGS) method. BFGS, although ‘only’ a Quasi-Newton method, tends to out-perform the Levenberg-Marquard implementation we present in Equation (5.24). Finally, BFGS also supports constrained optimization, significantly improving the optimization times for filters with box constrained such as κ -SLiSe.

Hyperparameter Optimization

Our optimization of the loss function $\mathcal{F}$ requires a weight function, and the ‘wrong’ choice can result in bad filter functions. In Section 5.2.4 we provide a large amount of detail regarding the weight function $\mathfrak{G}(t)$, including Guidelines on what should be considered when choosing the weights. We can regard $\mathfrak{G}(t)$ as a *hyperparameter*, that is, a parameter of the optimization that is chosen at the beginning of the optimization process and remains fixed throughought. The problem of choosing a good weight function can than be cast as a hyperparameter optimization problem.

Somewhat unconventionally, we choose a different optimization objective

for the hyperparameter optimization. A hyperparameter optimization that exclusively considers the value of the objective function has no provision to generate different filters. We want the ability to generate filters for different use-cases, for example γ -SLiSe as a replacement for the Gauss filter as well as η -SLiSe as a replacement for the Elliptic filter. As such, we want an objective function for the hyperparameter optimization that admits some problem specific information. The distinguishing property in the use-case of the Gauss and Elliptic filter is the number of eigenvalues around the end of the interval $\mathcal{I}$. The critical insight is that the difference between the use-cases between filters is, essentially, the existence (and size) of a spectral gap around the end-points of $\mathcal{I}$. We can leverage our insight by using the following criterion as the objective function for the hyperparameter optimization:

Theorem 4 (FEAST’s worst-case convergence rate[40, Theorem 2.2])

Given a rational filter f and a random initial subspace, FEAST exhibits a convergence rate no larger than

$$\mathfrak{F}_G(f) = \frac{\max_{t \in [-\infty, -G^{-1}] \cup [G^{-1}, \infty]} |f(t)|}{\min_{t \in [-G, G]} |f(t)|}, \quad (5.35)$$

for some gap parameter $G \in (0, 1)$, as long as no eigenvalues lie within $[-G^{-1}, -G] \cup [G, G^{-1}]$. For a given $G \in (0, 1)$, we refer to this ratio as the worst-case convergence rate.

For a fixed G , potentially specific to the problem at hand, we wish to optimize the weight function so that the SLiSe function results in good results. As we cannot feasibly optimize over the entire possible function space for $\mathfrak{G}$. We need to restrict the space of permissible weight functions even further. To this end we define the Least-Squares weight function for hyperparameter optimization

to be of the form:

$$\mathfrak{G}^{u,v}(t) := \begin{cases} u_1, & \text{if } |t| < v_1, \\ u_2, & \text{if } v_1 \leq |t| < v_1^{-1}, \\ u_3, & \text{if } v_1^{-1} \leq |t| < v_2, \\ u_4, & \text{if } v_2 \leq |t| < v_3, \\ u_5, & \text{otherwise,} \end{cases} \quad (5.36)$$

for two vectors $u \in \mathbb{R}^5$ and $v \in \mathbb{R}^3$. This particular parameterization tends to work quite well in practice. However, this definition can be expanded to $u \in \mathbb{R}^s$ and $v \in \mathbb{R}^{s-2}$.

For the hyperparameter optimization we need to assign our original optimization procure ([Equation \(5.24\)](#)) to a function

$$\text{SLiSe}(\mathfrak{G}^{u,v}) := \min_{w, \gamma} \mathcal{F}_{\mathfrak{G}^{u,v}}([w, \gamma]).$$

Now, we can define the hyperparameter optimization as

$$\min_{u \in \mathbb{R}^5, v \in \mathbb{R}^3} \mathfrak{F}_G(\text{SLiSe}(\mathfrak{G}^{u,v})).$$

In practical terms we nest two optimization processes, the original SLiSe optimization inside the hyperparameter optimization outlined above. The inner optimizations optimize the coefficients and poles, given a degree of the filter and the weight function. The hyperparameter (outer) optimization optimizes the weight function, given its parametrization (e.g., [Equation \(5.36\)](#)) and the gap parameter G . The hyperparameter optimization must be tackled with a derivate free optimization method, for example Nelder-Mead.

The above procedure results in an optimized filter function given only: a spectral gap G ; a parametrization of the Least-Squares weight function; and the desired degree of the filter function. This practically eliminates all of the problems from [Section 5.2](#), the hyperparameter optimization effectively ensures that these guidelines are adhered to.

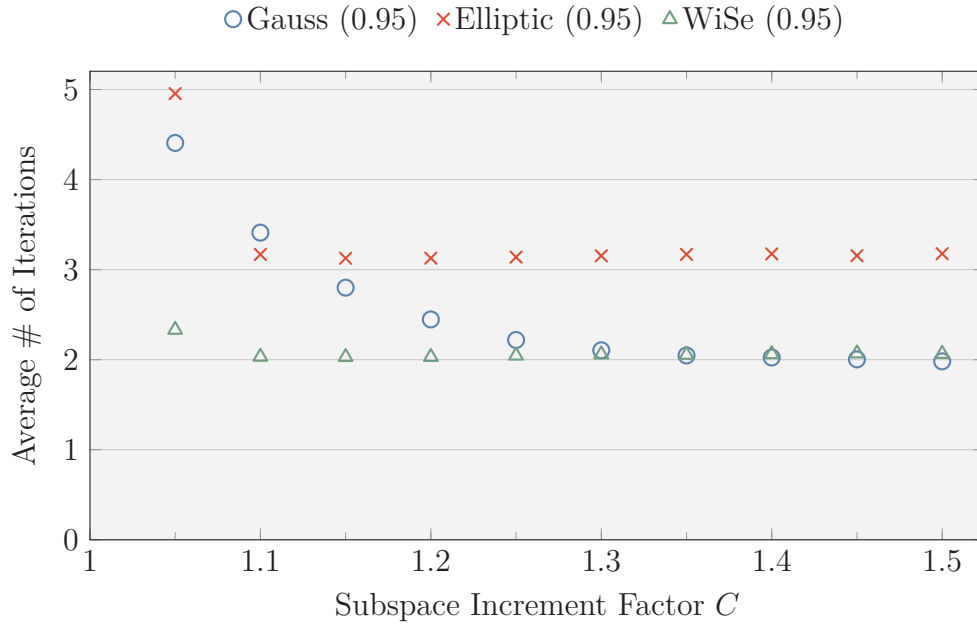


Figure 5.19: Average Numbers of Iterations Required by FEAST to Solve the Entire Si2 Problem Set
($C = \text{pm}^{-1}$; $G = 0.95$; 16 poles.)

We call the resulting filters **W**orst-Case **O**ptimized **L**east-**S**quares, or *WiSe* for short. The WiSe optimization works extremely well and is exceedingly easy to use, due to the small number of inputs and the high probability of success. However as obtaining consistently good results from derivate free optimizers is quite involved, the practical implementation of the optimization is not simple at all, for more details see [1].

Numerical Experiments

We quickly discuss some numerical experiments on WiSe filters, to show-case their potential. Now that we consider the gap G as a variable, it is not entirely fair to compare against FEASTs build-in filters. Güttel et al. [40] generate versions of the Elliptic and Gauss filters for a specific gap parameter. In the following we present results for the case of a moderate spectral gap of $G = 0.95$, including the Gauss and Elliptic filters.

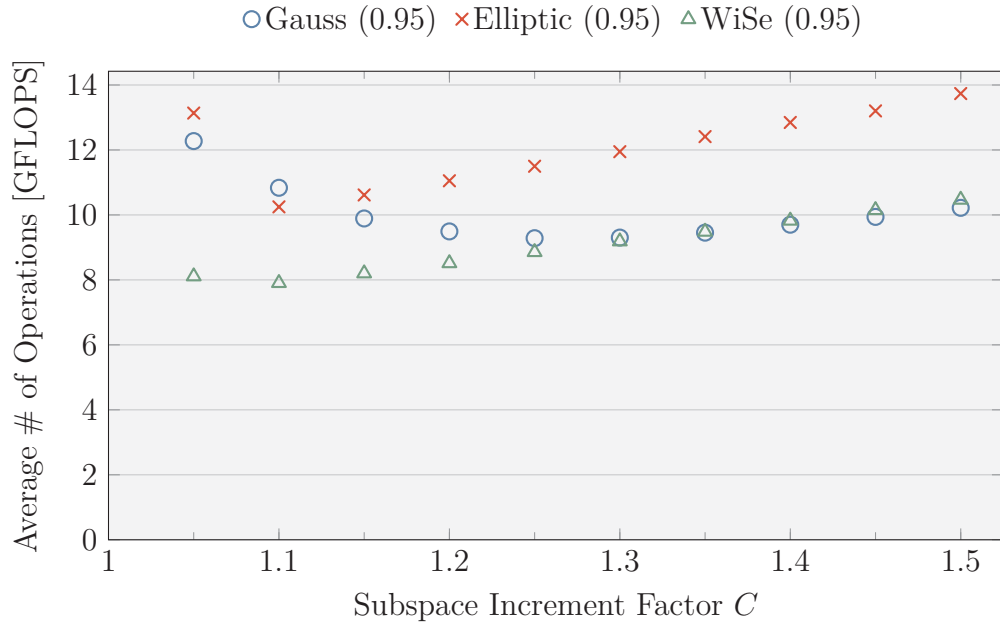


Figure 5.20: The Average Numbers of GFLOPs Required by FEAST to Solve the Entire Si2 Problem Set

($C = \mathbf{p}\mathbf{m}^{-1}$; $G = 0.95$; 16 poles.)

Figure 5.19 shows performance results for the Gauss, Elliptic, and Wise filters with $G = 0.95$. Shown is the average number of FEAST iterations required when solving the 2116 problems in the Si2 as a function of $C = \frac{\mathbf{p}}{\mathbf{m}}$. For the Gauss and Elliptic filters the plots shows an expected result: (1) A small C requires a large number of iterations of the Subspace Iteration procedure. This behavior can be understood through FEAST's convergence behavior. The rate of convergence is influenced by $|f(\lambda_{\mathbf{p}+1})|$, see Theorem 3, and a smaller value for C directly translates into a smaller value for $\mathbf{p}$. (2) The Elliptic filter can hardly make use of a larger input subspace size, i.e. $\mathbf{p} \gg \mathbf{m}$; the required number of iterations is identical for a $C \in [1.1, 1.5]$. This is a well known property of the Elliptic filter, that arises from the equi-oscillation of the filter. The Gauss filter can generally make use of larger C ; however, achieving fewer than a single iteration with random starting vectors is often unrealistic. (3) The WiSe filter performs just as well as the Gauss filter for large subspaces ($C > 1.3$), but performs

significantly better for small subspaces ($C \leq 1.3$). Even with $C = 1.1$, well below the current recommendation for the Gauss filter of $C = 1.5$, the WiSe filter requires, on average, only 2 iterations per problem.

Figure 5.20 shows the same experiment in a different light. Here we plot the number of operations required to solve each problem of the Si2 problem set. Raising C increases the number of linear system solves required for each FEAST iteration. Therefore, the number of iterations required to solve a problem do not capture the entire cost of a solve. Figure 5.20 plots the average number of required operations against C . The plots highlights the superiority of the WiSe filter in the small input subspace ($C \leq 1.3$) regime. Due to the small number of required linear system solves, the WiSe filter significantly out-performs both the Gauss and the Elliptic filter; for $C = 1.05$ the WiSe filter requires, on average, 30% fewer operations to converge a problem in the Si2 problem set.

5.5 Summary

We presented SLiSe, a weighted non-linear Least-Squares optimization framework for rational filters. SLiSe provides a rich framework of solutions that can be employed within FEAST. First, we stabilized the optimization process by explicitly formulating the filters to be CR invariant. A careful selection of the starting position and an appropriate optimization algorithm (e.g., the Levenberg-Marquardt or BFGS methods) lead to solutions with systematically smaller Least-Squares errors as the degree of the filter is increased. We showed numerical experiments that support our claim that SLiSe filters can improve convergence of FEAST for a large set of cases. We establish a number of use-cases for rational filters with different properties that cannot be covered by a single rational filter. Finally, we outlined a hyperparameter optimization approach for the Least-Squares weight function $\mathfrak{G}(t)$, resulting in an even more robust optimization scheme.

Significant effort went into easy usability of the optimization process. Our approach is designed to be “fire-and-forget”, that is both easily parametrized

and fast to run. The ultimate goal is to provide filters that adapt to a given specific HEP, ideally providing the optimal rational filter for every problem. This chapter is meant as a significant step towards this direction.

We can consider the rational filter function as a kind of parameter of FEAST. In that sense, this chapter connects with the idea of tuning numerical kernels for specific applications, as laid out in [Chapter 1](#).

*Verachte nur Vernunft und Wissenschaft,
Des Menschen allerhöchste Kraft,
Laß nur in Blend- und Zauberwerken
Dich von dem Lügengeist bestärken,
So hab ich dich schon unbedingt—
Ihm hat das Schicksal einen Geist gegeben,
Der ungebändigt immer vorwärts dringt,
Und dessen übereiltes Streben
Der Erde Freuden überspringt.*

Johann Wolfgang von Goethe

6

Conclusion

This dissertation set out to develop custom kernels for Materials Science simulation codes. We covered three separate kernels: (1) an adaptive numerical integration routine tailored towards TUF_{RG}, a simulation method for strongly interacting systems; (2) a solver for the solution of sequences of correlated Hermitian eigenproblems arising in Density Functional Theory (DFT) simulations; and (3) an optimization framework for the filters used in so-called “contour solvers”, a variant of eigensolvers that use spectral projection with rational functions, with a special focus on the FEAST solver.

We presented PAID, a high-performance numerical integration routine, that implements our new adaptivity criterion tailored towards the TUF_{RG} simulation method and code. Our adaptivity criterion’s main advantage is a smaller number of function evaluations required to obtain the desired numerical integrals. PAID touts a 5× runtime improvement for the numerical integrals in TUF_{RG} simulations, while remaining exceptionally scalable.

We discussed CHASE, a high-performance solver based on Subspace Iteration with a polynomial accelerator. Subspace Iteration can leverage correlation between eigenvalue problems into significant speedups. CHASE contains a

number of improvements to the state-of-the-art, including an adaptive choice of the filter polynomial and a novel approach to obtain spectral estimates required by the filter. Our kernel outperforms direct solvers on sequences of dense correlated eigenvalue problems.

Finally, we tackled spectral filtering via rational functions for Subspace Iteration methods by presenting SLiSe, an optimization framework for the rational filters based on a non-linear Least-Squares objective function. Our example filters result in significant improvements compared to FEAST’s default filters. More importantly, we made the argument that the rational filter is a parameter of the eigensolver. To this end, we suggested an approach to generate problem-specific filters, based on the size of the spectral gap around the endpoints of the search interval.

Our work is actively used in codes by our collaborations, resulting in significantly better accuracy, scalability, and time-to-solution in these simulation codes. What is more, all of these results are not specific to the codes of our collaborators; each kernel that we presented has potential for reuse in a related field. Our results for TUF_{RG} can be transported to other f_{RG} methods, indeed we are in the process of doing so. CHASE can benefit a number of different electronic structure codes in addition to FLEUR and the Bethe-Salpeter code where it is currently used. For our discussion of SLiSe, we have put a strong focus on FEAST, but the same filters could be used for other “contour solvers”, even those not based on Subspace Iteration.

The great thing about the Levenshtein distance is that it can turn any word into any other word.

Stefan Decker



Rational Filters

The filters presented in [Section 5.3](#) are provided here. For a given filter function

$$f(t, w, \gamma) = \sum_{k=1}^q \left[\frac{\gamma_k}{t - w_k} - \frac{\bar{\gamma}_k}{t + \bar{w}_k} + \frac{\gamma_k}{t - w_k} - \frac{\gamma_k}{t + w_k} \right]$$

we provide only the poles in the “first quadrant” of the complex plane—that is, the poles with positive real and imaginary parts—and the corresponding coefficients γ .

Poles ω	Coefficients γ
$0.999687712591797 + 0.0117367635577924i$	$0.006050012497458 - 0.000227036554136i$
$0.991596517222374 + 0.093208856178882i$	$0.021484299350510 - 0.003666847993474i$
$0.903848148311606 + 0.327740045699974i$	$0.055938387061383 - 0.032384567818443i$
$0.440319857798568 + 0.732970137475905i$	$0.054079510922005 - 0.122837701171251i$

Table A.1: γ -SLiSe, a Filter Meant to Replace the Gauss Filter

A Rational Filters

Poles ω	Coefficients γ
$0.999986323489133 + 0.002453510792541i$	$0.00110213725846833 - 7.98515806042 \cdot 10^{-6}i$
$0.999401600189637 + 0.024159213959740i$	$0.00768847889630669 - 0.000333793441122i$
$0.983469964312691 + 0.160816338804574i$	$0.04592568742949140 - 0.008858565519222i$
$0.628559997051189 + 0.718255201795431i$	$0.11139119375850727 - 0.147357486573937i$

Table A.2: η -SLiSe, a Filter Meant to Replace the Elliptic filter (Obtained with a Penalty Parameter of $c = -1.3 \cdot 10^{-6}$)

Poles ω	Coefficients γ
$0.999517437449349 + 0.0011346403206723i$	$0.000600799688893 - 0.0001380523176106i$
$0.996122208058289 + 0.0169588203859498i$	$0.006148402177611 - 0.0012349951550185i$
$0.971590779276380 + 0.1314326323772290i$	$0.039288214664051 - 0.0095779819369782i$
$0.632009932807876 + 0.6589465004506030i$	$0.111997418650841 - 0.1454789177402232i$

Table A.3: ζ -SLiSe, a Filter Meant to Replace the Elliptic Filter that Violates Guideline 1

Poles ω	Coefficients γ
$0.999997864241235 + 0.002199301304944i$	$0.000920693720425 - 2.62225614322 \cdot 10^{-6}i$
$0.999817083735386 + 0.019255687759249i$	$0.006309876577596 - 0.000106451402529i$
$0.993359173161426 + 0.135785991680408i$	$0.040771709455088 - 0.003927802442013i$
$0.694622923894908 + 0.732441949783473i$	$0.135841041422952 - 0.150283641573960i$

Table A.4: κ -SLiSe, a Filter Meant to Replace the Elliptic Filter with Krylov Solvers that Violates Guideline 1 (Obtained with a Box Constraint of $\text{lb} = 0.0022$)

The Least-Squares Weights $\mathfrak{G}(t)$ for the filters above are given in the tables below.

$ t \in$	$\mathfrak{G}_{\gamma\text{-SLiSe}}(t)$
$[0, .95)$	1
$ [.95, 1.05)$	.01
$[1.05, 1.4)$	10
$[1.4, 5)$	20
$[5, \infty)$	0

Table A.5: Weight Functions used to Generate the γ -SLiSe Filter

$ t \in$	$\mathfrak{G}_{\eta\text{-SLiSe}}(t)$	$\mathfrak{G}_{\text{Box-SLiSe}}(t)$
$[0, .95)$	1	1
$ [.95, .995)$	4	4
$ [.995, 1.005)$	0.5	2
$ [1.005, 1.05)$	4	4
$ [1.05, 1.1)$	0.6	0.6
$ [1.1, 1.3)$	1	1
$ [1.3, 1.8)$	0.3	0.3
$ [1.8, 3)$	0.1	0.1
$ [3, \infty)$	0	0

Table A.6: Weight Functions Used to Generate η -SLiSe and Box-SLiSe (Note the Difference in the Weight Assignment for $[.995, 1.005)$)

Definite Integrals

For the convenience of the reader, we provide the definite integrals required to compute the objective function $F(\gamma, \omega)$ of Equation (5.18), the gradient ∇F from Equation (5.22), and the approximation of the Hessian, given in Equation (5.31). For the sake of generality we denote poles as ω_1 and ω_2 , without conjugation although some formulation require the conjugation or sign change of ω_1 or ω_2 . Recall that the functions $\mathfrak{G}(t)$ and $h(t)$ are piecewise constant and determine the integral boundaries, written as a and b .

The calculation of $\mathcal{W}_{k,\ell}$, $\overline{\mathcal{W}}_{k,\ell}^\pi$, $\mathcal{X}_{k,\ell}$, $\overline{\mathcal{X}}_{k,\ell}^\pi$, $Y_{k,\ell}$, and $\mathcal{Z}_{k,\ell}$ requires the following integral:

$$\int_a^b \frac{dt}{(t - \omega_1)(t - \omega_2)} = \frac{1}{\omega_1 - \omega_2} \left[\ln \frac{(b - \omega_1)}{(a - \omega_1)} + \ln \frac{(a - \omega_2)}{(b - \omega_2)} \right].$$

For the ‘diagonal’ terms $\mathcal{W}_{k,k}$ and $\overline{\mathcal{X}}_{k,k}^\pi$ it holds than $\omega_1 = \omega_2$. In this case we can rewrite the expression without logarithms. This integral is also required for $\nabla \overline{\theta}_k$:

$$\int_a^b \frac{dt}{(t - \omega_1)^2} = \frac{b - a}{(b - \omega_1)(a - \omega_1)}.$$

Note that this is not the case for the $\mathcal{X}$, $\mathcal{Z}$, and $\mathcal{Y}$, where the signs and/or conjugations of ω_1 and ω_2 differ.

For θ_k and $\overline{\theta}_k^\pi$ we require the following integral:

$$\int_a^b \frac{dt}{t - \omega_1} = \ln \frac{(b - \omega_1)}{(a - \omega_1)}.$$

The calculation of $\nabla \overline{W}_{k,\ell}$, $\nabla X_{k,\ell}$, $\nabla \overline{Y}_{k,\ell}$, and $\nabla \mathcal{Z}_{k,\ell}$ requires the integral:

$$\int_a^b \frac{dt}{(t - \omega_1)^2(t - \omega_2)} = \frac{b - a}{(\omega_1 - \omega_2)(b - \omega_1)(a - \omega_1)} - \frac{\ln \frac{(b - \omega_1)}{(a - \omega_1)}}{(\omega_1 - \omega_2)^2} + \frac{\ln \frac{(b - \omega_2)}{(a - \omega_2)}}{(\omega_1 - \omega_2)^2}.$$

The ‘diagonal’ term $\nabla\overline{\mathcal{W}}_{k,k}$ can again be expressed without logarithm as

$$\int_a^b \frac{dt}{(t - \omega_1)^3} = (b - a) \left[\frac{b + a - 2\omega_1}{(b - \omega_1)^2(a - \omega_1)^2} \right].$$

In the approximation of the Hessian the terms $\nabla\overline{\mathcal{W}}_{i,j}$, $\overline{\nabla}\nabla\mathcal{X}_{i,j}$, $\nabla\overline{\nabla}\mathcal{Y}_{i,j}$, and $\overline{\nabla}\nabla\mathcal{Z}_{i,j}$ require yet another integral. This one is given by:

$$\begin{aligned} \int_a^b \frac{dt}{(t - \omega_1)^2(t - \omega_2)^2} &= \frac{a - b}{(\omega_1 - \omega_2)^2} \left[\frac{1}{(a - \omega_1)(-b + \omega_1)} + \frac{1}{(a - \omega_2)(-b + \omega_2)} \right] \\ &+ \frac{2}{(\omega_1 - \omega_2)^3} \left[\ln \frac{a - \omega_1}{b - \omega_1} + \ln \frac{b - \omega_2}{a - \omega_2} \right]. \end{aligned}$$

Here the ‘diagonal’ term is given by:

$$\int_a^b \frac{dt}{(t - \omega_1)^4} = \frac{1}{3} \left[\frac{1}{(a - \omega_1)^3} + \frac{1}{(-b + \omega_1)^3} \right].$$

B

Problem Sets

NaCl			
n	Sequence Length	Size on Disk	nev
3 893	16	232 MiB	256
9 273	16	1.3 GiB	256

Table B.1: All NaCl Matrices are Hermitian and in Double Complex Precision

AuAg			
n	Sequence Length	Size on Disk	nev
5 638	24	486 MiB	972
13 379	25	2.7 GiB	972

Table B.2: All AuAg Matrices are Hermitian and in Double Complex Precision

BSE			
n	Sequence Length	Size on Disk	nev
22 360	1	7.5 GiB	100
32 976	1	17 GiB	100
47 349	1	34 GiB	100
62 681	1	59 GiB	100
76 674	1	88 GiB	100

Table B.3: All BSE Matrices are Hermitian and in Double Complex Precision

Si2				
n	Sequence Length	Size on Disk	nev	
769	1	4.5 MiB	N/A	

Table B.4: The Si2 Matrix is Symmetric and in Double Precision

Si2 contains 2117 intervals containing between 5 and 20% of the spectrum, which corresponds to between 39 and 153 eigenvalues. Providing all 2117 of them explicitly is out of scope, even for this dissertation. We explain the algorithm that generates these intervals in the next section.

B.1 Obtaining Benchmark Problems From a Matrix

In this short appendix we discuss a method to generate a large set of intervals $\mathcal{I}$ from a given eigenproblem corresponding to an Hermitian matrix M . The objective is to obtain a benchmark test set for filtered subspace iteration, as a means to compare filters without having to solve each problem separately.

Interior eigensolvers based on filtered subspace iteration are sensitive to the spectrum of M near the search interval $\mathcal{I}$. Thus, the choice of the search intervals is critical for testing an interior eigensolver. Users often select the search interval motivated by some structure in the spectrum of M , such as the HOMO-LUMO gap in Density Functional Theory methods. Our aim is to create a large set of intervals that represent a variety of real-world use-cases, especially those based on the selection of intervals influenced by the structure of the spectrum. To this end, we propose a method to obtain search intervals by exploiting the distribution of the eigenvalues of M . Instead of choosing the intervals directly, we compute a set of endpoints $E = \{a_1, a_2, \dots\}$, and then form intervals $\mathcal{I}_{i,j} = [a_i, a_j]$ with $a_i, a_j \in E$ and $a_i < a_j$.

These endpoints should represent the neighborhood of an identifiable spectral feature, such as a spectral gap or a cluster; we refer to them as “feature points”.

B.1 Obtaining Benchmark Problems From a Matrix

Given a function $\phi \in C^2$ that approximates the eigenvalue density, we consider feature points to be the real zeros of the first and second derivative of ϕ . A good candidate for ϕ can be obtained, for example, via the Kernel Polynomial Method (KPM) which constructs a polynomial of degree M that approximates the spectral density. The zeros of ϕ' are stationary points; assuming that ϕ is a good approximation of the eigenvalue density, a local maximum may indicate a high density of eigenvalues or even a cluster. Inflection points of ϕ may indicate a change in the increase or decrease of the (approximated) eigenvalue density, signaling a relatively small spectral gap. By choosing a large $M \approx 45$ it is possible to obtain thousands of intervals for a single matrix M . The large problem set can be used to obtain the subspace convergence rates and the condition numbers of the linear system solves without solving each eigenproblem in the benchmark set. Such an approach is limited by the availability of the entire spectrum of M , and thus is only applicable for small to medium sized eigenproblems.

For instance, the subspace convergence rate (see [Theorem 3](#)) depends only on the filter value for the eigenvalues. If the entire spectrum of M is available, we can obtain the convergence rate by computing these values directly. In the same fashion, each filtered interval results in a number of shifted linear system solves of the form $M - \mathbb{I}z_i$ (see [Equation \(5.1\)](#)). When Krylov-based methods are used to solve these systems, the overall performance depends on the condition number. Since M is Hermitian, the shifted matrix is still normal. As a result the singular values of $M - \mathbb{I}z_i$ are given by $\sigma_i = |\lambda_i - z_i|$, where λ_i is the i^{th} eigenvalue of M . By proceeding in this manner for all the poles z_i , we can inexpensively obtain the condition number for all the system solves. Usually, the solve with the highest condition number dictates the overall performance, which is what we used to motivate our analysis of box constraints in [Section 5.2.5.2](#).



Supercomputers

Jureca

JURECA	
Operator	JSC
Construction Date	2015
Theoretical Peak	1.8 (CPU) + 0.44 (GPU) Petaflop per second
Number of Nodes	1872 compute nodes
Interconnect	EDR InfiniBand
Topology	non-blocking fat tree
Default Software Stack	Intel/2018 ParaStationMPI/5.2.1-1 CUDA/9.1.85

COMPUTE NODE		GPU NODE	
CPU Model	Xeon E5-2680 v3	CPU Model	Xeon E5-2680 v3
Architecture	Haswell	Architecture	Haswell
Number of Cores	2× 12 at 2.5 GHz	Number of Cores	2× 12 at 2.5 GHz
RAM per node	128 GiB	RAM per Node	128 GiB
		GPU per Node	2× NVIDIA K80
		Cuda Cores	2× 4992
		GPU Memory	2× 24 GiB

BlueWaters

BLUEWATERS	
Operator	NCSA
Construction Date	2012
Theoretical Peak	1.8 (CPU) + 0.44 (GPU) Petaflop per second
Number of Nodes	22 636 XE nodes, 4 228 XK nodes
Interconnect	Gemini
Topology	3D Torus
Default Software Stack	GCC v6.3.0
	MPICH v7.5
	LibSci v16.11
	CUDA v7.5

XE NODE		XK NODE	
CPU Model	AMD 6276	CPU Model	AMD 6276
Architecture	Interlagos	Architecture	Interlagos
	Bulldozer		Bulldozer
Number of Cores	2× 16 at 2.5 GHz	Number of Cores	1× 16 at 2.5 GHz
RAM per node	64 GiB	RAM per node	32 GiB
		GPU per Node	1× NVIDIA K20X
		Cuda Cores	1× 2688
		GPU Memory	1× 6 GiB

Bibliography

The bibliography is split into three parts: Papers of which I am a (co-)author are listed under [Publications](#) below, other scientific publications are collected in [References](#) on [Page 174](#), and websites and repositories are found under [Online Resources](#) on [Page 182](#).

Publications

- [1] K. A. S. Kollnig, J. Winkelmann, P. Bientinesi, and E. Di Napoli. “Rational Filters with Faster Convergence and Constraints for Contour-based Eigensolvers”. In: *Preparation* (2020).
- [2] J. Winkelmann and E. Di Napoli. “Non-linear Least-Squares Optimization of Rational Filters for the Solution of Interior Hermitian Eigenvalue Problems”. In: *Frontiers in Applied Mathematics and Statistics* 5 (2019). DOI: [10.3389/fams.2019.00005](#).
- [3] J. Winkelmann, P. Springer, and E. Di Napoli. “ChASE: Chebyshev Accelerated Subspace Iteration Eigensolver for Sequences of Hermitian Eigenvalue Problems”. In: *ACM Trans. Math. Softw.* 45.2 (Apr. 2019), 21:1–21:34. DOI: [10.1145/3313828](#).
- [4] J. Lichtenstein, J. Winkelmann, D. Sanchez de la Pena, T. Vidovic, and E. Di Napoli. “Parallel Adaptive Integration in High-Performance functional Renormalization Group Computations”. In: *Jülich Aachen Research Alliance High-Performance Computing Symposium 2016*. Lecture Notes in Computer Science. Springer-Verlag, 2017. DOI: [10.1007/978-3-319-53862-4_15](#).

References

- [5] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et al. *LAPACK Users' Guide*. Third Edition. SIAM, Jan. 1999. DOI: [10.1137/1.9780898719604](https://doi.org/10.1137/1.9780898719604).
- [6] A. P. Austin and L. N. Trefethen. “Computing Eigenvalues of Real Symmetric Matrices with Rational Filters in Real Arithmetic”. In: *SIAM J. Sci. Comput.* 37.3 (2015), A1365–A1387.
- [7] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. “Anasazi Software for the Numerical Solution of Large-scale Eigenvalue Problems”. In: *ACM Transactions on Mathematical Software* 36.3 (July 2009), 13:1–13:23. DOI: [10.1145/1527286.1527287](https://doi.org/10.1145/1527286.1527287).
- [8] A. S. Banerjee, L. Lin, W. Hu, C. Yang, and J. E. Pask. “Chebyshev polynomial filtered subspace iteration in the discontinuous Galerkin method for large-scale electronic structure calculations”. In: *The Journal of Chemical Physics* 145.15 (Oct. 2016), page 154101. DOI: [10.1063/1.4964861](https://doi.org/10.1063/1.4964861).
- [9] A. S. Banerjee, L. Lin, W. Hu, C. Yang, and J. E. Pask. “Chebyshev polynomial filtered subspace iteration in the discontinuous Galerkin method for large-scale electronic structure calculations”. In: *The Journal of Chemical Physics* 145.15 (Oct. 2016), page 154101. DOI: [10.1063/1.4964861](https://doi.org/10.1063/1.4964861).
- [10] M. V. Barel. “Designing Rational Filter Functions for Solving Eigenvalue Problems by Contour Integration”. In: *Linear Algebra Appl.* 502 (2016), pages 346–365. ISSN: 0024-3795.
- [11] F. L. Bauer. “Das Verfahren der Treppeniteration und verwandte Verfahren zur Lösung algebraischer Eigenwertprobleme”. In: *Zeitschrift für Angewandte Mathematik und Physik ZAMP* 8.3 (May 1957), pages 214–235. DOI: [10.1007/BF01600502](https://doi.org/10.1007/BF01600502).
- [12] C. Bekas, E. Kokiopoulou, and Y. Saad. “Computation of Large Invariant Subspaces Using Polynomial Filtered Lanczos Iterations with Applications in Density Functional Theory”. In: *SIAM Journal on Matrix Analysis and Applications* 30.1 (Jan. 2008), pages 397–418. DOI: [10.1137/060675435](https://doi.org/10.1137/060675435).
- [13] M. Berljafa and S. Güttel. “Parallelization of the Rational Arnoldi Algorithm”. In: *SIAM Journal on Scientific Computing* 39.5 (2017), S197–S221. DOI: [10.1137/16M1079178](https://doi.org/10.1137/16M1079178).
- [14] M. Berljafa, D. Wortmann, and E. Di Napoli. “An optimized and scalable eigensolver for sequences of eigenvalue problems”. In: *Concurrency and Computation: Practice and Experience* 27 (Sept. 2015), pages 905–922. DOI: [10.1002/cpe.3394](https://doi.org/10.1002/cpe.3394).

- [15] J. Berntsen, T. O. Espelid, and A. Genz. “Algorithm 698: DCUHRE: An Adaptive Multidimensional Integration Routine for a Vector of Integrals”. In: *ACM Trans. Math. Softw.* 17.4 (Dec. 1991), pages 452–456. DOI: [10.1145/210232.210234](https://doi.org/10.1145/210232.210234).
- [16] W.-J. Beyn. “An Integral Method for Solving Nonlinear Eigenvalue Problems”. In: *Linear Algebra and Its Applications* 436.10 (2012), pages 3839–3863.
- [17] L. S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users’ Guide*. SIAM, 1997. DOI: [10.1137/1.9780898719642](https://doi.org/10.1137/1.9780898719642).
- [18] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN: 0521833787.
- [19] M. Bull. “Parallel Algorithms for Globally Adaptive Quadrature”. Dissertation. University of Manchester, Oct. 1988, page 187.
- [20] C. W. Clenshaw and A. R. Curtis. “A method for numerical integration on an automatic computer”. In: *Numerische Mathematik* 2.1 (Dec. 1960), pages 197–205. DOI: [10.1007/bf01386223](https://doi.org/10.1007/bf01386223).
- [21] M. Clint and A. Jennings. “The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration”. In: *The Computer Journal* 13.1 (1970), pages 76–80. DOI: [10.1093/comjnl/13.1.76](https://doi.org/10.1093/comjnl/13.1.76).
- [22] R. Cools. “Advances in multidimensional integration”. In: *Journal of Computational and Applied Mathematics* 149.1 (2002), pages 1–12.
- [23] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Elsevier, 1984. DOI: [10.1016/c2013-0-10566-1](https://doi.org/10.1016/c2013-0-10566-1).
- [24] T. A. Davis and Y. Hu. “The University of Florida Sparse Matrix Collection”. In: *ACM Trans. Math. Softw.* 38.1 (Dec. 2011), 1:1–1:25. ISSN: 0098-3500.
- [25] E. Di Napoli. “Chebyshev acceleration of subspace iteration revisited: convergence and optimal polynomial degree”. In preparation. 2018.
- [26] E. Di Napoli and M. Berljafa. “Block iterative eigensolvers for sequences of correlated eigenvalue problems”. In: *Computer Physics Communications* 184.11 (Nov. 2013), pages 2478–2488. DOI: [10.1016/j.cpc.2013.06.017](https://doi.org/10.1016/j.cpc.2013.06.017).
- [27] E. Di Napoli, E. Polizzi, and Y. Saad. “Efficient Estimation of Eigenvalue Counts in an Interval”. In: *Numer. Linear Algebra Appl.* 23.4 (2016), pages 674–692. ISSN: 1099-1506.
- [28] E. D. Dolan and J. J. Moré. “Benchmarking Optimization Software With Performance Profiles”. In: *Mathematical Programming* 91.2 (2002), pages 201–213.

- [29] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. “A set of level 3 basic linear algebra subprograms”. In: *ACM Transactions on Mathematical Software* 16.1 (Mar. 1990), pages 1–17. DOI: [10.1145/77626.79170](https://doi.org/10.1145/77626.79170).
- [30] J. Dongarra, V. Eijkhout, and A. Kalhan. *Reverse Communication Interface for Linear Algebra Templates for Iterative Methods*. Technical report. University of Tennessee, 1995.
- [31] H.-r. Fang and Y. Saad. “A Filtered Lanczos Procedure for Extreme and Interior Eigenvalue Problems”. In: *SIAM Journal on Scientific Computing* 34.4 (Jan. 2012), A2220–A2246. DOI: [10.1137/110836535](https://doi.org/10.1137/110836535).
- [32] K. V. Fernando and B. N. Parlett. “Accurate singular values and differential qd algorithms”. In: *Numerische Mathematik* 67.2 (Mar. 1994), pages 191–229. DOI: [10.1007/s002110050024](https://doi.org/10.1007/s002110050024).
- [33] M. Galgon, L. Krämer, and B. Lang. “Improving projection-based eigensolvers via adaptive techniques”. In: *Numerical Linear Algebra with Applications* 25.1 (2018). e2124 nla.2124, e2124. DOI: [10.1002/nla.2124](https://doi.org/10.1002/nla.2124).
- [34] 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*. Springer International Publishing, 2017, pages 63–79. DOI: [10.1007/978-3-319-62426-6_5](https://doi.org/10.1007/978-3-319-62426-6_5).
- [35] C. F. Gauss. *Methodus nova integralium valores per approximationem inveniendi*. 1815, pages 29–76.
- [36] B. Gavin and E. Polizzi. “An Improved Krylov Eigenvalue Strategy Using the FEAST Algorithm With Inexact System Solves”. In: *ArXiv e-prints* (June 2017). arXiv: [1706.00692](https://arxiv.org/abs/1706.00692).
- [37] G. H. Golub and H. Zha. “The canonical correlations of matrix pairs and their numerical computation”. In: *Linear algebra for signal processing*. Springer, 1995, pages 27–49. DOI: [10.1007/978-1-4612-4228-4_3](https://doi.org/10.1007/978-1-4612-4228-4_3).
- [38] P. Gonnet. “A review of error estimation in adaptive quadrature”. In: *ACM Computing Surveys* 44.4 (Aug. 2012), pages 1–36. DOI: [10.1145/2333112.2333117](https://doi.org/10.1145/2333112.2333117).
- [39] S. Güttel. “Rational Krylov Approximation of Matrix Functions: Numerical Methods and Optimal Pole Selection”. In: *GAMM Mitteilungen* 36.1 (2013), pages 8–31.
- [40] S. Güttel, E. Polizzi, P. T. P. Tang, and G. Viaud. “Zolotarev Quadrature Rules and Load Balancing for the FEAST Eigensolver”. In: *SIAM J. Sci. Comput.* 37.4 (Jan. 2015), A2100–A2122. ISSN: 1095-7197.

- [41] S. Güttel, R. Van Beeumen, K. Meerbergen, and W. Michiels. “NLEIGS: A Class of Robust Fully Rational Krylov Methods for Nonlinear Eigenvalue Problems”. In: *SIAM J. Sci. Comput.* 36.6 (2014), A2842–A2864.
- [42] N. Hale. “On The Use Of Conformal Maps To Speed Up Numerical Computations”. PhD thesis. University of Oxford, 2009.
- [43] N. Hale, N. J. Higham, and L. N. Trefethen. “Computing A^α , $\log(A)$, and Related Matrix Functions By Contour Integrals”. In: *SIAM J. Numer. Anal.* 46.5 (2008), pages 2505–2523. DOI: [10.1137/070700607](https://doi.org/10.1137/070700607).
- [44] N. J. Higham. *Functions of Matrices*. Society for Industrial and Applied Mathematics, Jan. 2008. DOI: [10.1137/1.9780898717778](https://doi.org/10.1137/1.9780898717778).
- [45] P. Hohenberg and W. Kohn. “Inhomogeneous Electron Gas”. In: *Physical Review* 136 (3B Nov. 1964), B864–B871. DOI: [10.1103/PhysRev.136.B864](https://doi.org/10.1103/PhysRev.136.B864).
- [46] H. Hotelling. “Relations Between Two Sets of Variates”. In: *Biometrika* 28.3/4 (Dec. 1936), pages 321–377. DOI: [10.1007/978-1-4612-4380-9_14](https://doi.org/10.1007/978-1-4612-4380-9_14).
- [47] T. Ikegami and T. Sakurai. “Contour Integral Eigensolver for non-Hermitian Systems: A Rayleigh-Ritz-type Approach”. In: *Taiwanese J. Math.* 14.3A (2010), pp–825.
- [48] T. Ikegami, T. Sakurai, and U. Nagashima. “A Filter Diagonalization for Generalized Eigenvalue Problems Based on the Skaurai-Sugiuara Projection Method”. In: *J. Comput. Appl. Math.* 233.8 (2010), pages 1927–1936. ISSN: 0377-0427.
- [49] A. Imakura, L. Du, and T. Sakurai. “Relationships Among Contour Integral-Based Methods for Solving Generalized Eigenvalue Problems”. In: *Japan J. Indust. Appl. Math* 33.3 (2016), pages 721–750. ISSN: 1868-937X.
- [50] A. Imakura and T. Sakurai. “Block Krylov-type Complex Moment-based Eigensolvers for Solving Generalized Eigenvalue Problems”. In: *Numerical Algorithms* (2016), pages 1–21. ISSN: 1572-9265.
- [51] T. Imamura, S. Yamada, and M. Machida. “Development of a high performance eigensolver on the petascale next generation supercomputer system”. In: *Progress in Nuclear Science and Technology* 2 (2011), pages 643–650. DOI: [10.15669/pnst.2.643](https://doi.org/10.15669/pnst.2.643).
- [52] E. Isaacson. “Numerical Recipes in C: The Art of Scientific Computing”. In: *SIAM Review* 31.1 (Mar. 1989), pages 142–142. DOI: [10.1137/1031025](https://doi.org/10.1137/1031025).
- [53] H. J. F. Jansen and A. J. Freeman. “Total-Energy Full-Potential Linearized Augmented-Plane-Wave Method for Bulk Solids - Electronic and Structural Properties of Tungsten”. In: *Physical Review B* 30.2 (July 1984), pages 561–569. DOI: [10.1103/PhysRevB.30.561](https://doi.org/10.1103/PhysRevB.30.561).

- [54] V. Kalantzis, J. Kestyn, E. Polizzi, and Y. Saad. “Domain Decomposition Approaches for Accelerating Contour Integration Eigenvalue Solvers for Symmetric Eigenvalue Problems”. Preprint submitted to SISC.
- [55] T. Kato. *Perturbation Theory for Linear Operators*. Springer, 1995. DOI: [10.1007/978-3-642-66282-9](https://doi.org/10.1007/978-3-642-66282-9).
- [56] W. Kohn. “Nobel Lecture: Electronic structure of matter-wave functions and density functionals”. In: *Reviews of Modern Physics* 71.5 (Oct. 1999), pages 1253–1266. DOI: [10.1103/RevModPhys.71.1253](https://doi.org/10.1103/RevModPhys.71.1253).
- [57] A. R. Krommer and C. W. Ueberhuber. *Numerical integration on advanced computer systems*. Volume 848. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1994.
- [58] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users’ Guide*. Volume 6. SIAM, 1998. DOI: [10.1137/1.9780898719628](https://doi.org/10.1137/1.9780898719628).
- [59] K. Levenberg. “A Method for the Solution of Certain Non-Linear Problems in Least Squares”. In: *Q. Appl. Math.* 2.2 (1944), pages 164–168.
- [60] A. Levitt and M. Torrent. “Parallel eigensolvers in plane-wave Density Functional Theory”. In: *Computer Physics Communications* 187 (Feb. 2015), pages 98–105. DOI: [10.1016/j.cpc.2014.10.015](https://doi.org/10.1016/j.cpc.2014.10.015).
- [61] R. Li, Y. Xi, L. Erlandson, and Y. Saad. “The Eigenvalues Slicing Library (EVSL): Algorithms, Implementation, and Software”. In: (Feb. 14, 2018). arXiv: [1802.05215 \[math\]](https://arxiv.org/abs/1802.05215).
- [62] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad. “A Thick-Restart Lanczos Algorithm with Polynomial Filtering for Hermitian Eigenvalue Problems”. In: *SIAM Journal on Scientific Computing* 38.4 (Jan. 2016), A2512–A2534. DOI: [10.1137/15m1054493](https://doi.org/10.1137/15m1054493).
- [63] J. Lichtenstein, D. S. de la Peña, D. Rohe, E. D. Napoli, C. Honerkamp, and S. Maier. “High-performance functional Renormalization Group calculations for interacting fermions”. In: *Computer Physics Communications* 213 (2017), pages 100–110. DOI: <https://doi.org/10.1016/j.cpc.2016.12.013>.
- [64] J. Lichtenstein. “Functional renormalization group studies on competing orders in the square lattice”. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2018. Dissertation. RWTH Aachen University, 2018, page 104. DOI: [10.18154/RWTH-2018-225781](https://doi.org/10.18154/RWTH-2018-225781).
- [65] L. Lin, Y. Saad, and C. Yang. “Approximating spectral densities of large matrices”. In: *SIAM Review* 58.1 (2016), pages 34–65. DOI: [10.1137/130934283](https://doi.org/10.1137/130934283).

- [66] K. Madsen, H. B. Nielsen, and O. Tingleff. *Methods for Non-Linear Least Squares Problems*. Technical report. Technical University of Denmark, 2004.
- [67] 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”. In: *Journal of Physics: Condensed Matter* 26.21 (2014), page 213201. DOI: [10.1088/0953-8984/26/21/213201](https://doi.org/10.1088/0953-8984/26/21/213201).
- [68] D. W. Marquardt. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. In: *J. Soc. Ind. Appl. Math.* 11.2 (1963), pages 431–441.
- [69] H. Murakami. *An Experiment of the Filter Diagonalization Method for the Banded Generalized Symmetric-Definite Eigenproblem*. Technical report 59(2007-HPC-110). Tokyo Metropolitan University, 2007.
- [70] H. Murakami. “A Filter Diagonalization Method by the Linear Combination of Resolvents”. In: *IPJSJ Trans. ACS* 49.SIG2(ACS21) (2008), pages 66–87.
- [71] H. Murakami. *The Filter Diagonalization Method for the Unsymmetric Matrix Eigenproblem*. Technical report 43(2008-HPC-115). Tokyo Metropolitan University, 2008.
- [72] H. Murakami. *Experiments of Filter Diagonalization Method for Real Symmetric Definite Generalized Eigenproblems by the use of Elliptic Filters*. Technical report 2010-HPC-125. Tokyo Metropolitan University, June 2010.
- [73] H. Murakami. *Optimization of Bandpass Filters for Eigensolver*. Technical report 2010-HPC-124. Tokyo Metropolitan University, 2010.
- [74] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Jan. 1998. DOI: [10.1137/1.9781611971163](https://doi.org/10.1137/1.9781611971163).
- [75] M. Petschow, E. Peise, and P. Bientinesi. “High-Performance Solvers for Dense Hermitian Eigenproblems”. In: *SIAM Journal on Scientific Computing* 35.1 (2013), pages C1–C22. DOI: [10.1137/110848803](https://doi.org/10.1137/110848803).
- [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”. In: *Journal of Computational Physics* 325 (Nov. 2016), pages 226–243. DOI: [10.1016/j.jcp.2016.08.027](https://doi.org/10.1016/j.jcp.2016.08.027).
- [77] R. Piessens, E. de Doncker-Kapenga, C. W. Überhuber, and D. K. Kahaner. *Quadpack*. Springer Berlin Heidelberg, 1983. DOI: [10.1007/978-3-642-61786-7](https://doi.org/10.1007/978-3-642-61786-7).
- [78] S.-D. Poisson. “Sur le calcul numérique des intégrales définies”. In: *Mémoires de l’Académie Royale des Sciences de l’Institut de France* 4 (1827), pages 571–602.

- [79] E. Polizzi. “Density-matrix-based Algorithm for Solving Eigenvalue Problems”. In: *Physical Review B* 79 (Mar. 2009), page 115112. DOI: [10.1103/PhysRevB.79.115112](https://doi.org/10.1103/PhysRevB.79.115112).
- [80] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. “Elemental: A New Framework for Distributed Memory Dense Matrix Computations”. In: *ACM Transactions on Mathematical Software* 39.2 (Feb. 2013), 13:1–13:24. DOI: [10.1145/2427023.2427030](https://doi.org/10.1145/2427023.2427030).
- [81] J. R. Rice. “A Metalgorithm for Adaptive Quadrature”. In: *Journal of the ACM* 22.1 (Jan. 1975), pages 61–82. DOI: [10.1145/321864.321870](https://doi.org/10.1145/321864.321870).
- [82] E. H. Rogers. “A Minimax Theory for Overdamped Systems”. In: *Archive for Rational Mechanics and Analysis* 16.2 (1964), pages 89–96. DOI: [10.1007/bf00281333](https://doi.org/10.1007/bf00281333).
- [83] A. Ruhe. “Algorithms for the Nonlinear Eigenvalue Problem”. In: *SIAM Journal on Numerical Analysis* 10.4 (Sept. 1973), pages 674–689. DOI: [10.1137/0710059](https://doi.org/10.1137/0710059).
- [84] A. Ruhe. “Rational Krylov Sequence Methods for Eigenvalue Computation”. In: *Linear Algebra and its Applications* 58 (1984), pages 391–405.
- [85] H. Rutishauser. “Computational aspects of FL Bauer’s simultaneous iteration method”. In: *Numerische Mathematik* 13.1 (1969), pages 4–13. DOI: [10.1007/BF02165269](https://doi.org/10.1007/BF02165269).
- [86] H. Rutishauser. “Simultaneous iteration method for symmetric matrices”. In: *Numerische Mathematik* 16.3 (1970), pages 205–223. DOI: [10.1007/bf02219773](https://doi.org/10.1007/bf02219773).
- [87] Y. Saad. *Numerical methods for large eigenvalue problems*. SIAM, 2011. DOI: [10.1137/1.9781611970739](https://doi.org/10.1137/1.9781611970739).
- [88] Y. Saad, R. Li, Y. Xi, E. Vecharinsky, and C. Yang. *EigenValues Slicing Library*. <http://www-users.cs.umn.edu/~saad/software/EVSL/>. Accessed: 2016-11-05.
- [89] T. Sakurai and H. Sugiura. “A Projection Method for Generalized Eigenvalue Problems using Numerical Integration”. In: *J. Comput. Appl. Math.* 159.1 (2003), pages 119–128.
- [90] P. Salas, L. Giraud, Y. Saad, and S. Moreau. “Spectral recycling strategies for the solution of nonlinear eigenproblems in thermoacoustics”. In: *Numerical Linear Algebra with Applications* 22.6 (2015), pages 1039–1058. DOI: [10.1002/nla.1995](https://doi.org/10.1002/nla.1995).
- [91] L. Sorber, M. V. Barel, and L. D. Lathauwer. “Unconstrained Optimization of Real Functions in Complex Variables”. In: *SIAM Journal on Optimization* 22.3 (Jan. 2012), pages 879–898. DOI: [10.1137/110832124](https://doi.org/10.1137/110832124).

- [92] A. Stathopoulos. *Locking issues for finding a large number of eigenvectors of Hermitian matrices*. Technical report. Tech Report: WM-CS-2005-09. Department of Computer Science, College of William and Mary, 2006.
- [93] G. W. Stewart. “Accelerating the orthogonal iteration for the eigenvectors of a Hermitian matrix”. In: *Numerische Mathematik* 13.4 (Aug. 1969), pages 362–376. DOI: [10.1007/BF02165413](https://doi.org/10.1007/BF02165413).
- [94] G. W. Stewart. “Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices”. In: *Numerische Mathematik* 25.1 (1976), pages 123–136. DOI: [10.1007/BF01462265](https://doi.org/10.1007/BF01462265).
- [95] G. W. Stewart. *Matrix algorithms. Vol. II*. SIAM, 2001. DOI: [10.1137/1.9780898718058](https://doi.org/10.1137/1.9780898718058).
- [96] P. T. P. Tang and E. Polizzi. “FEAST As A Subspace Iteration Eigensolver Accelerated by Approximate Spectral Projection”. In: *SIAM. J. Matrix Anal. & Appl.* 35.2 (2014), pages 354–390.
- [97] L. N. Trefethen. “Is Gauss Quadrature Better than Clenshaw–Curtis?” In: *SIAM Review* 50.1 (Jan. 2008), pages 67–87.
- [98] L. N. Trefethen and J. A. C. Weideman. “The Exponentially Convergent Trapezoidal Rule”. In: *SIAM Review* 56.3 (Jan. 2014), pages 385–458. DOI: [10.1137/130932132](https://doi.org/10.1137/130932132).
- [99] C. Vömel, R. G. Veprek, U. Weber, and P. Arbenz. *Iterative solution of generalized eigenvalue problems from optoelectronics with trilos*. Technical report. Computer Science Department, ETH Zürich, 2012, page 15. DOI: <https://doi.org/10.3929/ethz-a-006821768>.
- [100] H. Voss and B. Werner. “A minimax principle for nonlinear eigenvalue problems with applications to nonoverdamped systems”. In: *Mathematical Methods in the Applied Sciences* 4 (1982), pages 415–424. DOI: [10.1002/mma.1670040126](https://doi.org/10.1002/mma.1670040126).
- [101] E. Wimmer, H. Krakauer, M. Weinert, and A. J. Freeman. “Full-potential self-consistent linearized-augmented-plane-wave method for calculating the electronic-structure of molecules and surfaces: O₂ Molecule”. In: *Physical Review B* 24.2 (July 1981), pages 864–875. DOI: [10.1103/PhysRevB.24.864](https://doi.org/10.1103/PhysRevB.24.864).
- [102] Y. Xi and Y. Saad. “Computing Partial Spectra with Least-Squares Rational Filters”. In: *SIAM J. Sci. Comput.* 38.5 (2016), A3020–A3045.
- [103] Y. Zhou, J. R. Chelikowsky, and Y. Saad. “Chebyshev-filtered subspace iteration method free of sparse diagonalization for solving the Kohn–Sham equation”. In: *Journal of Computational Physics* 274 (Oct. 2014), pages 770–782.

- [104] Y. Zhou and R.-C. Li. “Bounding the spectrum of large Hermitian matrices”. In: *Linear Algebra and its Applications* 435.3 (Aug. 2011), pages 480–493.
- [105] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. “Self-consistent-field calculations using Chebyshev-filtered subspace iteration”. In: *Journal of Computational Physics* 219.1 (Nov. 2006), pages 172–184. DOI: [10.1016/j.jcp.2006.03.017](https://doi.org/10.1016/j.jcp.2006.03.017).

Online Resources

All websites and repositories were accessible as of February 19, 2021.

- [106] S. Blügel, G. Bihlmayer, and D. Wortmann. *The FLEUR code*. URL: <http://www.flapw.de>.
- [107] Intel. *Intel Math Kernel Library*. URL: <https://software.intel.com/en-us/mkl>.
- [108] S. G. Johnson. *Cubature*. URL: <https://github.com/stevengj/cubature>.
- [109] E. Polizzi. *The FEAST Eigenvalue Solver*. URL: <http://feast-solver.org/>.
- [110] E. Polizzi. *The NESSIE Code*. URL: <http://www.ecs.umass.edu/~polizzi/software.htm>.