Domain decomposition algorithms for the solution of sparse symmetric generalized eigenvalue problems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Vasileios Kalantzis

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Yousef Saad

September, 2018

Acknowledgments

I am indebted to my father for living, but to my teacher for living well.

Alexander the Great, *King of the Ancient Greek (Hellenic) Kingdom of Macedon*

First, I would like to thank my doctoral supervisor, Professor Yousef Saad, for all of his guidance and support over the last five years. Professor Saad has been always willing to share his immense amount of knowledge with me and working with him has been a great privilege. I consider myself lucky to have him as my doctoral supervisor.

Second, I would like to thank the members of my thesis examination committee: Professor Daniel Boley, Professor George Karypis, and Professor John Sartori, for their kindness to serve on my committee and the time they devoted while on it.

Thinking back to how it all started, it is appropriate to thank Professor Efstratios Gallopoulos, the thesis supervisor of my M. Eng and M. Sc theses at University of Patras, for initiating and keep "watering" my interest in numerical methods and parallel computing. The course of my life would have been much different had I not walked into his "Scientific Computing II" class that Spring semester of 2010. Moreover, I would like to thank Eugenia Kontopoulou, Dr. Georgios Kollias, Professor Andreas Stathopoulos and Professor Petros Drineas for their constant support and advices over the past few years. Especially, Eugenia and Georgios has been close to me since the very beginning of my graduate life.

While carrying my graduate studies at University of Minnesota, I was fortunate enough to meet/collaborate with many great people. First, I would like to acknowledge the help and support of the people I was lucky to share the lab space with: Ruipeng Li, Thanh Ngo, Shashanka Ubaru, Yuanzhe Xi, Geoffrey Dillon, Zachary Bookey, Peter Solfest, and Agnieszka Miedlar. Second, my collaborators: Jared L. Aurentz, Anthony P. Austin, Michele Benzi, Yuanzhe Xi, James Kestyn, Eric Polizzi, Athanasio Nikolakopoulo, Ruipeng Li, and John Garofalaki.

My studies on this side of the Atlantic would not have been possible without the support of the National Science Foundation and the Department of Energy; and, by extension, the people of the United States of America. I owe them all a great debt of gratitude. Similarly, I owe the outmost to my native country, Greece, and its people, for educating me since the first day of my life.

My friends from back home should also get part of the credits of my accomplishments: Christos, Nikos, Nikos, Giorgos, Ilias, Ilias, Sotiris, Thanasis, Giannis, Thomas, Giannis, Giorgos, Giorgos, Panos, and Kostas. An even bigger part of the aforementioned credits should go to my family (my parents Giorgos and Eleni, and my siblings Andreas and Katerina) for being equally excited as I about my happiness and success. I am lucky to have them in my life.

Dedication

To Giorgos, Eleni, Andreas and Katerina.

Abstract

This dissertation focuses on the design, implementation, and evaluation of **domain decomposition** techniques for the solution of large and sparse algebraic symmetric generalized eigenvalue problems. Domain decomposition techniques begin by partitioning the global (discretized) domain into a number of subdomains. The solution of the algebraic eigenvalue problem is then decoupled into two separate subproblems: (1) one defined locally in the interior of each subdomain, and (2) one defined on the interface region connecting neighboring subdomains. As soon as the part of the solution associated with the interface region is computed, the part of the solution associated with the interior variables in each subdomain can be computed locally and independently of the rest of the subdomains.

The domain decomposition techniques proposed in this dissertation can be classified into two categories: (1) **filtering techniques**, and (2) **root-finding techniques**.

Filtering techniques are projection methods applied to a transformation of the original matrix pencil chosen so that, ideally, the eigenvalues of interest are mapped to the only nonzero eigenvalues in the transformed pencil. This dissertation combines domain decomposition with filtering approaches and demonstrates how this class of hybrid algorithms can outperform current state-of-the-art filtering techniques. Apart being well-suited for execution on distributed memory systems, an immediate advantage of such hybrid techniques is that any orthogonalization necessary needs to be performed only to vectors whose length is equal to the number of interface variables. Moreover, we show that if the filter is applied only to that portion of the pencil associated with the interface variables, it is possible to even achieve convergence within a number of steps that is smaller than the number of eigenvalues located inside the interval of interest. In contrast, any projection method applied to a transformation of the original matrix pencil must perform a number of steps that is at least equal to the number of eigenvalues located inside the interval of interest. Implementation aspects of the proposed numerical schemes on many-core/multi-core computer systems and experiments on serial/distributed computing environments are also discussed.

$Root-finding$ *techniques* convert the interface eigenvalue problem into one of computing roots of scalar functions, i.e., the eigenvalues of the original eigenvalue problem are roots of a carefully chosen scalar function. This allows the use of existing fast iterative root-finding schemes, e.g. Newton's method, for the solution of symmetric generalized eigenvalue problems. Root-finding techniques can be especially useful when only a few eigenvalues and associated eigenvectors are sought. The numerical schemes proposed in this part of the present dissertation are compared against other single-vector eigenvalue solvers such as the Rayleigh Quotient Iteration and Residual Inverse Iteration. Numerous theoretical details and practical considerations are considered. Implementation aspects of the proposed numerical schemes on multi-core computer systems and experiments on serial/distributed computing environments are also presented.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The numerical solution of large and sparse algebraic eigenvalue problems is a problem of major interest in many scientific and engineering applications. One such application is structural mechanics [1]. The frequency response function of an engineering structure with respect to a force vector f is often modeled as the solution of the linear system $A - \omega^2 M = f$, where A and M denote the stiffness and mass matrix, respectively, and $\omega \in [\omega_l, \omega_u]$ represents a frequency value within the range of interest [2, 3]. As the aforementioned linear system might need be solved for many different frequency values, e.g. hundreds or thousands, the standard approach of solving each linear system on a one-by-one basis can become computationally challenging; especially for modern Finite Element pencils the size of which runs in the order of millions. A popular approach to reduce the computational costs associated with the standard approach is the *modal superposition method*. Therein, the true frequency response of the structure is approximated by that of the projection of the equation $A - \omega^2 M = f$ on a basis formed by the eigenvectors associated with those eigenvalues of the matrix pencil (A, M) located within a certain real interval $[\alpha, \beta]$. The main computational procedure then is the solution of the associated generalized eigenvalue problem. Another important application of eigenvalue problems is in Density Functional Theory (DFT), a modeling technique used to calculate the electronic structure and ground-state properties of molecules [4]. In DFT, the solution of the all-electron Schrödinger equation is replaced by a one-electron Schrödinger equation with an effective potential which leads to a nonlinear eigenvalue problem known as the Kohn-Sham equation [5, 6]. The most widely used technique for solving the Kohn-sham equations is the self-consistent field iteration at each iteration of which a partial spectral decomposition of a symmetric matrix pencil must be computed. In particular, a typical electronic structure calculation with many atoms requires the calculation of (at least) the n_{occ} smallest eigenvalues and associated eigenvectors, where n_{occ} denotes the number of occupied states (for most systems of interest this is half the number of valence electrons). Thus, it is possible to see eigenvalue problems in the size of millions where tens of thousands of eigenvalues are needed. Other applications of large scale eigenvalue problems can be found in applications areas such as control theory, electrical networks and combustion processes [7], normal mode analysis in molecular dynamics simulations [8], recommender systems [9, 10], and Principal Component Analysis in the study of population genetics [11].

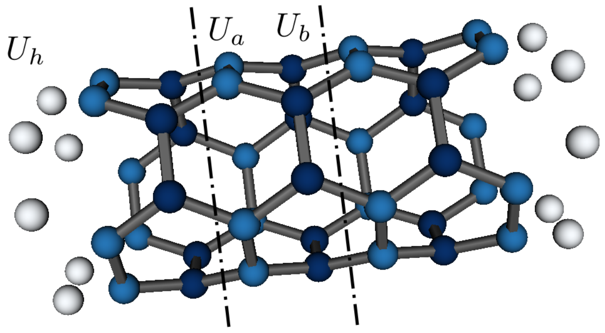


Figure 1.1: 3-dimensional plot of a 3 unit cell Carbon NanoTube (CNT-3). The molecule is composed of 3 $|U_aU_b|$ units cells, includes an additional U_a ring, and is terminated with hydrogen atoms U_h at each end.

1.1 The symmetric generalized eigenvalue problem

This dissertation focuses on the design, development, and implementation of numerical methods for the solution of large and sparse algebraic symmetric generalized eigenvalue problems. More specifically, our goal is to compute scalar-vector pairs (λ, x), $x \neq 0$, that satisfy the following matrix equation

$$\begin{aligned} Ax &= \lambda Mx, \\ (A - \lambda M)x &= 0. \end{aligned} \tag{1.1}$$

The matrices $A \in \mathbb{R}^{n\times n}$ and $M \in \mathbb{R}^{n\times n}$ are assumed large, sparse and symmetric, while matrix M is also positive-definite[1] (SPD). Matrices A and M are most often the result of a discretization scheme applied to a continuous problem of interest defined on a certain domain (either by Finite Difference, Finite Element, Finite Volume, or Spectral Element methods). When $M = I$, i.e., M is the identity matrix, the eigenvalue problem in (1.1) is of the *standard* form. While the numerical methods developed in this dissertation are concerned with the solution of the generalized eigenvalue problem, some of our experiments will be focusing on the solution of standard eigenvalue problems. Throughout this dissertation we will use the notation "pencil (A, M)" to refer to the eigenvalue problem in (1.1).

1.1.1 Notation

The matrix pencil (A, M) has n eigenvalue-eigenvector pairs (eigenpairs) and we will denote its ith eigenpair as $\left(\lambda_i, x^{(i)}\right),\ i = 1, \dots, n$. The eigenvalues of the matrix pencil (A, M) are all real. The eigenvectors $x^{(i)}$ can be chosen to be either real or complex. Throughout this dissertation we restrict our attention to the real vector space. As most engineering applications typically require only the computation of those eigenvalues (and, maybe, associated eigenvectors) which are located within a prescribed real interval, throughout this dissertation we will focus on computing the $nev \geq 1$ eigenpairs $\left(\lambda_i, x^{(i)}\right)$ for which $\lambda_i \in [\alpha, \beta]$. The exact value of nev is assumed[2] unknown, and the scalars $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$ that define the real interval $[\alpha, \beta]$ are either user-given or dictated by the application itself.

When only the interval $[\alpha, \beta]$ is given, we will assume that the eigenvalues of the matrix pencil (A, M) are indexed such that eigenvalues $\lambda_i,\ i = 1, \dots, nev$ (ordered by algebraic value) are located inside $[\alpha, \beta]$ while eigenvalues $\lambda_i,\ i = nev + 1, \dots, n$ are outside. When the interval of interest is given in the form $[\alpha, \beta] := [\lambda_k, \lambda_j]$ it should be inferred that we seek to compute the nev eigenvalues $\lambda_k, \lambda_{k+1}, \dots, \lambda_j$ (and, optionally, associated eigenvectors) where the subscripts $k, \dots, j$ indicate the order of the algebraic value of the sought eigenvalues with respect to all n eigenvalues of the matrix pencil (A, M), i.e., $\lambda_1 \leq \ldots \leq \lambda_k \leq \ldots \leq \lambda_j \leq \ldots \leq \lambda_n$.

While in this dissertation we compute each eigenpair (or invariant subspace) up to an abstract level of accuracy, we note that the level of accuracy up to which each approximate eigenpair

[1]This is equivalent to saying that all eigenvalues of matrix M are positive

[2]With the exception of the numerical technique presented in Chapter 4, the rest of the numerical techniques presented in this dissertation do not require the explicit knowledge (or approximation) of the value of nev

needs be computed in most engineering applications is typically not greater than the truncation error permitted during the discretization of the continuous problem.

Definition 1.1.1 *For any symmetric matrix pencil (L, C) with C a SPD matrix, we will denote*

$$\Lambda(L, C) := \{\lambda \mid \det(L - \lambda C) = 0\}.$$

Definition 1.1.2 *A subspace $\mathcal{Z}$ will be called invariant subspace if $M^{-1}A\mathcal{Z} \subseteq \mathcal{Z}$.*

1.2 The framework of domain decomposition

The numerical techniques developed in this dissertation are by and large based on the concept of domain decomposition [12, 13, 14]. Algebraic domain decomposition approaches start by splitting the computational domain into a set of $p \in \mathbb{Z}^+$ smaller subdomains with the help of a graph partitioner. Throughout this dissertation we consider non-overlapping subdomains whose union equals the original domain. The main idea then is to solve for the restriction of the original problem in the interior of each subdomain independently and in parallel of the rest of the subdomains. The part of the solution associated with the interface region is computed by solving a coarse problem in a process in which all all subdomains cooperate. Some of the main advantages of domain decomposition approaches include (1) inherent concurrency and ability to execute efficiently on distributed memory environments, (2) applicability to various PDEs and flexibility to handle problems of heterogeneous nature (e.g. different boundary conditions), (3) natural approach to implement hybrid numerical methods to balance cost/accuracy.

1.2.1 Graph partitioning and the local viewpoint of the eigenvalue problem

Figure 1.2 shows a common way of partitioning the graph of a 5×5 Finite Difference discretization of a Laplacian operator on a 2D square domain. On the left side is an edge-separator in which a vertex is an equation-unknown pair and the vertex set is subdivided into p partitions, i.e., p non-overlapping subsets whose union is equal to the original vertex set. On the right side is a vertex-separator in which each subdomain is formed by sets of edges. *Throughout this dissertation we will consider edge-separators only.*

A good partitioning is one in which each subdomain contains roughly n/p vertices, while, at the same time, either the number of edges connecting neighboring subdomains (for edge-separators) or the cardinality of the vertex separator set (for vertex-separators) is small. Computing an optimal partitioning is known to be NP-hard and in practice we have to rely on approximation algorithms. As was mentioned above, domain decomposition techniques are naturally suited for parallel computing. The value of p is then typically related to the number of cores or processors of the available computer system. As a general rule of thumb each subdomain is mapped to a single core or processor and is handled by a separate process. This means that a large value of p must be used in order to take advantage of the compute capabilities of large cluster computers with (tens of) thousands of processors. On the other hand, increasing the value of p also increases the total number of interface variables across the subdomains, i.e., vertices that are located on the boundaries of the subdomains, thus leading to higher inter-subdomain communication.

The focus of this dissertation lies on the development of novel numerical schemes and as such we will mostly consider the effects of the value of p with respect to the attainable accuracy and convergence of the proposed schemes. Nonetheless, some of the schemes proposed in this thesis are implemented so that they execute on distributed memory environment systems, and details on the parallel efficiency of the numerical scheme studied for different values of p will be given.

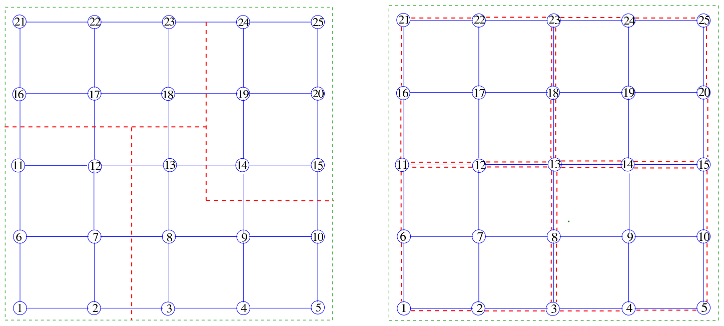


Figure 1.2: Two classical ways of partitioning a graph associated with a 5×5 discretization of a 2D Dirichlet eigenvalue problem discretized by Finite Differences. In this example the 2D discretized domain is partitioned into $p = 4$ subdomains. Left: edge-separator (vertex-based partitioning). Right: vertex-separator (edge-based) partitioning.

The eigenvalue problem Once a partitioning of the (discretized) domain is computed, three types of unknowns appear: (1) interior unknowns that are coupled only with local equations, (2) local interface unknowns that are coupled with both non-local (external) and local equations, and (3) external interface unknowns that belong to other subdomains and are coupled with local interface variables. Figure 1.3 sketches the local viewpoint of each subdomain after partitioning.

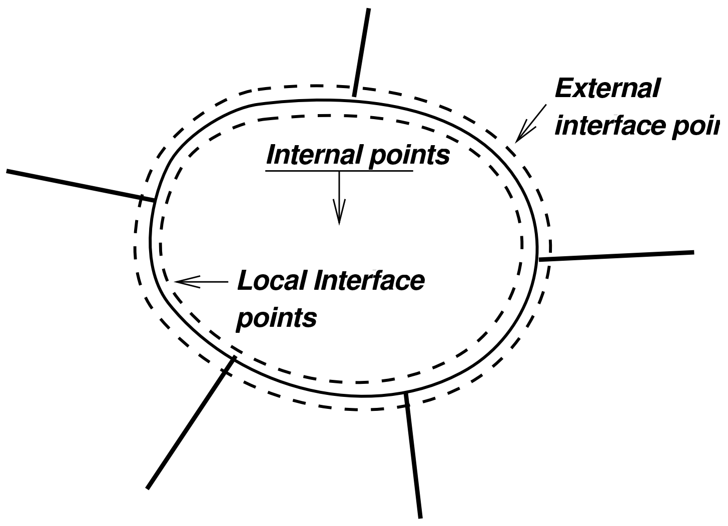


Figure 1.3: The local viewpoint of each subdomain after partitioning by an edge-separator.

The solution of the eigenvalue problem $Ax = \lambda Mx$ is then divided into the solution of the two following subproblems:

1. compute the part of the solution associated with the interface variables across all subdomains, and

2. compute the part of the solution associated with the interior variabels of each subdomain.

The order of the above two steps is not coincidental as the computation of the part of the solution associated with the interface variables is typically a prerequisite to compute the part of the solution associated with interior variables. Moreover, since the interior variables in each subdomain are not coupled with any external interface variables from other subdomains, the part of the solution associated with the interior variables in each subdomain can be computed independently, and in parallel with the rest of the subdomains. On the other hand, the solution of the interface part of the eigenvalue problem is *non-local* in the sense that exchange of information among the subdomains is required.

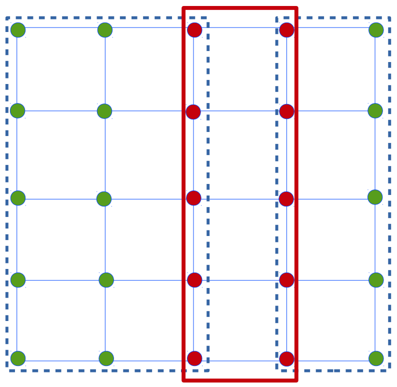


Figure 1.4: A 5×5 mesh partitioned into two subdomains. Interface variables are colored in red while interior variables are colored in green. Once the part of the solution associated with the interface variables is determined (solid line), the part of the solution associated with the interior variables is computed in parallel in each subdomain (dashed lines).

1.3 Domain decomposition eigenvalue solvers

While domain decomposition techniques have been studied extensively for the solution of sparse linear systems, e.g. see [12, 13] and the thesis in [15], relatively little is known for the application of these techniques to the solution of sparse (generalized) symmetric eigenvalue problems. Indeed, while for linear systems the interface problem is also a linear system, for eigenvalue problems the interface problem is a *nonlinear* eigenvalue problem, i.e., the matrix operator associated with the interface variables is nonlinear. Throughout this dissertation we will be commonly referring to this operator as *spectral Schur complement*. The idea of using domain decomposition for the numerical solution of eigenvalue problems is not new. Though not formulated in the framework of domain decomposition, the paper by Abramov and Chishov [16] is the earliest we know that introduced the concept of spectral Schur complements. Other earlier publications describing approaches that share some common features with domain decomposition can be found in [17, 18, 19, 20, 21, 22]. In particular, the articles [17, 18] establish some theory of spectral Schur complement techniques from a Partial Differential Equations viewpoint. The paper [19] also resorts to spectral Schur complements, but it is not a domain decomposition approach. Rather, it exploits a given subspace, on which the Schur complement is based, to extract approximate eigenpairs. Although not presented from a spectral Schur complements viewpoint, the articles [21, 22] discuss condensation techniques applied to the Raviart-Thomas and discontinuous Galerkin approximation of second order elliptic eigenvalue problems. Condensation leads to the solution of non-linear, but smaller, eigenvalue problems and the techniques described therein have similarities with spectral Schur complement-based approaches.

The literature of domain decomposition-based ideas for the numerical solution of sparse symmetric (generalized) eigenvalue problems appears to be richer in structural mechanics [23, 24], although these ideas are generally closer to the idea of *substructuring*.[3] Similarly to domain decomposition, substructuring techniques divide the structure (global domain) into a set of smaller, independent substructures. An approximation of the solution of the original problem is then obtained by a linear combination of the solutions associated with the substructures. A major step forward towards popularizing domain decomposition techniques for the solution of sparse algebraic symmetric eigenvalue problems was realized by the development of the the Automated Multi-Level Substructuring method [25] (AMLS), a multilevel extension of Component Mode Synthesis [26]. Some details on AMLS are given in Section 2.5.2 of this dissertation.

1.4 Organization and contribution of this dissertation

The domain decomposition techniques presented in this dissertation are organized into two separate parts: (1) Filtering-based techniques (Chapters 3-6), and (2) Root-finding techniques (Chapters 7-8). Chapter 2 discusses background material on a few well-known techniques used throughout this dissertation for comparison purposes. Finally, Chapter 9 summarizes the material presented in this dissertation and presents potential future research directions.

1.4.1 Filtering-based techniques

The first class of domain decomposition eigenvalue solvers presented in this dissertation consists of filtering-based approaches in which a transformation $\rho(.)$ (also called the "filter function") is applied to the matrix pencil (A, M). This transformation $\rho(.)$ can be either rational or polynomial. The eigenvalues of the transformed pencil $\rho(A, M)$ are equal to $\rho(\lambda)$ while the corresponding eigenvectors remain the same. In practice, the transformation $\rho(.)$ is constructed so that the *nev* eigenvalues of the pencil (A, M) located inside the interval $[\alpha, \beta]$ become approximately equal to a pre-specified value, e.g. one, while most of the *n-nev* eigenvalues located

[3]Substructuring techniques predate hybrid domain decomposition techniques and solve the interface eigenvalue problem by a direct method

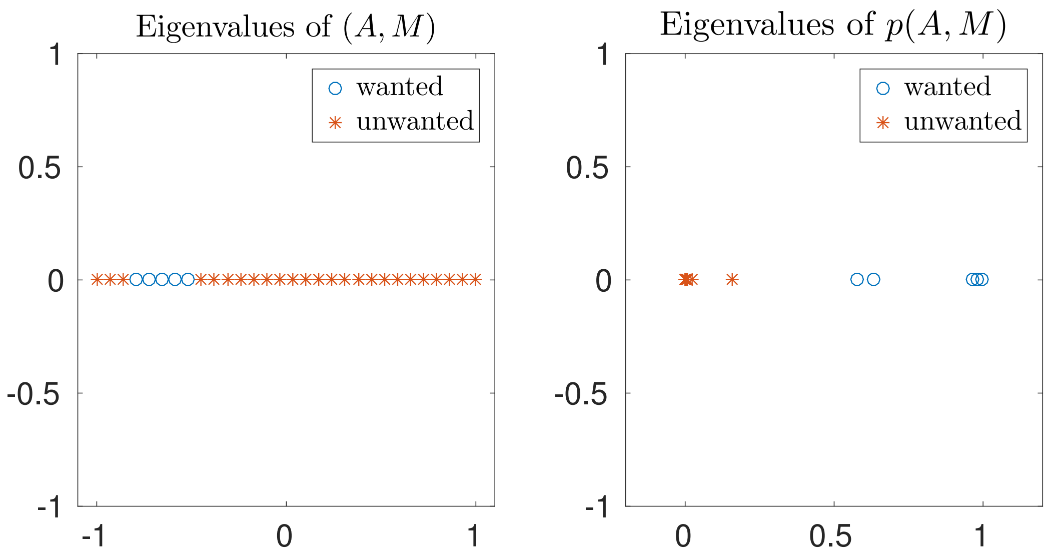


Figure 1.5: Filtering of the eigenvalues by a (polynomial) transformation of the pencil (A, M) to $p(A, M)$.

outside $[\alpha, \beta]$ are mapped to (approximately) zero. An example of a polynomial transformation is shown in Figure 1.5. Domain decomposition is then applied onto the transformed pencil $p(A, M)$. We present the proposed techniques, discuss their theoretical and practical details, and demonstrate via numerical experiments that this class of hybrid algorithms can outperform current state-of-the-art filtering-based approaches.

Chapter 3 presents RF-DDES (Rational Filtering Domain Decomposition Eigenvalue Solver), a numerical technique which combines domain decomposition with Krylov subspace projection methods and rational filters. Standard rational filtering techniques apply the rational filter to the entire matrix pencil, i.e., they require the solution of linear systems with complex coefficient matrices of the form $A - \zeta M$ for different complex values of $\zeta \in \mathbb{C}$. In contrast, RF-DDES applies the rational filter only to that part of $A - \zeta M$ that is associated with the interface variables of the domain. This approach has several advantages: (1) if a Krylov projection method is applied, orthonormalization needs to be applied to vectors whose length is equal to the number of interface variables only, (2) while RF-DDES also requires the solution of complex linear systems, the associated computational cost is lower than that of standard rational filtering approaches, (3) focusing on the interface variables only makes it possible to achieve convergence of the Krylov projection method in even fewer than *nev* iterations. In contrast, any Krylov projection method applied to a rational transformation of the original matrix pencil must perform at least *nev* iterations.

Chapter 4 discusses domain decomposition techniques in the context of contour integral eigenvalue solvers and proposes two numerical schemes. The first scheme can be seen as an extension of domain decomposition linear system solvers in the framework of contour integral methods for eigenvalue problems. In particular, contour integral eigenvalue solvers have mostly been associated with the use of sparse direct solvers to solve the linear systems which arise from the numerical approximation of the contour integral. This approach, however, is not always feasible due to the possible large amount of fill-in in the triangular factors, e.g. when factoring matrices that originate from discretizations of 3D computational domains. This chapter fills part of the gap that exists between contour integral eigenvalue solvers and the use of hybrid iterative solvers. The second scheme integrates the resolvent operator only partially. This leads to a scheme that is computationally cheaper than integrating over the entire resolvent operator but also requires special care if high accuracy is sought. A parallel implementation of the proposed schemes is described and results on distributed memory computing environments are reported. These results show that domain decomposition approaches can lead to reduced wall-clock times and improved scalability.

Chapter 5 considers the performance of contour integral eigenvalue solvers when additional levels of distributed memory parallelism are utilized. In particular, our main goal is to suggest a guideline on what are the opportunities for additional parallelism, and how a certain number of computational resources should be exploited so that the lowest possible wall-clock times can be achieved. We present results on the application of two levels of distributed memory in contour integral eigenvalue solvers such as those suggested in Chapter 4 and FEAST [27].

Chapter 6 describes the `Cucheb`[4] software package, a GPU implementation[5] of the filtered Lanczos procedure for the solution of large sparse symmetric eigenvalue problems. The filtered Lanczos procedure uses a carefully chosen polynomial spectral transformation to accelerate convergence of the Lanczos method when computing eigenvalues within a desired interval. Polynomial filtering approaches have been proven particularly effective for eigenvalue problems

[4]`https://github.com/jaurentz/cucheb`

[5]Although not based on Schur complements, the method presented in Chapter 6 is classified as a "domain decomposition" approach in the sense that the linear algebra operations underlying the numerical technique described therein are performed using the domain decomposition framework

that arise in electronic structure calculations and density functional theory. Compared to rational filtering techniques, polynomial filtering techniques access the matrix pencil only through Matrix-Vector products. The goal of this chapter is twofold. First, we describe our open source software package Cucheb that uses the filtered Lanczos procedure to accelerate large sparse eigenvalue computations using Nvidia brand GPUs. Second, we demonstrate the effectiveness of using GPUs to accelerate the filtered Lanczos procedure by solving a set of eigenvalue problems originating from electronic stucture calculations with Cucheb and comparing it with a similar CPU implementation. We compare Cucheb against a similar CPU-based implementation and show that using the GPU can reduce the wall-clock time by more than a factor of ten.

1.4.2 Root-finding techniques

The second class of domain decomposition eigenvalue solvers presented in this dissertation focuses on computing each sought eigenpair of the matrix pencil (A, M) independently of the others. In this sense, the techniques proposed in this section can be seen as alternatives to single-vector iterative schemes such as the Residual Inverse Iteration [28] and Rayleigh Quotient Iteration (RQI) [29].

Root-finding techniques compute an eigenpair (λ, x) of the matrix pencil (A, M) by first solving the interface eigenvalue problem to obtain the eigenvalue λ and interface part of eigenvector x. The remaining part of the eigenvector x, that associated with the interior variables, is computed independently in each subdomain. The main bottleneck is the solution of the interface eigenvalue problem, which although is of a much smaller dimension (equal to the number of interface variables), it is also of a nonlinear nature.

To solve the interface eigenvalue problem, root-finding techniques recast its solution into one of root-finding with a scalar function $\mathcal{F}(.)$ that satisfies the relation

$$\mathcal{F}(\sigma) : \mathbb{R} \to \mathbb{R} \mid \lambda \in \Lambda(A, M) \to \mathcal{F}(\sigma) \equiv 0,$$

i.e., the eigenvalues of (A, M) are roots of the scalar function $\mathcal{F}(\sigma)$. Note that the above definition allows $\mathcal{F}(.)$ to have roots which are not eigenvalues of the matrix pencil (A, M).

Chapter 7 presents a root-finding technique in which the scalar function $\mathcal{F}(.)$ is defined by parameterizing the eigenvalues of a zero-order approximation of the nonlinear matrix-valued function associated with the interface region, i.e., the spectral Schur complement operator. The root-finding problem is then solved by Newton's method. The primary goal of the approach presented in this chapter is to further extend current understanding of spectral Schur complement domain decomposition methods for the solution of symmetric eigenvalue problems, as well as to develop practical related algorithms. Distributed memory implementations of these algorithms are discussed and their performance is demonstrated.

Chapter 8 is similar to the techniques described in Chapter 7, i.e., the interface eigenvalue problem is recasted into one of computing roots of a scalar function. However, there is a notable difference. While the root-finding approach presented in Chapter 7 is applied on a scalar function defined by a zeroth-order approximation of the nonlinear matrix-valued function associated with the interface variables, the technique presented in this chapter exploits a first-order approximation instead. This choice requires a separate theoretical analysis, while it also leads to faster convergence of the root-finding scheme. The latter is verified experimentally where we demonstrate that the proposed scheme converges similarly, if not faster, than both RQI and the techniques described in Chapter 7, while is also more robust in the absence of accurate initial eigenvalue approximations.

Remark 1 *Throughout this dissertation the term "domain decomposition" has a dual meaning. In the first case, the domain decomposition framework is used as a mean to perform in parallel the various computational operations performed by a sparse eigenvalue solver applied to the matrix pencil (A, M), e.g. linear system solutions, Matrix-Vector products, inner products. In the second case, the domain decomposition framework is applied directly to the eigenvalue equation $Ax = \lambda x$, and the parts of the solution associated with the interior and interfaces variables are obtained in two separate phases. With the exception of Chapters 5 and 6, the focus of this dissertation lies on directly applying domain decomposition to the eigenvalue problem $Ax = \lambda Mx$.*

1.5 List of publications and conferences attended

A list of the core research articles associated with the present dissertation is as following:

1. V. Kalantzis, Y. Xi, and Y. Saad, **Beyond Automated MultiLevel Substructuring: Domain Decomposition with Rational Filtering**. *SIAM Journal on Scientific Computing, Vol. 40, No. 4, pp. C477-C502, 2018.*

2. V. Kalantzis, J. Kestyn, E. Polizzi, and Y. Saad, **Domain Decomposition Approaches for Accelerating Contour Integration Eigenvalue Solvers for Symmetric Eigenvalue Problems**, *Numerical Linear Algebra with Applications (To Appear).*

3. J. L. Aurentz, V. Kalantzis, and Y. Saad, **Cucheb: A GPU Implementation of the Filtered Lanczos Procedure**, *Computer Physics Communications, Vol. 220, pp. 332-340, 2017.*

4. V. Kalantzis, R. Li, and Y. Saad, **Spectral Schur Complement Techniques for Symmetric Eigenvalue Problems**, *Electronic Transactions on Numerical Analysis, Vol. 45, pp. 305-329, 2016.*

5. V. Kalantzis, **Spectral Schur complement Techniques for Symmetric Generalized Eigenvalue Problems**, *In Submission.*

In addition to the above research articles, the following articles are also related to this dissertation to some extent:

1. J. Kestyn, V. Kalantzis, E. Polizzi, and Y. Saad, **PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers**, *In Proceedings of the 2016 ACM/IEEE Supercomputing Conference (SC16).*

2. G. Dillon, V. Kalantzis, Y. Xi, and Y. Saad, **A Hierarchical Low-Rank Schur Complement Preconditioner for Indefinite Linear Systems**, *SIAM Journal on Scientific Computing, Vol. 40, No. 4, pp. A2234-A2252, 2018.*

Parts of the work appearing in this dissertation have been presented at the following conferences and institutions: 1) SIAM Conference on Computational Science and Engineering 2015 (Salt Lake City, Utah, USA), 2) SIAM Conference on Applied Linear Algebra 2015 (Atlanta, Georgia, USA), 3) ACM/IEEE Supercomputing Conference 2016 (Salt Lake City, Utah, USA), 4) SIAM Conference on Computational Science and Engineering 2017 (Atlanta, Georgia, USA), 5) PETSc User Meeting 2017 (Boulder, Colorado, US), 6) 21st Conference of the International Linear Algebra Society (Ames, Iowa, USA), 7) Argonne National Laboratory (Lemont, IL, USA), 8) Sandia National Laboratories (Albuquerque, NM, USA), and 9) Intel Parallel Computing Laboratories (Santa Clara, CA, USA). Travel support awards were granted by the funding committees of the following conferences: (a) SIAM Conference on Computational Science and

Engineering 2017, (b) PETSc User Meeting 2017, and (c) 21st Conference of the International Linear Algebra Society. The Computer Science and Engineering Department of the University of Minnesota also provided full funding for the (d) SIAM 2015 Conference on Computational Science and Engineering.

1.6 Computing environments and programming models

1.6.1 Hardware

The above software was designed, developed and tested on the computer systems of the Minnesota Supercomputing Institute. Some of the computer implementations used to perform the experiments presented in this dissertation can be found on the following webpage of the author: `https://www-users.cs.umn.edu/~kalan019/software.html`. More specifically, the experiments presented in this dissertation were performed on the following computing systems:

- The `Itasca` Linux cluster at Minnesota Supercomputing Institute. `Itasca` is an HP Linux cluster with 1,091 HP ProLiant BL280c G6 blade servers, each with two-socket, quad-core 2.8 GHz Intel Xeon X5560 "Nehalem EP" processors sharing 24 GB of system memory, with a 40-gigabit QDR InfiniBand (IB) interconnect. In all, Itasca consists of $8,728$ compute cores and 24 TB of main memory.

- The `Mesabi` Linux cluster at Minnesota Supercomputing Institute. `Mesabi` consists by 741 nodes of various configurations with a total of 17,784 compute cores provided by Intel Haswell E5-2680v3 processors. Each node features two sockets, each socket with twelve physical cores at 2.5 GHz and 32 GB of system memory. In total, `Mesabi` features a peak performance of 711 TeraFLOPS and 67 TB of memory.

1.6.2 Software

This dissertation features computer implementations written in Matlab, CUDA [30] and C/C++. The graphic illustrations appearing in this dissertation were generated by Matlab, and by the TikZ and PGF LATEX packages. All of the distributed memory implementations featured in this dissertation were written in C/C++ and on top of the Portable Extensible Toolkit for Scientific

Computation (PETSc) library [31, 32, 33]. PETSc is a collection of data structures and routines for the parallel solution of scientific applications, and is built on top of the Message Passing Interface (MPI) [34], a standardized and portable programming model for distributed memory computing environments. All linear algebra computations in the aforementioned C/C++ implementations were performed by calling the appropriate routines in the BLAS [35] and LAPACK [36] numerical libraries as those were implemented in the Math Kernel Library (MKL) [37].

1.7 Funding and access to computational resources

The research output presented in this dissertation was supported by (a) NSF under award CCF-1505970, (b) the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences under award number DE-SC0008877; and (c) a Gerondelis foundation scholarship. The Department of Computer Science and Engineering at University of Minnesota also contributed to the research that led to this dissertation by offering a full academic year's salary and tuition benefits through its Teaching Assistantships program.

The author would also like to acknowledge the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing resources that contributed to the research results reported in this dissertation.[6]

[6]The MSI, NSF, and DOE logos are listed in this dissertation with the sole purpose of acknowledging the support made available by the aforementioned institutions/departments.

Chapter 2

Related Work

The standard approach to compute a few, e.g. $nev \ll n$, eigenpairs of large and sparse symmetric eigenvalue problems of the form $Ax = \lambda Mx$ is to perform a Rayleigh-Ritz projection onto a low-dimensional subspace that spans an invariant subspace associated with the nev eigenvalues located inside the real interval $[\alpha, \beta]$. This section reviews some well-known techniques to build an efficient Rayleigh-Ritz projection subspace.

2.1 The Rayleigh-Ritz procedure

Algorithm 2.1.1 sketches the Rayleigh-Ritz procedure to approximate eigenpairs of eigenvalue problems of the form $Ax = \lambda Mx$. The full column rank matrix $Z \in \mathbb{R}^{n\times\kappa}$ represents a basis of the projection subspace $\mathcal{Z}$.

Algorithm **2.1.1** *The Rayleigh-Ritz procedure*

```
0.  Input: $A,\ M,\ Z \in \mathbb{R}^{n\times\kappa}$
1.  Form $H = Z^T AZ$ and $W = Z^T MZ$
2.  Solve the eigenvalue problem $Hg^{(i)} = \hat{\lambda}_i W g^{(i)},\ i = 1,\ldots,\kappa$
3.  Compute the Ritz vectors $\hat{x}^{(i)} = Zg^{(i)},\ i = 1,\ldots,\kappa$
```

The approximate eigenvalues $\hat{\lambda}_i$ computed by the Rayleigh-Ritz procedure are referred to as *Ritz values* while the corresponding approximate eigenvectors $\hat{x}^{(i)}$ are the associated *Ritz vectors*. It is straightforward to verify that the Rayleigh-Ritz procedure extracts approximate eigenpairs $\left(\hat{\lambda}_i, \hat{x}^{(i)}\right)$ which satisfy the *Galerkin* condition:

$$\hat{x}^{(i)} \in \mathcal{Z}, \qquad A\hat{x}^{(i)} - \hat{\lambda}_i M\hat{x}^{(i)} \perp \mathcal{Z}.$$

After a Ritz pair $\left(\hat{\lambda}_i, \hat{x}^{(i)}\right)$ is computed, we can determine its accuracy by computing the (relative) norm of the residual $r^{(i)} = Ax^{(i)} - \hat{\lambda}_i Mx^{(i)}$.

2.1.1 Optimality of the Rayleigh-Ritz procedure

Let $\mathcal{Z}$ denote the projection subspace and, as previously, let $Z \in \mathbb{R}^{n\times\kappa}$ denote a corresponding basis such that $Q = [Z, K]$ is an orthogonal matrix. Then, if $M = I$, i.e., M is the identity matrix, we can show the following.

Theorem 2.1.1 *Let $T = Z^T AZ$. The minimum of $\|AZ - ZR\|_2$ over all κ-by-κ symmetric matrices R is equal to $\|K^T AZ\|_2$ and is attained when $R = T$. Similarly, let $T = Y\hat{\Lambda}Y^T$ be the eigenvalue decomposition of the matrix T. Then, the minimum of $\|AP - PD\|_2$ over all $n \times \kappa$ orthonormal matrices P where* $\mathbf{span}(P) = \mathbf{span}(Z)$ *and diagonal matrices D is also equal to $\|K^T AZ\|_2$ and is attained by $P = ZY$ and $D = \hat{\Lambda}$.*

Proof: See [38]. □

According to Theorem 2.1.1, the columns of matrix $P = ZY$ (the Ritz vectors) are the "optimal" approximate eigenvectors and the diagonal entries of $\hat{\Lambda}$ are the "optimal" approximate eigenvalues in terms of minimizing the residual norm $\|AP - PD\|$.

A similar result can be also shown for the case where M is a general SPD matrix through converting the matrix pencil (A, M) to an equivalent one of the form $(L^{-1}AL^{-T}, I)$ where $M = LL^T$. In the latter case, the computed approximate eigenvectors will have to be multiplied by L^{-T} in order to retrieve the approximate eigenvectors $\hat{x}^{(i)}$ of the original matrix pencil (A, M).

Theorem 2.1.2 *Let the Ritz pairs of (A, M) computed by applying the Rayleigh-Ritz procedure on the projection subspace $\mathcal{Z} \equiv$ $\mathbf{span}(Z)$, $Z \in \mathbb{R}^{n\times\kappa}$, be of the form $(\hat{\lambda}_i, Zg^{(i)})$ where $g^{(i)}$*

satisfies $(Z^TAZ - \hat{\lambda}_i Z^TMZ)g^{(i)} = 0$*, and* $M = LL^T$*. Then, the Ritz pairs minimize*

$$\left\|L^{-1}R\right\|_F^2 = \text{trace}\left(R^TM^{-1}R\right),$$

where $R = \left[r^{(1)}, \ldots, r^{(\kappa)}\right]$ *with* $r^{(i)} = Ax^{(i)} - \hat{\lambda}_i Mx^{(i)}$*, and* $x^{(i)} \in \textbf{span}\,(Z)$*.*

Proof: See [29]. □

2.2 The shift-and-invert Lanczos method for symmetric generalized eigenvalue problems

The Lanczos method has become the standard method for approximating a few of the extremal eigenvalues and associated eigenvectors of large and sparse matrix pencils [7]. In a nutshell, the Lanczos method generates a low-dimensional (Krylov) subspace onto which the original eigenvalue problem is projected by imposing the Galerkin criterion. This projection is carried out iteratively and is represented by a tridiagonal matrix. The solution of the projected eigenvalue problem is then exploited to approximate the eigenpairs of the matrix pencil (A, M).

Throughout the rest of this section we describe the shift-and-invert variant of Lanczos. In particular, let $\sigma \in \mathbb{R} \setminus \Lambda(A, M)$, and consider the following transformation

$$\begin{aligned} Ax &= \lambda Mx \\ (A - \sigma M)x &= (\lambda - \sigma)Mx \\ M^{-1}(A - \sigma M)x &= (\lambda - \sigma)x \\ Cx &= \left(\frac{1}{\lambda - \sigma}\right)x \end{aligned}$$

where

$$C := (A - \sigma M)^{-1}M.$$

The eigenvectors of matrix C are identical to those of the matrix pencil (A, M). Moreover, each eigenvalue $\lambda_i,\ i = 1, \ldots, n$, of the matrix pencil (A, M) is mapped to the eigenvalue $\dfrac{1}{\lambda_i - \sigma}$ of matrix C. Since Lanczos typically converges towards the extremal eigenvalues first [39], the shift σ is chosen so that the eigenvalues $\dfrac{1}{\lambda_1 - \sigma}, \ldots, \dfrac{1}{\lambda_{nev} - \sigma}$ of matrix C are those of largest magnitude. Note here that while matrix C is nonsymmetric, the symmetry can be recovered by replacing the inner product $< x, y >= x^T y$ by $< x, y >_M = x^T M y$.

More specifically, the shift-and-invert Lanczos method builds a recurrence

$$CW_\mu = W_\mu T_\mu + r e_\mu^T$$

where $W_\mu^T M W_\mu = I$, $e_\mu \in \mathbb{R}^\mu$ denotes the μth canonical vector, and

$$W_\mu = \left[w^{(1)}, \ldots, w^{(\mu)} \right].$$

The columns of W_μ form an M-orthogonal basis for the Krylov subspace

$$\mathcal{K}_j(C, w^{(1)}) = \textbf{span}\left(\left[w^{(1)}, C w^{(1)}, \ldots, C^{j-1} w^{(1)} \right] \right).$$

The matrix

$$T_\mu = (M W_\mu)^T (A - \sigma M)^{-1} M W_\mu$$

is symmetric and tridiagonal, i.e., T_μ has the form

$$T_\mu = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_\mu \\ & & \beta_\mu & \alpha_\mu \end{pmatrix}.$$

Algorithm 2.2.1 lists the shift-and-invert variant of the Lanczos method combined with full orthogonalization. Each iteration shift-and-invert Lanczos requires one linear system solution with matrix $A - \sigma M$ and $O(n\mu)$ floating point operations to perform the orthonormalization at Steps (5)-(7). The matrix $A - \sigma M$ need be factorized only once at the outset. When accounting for all μ steps, the total computational cost associated with the full orthogonalization runs at $O(n\mu^2)$ floating point operations. Note that Algorithm 2.2.1 can be applied also in cases where M is singular[1] [40].

[1]As long as components in the direction of the nullspace of M are kept away from entering the computations

Algorithm **2.2.1** *The shift-and-invert Lanczos method with full orthogonalization*

```
0.  Start with $w^{(0)} = 0,\ \beta_1 = 0,\ \left(w^{(1)}\right)^T M w^{(1)} = 1$
1.  For $\mu = 1, 2, \ldots$
2.      Compute $r = \underbrace{(A - \sigma M)^{-1} M}_{C} w^{(\mu)} - w^{(\mu-1)}\beta_\mu$
3.      $\alpha_\mu = r^T M w^{(\mu)}$
4.      $r = r - w^{(\mu)}\alpha_\mu$
5.      $r = r - W_\mu (W_\mu^T (Mr))$
6.      $\beta_{\mu+1} = \|r^T M r\|_2$
7.      $w^{(\mu+1)} = r/\beta_{\mu+1}$
8.      Compute the eigendecomposition $T_\mu = S_\mu \Theta_\mu S_\mu^T$
        and test the accuracy of the Ritz pairs
9.  End
```

The ith eigenpair $\left(\theta_i^{(\mu)}, s_i^{(\mu)}\right)$ of matrix T_μ satisfies the equation

$$T_\mu s_i^{(\mu)} = \theta_i^{(\mu)} s_i^{(\mu)},\ i = 1, \ldots, \mu,$$

and it is used to compute a Ritz value

$$\lambda_i^{(\mu)} = \sigma + \frac{1}{\theta_i^{(\mu)}},$$

and Ritz vector

$$x_i^{(\mu)} = W_\mu s_i^{(\mu)},$$

of the matrix pencil (A, M). The residual $r_i^{(\mu)} = Ax_i^{(\mu)} - \lambda_i^{(\mu)} M x_i^{(\mu)}$ of the approximate eigenpair $\left(\lambda_i^{(\mu)}, x_i^{(\mu)}\right)$ of (A, M) satisfies the equation $\left(\theta_i^{(\mu)} \neq 0\right)$

$$r_i^{(\mu)} = -\frac{1}{\theta_i^{(\mu)}} (A - \sigma M) w^{(\mu+1)} \beta_{\mu+1} s_{\mu,i}^{(\mu)}.$$

Therefore, the norm of the residual $r_i^{(\mu)}$ is small whenever the absolute value of the ratio $\beta_\mu s_{\mu,i}^{(\mu)}/\theta_i^{(\mu)}$ is also small.

2.3 Rational filtering

When the number of eigenvalues *nev* located inside the interval of interest $[\alpha, \beta]$ is very large, e.g. thousands, the shift-and-invert Lanczos method presented in Section 2.2 becomes less practical due to the requirement to store and retain the orthonormality of the Krylov basis. Moreover, convergence towards invariant subspaces associated with eigenvalues lying further away from the shift σ can be slow. An idea to reduce memory and orthogonalization costs is to exploit partial orthogonalization [41] and/or thick restarting [42]. Another option is to reduce the memory and orthogonalization costs by computing the eigenpairs of interest by "slices", i.e., by subdividing the interval $[\alpha, \beta]$ into smaller subintervals and computing the eigenvalues and eigenvectors associated with each slice independently of the others [43]. This is the main idea behind the EigenValues Slicing Library[2] [44] (EVSL).

In this section we consider the acceleration of the convergence of Krylov subspace and Subspace Iteration projection methods by *rational filtering*. Rational filtering techniques are projection methods applied to a rational transformation of the pencil (A, M) so that (1) the eigenvalues located inside the interval $[\alpha, \beta]$ are mapped to an a-priori chosen value (typically one), and (2) those eigenvalues of (A, M) located outside the interval $[\alpha, \beta]$ are mapped to (approximately) zero. It is easy to verify that the shift-and-invert Lanczos method presented in Section 2.2 is also based on rational transformations, i.e., of the form $\rho(\zeta) = \dfrac{1}{\zeta - \sigma}$, but does not necessarily satisfy the second condition mentioned above.

In this section we discuss rational filtering approaches for the solution of real symmetric generalized eigenvalue problems, and focus on two specific methods: FEAST [27], and RF-KRYLOV [45].

2.3.1 Construction of the rational filter

The classic approach to construct a rational filter function $\rho(\zeta) : \mathbb{C} \rightarrow \mathbb{R}$ is to exploit the Cauchy integral representation of the indicator function

$$I_{[\alpha,\beta]}(\zeta) = \begin{cases} 1, \text{ iff } \zeta \in [\alpha, \beta] \\ 0, \text{ iff } \zeta \notin [\alpha, \beta] \end{cases}. \tag{2.1}$$

[2]`https://www-users.cs.umn.edu/~saad/software/EVSL/index.html`

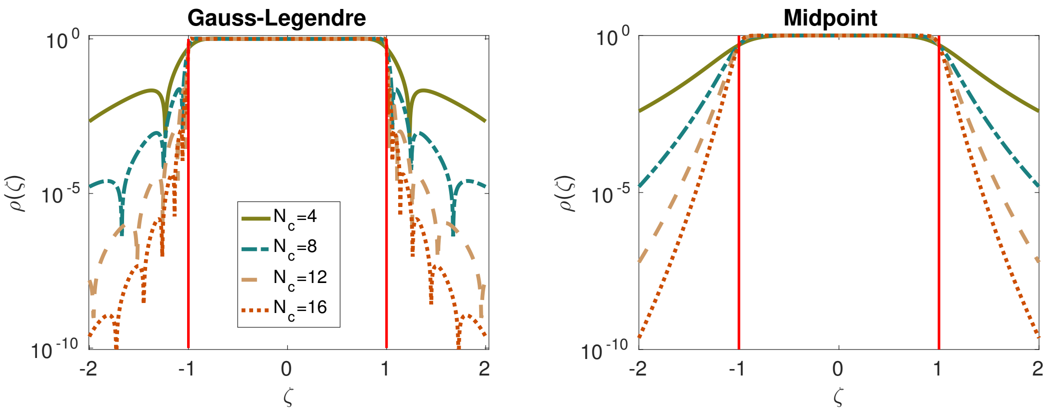


Figure 2.1: The modulus of the rational filter function $\rho(\zeta)$ when $\zeta \in [-2, 2]$. Left: Gauss-Legendre rule. Right: Midpoint rule.

In particular, let $\Gamma_{[\alpha,\beta]}$ be a smooth, closed contour that encloses only those nev eigenvalues of (A, M) which are located inside $[\alpha, \beta]$, e.g. a circle centered at $(\alpha+\beta)/2$ with radius $(\beta-\alpha)/2$. We then have

$$I_{[\alpha,\beta]}(\zeta) = \frac{1}{2\pi i}\int_{\Gamma_{[\alpha,\beta]}} \frac{1}{\nu-\zeta} d\nu, \tag{2.2}$$

where the integration is performed counter-clockwise. The filter function $\rho(\zeta)$ can be obtained by applying a quadrature rule to discretize the right-hand side in (2.2):

$$\rho(\zeta) = \sum_{\ell=1}^{2N_c} \frac{\hat{\omega}_\ell}{\zeta_\ell - \zeta} = \sum_{\ell=1}^{2N_c} \frac{\omega_\ell}{\zeta - \zeta_\ell}, \tag{2.3}$$

where $\{\zeta_\ell, \omega_\ell\}_{1\leq \ell \leq 2N_c} \equiv \{\zeta_\ell, -\hat{\omega}_\ell\}_{1\leq \ell \leq 2N_c}$ and $\{\zeta_\ell, \hat{\omega}_\ell\}_{1\leq \ell \leq 2N_c}$ are the poles and weights of the quadrature rule. Note that if the $2N_c$ poles in (2.3) come in conjugate pairs, and the first N_c poles lie on the upper half plane, the expression in (2.3) can be simplified into

$$\rho(\zeta) = 2\Re e \left\{ \sum_{\ell=1}^{N_c} \frac{\omega_\ell}{\zeta - \zeta_\ell} \right\}, \text{ when } \zeta \in \mathbb{R}. \tag{2.4}$$

Since the spectrum of the matrix pencil (A, M) is real, we will be relying on the expression in (2.4). Other approaches to construct rational filters (not related to Cauchy's formula) can be found in [46, 47, 48, 49].

Figure 2.1 plots the modulus of the rational function $\rho(\zeta)$ (scaled such that $\rho(\alpha) \equiv \rho(\beta) \equiv$

1/2) in the interval $\zeta \in [-2, 2]$, where $\{\zeta_\ell, \omega_\ell\}_{1\leq\ell\leq 2N_c}$ are obtained by numerically approximating $I_{[-1,1]}(\zeta)$ by the Gauss-Legendre rule (left) and Midpoint rule (right) [50]. Notice that as N_c increases, $\rho(\zeta)$ becomes a more accurate approximation of $I_{[-1,1]}(\zeta)$ [48].

2.3.2 Rational filtering projection methods

The eigenvalue problem $Ax = \lambda Mx$ is equivalent to the eigenvalue problem $M^{-1}Ax = \lambda x$. The spectral projector associated with the eigenvalues of the pencil $(A, M) \equiv (M^{-1}A, I)$ located inside the interval $[\alpha, \beta]$ can be written as

$$P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} = \frac{1}{2\pi i}\int_{\Gamma_{[\alpha,\beta]}} (\nu I - M^{-1}A)^{-1} d\nu \tag{2.5}$$

where, similarly to (2.2), $\Gamma_{[\alpha,\beta]}$ is a smooth, closed contour that encloses only those *nev* eigenvalues of (A, M) which are located inside $[\alpha, \beta]$, the integration is performed counter-clockwise. The spectral projector can be written equivalently as

$$\begin{aligned} P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} &= \frac{1}{2\pi i}\int_{\Gamma_{[\alpha,\beta]}} (\nu M^{-1}M - M^{-1}A)^{-1} d\nu \\ &= \frac{1}{2\pi i}\int_{\Gamma_{[\alpha,\beta]}} \left[M^{-1}(\nu M - A)\right]^{-1} d\nu \\ &= \frac{1}{2\pi i}\int_{\Gamma_{[\alpha,\beta]}} (\nu M - A)^{-1} M d\nu \\ &= \sum_{i=1}^{nev} x^{(i)} \left(x^{(i)}\right)^T M. \end{aligned} \tag{2.6}$$

In practice, the spectral projector in (2.6) will be discretized by a quadrature rule. This approximation $\rho(M^{-1}A) \approx P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}$ can be written as

$$\rho(M^{-1}A) = \sum_{\ell=1}^{N_c} \hat{\omega}_\ell (\zeta_\ell M - A)^{-1} M,$$

where $\{\zeta_\ell, \hat{\omega}_\ell\}_{1\leq\ell\leq 2N_c}$ are the same as in Section (2.3.1). Let $\Lambda := \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ and recall the identity $\omega_\ell = -\hat{\omega}_\ell$. Then, $\rho(M^{-1}A)$ can be written as

$$\begin{aligned}
\rho(M^{-1}A) &= \sum_{\ell=1}^{N_c} \omega_\ell (A - \zeta_\ell M)^{-1} M \\
&= 2\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell (A - \zeta_\ell M)^{-1} M \right\} \\
&= X \left(2\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell (\Lambda - \zeta_\ell I)^{-1} \right\} \right) X^T M \\
&= X \rho(\Lambda) X^T M \\
&= \sum_{i=1}^{n} \rho(\lambda_i) x^{(i)} \left(x^{(i)} \right)^T M
\end{aligned} \tag{2.7}$$

where $\rho(.)$ is the filter function defined in (2.4) and

$$\rho(\Lambda) = \begin{pmatrix} \rho(\lambda_1) & & & \\ & \rho(\lambda_2) & & \\ & & \ddots & \\ & & & \rho(\lambda_n) \end{pmatrix}.$$

The eigenvectors of $\rho(M^{-1}A)$ are identical to those of the matrix pencil (A, M), while the corresponding eigenvalues are transformed to $\{\rho(\lambda_j)\}_{j=1,\ldots,n}$. Since $\rho(\lambda_1), \ldots, \rho(\lambda_{nev})$ are all larger than $\rho(\lambda_{nev+1}), \ldots, \rho(\lambda_n)$, the *nev* eigenvalues of (A, M) located inside $[\alpha, \beta]$ become the *nev* dominant eigenvalues of matrix $\rho(M^{-1}A)$. Applying a projection method to $\rho(M^{-1}A)$ can then lead to rapid convergence towards an invariant subspace associated with the *nev* eigenvalues of (A, M) located inside $[\alpha, \beta]$.

The FEAST method

The FEAST[3] method, summarized in Algorithm 2.3.1, is a rational filtering scheme based on Subspace Iteration. In a nutshell, FEAST applies the Subspace Iteration projection scheme to the rational transformation of the matrix pencil (A, M) in (2.7). By recalling standard results on Subspace Iteration [7], the convergence rate of FEAST is then determined by the ratio

[3] `http://www.feast-solver.org/`

$\rho(\lambda_r)/\rho(\lambda_{nev})$, where $r \geq nev$ denotes the dimension of the initial approximation subspace $\mathcal{Z}$. A detailed numerical analysis of FEAST is presented in [51]. An extension to nonsymmetric eigenvalue problems can be found in [52].

Algorithm **2.3.1** *FEAST*

```
0.  Start with $Z \in \mathbb{R}^{n\times r}$ $(r \geq nev)$
1.  For $\mu = 1, 2, \ldots$
2.      Compute $W = 2\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell (A - \zeta_\ell M)^{-1} MZ\right\}$
3.      Set $A_W = W^T AW$ and $M_W = W^T MW$
4.      Solve $A_W Q = M_W Q\hat{\Lambda}$
5.      $Z = WQ$
6.  End
```

FEAST naturally captures all multiple eigenvalues. As FEAST is based on Subspace Iteration, a reasonably accurate estimation of *nev* (i.e., we must have $r \geq nev$). Techniques to estimate the number of eigenvalues located inside an interval $[\alpha, \beta]$ can be found in [53]. FEAST terminates as soon as the trace of matrix $\hat{\Lambda}$ remains the same up to a certain tolerance. Alternatively, we can determine the convergence of each approximate eigenpair independently by computing its residual norm explicitly and signal it as an accurate enough approximation if the latter norm is less than some user-given threshold. In practice, a robust implementation of FEAST should also take care of spurious solutions, e.g. see Section 3.7 in [52].

The dominant computational procedure of the FEAST method is that of computing the solution of the sparse linear systems with matrices $A - \zeta_1 M, \ldots, A - \zeta_{N_c} M$ at each iteration. These linear system solutions are performed in complex arithmetic and only their real part is retained.

Rational filtering Arnoldi

An alternative to Subspace Iteration is to exploit Krylov projection schemes [54, 55] as the projection scheme of choice. One immediate advantage of this approach is that no estimation of *nev* is then necessary. Additionally, Krylov projection schemes typically requires less computational work per eigenpair than Subspace Iteration. On the other hand, standard Krylov subspace approaches construct the projection subspace one vector at a time, and thus their efficiency might be lower than techniques based in Subspace Iteration which can take advantage of specialized multiple right-hand side linear system solvers. To remedy the latter drawback, block versions of Krylov projection schemes are possible, however these are not considered throughout this dissertation [56].

Algorithm 2.3.2 sketches the Arnoldi procedure applied to matrix $\rho(M^{-1}A)$ for the computation of all eigenvalues of the matrix pencil (A, M) located inside the interval $[\alpha, \beta]$ and associated eigenvectors. Step (2) computes the "filtered" vector w by applying the matrix function $\rho(M^{-1}A)$ to $q^{(\mu)}$, which in turn requires the solution of the N_c linear systems associated with matrices $A - \zeta_\ell M,\ \ell = 1, \ldots, N_c$. Steps (3)-(12) orthonormalize w against the previous Arnoldi vectors $q^{(1)}, \ldots, q^{(\mu)}$ to produce the next Arnoldi vector $q^{(\mu+1)}$. Step (13) checks the sum of those eigenvalues of the upper-Hessenberg matrix H_μ which are no less than 1/2. If this sum remains constant up to a certain tolerance, the outer loop stops. Finally, Step (16) computes the Rayleigh quotients associated with the approximate eigenvectors of $\rho(M^{-1}A)$ (the Ritz vectors obtained in Step (15)).

Algorithm **2.3.2** *RF-KRYLOV*

```
0.  Start with $q^{(1)} \in \mathbb{R}^n$ s.t. $\left\|q^{(1)}\right\|_2 = 1$
1.  For $\mu = 1, 2, \ldots$
2.      Compute $w = 2\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell (A - \zeta_\ell M)^{-1} M q^{(\mu)}\right\}$
3.      For $\kappa = 1, \ldots, \mu$
4.          $h_{\kappa,\mu} = w^T q^{(\kappa)}$
5.          $w = w - h_{\kappa,\mu} q^{(\kappa)}$
6.      End
7.      $h_{\mu+1,\mu} = \|w\|_2$
8.      If $h_{\mu+1,\mu} = 0$
9.          generate a unit-norm $q^{(\mu+1)}$ orthogonal to $q^{(1)}, \ldots, q^{(\mu)}$
10.     Else
11.         $q^{(\mu+1)} = w / h_{\mu+1,\mu}$
12      EndIf
13.     If the sum of eigenvalues of $H_\mu$ no less than $1/2$ is unchanged
        during the last few iterations; BREAK; EndIf
14. End
15. Compute the eigenvectors of $H_\mu$ and form the Ritz vectors of $(A, M)$
16. For each Ritz vector $\hat{q}$, compute the corresponding approximate
    Ritz value as the Rayleigh quotient $\hat{q}^T A \hat{q} / \hat{q}^T M \hat{q}$
```

Throughout the rest of this dissertation, Algorithm 2.3.2 will be abbreviated as RF-KRYLOV. We note at this point that while RF-KRYLOV is based on the Arnoldi scheme, it is also possible to convert the eigenvalue problem with the matrix transformation $\rho(M^{-1}A)$ to a symmetric one and use the Lanczos method instead. In particular, we can multiply both sides of the eigenvalue equation $\rho(M^{-1}A)x = \rho(\lambda)x$ from the left side by matrix M to get

$$M\rho(M^{-1}A)x = \rho(\lambda)Mx,$$

$$2\left(M\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell (A - \zeta_\ell M)^{-1}\right\} M\right) x = \rho(\lambda)Mx.$$

This is a SPD eigenvalue problem since M is SPD and $2\left(M\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell(A-\zeta_\ell M)^{-1}\right\}M\right)$ is symmetric. Alternatively, it is possible to set $x = M^{-1}y$ and write the eigenvalue equation $\rho(M^{-1}A)x = \rho(\lambda)x$ as

$$\begin{aligned}\rho(M^{-1}A)M^{-1}y &= \rho(\lambda)M^{-1}y,\\ 2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell(A-\zeta_\ell M)^{-1}\right\}y &= \rho(\lambda)M^{-1}y.\end{aligned}$$

Again, this is a SPD eigenvalue problem since M^{-1} is SPD and $2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell(A-\zeta_\ell M)^{-1}\right\}$ is symmetric. More details on the above can be found in [44].

2.3.3 Discussion

Both RF-KRYLOV and FEAST are highly suitable for execution on current high-end computers. In addition to the ability to slice the interval of interest $[\alpha, \beta]$ to non-overlapping subintervals and process each subinterval in parallel [43, 57], the application of the matrix $2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell(A-\zeta_\ell M)^{-1}M\right\}$ to a vector (or a set of vectors) can be performed in parallel by assigning each one of the N_c linear system solutions to different groups of processors. Clearly, this level of parallelism is restrained by N_c. A third level of parallelism is also possible by solving each one of the N_c linear systems by a domain decomposition distributed linear system solver. More details on the parallelization strategies of rational filtering approaches, FEAST in particular, can be found in [58] and Section 5 of the present dissertation.

By default the linear systems in both FEAST and RF-KRYLOV are solved by a direct sparse linear system solver. When iterative solvers are the only practical option, FEAST is more robust than RF-KRYLOV due to the fact that Subspace Iteration is by itself more robust than Krylov subspace approaches in the presence of inexact applications of the coefficient matrix. Note here that the application of iterative solvers in both FEAST and RF-KRYLOV can be particularly challenging as N_c increases since in this case the quadrature nodes $\zeta_j,\ j = 1, \ldots, N_c$ will come closer to the real axis. Therefore, when iterative linear system solvers are considered, it might be more practical to set N_c to a low value, e.g. one or two. Another idea is to try to fix the location of the quadrature node(s) a-priori, and compute the associated weights of the rational filter $\rho(.)$ by a least-squares approximation of the indicator function $I_{[\alpha,\beta]}(.)$ [48]. Some results on this subject from the viewpoint of contour integral filters can be found in [59, 60] (see also

Section 4 of the present dissertation).

While not discussed in this dissertation, it is possible to consider contour integrals of other rational functions, e.g., the scalar function $u^H(\zeta M - A)^{-1}v$, with $u,\ v,\ \in \mathbb{C}^n$, as proposed by Sakurai and Sugiura [61, 62, 63]. The poles of this scalar function are the eigenvalues of the matrix pencil (A, M). A similar algorithm, based on pole-finding of rational interpolants of $u^*(\zeta M - A)^{-1}v$, can be found in [46].

2.4 Polynomial filtering

Similarly to rational filtering techniques, polynomial filtering approaches aim to accelerate the convergence rate of the projection method of choice by applying a polynomial transformation to matrix pencil (A, M). More specifically, let $M = LL^T$ be the Cholesky decomposition of matrix M. Then, the original eigenvalue problem $Ax = \lambda Mx$ can be written in an equivalent form $L^{-1}AL^{-T}\left(L^T x\right) = \lambda\left(L^T x\right)$. The idea then is to apply a projection method, i.e., Lanczos, to a polynomial transformation of the matrix $L^{-1}AL^{-T},\ p\left(L^{-1}AL^{-T}\right)$, where $p(.) : \mathbb{R} \to \mathbb{R}$ is chosen so that $p(\lambda_i) \approx 0$ for any $\lambda_i \notin [\alpha, \beta]$.

In practice, the polynomial filter $p(.)$ is typically an approximation of the indicator function $I_{[\alpha,\beta]}(.)$. Another option is to construct $p(.)$ by considering a polynomial approximation of the Dirac delta function. This technique is implemented in the EVSL library [44]. In any case, Chebyshev polynomials are used as the basis functions. Then, $p(.)$ can be written as

$$p(\zeta) = \sum_{i=0}^{m} b_i T_i(\zeta),\ \zeta \in \mathbb{R}, \tag{2.8}$$

where $\{b_i\}_{i=0}^m$ are the expansion coefficients, and T_i denotes the ith Chebyshev polynomial of the first kind. The expansion coefficients can be computed in a few different ways, e.g. by either truncating the Chebyshev series approximation of $I_{[\alpha,\beta]}(.)$ or solving a least-squares problem [43].

Since Chebyshev polynomials are defined over the reference interval $[-1,1]$, a linear transformation is needed to map the eigenvalues of $L^{-1}AL^{-T}$ to this reference interval. This can be achieved by shifting the origin, i.e.,

$$\overline{L^{-1}AL^{-T}} = \frac{L^{-1}AL^{-T} - xI}{e},\quad x = \frac{\lambda_{\min} + \lambda_{\max}}{2},\quad e = \frac{\lambda_{\min} - \lambda_{\max}}{2},$$

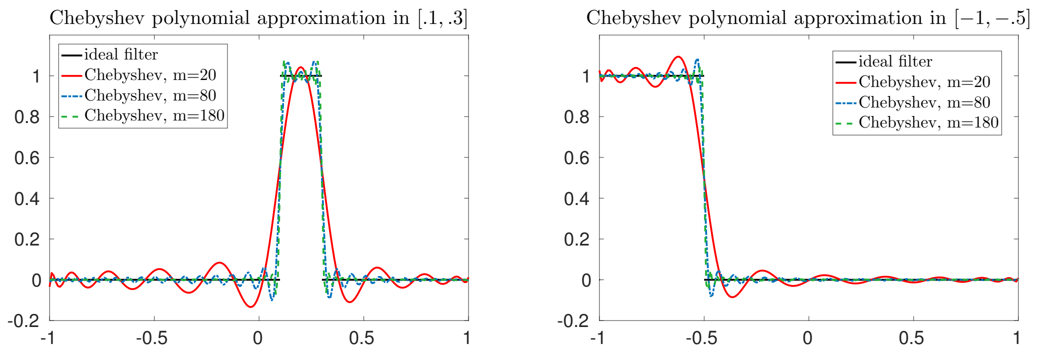


Figure 2.2: Chebyshev polynomial approximation of $I_{[\alpha,\beta]}$, $[\alpha,\beta] \subseteq [-1,1]$, for different degree values. Left: $[\alpha,\beta] = [.1,.3]$. Right: $[\alpha,\beta] = [-1,-.5]$.

where I denotes the identity matrix of the same dimension as A, and $\lambda_{\min}$, $\lambda_{\max}$ denote the algebraically smallest and largest eigenvalues of $L^{-1}AL^{-T}$ (i.e., (A, M)), respectively. In practice, we do not need to know the exact values of $\lambda_{\min}$ and $\lambda_{\max}$, but rather a tight underestimation and overestimation, respectively. Note that the same transformation should be also applied to the interval $[\alpha,\beta]$, i.e., $[\alpha,\beta] := [(\alpha - x)/e, (\beta - x)/e]$.

Figure 2.2 shows a Chebyshev polynomial approximation of $I_{[\alpha,\beta]}$ for different degree values m and intervals $[\alpha,\beta]$. In practice the degree m will have to be set automatically, e.g. see Section 6.2.1 of this dissertation and the discussion in [43]. Note that higher values of m do not necessarily lead to (much) faster convergence. Indeed, our experience tells us that increasing m typically pays off only for heavily interior eigenvalue problems.

2.4.1 The FILTLAN library

Algorithm 2.4.1 (FILTLAN) applies the Lanczos method combined with full orthogonalization to matrix $p(\overline{L^{-1}AL^{-T}})$. Since A is symmetric, the matrix H_μ is actually tridiagonal, and no entries above the leading superdiagonal need be retained. Each iteration of FILTLAN requires $2m$ triangular substitutions with matrix L, m Matrix-Vector products with matrix A, as well as $O(n\mu)$ floating point operations to orthonormalize w with against the Krylov basis generated up to the μth iteration. The total orthogonalization cost runs at $O(n\mu^2)$ floating point operations.

Algorithm **2.4.1** *FILTLAN with full orthogonalization*

```
0.  Start with $q^{(0)} = 0,\ \beta_1 = 0,\ q^{(1)} \in \mathbb{R}^n$ s.t. $\left\|q^{(1)}\right\|_2 = 1$
1.  For $\mu = 1, 2, \ldots$
2.      Compute $w = p(\overline{L^{-1}AL^{-T}})q^{(\mu)}$-$q^{(\mu-1)}\beta_\mu$
3.      $h_{\mu,\mu} = w^T q^{(\mu)}$
4.      For $\kappa = 1, \ldots, \mu$
5.          $\gamma = w^T q^{(\kappa)}$
6.          $w = w - \gamma q^{(\kappa)}$
7.      End
8.      $h_{\mu+1,\mu} = h_{\mu,\mu+1} = \|w\|_2$
9.      If $H_{\mu+1,\mu} = 0$
10.         generate a unit-norm $q^{(\mu+1)}$ orthogonal to $q^{(1)}, \ldots, q^{(\mu)}$
11.     Else
12.         $q^{(\mu+1)} = w/H_{\mu+1,\mu}$
13.     EndIf
14.     If the sum of eigenvalues of $H_\mu$ no less than $1/2$ is unchanged
        during the last few iterations; BREAK; EndIf
15. End
16. Compute the eigenvectors of $H_\mu$ and form the Ritz vectors of $(A, M)$
17. For each Ritz vector $\hat{q}$, compute the corresponding approximate
    Ritz value as the Rayleigh quotient $\hat{q}^T A\hat{q}/\hat{q}^T M\hat{q}$
```

By the above discussion it becomes apparent that unless M has a very particular structure, e.g. diagonal, the FILTLAN implementation presented in Algorithm 2.4.1 can be considerably expensive due to the repeated triangular substitutions with the triangular matrices L and L^T. Indeed, the true strength of the method is realized when (1) $M = I$, (2) A is large and sparse, and (3) a factorization of A is impractical. Polynomial filtering has proven itself highly successful when applied to eigenvalue problems associated with matrices originating structure calculations [64], e.g. see [65, 66, 67].

A computer implementation[4] of FILTLAN combined with partial orthogonalization [41] and

[4] `http://www-users.cs.umn.edu/~saad/software/filtlan/`

least-squares polynomial filters can be found in [67]. For an implementation of polynomial filtering approaches on multi-core CPUs and distributed memory environments we refer to [68, 44].

2.5 Domain decomposition techniques

An alternative to reduce the orthonormalization costs in large scale eigenvalue computations is to consider domain decomposition-type approaches (we refer to [13, 12] for an in-depth discussion of domain decomposition). Domain decomposition approaches decouple the original eigenvalue problem into two separate subproblems: (a) one defined locally in the interior of each subdomain, and (b) one defined on the interface region connecting neighboring subdomains. Once the original eigenvalue problem is solved for the interface region, the solution associated with the interior of each subdomain is computed independently of the rest of the subdomains [69, 70, 25, 71, 17, 18, 20]. When the number of variables associated with the interface region is much smaller than the number of global variables, domain decomposition approaches can provide approximations to thousands of eigenpairs while avoiding excessive orthogonalization costs.

Algebraic domain decomposition eigensolvers begin by calling a graph partitioner [72, 73] to decompose the adjacency graph of $|A| + |M|$ into $p \in \mathbb{Z}^+$ non-overlapping subdomains. If we then order the interior variables in each subdomain before the interface variables across all subdomains, matrices A and M take the following block structures:

$$\begin{aligned}
A &= \begin{pmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_p & E_p \\ E_1^T & E_2^T & \dots & E_p^T & C \end{pmatrix}, \\
M &= \begin{pmatrix} M_B^{(1)} & & & & M_E^{(1)} \\ & M_B^{(2)} & & & M_E^{(2)} \\ & & \ddots & & \vdots \\ & & & M_B^{(p)} & M_E^{(p)} \\ \left(M_E^{(1)}\right)^T & \left(M_E^{(2)}\right)^T & \dots & \left(M_E^{(p)}\right)^T & M_C \end{pmatrix}.
\end{aligned} \tag{2.9}$$

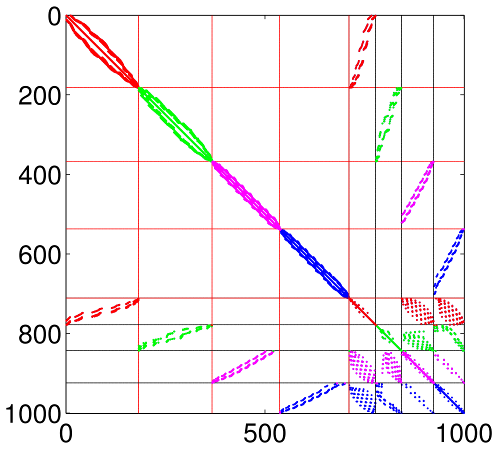


Figure 2.3: Sparsity pattern of A and M obtained by partitioning the graph of $|A| + |M|$ in $p = 4$ subdomains by METIS. Different colors indicate entries that are associated with different subdomains.

The matrices B_j and $M_B^{(j)}$ are of size $d_j \times d_j$, where d_j denotes the number of interior variables residing in the jth subdomain. On the other hand, matrices E_j and $M_E^{(j)}$ are rectangular matrices of size $d_j \times s$. In particular, the matrices E_j, $M_E^{(j)}$ have a special nonzero pattern of the form $E_j = \left[0_{d_j,\ell_j}, \hat{E}_j, 0_{d_j,\nu_j}\right]$, and $M_E^{(j)} = \left[0_{d_j,\ell_j}, \hat{M}_E^{(j)}, 0_{d_j,\nu_j}\right]$, where s_j denotes the number of interfaces variables in the jth subdomain (note that $s = \sum_{j=1}^{p} s_j$), $\ell_j = \sum_{k=1}^{k<j} s_k$, $\nu_j = \sum_{k>j}^{k=p} s_k$, and $0_{\chi,\psi}$ denotes the zero matrix of size $\chi \times \psi$. Throughout the rest of this dissertation we will partition the adjacency graph of $|A| + |M|$ by the METIS library [72].

Under the permutation in (2.9), A and M can be also written as:

$$A = \begin{pmatrix} B & E \\ E^T & C \end{pmatrix}, \quad M = \begin{pmatrix} M_B & M_E \\ M_E^T & M_C \end{pmatrix}. \tag{2.10}$$

The block-diagonal matrices B and M_B are of size $d \times d$, where $d = \sum_{i=1}^{p} d_i$, while E and M_E are of size $d \times s$. Matrices C and M_C are square matrices of size $s \times s$, where $s = \sum_{j=1}^{p} s_j$.

Figure 2.3 plots the sparsity pattern of A and M after a reordering obtained by partitioning the graph of $|A| + |M|$ in $p = 4$ subdomains by METIS. Matrix A was chosen as 7-pt stencil Finite Difference discretization of the Laplace operator on the unit cube using a regular grid with the same step size in each direction.

Throughout the rest of this dissertation we will retain the symbols A and M to denote the permuted matrices in (2.9).

2.5.1 Invariant subspaces from a Schur complement viewpoint

Domain decomposition eigenvalue solvers can be seen as Rayleigh-Ritz projection techniques in which the subspace $\mathcal{Z}$ to perform the projection onto is of the form

$$\mathcal{Z} = \mathcal{U} \oplus \mathcal{Y}, \tag{2.11}$$

where $\mathcal{U}$ and $\mathcal{Y}$ are (structurally) orthogonal subspaces that approximate the part of the solution associated with the interior and interface variables, respectively.

Let the ith eigenvector of (A, M) be partitioned as

$$x^{(i)} = \begin{pmatrix} u^{(i)} \\ y^{(i)} \end{pmatrix}, \; i = 1, \ldots, n, \tag{2.12}$$

where $u^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}^s$ correspond to the eigenvector parts associated with the interior and interface variables, respectively. We can then rewrite $Ax^{(i)} = \lambda_i M x^{(i)}$ in the following block form

$$\begin{pmatrix} B - \lambda_i M_B & E - \lambda_i M_E \\ E^T - \lambda_i M_E^T & C - \lambda_i M_C \end{pmatrix} \begin{pmatrix} u^{(i)} \\ y^{(i)} \end{pmatrix} = 0. \tag{2.13}$$

Eliminating $u^{(i)}$ from the second equation in (2.13) leads to the following nonlinear eigenvalue problem of size $s \times s$:

$$\Big[C - \lambda_i M_C - (E - \lambda_i M_E)^T (B - \lambda_i M_B)^{-1} (E - \lambda_i M_E)\Big] y^{(i)} = 0. \tag{2.14}$$

Once λ_i and $y^{(i)}$ are computed in the above equation, $u^{(i)}$ can be recovered by the following linear system solution

$$(B - \lambda_i M_B) u^{(i)} = -(E - \lambda_i M_E) y^{(i)}. \tag{2.15}$$

In practice, since matrices B and M_B in (2.9) are block-diagonal, the p sub-vectors $u_j^{(i)} \in \mathbb{R}^{d_j}$ of $u^{(i)} = \left[\left(u_1^{(i)}\right)^T, \ldots, \left(u_p^{(i)}\right)^T\right]^T$ can be computed in a decoupled fashion among the p

subdomains as

$$\left(B_j - \lambda_i M_B^{(j)}\right) u_j^{(i)} = -\left(\hat{E}_j - \lambda_i \hat{M}_E^{(j)}\right) y_j^{(i)},\ j = 1, \ldots, p, \tag{2.16}$$

where $y_j^{(i)} \in \mathbb{R}^{s_j}$ is the subvector of $y^{(i)} = \left[\left(y_1^{(i)}\right)^T, \ldots, \left(y_p^{(i)}\right)^T\right]^T$ that corresponds to the jth subdomain.

By (2.14) and (2.15) we see that the subspaces $\mathcal{U}$ and $\mathcal{Y}$ in (2.11) should ideally be chosen as

$$\mathcal{Y} = \mathbf{span}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right), \tag{2.17}$$

$$\mathcal{U} = \mathbf{span}\left(\left[(B - \lambda_1 M_B)^{-1}(E - \lambda_1 M_E)y^{(1)}, \ldots, (B - \lambda_{nev} M_B)^{-1}(E - \lambda_{nev} M_E)y^{(nev)}\right]\right). \tag{2.18}$$

2.5.2 The Automated Multi-Level Substructuring method

The Automated Multi-Level Substructuring (AMLS) method [25, 70, 74], originally developed by the structural engineering community for the frequency response analysis of Finite Element automobile bodies, is one of the best known algebraic substructuring techniques eigenvalue solvers. AMLS has been proven considerably faster than the block Lanczos shift-and-invert (SIBL) [75] approach implemented in the NASTRAN industrial package [76] in applications where $nev \gg 1$. One of the main reasons of this superiority of AMLS versus SIBL lies on the fact that AMLS avoids the orthonormalization of n-dimensional subspaces. On the other hand, AMLS is a non-iterative method and as such its accuracy is relatively modest only for those of (A, M) located close to some user-given shift $\sigma \in \mathbb{R}$. Some techniques to improve the accuracy provided of AMLS by means of post-processing can be found in [70, 77].

AMLS is essentially a Rayleigh-Ritz projection technique onto a subspace defined as

$$\mathcal{Z}_{AMLS} = \mathcal{U}_{AMLS} \oplus \mathcal{Y}_{AMLS},$$

where $\mathcal{U}_{AMLS}$ and $\mathcal{Y}_{AMLS}$ are structurally orthogonal subspaces associated with the interior and interface variables, respectively. In particular, AMLS approximates the part of the solution associated with the interface variables, $\mathbf{span}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$, by the subspace

$$\mathcal{Y}_{AMLS} = \mathbf{span}\left(\left[\hat{y}^{(1)}, \ldots, \hat{y}^{(k)}\right]\right), \tag{2.19}$$

where $\hat{y}^{(i)},\ i = 1,\dots,k$, denote the eigenvectors associated with the k eigenvalues of smallest magnitude of the SPD matrix pencil $(S(\sigma), -S'(\sigma))$ where $S'(\sigma)$ denotes the derivative of the matrix-valued function $S(.)$ evaluated at $\sigma \in \mathbb{R}$. In AMLS this shift σ is typically zero since the method was originally developed for the analysis of low-frequency response of automobile bodies. Similarly, AMLS approximates the part of the solution associated with the interior variables of the domain by the subspace

$$\mathcal{U}_{AMLS} = \textbf{span}\left(\left[(B - \sigma M_B)^{-1}(E - \sigma M_E)\left[\hat{y}^{(1)},\dots,\hat{y}^{(k)}\right], V\right]\right),$$

where $V \in \mathbb{R}^{b \times pnev_B}$ is a block-diagonal matrix that is distributed row-wise among the p different subdomains,

$$V = \begin{pmatrix} V_1 & & & \\ & V_2 & & \\ & & \ddots & \\ & & & V_p \end{pmatrix},$$

and V_j holds the eigenvectors associated with the nev_B smallest magnitude eigenvalues of each matrix pencil $\left(B_j - \sigma M_B^{(j)}, M_B^{(j)}\right)$.

In order to reduce the computational costs associated with each subdomain, AMLS computes each matrix V_j by applying the same technique recursively, i.e., V_j is also computed by applying AMLS to each matrix pencil $\left(B_j - \sigma M_B^{(j)}, M_B^{(j)}\right),\ j = 1,\dots,p$. As a result, AMLS is typically combined with a recursive Nested Dissection ordering of the pencil (A, M) [78].

2.6 Test matrices

This section provides details on the test matrices used throughout this dissertation.

2.6.1 The Dirichlet eigenvalue problem

Throughout this dissertation we will use the solution of the Dirichlet eigenvalue problem as a testbed to provide insights on the performance of various eigenvalue solvers. The Dirichlet

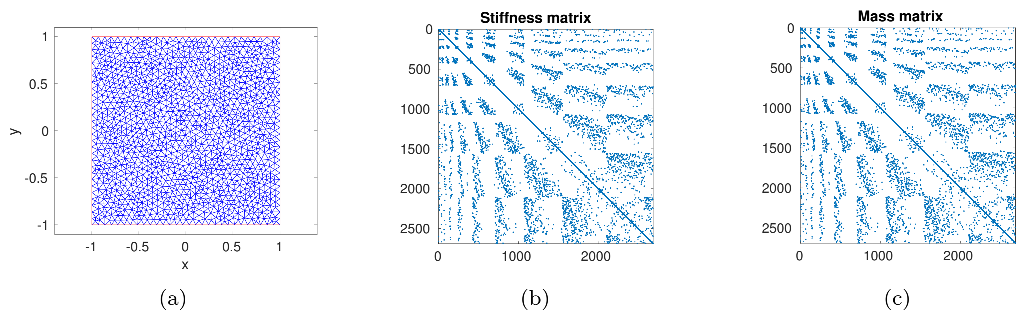


Figure 2.4: a) The FE mesh, b) the sparsity pattern of matrix A, and c) the sparsity pattern of matrix M.

eigenvalue problem can be formally defined as

$$\begin{aligned} \Delta u + \lambda u &= 0 \text{ in } \Omega \\ u_{|\partial\Omega} &= 0 \end{aligned} \tag{2.20}$$

where $\Delta = \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2}$ is the Laplace operator. We will consider two different discretization techniques, Finite Difference and Finite Element. In both cases, the eigenvalues of the discretized Dirichlet eigenvalue problem will be real and positive.

Finite Difference discretization We consider the solution of the Dirichlet eigenvalue problem in (2.20) on 3D (2D) domains $\Omega := [0,1] \times [0,1] \times [0,1]$ ($\Omega := [0,1] \times [0,1]$). For the Finite Difference discretization, we consider 5-pt (2D) and 7-pt (3D) stencils. We will denote the mesh size of the domain by $n_x \times n_y \times n_z$ and the discretized Laplacian matrix A will be of size $n = n_x \times n_y \times n_z$ (or $n = n_x \times n_y$ for the 2D case).

Finite Element discretization We consider the solution of the Dirichlet eigenvalue problem in (2.20) on 2D domains $\Omega := [-1,1] \times [-1,1]$. Unless noted otherwise, we will use linear elements with target maximum mesh edge length of $h = 0.05$.

Figure 2.4 plots the FE mesh and sparsity pattern of matrices A and M of a Finite Element discretization of the $[-1,1] \times [-1,1]$ plane using linear elements with target maximum mesh edge length of $h = 0.05$.[5]

[5]We used Matlab's PDE toolbox to generate these matrices

2.6.2 General matrices

The second class of matrices used throughout this dissertation can be found in the SuiteSparse[6] Matrix Collection [79], a freely accessible set of sparse matrices that arise in various real-world applications.

2.7 Details on notation

Throughout the rest of this dissertation we will frequently need to refer to matrices $B - \sigma M_B$, $E - \sigma M_E$, $C - \sigma M_C$ and $C - \sigma M_C - (E - \sigma M_E)^T (B - \sigma M_B)^{-1} (E - \sigma M_E)$. Therefore, we introduce the following notation.

Definition 2.7.1 *For any $\sigma \in \mathbb{C}$, we define the following matrix-valued functions:*

$$\begin{aligned} B_\sigma &= B - \sigma M_B, \\ E_\sigma &= E - \sigma M_E, \\ C_\sigma &= C - \sigma M_C. \end{aligned}$$

and

$$\begin{aligned} S(\sigma) &= C - \sigma M_C - (E - \sigma M_E)^T (B - \sigma M_B)^{-1} (E - \sigma M_E), \\ &= C_\sigma - E_\sigma^T B_\sigma^{-1} E_\sigma. \end{aligned} \tag{2.21}$$

[6]https://sparse.tamu.edu/

Part I

Filtering techniques

Chapter 3

A Krylov-based rational filtering domain decomposition technique

This section presents a Krylov-based rational filtering domain decomposition technique.[1] The main idea behind the proposed technique is to perform a Rayleigh-Ritz projection onto a subspace $\mathcal{Z}$ of the form

$$\mathcal{Z} = \mathcal{U} \oplus \mathcal{Y},$$

where $\mathcal{U}$ and $\mathcal{Y}$ are (structurally) orthogonal subspaces that approximate the part of the solution associated with the interior and interface variables, respectively. As was already discussed in Section 2.5.1, the ideal choice for subspaces $\mathcal{U}$ and $\mathcal{Y}$ is to set

$$\begin{aligned}
&\mathcal{Y} = \mathbf{span}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right), \\
&\mathcal{U} = \mathbf{span}\left(\left[(B-\lambda_1 M_B)^{-1}(E-\lambda_1 M_E)y^{(1)}, \ldots, (B-\lambda_{nev} M_B)^{-1}(E-\lambda_{nev} M_E)y^{(nev)}\right]\right).
\end{aligned}$$

The technique presented in this section is abbreviated as Rational Filtering Domain Decomposition Eigenvalue Solver (RF-DDES). RF-DDES builds the subspace associated with the interface variables of the subdomains by leveraging rational filters. On the other hand, the subspace associated with the interior variables of each subdomain is built independently from the rest of the subdomains using only real arithmetic. Compared to shift-and-invert Krylov projection

[1]This is joint work with Yuanzhe Xi and Yousef Saad (University of Minnesota, Twin Cities)

schemes applied to the matrix pencil (A, M), RF-DDES applies orthonormalization to vectors whose length is equal only to the number of interface variables. Numerical experiments performed in distributed memory architectures illustrate the competitiveness of the proposed technique against rational filtering Krylov approaches.

The structure of this chapter is as follows: Section 3.1 describes computational approaches for the solution of the eigenvalue problem associated with the interface variables. Section 3.2 describes the solution of the eigenvalue problem associated with the interior variables in each subdomain. Section 3.3 combines all previous discussion into the form of an algorithm. Section 3.4 presents experiments performed on model and general matrix pencils. Finally, Section 3.5 contains our concluding remarks.

3.1 Approximation of span $\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$

In this section we propose a numerical scheme to approximate **span** $\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$. Before we proceed, we remind the reader of the following identities:

$$\rho(\zeta) = 2\Re e \left\{ \sum_{\ell=1}^{N_c} \frac{\omega_\ell}{\zeta - \zeta_\ell} \right\},$$

and

$$\rho(M^{-1}A) = 2\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell (A - \zeta_\ell M)^{-1} M \right\}.$$

where $\{\zeta_\ell, \omega_\ell\}_{1 \leq \ell \leq N_c}$ are complex pairs.

3.1.1 Rational filtering restricted to the interface region

Each matrix $(A - \zeta_\ell M)^{-1}$ can be written in a 2×2 block form as

$$(A - \zeta_\ell M)^{-1} = \begin{pmatrix} B_{\zeta_\ell}^{-1} + B_{\zeta_\ell}^{-1} E_{\zeta_\ell} S(\zeta_\ell)^{-1} E_{\zeta_\ell}^T B_{\zeta_\ell}^{-1} & -B_{\zeta_\ell}^{-1} E_{\zeta_\ell} S(\zeta_\ell)^{-1} \\ -S(\zeta_\ell)^{-1} E_{\zeta_\ell}^T B_{\zeta_\ell}^{-1} & S(\zeta_\ell)^{-1} \end{pmatrix}. \tag{3.1}$$

Substituting (3.1) into $\rho(M^{-1}A)$ leads to

$$\rho(M^{-1}A) = 2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell\begin{bmatrix}\left[B_{\zeta_\ell}^{-1}+B_{\zeta_\ell}^{-1}E_{\zeta_\ell}S(\zeta_\ell)^{-1}E_{\zeta_\ell}^T B_{\zeta_\ell}^{-1}\right] & -B_{\zeta_\ell}^{-1}E_{\zeta_\ell}S(\zeta_\ell)^{-1}\\ -S(\zeta_\ell)^{-1}E_{\zeta_\ell}^T B_{\zeta_\ell}^{-1} & S(\zeta_\ell)^{-1}\end{bmatrix}\right\}M. \tag{3.2}$$

On the other hand, for any $\zeta \notin \Lambda(A,M)$ we can write

$$(A-\zeta M)^{-1} = \sum_{i=1}^{n}\frac{x^{(i)}\left(x^{(i)}\right)^T}{\lambda_i-\zeta}. \tag{3.3}$$

The above equality yields another equivalent expression for $\rho(M^{-1}A)$:

$$\rho(M^{-1}A) = \sum_{i=1}^{n}\rho(\lambda_i)x^{(i)}\left(x^{(i)}\right)^T M \tag{3.4}$$

$$= \sum_{i=1}^{n}\rho(\lambda_i)\begin{bmatrix}u^{(i)}\left(u^{(i)}\right)^T & u^{(i)}\left(y^{(i)}\right)^T\\ y^{(i)}\left(u^{(i)}\right)^T & y^{(i)}\left(y^{(i)}\right)^T\end{bmatrix}M. \tag{3.5}$$

Equating the (2,2) blocks of the right-hand sides in (3.2) and (3.5), yields

$$2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\} = \sum_{i=1}^{n}\rho(\lambda_i)\left[y^{(i)}\left(y^{(i)}\right)^T\right]. \tag{3.6}$$

Equation (3.6) provides a way to approximate $\mathbf{span}\left(\left[y^{(1)},\ldots,y^{(nev)}\right]\right)$ through the information in matrix $2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}$. In particular, the magnitude of $\rho(\lambda_i)$ can be interpreted as the contribution of the direction $y^{(i)}$ in $2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}$. In the ideal case where $\rho(\zeta)\equiv \pm I_{[\alpha,\beta]}(\zeta)$, we have $\sum_{i=1}^{n}\rho(\lambda_i)y^{(i)}\left(y^{(i)}\right)^T = \pm\sum_{i=1}^{nev}y^{(i)}\left(y^{(i)}\right)^T$. In practice, $\rho(\zeta)$ will only be an approximation to $\pm I_{[\alpha,\beta]}(\zeta)$, and since $\rho(\lambda_1),\ldots,\rho(\lambda_{nev})$ are all nonzero, the following relation holds:

$$\mathbf{span}\left(\left[y^{(1)},\ldots,y^{(nev)}\right]\right) \subseteq \mathbf{range}\left(\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}\right). \tag{3.7}$$

The above relation suggests to compute an approximation to $\mathbf{span}\left(\left[y^{(1)},\ldots,y^{(nev)}\right]\right)$ by capturing the range space of $\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}$.

3.1.2 A Krylov-based approach

To capture **range** $\left(\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}\right)$ we consider the numerical scheme outlined in Algorithm 3.1.1. In contrast with RF-KRYLOV (see Section 2.3.2), Algorithm 3.1.1 is based on the Lanczos process [7]. Variable T_μ denotes a $\mu \times \mu$ symmetric tridiagonal matrix with $\alpha_1, \ldots, \alpha_\mu$ as its diagonal entries, and $\beta_2, \ldots, \beta_\mu$ as its off-diagonal entries, respectively. Step (2) computes the "filtered" vector w by applying $\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}$ to $q^{(\mu)}$ by solving the N_c linear systems associated with matrices $S(\zeta_\ell),\ \ell = 1, \ldots, N_c$. Steps (4)-(12) orthonormalize w against vectors $q^{(1)}, \ldots, q^{(\mu)}$ in order to generate the next vector $q^{(\mu+1)}$. Throughout this chapter we apply full orthogonalization but other choices, e.g. partial orthogonalization [41], are possible. Algorithm 3.1.1 terminates when the trace of the tridiagonal matrices T_μ and $T_{\mu-1}$ remains the same up to a certain tolerance.

Algorithm **3.1.1** *Krylov restricted to the interface variables*

```
0.  Start with $q^{(1)} \in \mathbb{R}^s$, s.t. $\left\|q^{(1)}\right\|_2 = 1$, $q_0 := 0$, $\beta_1 = 0$, $\mathrm{tol} \in \mathbb{R}$
1.  For $\mu = 1, 2, \ldots$
2.      Compute $w = \Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}q^{(\mu)}\right\} - \beta_\mu q^{(\mu-1)}$
3.      $\alpha_\mu = w^T q^{(\mu)}$
4.      For $\kappa = 1, \ldots, \mu$
5.          $w = w - q^{(\kappa)}\left(w^T q^{(\kappa)}\right)$
6.      End
7.      $\beta_{\mu+1} = \|w\|_2$
8.      If $\beta_{\mu+1} = 0$
9.          generate a unit-norm $q^{(\mu+1)}$ orthogonal to $q^{(1)}, \ldots, q^{(\mu)}$
10.     Else
11.         $q^{(\mu+1)} = w/\beta_{\mu+1}$
12      EndIf
13.     If the sum of the eigenvalues of $T_\mu$ remains unchanged (up to $\mathrm{tol}$)
        during the last few iterations; BREAK; EndIf
14. End
15. Return $Q_\mu = \left[q^{(1)}, \ldots, q^{(\mu)}\right]$
```

Algorithm 3.1.1 shares a few key differences with RF-KRYLOV. First, Algorithm 3.1.1 restricts orthonormalization to vectors of length s instead of n. In addition, Algorithm 3.1.1 only requires linear system solutions with $S(\zeta)$ instead of $A - \zeta M$. As can be verified by (8.1.3), a computation of the form $(A-\zeta M)^{-1}v = w$ requires, in addition to a linear system solution with matrix $S(\zeta)$, two linear system solutions with B_ζ as well as two Matrix-Vector multiplications with E_ζ. Finally, in contrast to RF-KRYLOV which requires at least *nev* iterations to compute any *nev* eigenpairs of the pencil (A, M), Algorithm 3.1.1 might terminate in fewer than *nev* iterations. This possible "early termination" of Algorithm 3.1.1 is explained in more detail by Proposition 3.1.1.

Proposition 3.1.1 *The rank of the matrix* $\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1}\right\}$,

$$r(S) = \mathbf{rank}\left(\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1}\right\}\right), \tag{3.8}$$

satisfies the inequality

$$\mathbf{rank}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right) \leq r(S) \leq s. \tag{3.9}$$

Proof: We first prove the upper bound of $r(S)$. Since $\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1}\right\}$ is of size $s \times s$, $r(S)$ can not exceed s. To get the lower bound, let $\rho(\lambda_i) = 0,\ i = nev + \kappa, \ldots, n$, where $0 \leq \kappa \leq n - nev$. We then have

$$\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1}\right\} = \sum_{i=1}^{nev+\kappa} \rho(\lambda_i) y^{(i)} \left(y^{(i)}\right)^T,$$

and $\mathbf{rank}\left(\Re e\left\{\sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1}\right\}\right) = \mathbf{rank}\left(\left[y^{(1)}, \ldots, y^{(nev+\kappa)}\right]\right)$. Since $\rho(\lambda_i) \neq 0,\ i = 1, \ldots, nev$, we have

$$r(S) = \mathbf{rank}\left(\left[y^{(1)}, \ldots, y^{(nev+\kappa)}\right]\right) \geq \mathbf{rank}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right).$$

□

By Proposition 3.1.1, Algorithm 3.1.1 will perform at most $r(S)$ iterations, and $r(S)$ can be as small as $\mathbf{rank}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$. We quantify this with a short example for a 2D Laplacian matrix generated by a Finite Difference discretization with Dirichlet boundary conditions (for more details on this matrix see entry "FDmesh1" in Table 8.1) where we set

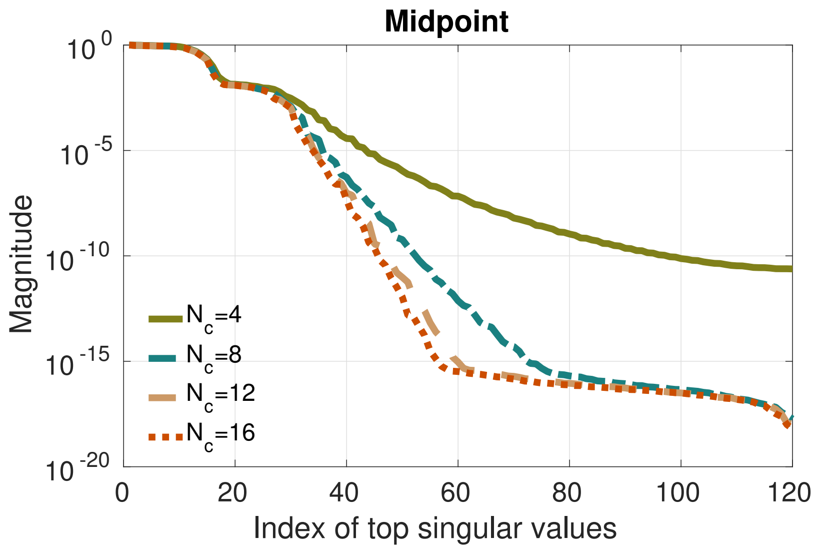


Figure 3.1: The leading singular values of $\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}$ for different values of N_c when (2.2) is discretized by the Midpoint rule.

$[\alpha,\beta]=[\lambda_1,\lambda_{100}]$ (thus $nev=100$). After computing the vectors $y^{(1)},\ldots,y^{(nev)}$ explicitly, we found that $\mathbf{rank}\left(\left[y^{(1)},\ldots,y^{(nev)}\right]\right)=48$. Figure 3.1 plots the 120 (after normalization) leading singular values of matrix $\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell S(\zeta_\ell)^{-1}\right\}$. As N_c increases, the trailing $s-\mathbf{rank}\left(\left[y^{(1)},\ldots,y^{(nev)}\right]\right)$ singular values approach zero. Moreover, even for those singular values which are not zero, their magnitude might be small, in which case Algorithm 3.1.1 might still converge in fewer than $r(S)$ iterations. Indeed, when $N_c=16$, Algorithm 3.1.1 terminates after exactly 36 iterations which is lower than $r(S)$ and only one third of the minimum number of iterations required by RF-KRYLOV for any value of N_c. As a sidenote, when $N_c=2$, Algorithm 3.1.1 terminates after 70 iterations.

3.2 Approximation of span $\left(\left[u^{(1)},\ldots,u^{(nev)}\right]\right)$

Recall the eigenvector partitioning $\left(x^{(i)}\right)^T=\left[\left(u^{(i)}\right)^T,\left(y^{(i)}\right)^T\right]^T$ for each $x^{(i)}$, λ_i, $i=1,\ldots,nev$. A straightforward approach to compute the part $u^{(i)}$ of $x^{(i)}$ is then to solve the block-diagonal linear system $-B_\lambda u^{(i)}=E_\lambda y^{(i)}$. However, this approach entails two main drawbacks. First, solving the linear systems with matrix B_{λ_i} for all λ_i, $i=1,\ldots,nev$ becomes prohibitively expensive when $nev\gg 1$. In addition, Algorithm 3.1.1 only returns an approximation of

span $\left(\left[y^{(1)},\ldots,y^{(nev)}\right]\right)$, rather than the individual vectors $y^{(1)},\ldots,y^{(nev)}$, or the eigenvalues $\lambda_1,\ldots,\lambda_{nev}$.

In this section we consider alternative approaches to approximate the subspace **span** $\left(\left[u^{(1)},\ldots,u^{(nev)}\right]\right)$. Since the following discussion applies to any of the n eigenpairs $(\lambda, x^{(i)})$ of (A,M), we will drop the superscripts in vectors $u^{(i)}$ and $y^{(i)}$.

3.2.1 The basic approximation

Let $x^T = \left[u^T, y^T\right]^T$ be the eigenvector associated with (the unknown) eigenvalue λ of (A,M). The part associated with the interior variables, u, can be then approximated as

$$\hat{u} = -B_\sigma^{-1}E_\sigma y \tag{3.10}$$

where $\sigma \in \mathbb{R}$. As it is trivial to verify, when $\sigma \equiv \lambda$, the formula in (3.10) leads to $\hat{u} \equiv u$. In the general case, the approximation error is of the following form.

Lemma 3.2.1 *Suppose u and $\hat{u}$ are computed as in* (2.15) *and* (3.10)*, respectively. Then:*

$$u - \hat{u} = -\left[B_\lambda^{-1} - B_\sigma^{-1}\right]E_\sigma y + (\lambda - \sigma)B_\lambda^{-1}M_E y. \tag{3.11}$$

Proof: We can write u as

$$\begin{aligned} u &= -B_\lambda^{-1}E_\lambda y \\ &= -B_\lambda^{-1}(E_\sigma - (\lambda-\sigma)M_E)y \\ &= -B_\lambda^{-1}E_\sigma y + (\lambda-\sigma)B_\lambda^{-1}M_E y. \end{aligned} \tag{3.12}$$

The result in (3.11) follows by combining (3.10) and (3.12). □

We are now ready to compute an upper bound of $u - \hat{u}$ measured in the M_B-norm.[2]

Theorem 3.2.2 *Let the eigendecomposition of (B, M_B) be written as*

$$BV = M_B V D, \tag{3.13}$$

where $D = \mathrm{diag}(\delta_1,\ldots,\delta_d)$ and $V = \left[v^{(1)},\ldots,v^{(d)}\right]$. If $\hat{u}$ is defined as in (3.10) and $\left(\delta_\ell, v^{(\ell)}\right)$, $\ell =$

[2]We define the X-norm of any nonzero vector y and SPD matrix X as $||y||_X = \sqrt{y^T X y}$.

$1, \ldots, d$, denote the eigenpairs of (B, M_B) where each eigenvector $v^{(\ell)}$ is scaled such that $\left(v^{(\ell)}\right)^T M_B v^{(\ell)} = 1$, then

$$\|u - \hat{u}\|_{M_B} \leq \max_{\ell} \frac{|\lambda - \sigma|}{|(\lambda - \delta_\ell)(\sigma - \delta_\ell)|} ||E_\sigma y||_{M_B^{-1}} + \max_{\ell} \frac{|\lambda - \sigma|}{|\lambda - \delta_\ell|} ||M_E y||_{M_B^{-1}}, \tag{3.14}$$

Proof: Since M_B is SPD, vectors $E_\sigma y$ and $M_E y$ in (3.12) can be expanded in the basis $M_B v^{(\ell)}$ as:

$$E_\sigma y = M_B \sum_{\ell=1}^{\ell=d} \epsilon_\ell v^{(\ell)}, \quad M_E y = M_B \sum_{\ell=1}^{\ell=d} \gamma_\ell v^{(\ell)}, \tag{3.15}$$

where $\epsilon_\ell,\ \gamma_\ell \in \mathbb{R}$ are the expansion coefficients. By recalling (3.13) and noticing that $V^T M_B V = I$, we get

$$B_\sigma^{-1} = V(D - \sigma I)^{-1} V^T, \qquad B_\lambda^{-1} = V(D - \lambda I)^{-1} V^T. \tag{3.16}$$

Substituting the expressions in (3.15) and (3.16) into the right-hand side of (3.11) gives

$$\begin{aligned} u - \hat{u} &= -V\left[(D - \lambda I)^{-1} - (D - \sigma I)^{-1}\right] V^T \left(M_B \sum_{\ell=1}^{\ell=d} \epsilon_\ell v^{(\ell)}\right) \\ &\quad + (\lambda - \sigma) V (D - \lambda I)^{-1} V^T \left(M_B \sum_{\ell=1}^{\ell=d} \gamma_\ell v^{(\ell)}\right) \\ &= -\sum_{\ell=1}^{\ell=d} \frac{\epsilon_\ell(\lambda - \sigma)}{(\delta_\ell - \lambda)(\delta_\ell - \sigma)} v^{(\ell)} + \sum_{\ell=1}^{\ell=d} \frac{\gamma_\ell(\lambda - \sigma)}{\delta_\ell - \lambda} v^{(\ell)}. \end{aligned} \tag{3.17}$$

Now, taking the M_B-norm of (3.17), we finally obtain

$$\begin{aligned} ||u - \hat{u}||_{M_B} &\leq \left\| \sum_{\ell=1}^{\ell=d} \frac{-\epsilon_\ell(\lambda - \sigma)}{(\delta_\ell - \lambda)(\delta_\ell - \sigma)} v^{(\ell)} \right\|_{M_B} + \left\| \sum_{\ell=1}^{\ell=d} \frac{\gamma_\ell(\lambda - \sigma)}{\delta_\ell - \lambda} v^{(\ell)} \right\|_{M_B} \\ &= \left\| \sum_{\ell=1}^{\ell=d} \left| \frac{(\lambda - \sigma)}{(\delta_\ell - \lambda)(\delta_\ell - \sigma)} \right| \epsilon_\ell v^{(\ell)} \right\|_{M_B} + \left\| \sum_{\ell=1}^{\ell=d} \left| \frac{(\lambda - \sigma)}{\delta_\ell - \lambda} \right| \gamma_\ell v^{(\ell)} \right\|_{M_B} \\ &\leq \max_{\ell} \frac{|\lambda - \sigma|}{|(\lambda - \delta_\ell)(\sigma - \delta_\ell)|} \left\| \sum_{\ell=1}^{\ell=d} \epsilon_\ell v^{(\ell)} \right\|_{M_B} + \max_{\ell} \frac{|\lambda - \sigma|}{|\lambda - \delta_\ell|} \left\| \sum_{\ell=1}^{\ell=d} \gamma_\ell v^{(\ell)} \right\|_{M_B} \\ &= \max_{\ell} \frac{|\lambda - \sigma|}{|(\lambda - \delta_\ell)(\sigma - \delta_\ell)|} \left\| M_B^{-1} E_\sigma y \right\|_{M_B} + \max_{\ell} \frac{|\lambda - \sigma|}{|\lambda - \delta_\ell|} \left\| M_B^{-1} M_E y \right\|_{M_B} \\ &= \max_{\ell} \frac{|\lambda - \sigma|}{|(\lambda - \delta_\ell)(\sigma - \delta_\ell)|} \left\| E_\sigma y \right\|_{M_B^{-1}} + \max_{\ell} \frac{|\lambda - \sigma|}{|\lambda - \delta_\ell|} \left\| M_E y \right\|_{M_B^{-1}}. \end{aligned}$$

□

Theorem 3.2.2 indicates that the upper bound of $||u-\hat{u}||_{M_B}$ depends on the distance between σ and λ, as well as the distance of these values from the eigenvalues of (B, M_B). In particular, this upper bound becomes small when λ and σ lie close to each other, and far from the eigenvalues of (B, M_B).

3.2.2 Enhancing accuracy by resolvent expansions

Consider the resolvent expansion of B_λ^{-1} around $\sigma \in \mathbb{R}$:

$$B_\lambda^{-1} = B_\sigma^{-1} \sum_{\theta=0}^{\infty} \Big[(\lambda-\sigma) M_B B_\sigma^{-1}\Big]^\theta. \tag{3.18}$$

By recalling (3.11), the error $u - \hat{u}$ consists of two components: *i)* $(B_\lambda^{-1} - B_\sigma^{-1})E_\sigma y$; and *ii)* $(\lambda-\sigma)B_\lambda^{-1} M_E y$. A straightforward idea then is to approximate B_λ^{-1} by also considering higher-order terms in (3.18) instead of B_σ^{-1} only. Furthermore, the same idea can be repeated for the second error component. Thus, we can extract $\hat{u}$ by a projection step from the following subspace

$$\hat{u} \in \Big\{B_\sigma^{-1}E_\sigma y, \ldots, B_\sigma^{-1}\left(M_B B_\sigma^{-1}\right)^{\psi-1} E_\sigma y, B_\sigma^{-1} M_E y, \ldots, B_\sigma^{-1}\left(M_B B_\sigma^{-1}\right)^{\psi-1} M_E y\Big\}. \tag{3.19}$$

The following theorem refines the upper bound of $\|u - \hat{u}\|_{M_B}$ presented in Theorem 3.2.2.

Theorem 3.2.3 *Let* $\mathcal{U} = \textbf{span}\,(U_1, U_2)$ *where*

$$\begin{aligned} U_1 &= \Big[B_\sigma^{-1}E_\sigma y, \ldots, B_\sigma^{-1}\left(M_B B_\sigma^{-1}\right)^{\psi-1} E_\sigma y\Big], \\ U_2 &= \Big[B_\sigma^{-1}M_E y, \ldots, B_\sigma^{-1}\left(M_B B_\sigma^{-1}\right)^{\psi-1} M_E y\Big]. \end{aligned} \tag{3.20}$$

If $\hat{u} := \arg\min_{g\in\mathcal{U}} \|u-g\|_{M_B}$*, and* $\left(\delta_\ell, v^{(\ell)}\right)$*,* $\ell = 1, \ldots, d$*, denote the eigenpairs of* (B, M_B)*, then:*

$$\|u-\hat{u}\|_{M_B} \leq \max_\ell \frac{|\lambda-\sigma|^\psi}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|}||E_\sigma y||_{M_B^{-1}} + \max_\ell \frac{|\lambda-\sigma|^{\psi+1}}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|}||M_E y||_{M_B^{-1}}. \tag{3.21}$$

Proof: Define the vector $g := U_1\mathbf{c}_1 + U_2\mathbf{c}_2$ where

$$\mathbf{c}_1 = -\Big[1, \lambda-\sigma, \ldots, (\lambda-\sigma)^{\psi-1}\Big]^T, \quad \mathbf{c}_2 = \Big[\lambda-\sigma, \ldots, (\lambda-\sigma)^{\psi}\Big]^T.$$

The difference $u-g$ then is equal to

$$\begin{aligned} u-g = &-\left[B_\lambda^{-1} - B_\sigma^{-1}\sum_{\theta=0}^{\psi-1}\Big[(\lambda-\sigma)M_BB_\sigma^{-1}\Big]^\theta\right]E_\sigma y \\ &+(\lambda-\sigma)\left[B_\lambda^{-1} - B_\sigma^{-1}\sum_{\theta=0}^{\psi-1}\Big[(\lambda-\sigma)M_BB_\sigma^{-1}\Big]^\theta\right]M_E y. \end{aligned} \tag{3.22}$$

Expanding B_σ^{-1} and B_λ^{-1} in the eigenbasis of (B, M_B) gives

$$B_\lambda^{-1} - B_\sigma^{-1}\sum_{\theta=0}^{\psi-1}\Big[(\lambda-\sigma)M_BB_\sigma^{-1}\Big]^\theta = (\lambda-\sigma)^\psi V(D-\lambda I)^{-1}(D-\sigma I)^{-\psi}V^T, \tag{3.23}$$

and thus (3.22) can be simplified as

$$\begin{aligned} u-g = &-(\lambda-\sigma)^\psi V(D-\lambda I)^{-1}(D-\sigma I)^{-\psi}V^TE_\sigma y \\ &+(\lambda-\sigma)^{\psi+1}V(D-\lambda I)^{-1}(D-\sigma I)^{-\psi}V^TM_E y. \end{aligned}$$

Plugging in the expansion of $E_\sigma y$ and $M_E y$ defined in (3.15) finally leads to

$$u-g = -\sum_{\ell=1}^{\ell=d}\frac{\epsilon_\ell(\lambda-\sigma)^\psi}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi}v^{(\ell)} + \sum_{\ell=1}^{\ell=d}\frac{\gamma_\ell(\lambda-\sigma)^{\psi+1}}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi}v^{(\ell)}. \tag{3.24}$$

Similarly to Theorem 3.2.2, we consider the M_B-norm of (3.24):

$$\begin{aligned} \|u-g\|_{M_B} \leq &\left\|\sum_{\ell=1}^{\ell=d}\frac{-\epsilon_\ell(\lambda-\sigma)^\psi}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi}v^{(\ell)}\right\|_{M_B} + \left\|\sum_{\ell=1}^{\ell=d}\frac{\gamma_\ell(\lambda-\sigma)^{\psi+1}}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi}v^{(\ell)}\right\|_{M_B} \\ \leq &\max_\ell \frac{|\lambda-\sigma|^\psi}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|}\left\|\sum_{\ell=1}^{\ell=d}\epsilon_\ell v^{(\ell)}\right\|_{M_B} \\ &+\max_\ell \frac{|\lambda-\sigma|^{\psi+1}}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|}\left\|\sum_{\ell=1}^{\ell=d}\gamma_\ell v^{(\ell)}\right\|_{M_B} \\ = &\max_\ell \frac{|\lambda-\sigma|^\psi}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|}||E_\sigma y||_{M_B^{-1}} + \max_\ell \frac{|\lambda-\sigma|^{\psi+1}}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|}||M_E y||_{M_B^{-1}}. \end{aligned}$$

Recalling that $\hat{u} := \arg\min_{g \in \mathcal{U}} \|u - g\|_{M_B}$ finishes the proof. $\square$

A comparison of the bound in Theorem 3.2.3 with the bound in Theorem 3.2.2 indicates that one may expect an improved approximation when σ is close to λ. The numerical examples in Section 3.4 will verify that this approach enhances accuracy even when $|\sigma - \lambda|$ is not very small.

3.2.3 Enhancing accuracy by eigenvector deflation

Both Theorem 3.2.2 and Theorem 3.2.3 imply that the approximation error $u - \hat{u}$ might have its largest components along those eigenvector directions associated with those eigenvalues of (B, M_B) located the closest to σ. We can remove these error directions explicitly by augmenting the projection subspace with the corresponding eigenvectors of (B, M_B).

Theorem 3.2.4 *Let $\delta_1, \delta_2, \ldots, \delta_\kappa$ be the κ eigenvalues of (B, M_B) that lie the closest to σ, and let $v^{(1)}, v^{(2)}, \ldots, v^{(\kappa)}$ denote the corresponding eigenvectors. Moreover, let $\mathcal{U} = \mathbf{span}\,(U_1, U_2, U_3)$ where*

$$\begin{aligned} U_1 &= \left[B_\sigma^{-1} E_\sigma y, \ldots, B_\sigma^{-1} \left(M_B B_\sigma^{-1}\right)^{\psi-1} E_\sigma y\right], \\ U_2 &= \left[B_\sigma^{-1} M_E y, \ldots, B_\sigma^{-1} \left(M_B B_\sigma^{-1}\right)^{\psi-1} M_E y\right], \\ U_3 &= \left[v^{(1)}, v^{(2)}, \ldots, v^{(\kappa)}\right]. \end{aligned} \tag{3.25}$$

If $\hat{u} := \arg\min_{g \in \mathcal{U}} \|u - g\|_{M_B}$ and $\left(\delta_\ell, v^{(\ell)}\right),\ \ell = 1, \ldots, d$, denote the eigenpairs of (B, M_B), then:

$$\|u - \hat{u}\|_{M_B} \leq \max_{\ell > \kappa} \frac{|\lambda - \sigma|^\psi}{|(\lambda - \delta_\ell)(\sigma - \delta_\ell)^\psi|} ||E_\sigma y||_{M_B^{-1}} + \max_{\ell > \kappa} \frac{|\lambda - \sigma|^{\psi+1}}{|(\lambda - \delta_\ell)(\sigma - \delta_\ell)^\psi|} ||M_E y||_{M_B^{-1}}. \tag{3.26}$$

Proof: Let us define the vector $g := U_1 \mathbf{c}_1 + U_2 \mathbf{c}_2 + U_3 \mathbf{c}_3$ where

$$\mathbf{c}_1 = -\left[1, \lambda - \sigma, \ldots, (\lambda - \sigma)^{\psi-1}\right]^T, \quad \mathbf{c}_2 = \left[\lambda - \sigma, \ldots, (\lambda - \sigma)^\psi\right]^T,$$
$$\mathbf{c}_3 = \left[\frac{\gamma_1(\lambda - \sigma)^{\psi+1} - \epsilon_1(\lambda - \sigma)^\psi}{(\delta_1 - \lambda)(\delta_1 - \sigma)^\psi}, \ldots, \frac{\gamma_\kappa(\lambda - \sigma)^{\psi+1} - \epsilon_\kappa(\lambda - \sigma)^\psi}{(\delta_\kappa - \lambda)(\delta_\kappa - \sigma)^\psi}\right]^T.$$

We then have

$$\begin{aligned} u-g =& \sum_{\ell=1}^{\ell=d} \frac{-\epsilon_\ell(\lambda-\sigma)^\psi}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} + \sum_{\ell=1}^{\ell=d} \frac{\gamma_\ell(\lambda-\sigma)^{\psi+1}}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} \\ & - \sum_{\ell=1}^{\ell=\kappa} \frac{-\epsilon_\ell(\lambda-\sigma)^\psi}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} - \sum_{\ell=1}^{\ell=\kappa} \frac{\gamma_\ell(\lambda-\sigma)^{\psi+1}}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} \\ =& \sum_{\ell=\kappa+1}^{\ell=d} \frac{-\epsilon_\ell(\lambda-\sigma)^\psi}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} + \sum_{\ell=\kappa+1}^{\ell=d} \frac{\gamma_\ell(\lambda-\sigma)^{\psi+1}}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)}, \end{aligned} \tag{3.27}$$

and taking the M_B-norm of $u-g$ finally leads to

$$\begin{aligned} \|u-g\|_{M_B} \leq & \left\| \sum_{\ell=\kappa+1}^{\ell=d} \frac{-\epsilon_\ell(\lambda-\sigma)^\psi}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} \right\|_{M_B} + \left\| \sum_{\ell=\kappa+1}^{\ell=d} \frac{\gamma_\ell(\lambda-\sigma)^{\psi+1}}{(\delta_\ell-\lambda)(\delta_\ell-\sigma)^\psi} v^{(\ell)} \right\|_{M_B} \\ \leq & \max_{\ell>\kappa} \frac{|\lambda-\sigma|^\psi}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|} ||E_\sigma y||_{M_B^{-1}} + \max_{\ell>\kappa} \frac{|\lambda-\sigma|^{\psi+1}}{|(\lambda-\delta_\ell)(\sigma-\delta_\ell)^\psi|} ||M_E y||_{M_B^{-1}}, \end{aligned}$$

Recalling that $\hat{u} := \arg\min_{g\in\mathcal{U}} \|u-g\|_{M_B}$ concludes the proof. □

3.3 The RF-DDES algorithm

RF-DDES starts by calling a graph partitioner to partition the graph of $|A|+|M|$ into p subdomains and reorders the matrix pencil (A, M) as in (2.9). RF-DDES then proceeds to the computation of those eigenvectors associated with the $nev_B^{(j)}$ smallest (in magnitude) eigenvalues of each matrix pencil $\left(B_j - \sigma M_B^{(j)}, M_B^{(j)}\right)$, and stores these eigenvectors in $V_j \in \mathbb{R}^{d_j \times nev_B^{(j)}}$, $j = 1, \ldots, p$. As our current implementation stands, these eigenvectors are computed by Lanczos combined with shift-and-invert [75]. While we do not consider any special mechanisms to set the value of $nev_B^{(j)}$, it is possible to adapt the work in [80]. The next step of RF-DDES is to call Algorithm 3.1.1 and approximate **span** $\left(y^{(1)}, \ldots, y^{(nev)}\right)$ by **range**$\{Q\}$, where Q denotes the orthonormal matrix returned by Algorithm 3.1.1. RF-DDES then builds an approximation subspace as described in Section 3.3.1 and performs a Rayleigh-Ritz (RR) projection to extract approximate eigenpairs of (A, M). The complete procedure is shown in Algorithm 3.3.1.

Algorithm **3.3.1** *RF-DDES*

```
0.  Input: $A,\ M,\ \alpha,\ \beta,\ \sigma,\ p,\ \{\omega_\ell,\zeta_\ell\}_{\ell=1,\ldots,N_c},\ \left\{nev_B^{(j)}\right\}_{j=1,\ldots,p},\ \psi$
1.  Reorder $A$ and $M$ as in (2.9)
2.  For $j = 1,\ldots,p$:
3.      Compute the eigenvectors associated with the $nev_B^{(j)}$ smallest
        (in magnitude) eigenvalues of $\left(B_\sigma^{(j)}, M_B^{(j)}\right)$ and store them in $V_j$
4.  End
5.  Compute $Q$ by Algorithm_3.1.1
6.  Form $Z$ as in (3.30)
7.  Solve the Rayleigh-Ritz eigenvalue problem: $Z^T AZG = Z^T MZG\hat{\Lambda}$
8.  If eigenvectors were also sought, permute the entries of each
    approximate eigenvector back to their original ordering
```

The Rayleigh-Ritz eigenvalue problem at Step (7) of RF-DDES can be solved either by a shift-and-invert procedure or by the appropriate routine in LAPACK [36].

3.3.1 The projection matrix Z

Let matrix Q returned by Algorithm 3.1.1 be written in its distributed form among the p subdomains,

$$Q = \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_p \end{pmatrix}, \tag{3.28}$$

where $Q_j \in \mathbb{R}^{s_i \times \mu},\ j = 1,\ldots,p$, is local to the jth subdmonain and $\mu \in \mathbb{N}^*$ denotes the total number of iterations performed by Algorithm 3.1.1. By defining

$$\begin{aligned} B_\sigma^{(j)} &= B_j - \sigma M_B^{(j)}, \\ \Phi_\sigma^{(j)} &= -\left(\hat{E}_j - \sigma \hat{M}_E^{(j)}\right) Q_j, \\ \Psi^{(j)} &= \hat{M}_E^{(j)} Q_j, \end{aligned} \tag{3.29}$$

the Rayleigh-Ritz projection matrix Z in RF-DDES can be written as:

$$Z = \begin{pmatrix} V_1 & & & & \Sigma_1^{(\psi)} & \Gamma_1^{(\psi)} \\ & V_2 & & & \Sigma_2^{(\psi)} & \Gamma_2^{(\psi)} \\ & & \ddots & & \vdots & \vdots \\ & & & V_p & \Sigma_p^{(\psi)} & \Gamma_p^{(\psi)} \\ & & & & [Q, 0_{s,(\psi-1)\mu}] & \end{pmatrix}, \tag{3.30}$$

where $0_{\chi,\zeta}$ denotes a zero matrix of size $\chi \times \zeta$, and

$$\begin{aligned} \Sigma_j^{(\psi)} &= \left[(B_\sigma^{(j)})^{-1}\Phi_\sigma^{(j)}, (B_\sigma^{(j)})^{-1}M_B^{(j)}(B_\sigma^{(j)})^{-1}\Phi_\sigma^{(j)}, \ldots, (B_\sigma^{(j)})^{-1}\left(M_B^{(j)}(B_\sigma^{(j)})^{-1}\right)^{\psi-1}\Phi_\sigma^{(j)}\right], \\ \Gamma_j^{(\psi)} &= \left[(B_\sigma^{(j)})^{-1}\Psi^{(j)}, (B_\sigma^{(j)})^{-1}M_B^{(j)}(B_\sigma^{(j)})^{-1}\Psi^{(j)}, \ldots, (B_\sigma^{(j)})^{-1}\left(M_B^{(j)}(B_\sigma^{(j)})^{-1}\right)^{\psi-1}\Psi^{(j)}\right]. \end{aligned} \tag{3.31}$$

When M_E is a nonzero matrix, the size of matrix Z is $n \times (\kappa + 2\psi\mu)$, $\kappa = \sum_{j=1}^{p} nev_B^{(j)}$. When $M_E \equiv 0_{d,s}$ (as is the case for example when M is the identity matrix) the size of Z reduces to $n \times (\kappa + \psi\mu)$ since $\Gamma_j^{(\psi)} \equiv 0_{d_j,\psi\mu}$. The total memory overhead of RF-DDES associated with the jth subdomain is then at most that of storing $d_j(nev_B^{(j)} + 2\psi\mu) + s_i\mu$ floating point scalars.

3.3.2 Comparison with AMLS

Both RF-DDES and AMLS exploit the domain decomposition framework discussed in Section 2.5. However, the two methods differ in a few points.

In contrast to RF-DDES which exploits Algorithm 3.1.1, AMLS approximates the part of the solution associated with the interface variables of (A, M) by solving a generalized eigenvalue problem stemming by a first-order approximation of the nonlinear eigenvalue problem in (2.14). More specifically, AMLS approximates **span** $\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$ by the **span** of the eigenvectors associated with a few of the eigenvalues of smallest magnitude of the SPD pencil $(S(\sigma), -S'(\sigma))$, where σ is some real shift and $S'(\sigma)$ denotes the derivative of the Schur complement matrix at σ. As such, only the **span** of those vectors $y^{(i)}$ for which λ_i lies sufficiently close to σ can be captured very accurately. In the standard AMLS method the shift σ is zero. In contrast, RF-DDES can capture all of **span** $\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$ to high accuracy regardless of where λ_i is located inside the interval $[\alpha, \beta]$ of interest.

Another difference between RF-DDES and AMLS concerns the way in which the two schemes approximate $\mathbf{span}\left(\left[u^{(1)}, \ldots, u^{(nev)}\right]\right)$. As can be easily verified, AMLS is similar to RF-DDES with the choice $\psi = 1$ [70]. While it is possible to combine AMLS with higher values of ψ, this might not always lead to a significant increase in the accuracy of the approximate eigenpairs of (A, M) due to the inaccuracies in the approximation of $\mathbf{span}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$. In contrast, because RF-DDES can compute a good approximation to the entire space $\mathbf{span}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$, the accuracy of the approximate eigenpairs of (A, M) can be improved by simply increasing ψ and/or $nev_B^{(j)}$ and repeating the Rayleigh-Ritz projection.

From a computational viewpoint, AMLS computes the factorization of $S(\sigma)$ in real arithmetic and proceeds to compute a partial solution of the eigenvalue problem with the matrix pencil $(S(\sigma), -S'(\sigma))$. When the accuracy provided by AMLS is deemed accurate enough without computing a large number of eigenvectors of $(S(\sigma), -S'(\sigma))$, AMLS can be potentially faster than RF-DDES. Note, however, that even in this scenario RF-DDES might also be an appealing approach due to its good parallelization properties, e.g. when a distributed memory environment is available.

3.4 Experiments

In this section we present numerical experiments performed in serial and distributed memory computing environments. The RF-KRYLOV and RF-DDES schemes were written in C/C++ and built on top of the PETSc and MKL scientific libraries. The source files were compiled with the Intel MPI compiler `mpiicpc`, using the -O3 optimization level. For RF-DDES, the computational domain was partitioned to p non-overlapping subdomains by the METIS graph partitioner, and each subdomain was then assigned to a distinct processor group. Communication among different processor groups was achieved by means of the MPI standard. The linear system solutions with matrices $A - \zeta_1 M, \ldots, A - \zeta_{N_c} M$ and $S(\zeta_1), \ldots, S(\zeta_{N_c})$ were computed by the Multifrontal Massively Parallel Sparse Direct Solver (MUMPS) [81], while those with the block-diagonal matrices $B_{\zeta_1}, \ldots, B_{\zeta_{N_c}}$, and B_σ by MKL PARDISO.

The quadrature node-weight pairs $\{\omega_\ell, \zeta_\ell\},\ \ell = 1, \ldots, N_c$ were computed by the Midpoint quadrature rule of order $2N_c$, retaining only the N_c quadrature nodes (and associated weights) with positive imaginary part. Unless stated otherwise, the default values used throughout the experiments were $p = 2$, $N_c = 2$, and $\sigma = 0$, while we set $nev_B^{(1)} = \ldots = nev_B^{(p)} = nev_B$. The

Table 3.1: n: size of A and M, $nnz(X)$: number of nonzero entries in matrix X.

#	Mat. pencil	n	$nnz(A)/n$	$nnz(M)/n$	$[\alpha, \beta]$	nev
1.	bcsst24	3,562	44.89	1.00	[0, 352.55]	100
2.	Kuu/Muu	7,102	47.90	23.95	[0, 934.30]	100
3.	FDmesh1	24,000	4.97	1.00	[0, 0.0568]	100
4.	bcsst39	46,772	44.05	1.00	[-11.76, 3915.7]	100
5.	qa8fk/qa8fm	66,127	25.11	25.11	[0, 15.530]	100

Table 3.2: Maximum relative errors of the approximation of the lowest $nev = 100$ eigenvalues returned by RF-DDES for the matrix pencils listed in Table 8.1.

	$nev_B = 50$			$nev_B = 100$			$nev_B = 200$		
Mat. pencil	$\psi = 1$	$\psi = 2$	$\psi = 3$	$\psi = 1$	$\psi = 2$	$\psi = 3$	$\psi = 1$	$\psi = 2$	$\psi = 3$
bcsst24	2.2e-2	1.8e-3	3.7e-5	9.2e-3	1.5e-5	1.4e-7	7.2e-4	2.1e-8	4.1e-11
Kuu/Muu	2.4e-2	5.8e-3	7.5e-4	5.5e-3	6.6e-5	1.5e-6	1.7e-3	2.0e-6	2.3e-8
FDmesh1	1.8e-2	5.8e-3	5.2e-3	6.8e-3	2.2e-4	5.5e-6	2.3e-3	1.3e-5	6.6e-8
bcsst39	2.5e-2	1.1e-2	8.6e-3	1.2e-2	7.8e-5	2.3e-6	4.7e-3	4.4e-6	5.9e-7
qa8fk/qa8fm	1.6e-1	9.0e-2	2.0e-2	7.7e-2	5.6e-3	1.4e-4	5.9e-2	4.4e-4	3.4e-6

stopping criterion in Algorithm 3.1.1, was set to tol $= 1e$-6. All computations were carried out in 64-bit (double) precision, and all wall-clock times reported throughout the rest of this section will be listed in seconds.

3.4.1 Numerical illustration of RF-DDES

We tested RF-DDES on the matrix pencils listed in Table 8.1. For each pencil, the interval of interest $[\alpha, \beta]$ was chosen so that $nev = 100$. Matrix pencils 1), 2), 4), and 5) can be found in the SuiteSparse matrix collection. Matrix pencil 3) was obtained by a discretization of a differential eigenvalue problem associated with a membrane on the unit square with Dirichlet boundary conditions on all four edges using Finite Difference, and is of the standard form, i.e., $M = I$, where I denotes the identity matrix of appropriate size.

Table 3.2 lists the maximum (worst-case) relative error among all nev approximate eigenvalues returned by RF-DDES. In agreement with the discussion in Section 3.2, exploiting higher values of ψ and/or nev_B leads to enhanced accuracy. Figure 3.2 plots the relative errors among all nev approximate eigenvalues (not just the worst-case errors) for the largest matrix pencil listed in Table 8.1. Since all eigenvalues of "qa8fk/qa8fm" are positive, and $\sigma = 0$, we expect the algebraically smallest eigenvalues of (A, M) to be approximated more accurately. Increasing

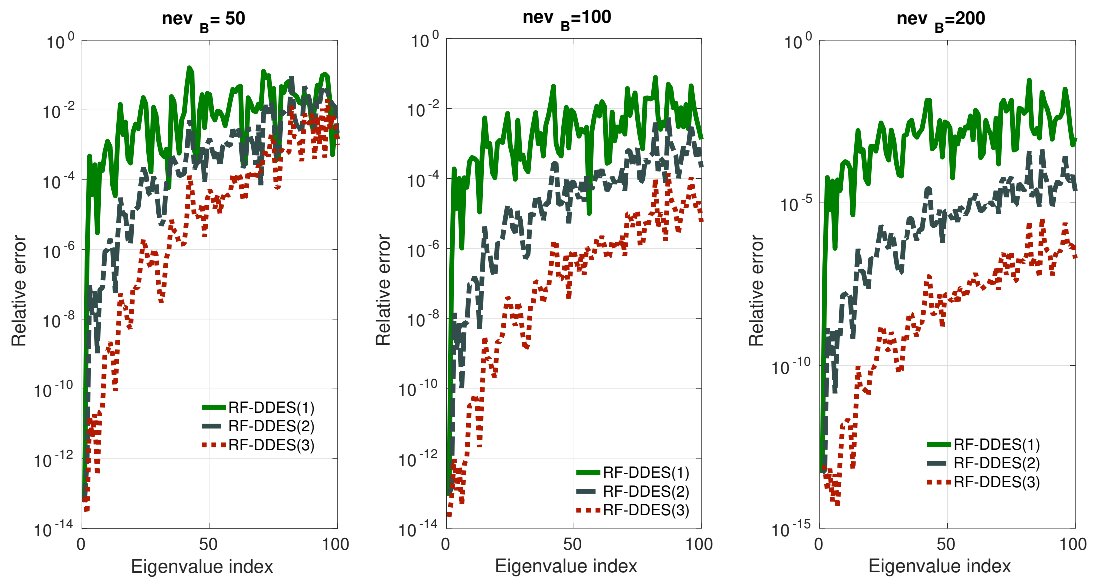


Figure 3.2: Relative errors of the approximation of the lowest $nev = 100$ eigenvalues for the "qa8fk/qafm" matrix pencil. Left: $nev_B = 50$. Center: $nev_B = 100$. Right: $nev_B = 200$.

Table 3.3: Number of iterations performed by Algorithm 3.1.1 for the matrix pencils listed in Table 8.1. 's' denotes the number of interface variables.

Mat. pencil	s	s/n	$N_c = 2$	$N_c = 4$	$N_c = 8$	$N_c = 12$	$N_c = 16$
bcsst24	449	0.12	164	133	111	106	104
Kuu/Muu	720	0.10	116	74	66	66	66
FDmesh1	300	0.01	58	40	36	35	34
bcsst39	475	0.01	139	93	75	73	72
qa8fk/qa8fm	1272	0.01	221	132	89	86	86

the value of ψ and/or nev_B then will mainly improve the accuracy in the approximation of those eigenvalues λ located farther away from σ. A similar pattern was also observed for the rest of the matrix pencils listed in Table 8.1.

Table 3.3 lists the number of iterations performed by Algorithm 3.1.1 as the value of N_c increases. Observe that for matrix pencils 2), 3), 4) and 5) this number can be less than nev (recall the "early termination" property discussed in Proposition 3.1.1), even for values of N_c as low as $N_c = 2$. Moreover, Figure 3.3 plots the 150 leading[3] singular values of matrix $\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1} \right\}$ for matrix pencils "bcsst24" and "Kuu/Muu" as $N_c = 4$, 8, 12 and

[3]After normalization by the spectral norm

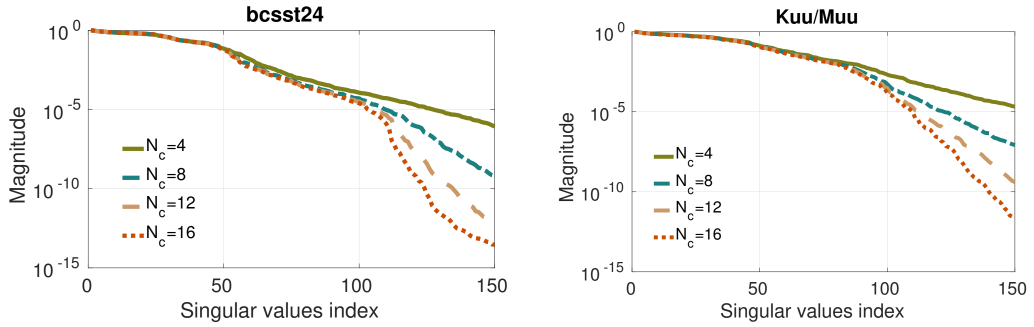


Figure 3.3: The 150 leading singular values of $\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1} \right\}$ for the matrix pencils "bcsst24" and "Kuu/Muu".

$N_c = 16$. In agreement with the discussion in Section 3.1.2, as the value of N_c increases the magnitude of the trailing $s - \textbf{rank}\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$ singular values approaches zero.

Except the value of N_c, the number of subdomains p might also affect the number of iterations performed by Algorithm 3.1.1. Figure 3.4 shows the total number of iterations performed by Algorithm 3.1.1 when applied to matrix "FDmesh1" for $p = 2,\ 4,\ 8$ and $p = 16$ subdomains. For each different value of p we considered $N_c = 2,\ 4,\ 8,\ 12,$ and $N_c = 16$ quadrature nodes. The interval $[\alpha, \beta]$ was set so that it included only eigenvalues $\lambda_1, \ldots, \lambda_{200}$ ($nev = 200$). Observe that higher values of p might lead to an increase in the number of iterations performed by Algorithm 3.1.1. For example, when the number of subdomains is set to $p = 2$ or $p = 4$, setting $N_c = 2$ is sufficient for Algorithm 3.1.1 to terminate in less than nev iterations. On the other hand, when $p \geq 8$, we need at least $N_c \geq 4$ if a similar number of iterations is to be performed. This potential increase in the number of iterations performed by Algorithm 3.1.1 for larger values of p is a consequence of the fact that the columns of matrix $Y = \left[y^{(1)}, \ldots, y^{(nev)}\right]$ now lie in a higher-dimensional subspace. This might not only increase the rank of Y, but also affect the decay of the singular values of $\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1} \right\}$. This can be seen more clearly in Figure 3.5 where we plot the leading 250 singular values of $\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1} \right\}$ of the problem in Figure 3.4 for two different values of p, $p = 2$ and $p = 8$. Notice how the leading singular values decay more slowly for the case $p = 8$. Similar results were observed for different values of p and for all matrix pencils listed in Table 8.1.

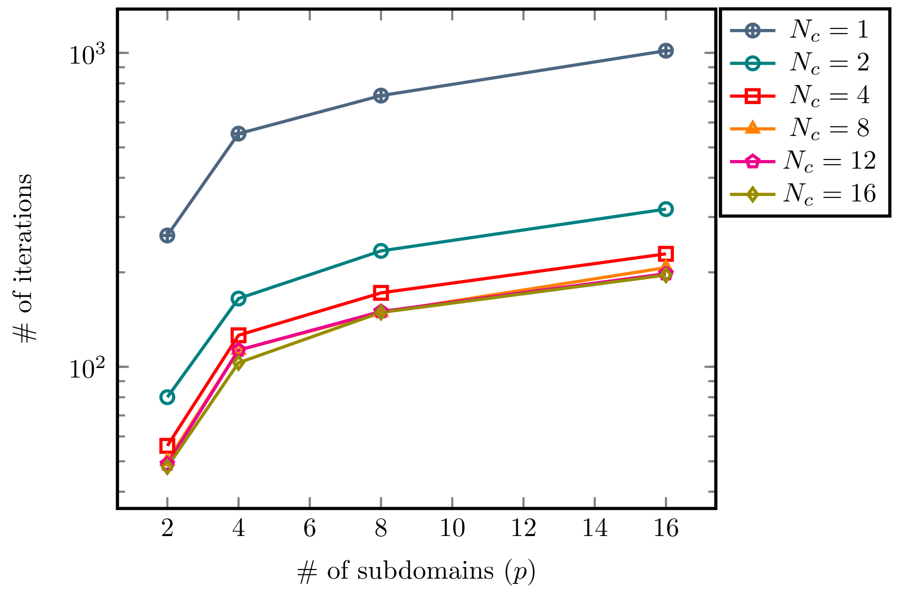


Figure 3.4: Total number of iterations performed by Algorithm 3.1.1 when applied to matrix "FDmesh1" with $[\alpha, \beta] = [\lambda_1, \lambda_{200}]$. Results reported are for all different combinations of $p = 2,\ 4,\ 8$ and $p = 16$, and $N_c = 1,\ 2,\ 4,\ 8$ and $N_c = 16$.

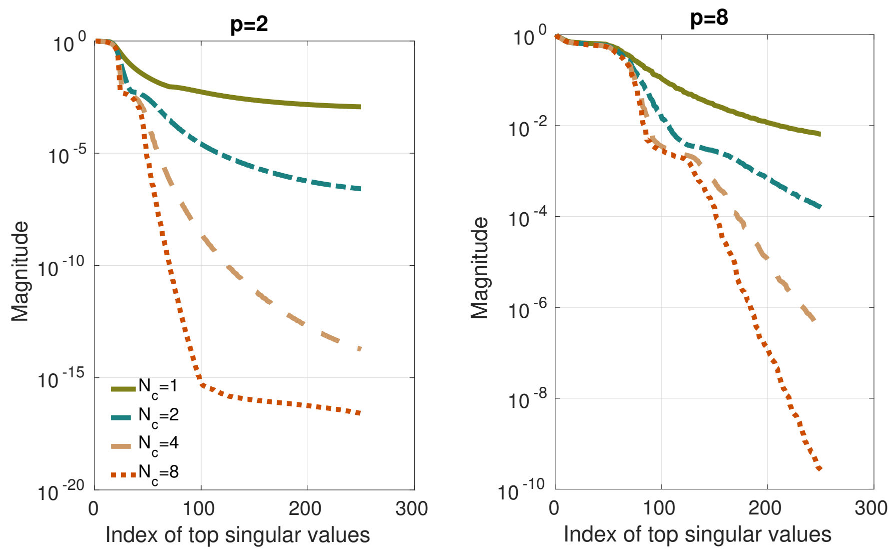


Figure 3.5: The leading 250 singular values of $\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1} \right\}$ for the same problem as in Figure 3.4. Left: $p = 2$. Right: $p = 8$. For both values of p we set $N_c = 1,\ 2,\ 4$, and $N_c = 8$.

Table 3.4: n: size of A, $nnz(A)$: number of nonzero entries in matrix A. s_2 and s_4 denote the number of interface variables when $p = 2$ and $p = 4$, respectively.

#	Matrix	n	$nnz(A)/n$	s_2	s_4	$[\lambda_1, \lambda_{101}, \lambda_{201}, \lambda_{300}]$
1.	shipsec8	114,919	28.74	4,534	9,001	[3.2e-2, 1.14e-1, 1.57e-2, 0.20]
2.	boneS01	172,224	32.03	10,018	20,451	[2.8e-3, 24.60, 45.42, 64.43]
3.	FDmesh2	250,000	4.99	1,098	2,218	[7.8e-5, 5.7e-3, 1.08e-2, 1.6e-2]
4.	FDmesh3	1,000,000	4.99	2,196	4,407	[1.97e-5, 1.4e-3, 2.7e-3, 4.0e-3]

3.4.2 A comparison of RF-DDES and RF-KRYLOV in distributed computing environments

In this section we compare the performance of RF-KRYLOV and RF-DDES on distributed computing environments for the matrices listed in Table 3.4. All eigenvalue problems in this section are of the form (A, I), i.e., standard eigenvalue problems. Matrices “boneS01” and “shipsec8” can be found in the SuiteSparse matrix collection. Similarly to “FDmesh1”, matrices “FDmesh2” and “FDmesh3” were generated by a Finite Difference discretization of the Laplacian operator on the unit plane using Dirichlet boundary conditions and two different mesh sizes so that $n = 250,000$ (“FDmesh2”) and $n = 1,000,000$ (“FDmesh3”).

Throughout the rest of this section we will keep $N_c = 2$ fixed, since this option was found the best both for RF-KRYLOV and RF-DDES.

Wall-clock time comparisons

We now consider the wall-clock times achieved by RF-KRYLOV and RF-DDES when executing both schemes on $\tau = 2,\ 4,\ 8,\ 16$ and $\tau = 32$ computer cores. For RF-KRYLOV, the value of τ will denote the number of single-threaded MPI processes. For RF-DDES, the number of MPI processes will be equal to the number of subdomains, p, and each MPI process will utilize τ/p compute threads. Unless mentioned otherwise, we will assume that RF-DDES is executed with $\psi = 3$ and $nev_B = 100$.

Table 3.7 lists the wall-clock time required by RF-KRYLOV and RF-DDES to approximate the $nev = 100$, $nev = 200$, and $nev = 300$ algebraically smallest eigenvalues of the matrices listed in Table 3.4. For RF-DDES we considered two different values of p; $p = 2$ and $p = 4$. Overall, RF-DDES was found to be faster than RF-KRYLOV, especially for higher values of nev. Table 3.5 lists the number of iterations performed by RF-KRYLOV and Algorithm 3.1.1 in

Table 3.5: Number of iterations performed by RF-KRYLOV (denoted as RFK) and Algorithm 3.1.1 in RF-DDES (denoted by RFD(2) and RFD(4), with the number inside the parentheses denoting the value of p) for the matrix pencils listed in Table 3.4. The convergence criterion in both RF-KRYLOV and Algorithm 3.1.1 was tested every ten iterations.

	$nev = 100$			$nev = 200$			$nev = 300$		
Matrix	RFK	RFD(2)	RFD(4)	RFK	RFD(2)	RFD(4)	RFK	RFD(2)	RFD(4)
shipsec8	280	170	180	500	180	280	720	190	290
boneS01	240	350	410	480	520	600	620	640	740
FDmesh2	200	100	170	450	130	230	680	160	270
FDmesh3	280	150	230	460	180	290	690	200	380

Table 3.6: Maximum relative error of the approximate eigenvalues returned by RF-DDES for the matrix pencils listed in Table 3.4.

	$nev = 100$			$nev = 200$			$nev = 300$		
Matrix	nev_B=25	nev_B=50	nev_B=100	nev_B=25	nev_B=50	nev_B=100	nev_B=25	nev_B=50	nev_B=100
shipsec8	1.4e-3	2.2e-5	2.4e-6	3.4e-3	1.9e-3	1.3e-5	4.2e-3	1.9e-3	5.6e-4
boneS01	5.2e-3	7.1e-4	2.2e-4	3.8e-3	5.9e-4	4.1e-4	3.4e-3	9.1e-4	5.1e-4
FDmesh2	4.0e-5	2.5e-6	1.9e-7	3.5e-4	9.6e-5	2.6e-6	3.2e-4	2.0e-4	2.6e-5
FDmesh3	6.2e-5	8.5e-6	4.3e-6	6.3e-4	1.1e-4	3.1e-5	9.1e-4	5.3e-4	5.3e-5

RF-DDES. For all matrices but "boneS01", Algorithm 3.1.1 required fewer iterations than RF-KRYLOV. Table 3.6 lists the maximum relative error of the approximate eigenvalues returned by RF-DDES when $p = 4$. The decrease in the accuracy of RF-DDES as nev increases is due the fact that nev_B remains constant. More specifically, an increase in the value of nev should be also accompanied by an increase in the value of nev_B, if the same level of maximum relative error need be retained. On the other hand, RF-KRYLOV computed all nev eigenpairs within an accuracy which was of the order $O(10^{-13})$ or lower for all four matrices considered.

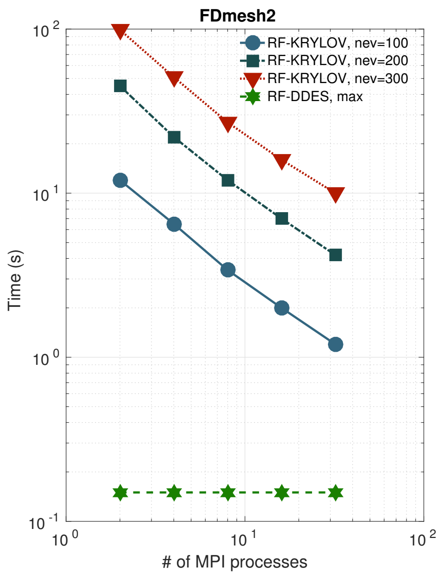
Table 3.8 lists the amount of time spent on the triangular substitutions required to apply the rational filter in RF-KRYLOV, as well as the amount of time spent on forming and factorizing the Schur complement matrices and applying the rational filter in RF-DDES. For the values of nev tested in this section, these procedures were found to be the computationally most expensive ones. Figure 3.6 plots the total amount of time spent on orthonormalization by RF-KRYLOV and RF-DDES when applied to matrices "FDmesh2" and "FDmesh3". For RF-KRYLOV, we report results for all different values of nev and number of MPI processes. For RF-DDES we only report the highest times across all different values of nev, τ and p. RF-DDES was

Table 3.7: Wall-clock times of RF-KRYLOV and RF-DDES using $\tau = 2,\ 4,\ 8,\ 16$ and $\tau = 32$ computational cores. RFD(2) and RFD(4) denote RF-DDES with $p = 2$ and $p = 4$ subdomains, respectively.

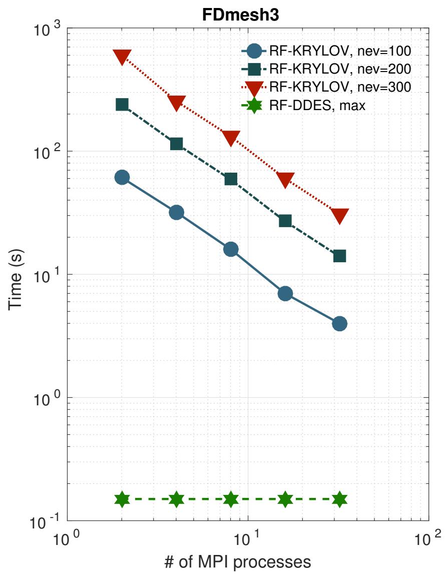
	$nev = 100$			$nev = 200$			$nev = 300$		
Matrix	RFK	RFD(2)	RFD(4)	RFK	RFD(2)	RFD(4)	RFK	RFD(2)	RFD(4)
shipsec8($\tau = 2$)	114	195	-	195	207	-	279	213	-
($\tau = 4$)	76	129	93	123	133	103	168	139	107
($\tau = 8$)	65	74	56	90	75	62	127	79	68
($\tau = 16$)	40	51	36	66	55	41	92	57	45
($\tau = 32$)	40	36	28	62	41	30	75	43	34
boneS01($\tau = 2$)	94	292	-	194	356	-	260	424	-
($\tau = 4$)	68	182	162	131	230	213	179	277	260
($\tau = 8$)	49	115	113	94	148	152	121	180	187
($\tau = 16$)	44	86	82	80	112	109	93	137	132
($\tau = 32$)	51	66	60	74	86	71	89	105	79
FDmesh2($\tau = 2$)	241	85	-	480	99	-	731	116	-
($\tau = 4$)	159	34	63	305	37	78	473	43	85
($\tau = 8$)	126	22	23	228	24	27	358	27	31
($\tau = 16$)	89	16	15	171	17	18	256	20	21
($\tau = 32$)	51	12	12	94	13	14	138	15	20
FDmesh3($\tau = 2$)	1021	446	-	2062	502	-	3328	564	-
($\tau = 4$)	718	201	281	1281	217	338	1844	237	362
($\tau = 8$)	423	119	111	825	132	126	1250	143	141
($\tau = 16$)	355	70	66	684	77	81	1038	88	93
($\tau = 32$)	177	47	49	343	51	58	706	62	82

Table 3.8: Time elapsed to apply the rational filter in RF-KRYLOV and RF-DDES using $\tau = 2,\ 4,\ 8,\ 16$ and $\tau = 32$ computer cores. RFD(2) and RFD(4) denote RF-DDES with $p = 2$ and $p = 4$ subdomains, respectively. For RF-KRYLOV the times listed also include the amount of time spent on factorizing matrices $A - \zeta_\ell M,\ \ell = 1, \ldots, N_c$. For RF-DDES, the times listed also include the amount of time spent in forming and factorizing matrices $S(\zeta_\ell),\ \ell = 1, \ldots, N_c$.

	$nev = 100$			$nev = 200$			$nev = 300$		
Matrix	RFK	RFD(2)	RFD(4)	RFK	RFD(2)	RFD(4)	RFK	RFD(2)	RFD(4)
shipsec8($\tau = 2$)	104	153	-	166	155	-	222	157	-
($\tau = 4$)	71	93	75	107	96	80	137	96	82
($\tau = 8$)	62	49	43	82	50	45	110	51	47
($\tau = 16$)	38	32	26	61	33	28	83	34	20
($\tau = 32$)	39	21	19	59	23	20	68	24	22
boneS01($\tau = 2$)	86	219	-	172	256	-	202	291	-
($\tau = 4$)	64	125	128	119	152	168	150	178	199
($\tau = 8$)	46	77	88	84	95	117	104	112	140
($\tau = 16$)	43	56	62	75	70	85	86	84	102
($\tau = 32$)	50	42	44	72	51	60	82	63	61
FDmesh2($\tau = 2$)	227	52	-	432	59	-	631	65	-
($\tau = 4$)	152	22	36	287	24	42	426	26	45
($\tau = 8$)	122	13	14	215	14	16	335	15	18
($\tau = 16$)	85	9	8	164	10	10	242	11	11
($\tau = 32$)	50	6	6	90	7	8	127	8	10
FDmesh3($\tau = 2$)	960	320	-	1817	341	-	2717	359	-
($\tau = 4$)	684	158	174	1162	164	192	1582	170	201
($\tau = 8$)	406	88	76	764	91	82	1114	94	88
($\tau = 16$)	347	45	43	656	48	49	976	51	52
($\tau = 32$)	173	28	26	328	28	32	674	31	41

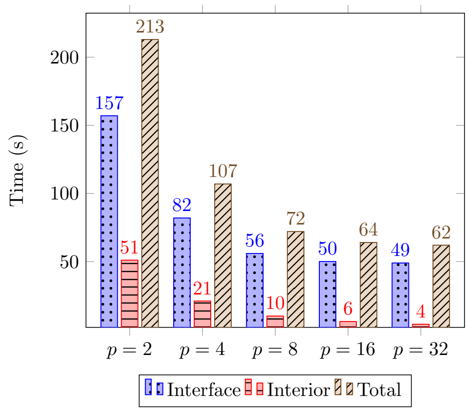


(a) FDmesh2 ($n = 250,000$).



(b) FDmesh3 ($n = 1,000,000$).

Figure 3.6: Time spent on orthonormalization in RF-KRYLOV and RF-DDES when computing the $nev = 100,\ 200$ and $nev = 300$ algebraically smallest eigenvalues and associated eigenvectors of matrices “FDmesh2” and “FDmesh3”.



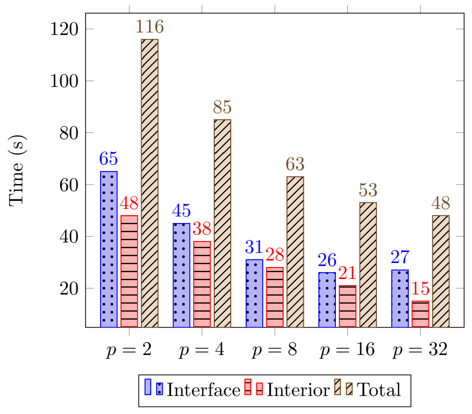


Figure 3.7: Amount of time required to apply the rational filter ("Interface"), form the subspace associated with the interior variables ("Interior"), and total wall-clock time ("Total") obtained by an MPI-only execution of RF-DDES for the case where $nev = 300$. Left: "shipsec8". Right: "FDmesh2".

found to spend a considerably smaller amount of time on orthonormalization than what RF-KRYLOV did, mainly because s was much smaller than n (the values of s for $p = 2$ and $p = 4$ can be found in Table 3.4). Indeed, if both RF-KRYLOV and Algorithm 3.1.1 in RF-DDES perform a similar number of iterations, we expect the former to spend roughly n/s more time on orthonormalization compared to RF-DDES.

Figure 8.2 lists the wall-clock times achieved by an MPI-only implementation of RF-DDES, i.e., p still denotes the number of subdomains but each subdomain is handled by a separate (single-threaded) MPI process, for matrices "shipsec8" and "FDmesh2". In all cases, the MPI-only implementation of RF-DDES led to higher wall-clock times than those achieved by the hybrid implementations discussed in Tables 3.7 and 3.8. More specifically, while the MPI-only implementation reduced the cost to construct and factorize the distributed $S(\zeta_\ell)$ matrices, the application of the rational filter in Algorithm 3.1.1 became more expensive due to: a) each linear system solution with $S(\zeta_\ell)$ required more time, b) a larger number of iterations had to be performed as p increased (Algorithm 3.1.1 required 190, 290, 300, 340 and 370 iterations for "shipsec8", and 160, 270, 320, 350, and 410 iterations for "FDmesh2" as $p = 2,\ 4,\ 8,\ 16$ and $p = 32$, respectively). Note that the scalability of the MPI-only version of RF-DDES for increasing values of p is limited by the scalability of the linear system used, which in this particular case was not high. This suggests that reducing p and applying RF-DDES recursively to the local pencils $\left(B_\sigma^{(j)}, M_B^{(j)}\right),\ j = 1, \ldots, p$ might be the best combination when only distributed memory parallelism is considered.

3.5 Summary

In this chapter we proposed a rational filtering domain decomposition approach (termed as RF-DDES) for the computation of all eigenpairs of real symmetric pencils located inside a given interval $[\alpha, \beta]$. In contrast with rational filtering Krylov approaches, RF-DDES applies the rational filter only to the interface variables. This has several advantages. First, orthogonalization is performed on vectors whose length is equal to the number of interface variables only. Second, the Krylov projection method may converge in fewer than nev iterations. Third, it is possible to solve the original eigenvalue problem associated with the interior variables in real arithmetic and with trivial parallelism with respect to each subdomain. RF-DDES can be considerably faster than rational filtering Krylov approaches, especially when nev is large.

Chapter 4

Domain decomposition from a contour integral viewpoint

In this chapter we present[1] two domain decomposition eigenvalue solvers which take advantage of the contour integral representation of the spectral projector[2]

$$
\begin{aligned}
P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} &= \frac{1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}} (\zeta M - A)^{-1} M d\zeta \\
&= \sum_{i=1}^{nev} x^{(i)} \left(x^{(i)}\right)^T M,
\end{aligned}
\tag{4.1}
$$

where $\Gamma_{[\alpha,\beta]}$ is a counter-clockwise oriented smooth Jordan curve, e.g. a circle, that encloses only the nev eigenvalues of (A, M) located inside the interval $[\alpha, \beta]$.

The first scheme proposed can be seen as an extension of domain decomposition linear system solvers in the framework of contour integral techniques, such as FEAST, for the solution of symmetric eigenvalue problems. The second scheme proposed shares similarities with the work of Beyn in [82] and approximates only certain parts of the contour integral of the matrix resolvent $(\zeta M - A)^{-1}$. This approach leads to a numerical scheme which can be computationally more efficient than following the standard approach of numerically integrating the entire matrix

[1]This is joint work with James Kestyn and Eric Polizzi (University of Massachusetts, Amherst) and Yousef Saad (University of Minnesota, Twin Cities)

[2]Background material in contour integral spectral projectors is presented in Sections 2.3.1 and 2.3.2 of the present dissertation

resolvent, especially when the linear system solutions associated with the interior variables of the subdomains are relatively expensive. Numerical experiments performed in distributed memory environments using the MPI programming model are also reported.

The organization of this chapter is as follows. Section 4.1 presents a scheme that is based on a domain decomposition of the contour integral of the matrix resolvent. Section 4.2 presents a scheme that integrates the contour integral of the matrix resolvent only partially. Section 4.3 focuses on the solution of the linear systems during the numerical integration phase and presents details on the implementation of domain decomposition-based preconditioners in distributed memory environments. Section 4.4 presents computational experiments. Finally, in Section 4.5 we state our concluding remarks.

4.1 Full integration of the matrix resolvent

We start by noticing that $(\zeta M - A)^{-1} = -(A - \zeta M)^{-1}$ and thus

$$(\zeta M - A)^{-1} = \begin{pmatrix} -\left[B_\zeta^{-1} + F(\zeta)S(\zeta)^{-1}F(\zeta)^T\right] & F(\zeta)S(\zeta)^{-1} \\ S(\zeta)^{-1}F(\zeta)^T & -S(\zeta)^{-1} \end{pmatrix}, \tag{4.2}$$

where

$$F(\zeta) = B_\zeta^{-1}E_\zeta. \tag{4.3}$$

The spectral projector $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}$ in (4.1) can be then written in a 2×2 block form by integrating each block of $(\zeta M - A)^{-1}$ separately:

$$\begin{aligned} P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} &= \frac{1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}} (\zeta M - A)^{-1} M d\zeta \\ &= \begin{pmatrix} \mathcal{H} & \mathcal{W} \\ \mathcal{W}^T & \mathcal{G} \end{pmatrix} M, \end{aligned} \tag{4.4}$$

with

$$\begin{aligned} \mathcal{H} &= \frac{-1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}[B_\zeta^{-1}+F(\zeta)S(\zeta)^{-1}F(\zeta)^T]d\zeta \\ \mathcal{G} &= \frac{-1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}S(\zeta)^{-1}d\zeta \\ \mathcal{W} &= \frac{1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}F(\zeta)S(\zeta)^{-1}d\zeta. \end{aligned} \tag{4.5}$$

In order to extract an eigenspace from the expression of $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}$ in (4.4), we consider the product $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}V$, where $V = MZ,\ Z \in \mathbb{R}^{n\times r},\ r \geq nev$, written as

$$P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}\begin{pmatrix} V_u \\ V_s \end{pmatrix} = \begin{pmatrix} \mathcal{H}V_u + \mathcal{W}V_s \\ \mathcal{W}^T V_u + \mathcal{G}V_s \end{pmatrix} \equiv \begin{pmatrix} Z_u \\ Z_s \end{pmatrix}, \tag{4.6}$$

where $V = [V_u^T, V_s^T]^T$, and $V_u \in \mathbb{R}^{d\times r},\ V_s \in \mathbb{R}^{s\times r}$, respectively.

We finally get

$$\begin{aligned} Z_u &= \frac{-1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}B_\zeta^{-1}V_u d\zeta - \frac{-1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}F(\zeta)S(\zeta)^{-1}[V_s - F(\zeta)^T V_u]d\zeta \\ Z_s &= \frac{-1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}S(\zeta)^{-1}[V_s - F(\zeta)^T V_u]d\zeta. \end{aligned} \tag{4.7}$$

Let $\hat{X} = \left[x^{(1)}, \ldots, x^{(nev)}\right]$ and assume that $V \in \mathbb{R}^{n\times r}$ is chosen such that $V^T\hat{X}$ has rank nev. Then, the expression in (4.7) captures the exact invariant subspace of (A, M) associated with the eigenvalues $\lambda_1, \ldots, \lambda_{nev}$. In the process, the matrix $Z \equiv P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}V$ can be exploited in a Rayleigh-Ritz projection to recover the actual eigenpairs of (A, M). Because the above discussed scheme considers all blocks of $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}$, we will refer to it as "Domain Decomposition Full Projector" (DD-FP). In the following, we summarize the practical details of the DD-FP scheme.

4.1.1 Practical aspects of the DD-FP scheme

In practice, the contour integrals in (4.7) will have to be approximated numerically. Once a quadrature rule is selected, with quadrature nodes and weights[3] $\{\zeta_j, \hat{\omega}_j\},\ j = 1, \ldots, 2N_c$, the

[3]The pairs $\{\zeta_j, \hat{\omega}_j\},\ j = 1, \ldots, 2N_c$ are computed by discretizing the expression $\frac{1}{2i\pi}\int_{\Gamma_{[\alpha,\beta]}}(\zeta M - A)^{-1}M d\zeta$

expression in (4.7) is approximated by the following summations:

$$\tilde{Z}_u = -\sum_{j=1}^{2N_c} \hat{\omega}_j (B - \zeta_j M_B)^{-1} V_u + \sum_{j=1}^{2N_c} \hat{\omega}_j F(\zeta_j) S(\zeta_j)^{-1} [V_s - F(\zeta_j)^T V_u], \tag{4.8}$$

$$\tilde{Z}_s = -\sum_{j=1}^{2N_c} \hat{\omega}_j S(\zeta_j)^{-1} [V_s - F(\zeta_j)^T V_u]. \tag{4.9}$$

The numerical integration can be performed by one of the available quadrature rules, e.g., the Gauss-Legendre [27] or the Midpoint [50] rules. Since the eigenvalues of A are real, using a rule in which[4] the quadrature nodes appear in conjugate pairs, i.e., $\zeta_j = \overline{\zeta_{j+N_c}},\ j = 1, \ldots, N_c$, reduces the cost of the numerical approximation by a factor of two, since

$$B - \zeta_j M_B = \overline{B - \zeta_{j+N_c} M_B}, \quad S(\zeta_j) = \overline{S(\zeta_{j+N_c})}, \quad j = 1, \ldots, N_c.$$

For each quadrature node $\zeta_j,\ j = 1, \ldots, N_c$, and each one of the r columns of matrix V we must solve two linear systems with B_{ζ_j} and one linear system with $S(\zeta_j)$. The calculation takes four steps that accumulate the sums (4.8)-(4.9) into $\tilde{Z}_u,\ \tilde{Z}_s$, and is shown in Algorithm 4.1.1:

Algorithm **4.1.1** *DD-FP*

```
0.  Start with random $V \in \mathbb{R}^{n \times r}$ and set $\tilde{Z} = [\tilde{Z}_u^T, \tilde{Z}_s^T]^T = 0$
1.  Do until convergence
2.      For $j = 1, \ldots, N_c$:
3.          $W_u := B_{\zeta_j}^{-1} V_u$
4.          $W_s := V_s - E_{\zeta_j}^T W_u$
5.          $W_s := S(\zeta_j)^{-1} W_s,$              $\tilde{Z}_s := \tilde{Z}_s - \Re e(\hat{\omega}_j W_s)$
6.          $W_u := W_u - F(\zeta_j) W_s,$            $\tilde{Z}_u := \tilde{Z}_u - \Re e(\hat{\omega}_j W_u)$
7.      End
8.      Rayleigh-Ritz: solve the eigenvalue problem $\left(\tilde{Z}^T A \tilde{Z}\right) Q = \left(\tilde{Z}^T M \tilde{Z}\right) Q \hat{\Lambda}$
-.      If not satisfied with accuracy, repeat with $V_u = \tilde{Z}_u Q,\ V_s = \tilde{Z}_s Q$
9.  EndDo
```

[4]We assume here that none of the quadrature nodes lies on the real axis

The factorization of each block-diagonal matrix $B_{\zeta_j},\ j = 1,\ldots,N_c$ is decoupled into the factorization of the matrices $B_i - \zeta_j M_B^{(i)},\ i = 1,\ldots,p$. Recall here that $B_i - \zeta_j M_B^{(i)}$ is local to the ith subdomain. Moreover, only the real parts of $\tilde{Z}_s$ and $\tilde{Z}_u$ need be retained. Step (8) of Algorithm 4.1.1 extracts the approximate eigenpairs of A by a Rayleigh-Ritz projection, and also verifies whether all eigenpairs inside $[\alpha,\beta]$ are approximated up to a sufficient accuracy (this part is omitted from the description of the algorithm). If not satisfied with the accuracy achieved, we can repeat Steps (2)-(7) using the current approximate eigenvectors as the new matrix V.

If a direct solver is utilized to solve the linear systems with matrices $B_{\zeta_j},\ S(\zeta_j),\ j = 1,\ldots,N_c$ then the DD-FP scheme is equivalent to the FEAST method tied to a domain decomposition solver to compute the products $(A - \zeta_j M)^{-1}MV,\ j = 1,\ldots,N_c$. However, a factorization of $S(\zeta)$ is not always feasible. In such scenarios, the DD-FP scheme can leverage hybrid iterative solvers which might be more practical.

4.2 Partial integration of the matrix resolvent

In this section we describe an alternative scheme, also based on domain decomposition, which attempts to extract approximate eigenpairs at a lower cost than the DD-FP scheme.

Recall the matrix $\hat{X} = \left[x^{(1)},\ldots,x^{(nev)}\right]$ and the identity of the spectral projector $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}$ defined in (4.4), $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} = \hat{X}\hat{X}^T M$. Now, let $\hat{X}$ be written as $\hat{X} = [X_u^T,\ X_s^T]^T$ with $X_u \in \mathbb{R}^{d\times nev},\ X_s \in \mathbb{R}^{s\times nev}$. Then, $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)}$ can be also expressed in a block-partitioned form:

$$\hat{X} \equiv \begin{pmatrix} X_u \\ X_s \end{pmatrix}, P = \hat{X}\hat{X}^T M \rightarrow P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} = [\mathcal{P}_1, \mathcal{P}_2]M = \begin{pmatrix} X_u X_u^T & X_u X_s^T \\ X_s X_u^T & X_s X_s^T \end{pmatrix} M. \tag{4.10}$$

Under the mild assumption that $nev \leq s$, i.e., the number of interface variables s is greater than the number of eigenvalues of the matrix pencil (A, M) located inside the interval $[\alpha,\beta]$. The range of $P_{\Gamma_{[\alpha,\beta]}}^{(M^{-1}A)} M^{-1}$ can be then captured by the range of $\mathcal{P}_2 = \hat{X}X_s^T = [X_u^T,\ X_s^T]^T X_s^T$. Equating (4.10) with (4.4) shows that $X_s X_s^T \equiv \mathcal{G}$ and $X_u X_s^T \equiv \mathcal{W}$, and thus, in contrast with the DD-FP scheme, we only need to compute the contour integrals $\mathcal{W}$ and $\mathcal{G}$, and ignore the block $\mathcal{H}$. As discussed in Section 4.2.1, and confirmed via experiments in Section 4.4, avoiding the computation of $\mathcal{H}$ can lead to considerable savings in certain cases.

Because the above scheme approximates the spectral projector $\mathcal{P}$ only partially, we will refer to it as "Domain Decomposition Partial Projector" (DD-PP).

The DD-PP scheme

The range of $\mathcal{G}$ and $-\mathcal{W}$ can be captured by the range of:

$$\mathcal{G}R = \frac{-1}{2i\pi}\int_\Gamma S(\zeta)^{-1}Rd\zeta, \quad \mathcal{W}R = \frac{1}{2i\pi}\int_\Gamma B_\zeta^{-1}E_\zeta Rd\zeta, \tag{4.11}$$

for any $R \in \mathbb{R}^{s\times r}$ such that $\mathbf{rank}(X_s^T R) \equiv \mathbf{rank}(X_s)$.

In practice, (4.11) will be approximated numerically by a quadrature rule, and thus

$$\mathcal{G}R \approx \tilde{\mathcal{G}}R = -\sum_{j=1}^{2N_c}\hat{\omega}_j S(\zeta_j)^{-1}R, \quad \mathcal{W}R \approx \tilde{\mathcal{W}}R = \sum_{j=1}^{2N_c}\hat{\omega}_j B_\zeta^{-1}E_\zeta S(\zeta_j)^{-1}R. \tag{4.12}$$

Combining the contribution of all quadrature nodes together, the final subspace accumulation proceeds as in Algorithm 4.2.1, which we abbreviate as DD-PP.

Algorithm **4.2.1** *DD-PP*

```
0.  Start with a random $R \in \mathbb{R}^{s\times r}$ and set $\tilde{Z} = [\tilde{Z}_u^T, \tilde{Z}_s^T]^T = 0$
1.  For $j = 1, \ldots, N_c$:
2.      $W_s := S(\zeta_j)^{-1}R$,        $\tilde{Z}_s := \tilde{Z}_s - \Re e(\hat{\omega}_j W_s)$
3.      $W_u := -F(\zeta_j)W_s$,          $\tilde{Z}_u := \tilde{Z}_u - \Re e(\hat{\omega}_j W_u)$
4.  End
5.  Rayleigh-Ritz: solve the eigenvalue problem $\left(\tilde{Z}^T A\tilde{Z}\right)Q = \left(\tilde{Z}^T M\tilde{Z}\right)Q\hat{\Lambda}$
```

Analysis of the DD-PP scheme

The matrix inverse $(\zeta M - A)^{-1}$ in (8.1.3) can be also written in terms of the eigenvectors of (A, M) as

$$(\zeta M - A)^{-1} = \sum_{i=1}^{n}\frac{x^{(i)}\left(x^{(i)}\right)^T}{\zeta - \lambda_i}. \tag{4.13}$$

If we partition each eigenvector $x^{(i)}$ of (A, M) as $x^{(i)} = \left[\left(u^{(i)}\right)^T, \left(y^{(i)}\right)^T\right]^T$, $i = 1, \ldots, n$, and combine (4.13) with (8.1.3) for all $\zeta \equiv \zeta_j,\ j = 1, \ldots, 2N_c$, we get

$$\sum_{j=1}^{2N_c} \hat{\omega}_j (\zeta_j M - A)^{-1} = \sum_{i=1}^{n} \rho(\lambda_i) \begin{bmatrix} u^{(i)}(u^{(i)})^T & u^{(i)}\left(y^{(i)}\right)^T \\ y^{(i)}(u^{(i)})^T & y^{(i)}\left(y^{(i)}\right)^T \end{bmatrix} \tag{4.14}$$

where $\rho(\zeta) = \sum_{\ell=1}^{2N_c} \frac{\hat{\omega}_\ell}{\zeta_\ell - \zeta}$ is the filter function defined in (2.3) and which approximated. Equating the (1,2) and (2,2) blocks of (8.1.3) and (4.14) gives

$$\begin{aligned} \tilde{\mathcal{G}} &= -\sum_{j=1}^{2N_c} \hat{\omega}_j S(\zeta_j)^{-1} = \sum_{i=1}^{n} \rho(\lambda_i) y^{(i)} \left(y^{(i)}\right)^T \\ \tilde{\mathcal{W}} &= \sum_{j=1}^{2N_c} \hat{\omega}_j B_\zeta^{-1} E_\zeta S(\zeta_j)^{-1} = \sum_{i=1}^{n} \rho(\lambda_i) u^{(i)} \left(y^{(i)}\right)^T . \end{aligned} \tag{4.15}$$

By the expression in (4.15) we see that if $\rho(\lambda_{nev+1}), \ldots, \rho(\lambda_n)$ damp sufficiently close to zero, then **range**$\{\tilde{\mathcal{G}}\} \approx$ **span** $\left(\left[y^{(1)}, \ldots, y^{(nev)}\right]\right)$, and **range** $\left(\tilde{\mathcal{W}}\right) \approx$ **span** $\left(\left[u^{(1)}, \ldots, u^{(nev)}\right]\right)$, which are exactly the subspaces required to retrieve the sought eigenpairs $\left(\lambda_1, x^{(1)}\right), \ldots, \left(\lambda_{nev}, x^{(nev)}\right)$ of the matrix pencil (A, M). In the opposite case, an increase in r, the number of columns in matrix R, becomes necessary.

Figure 4.1 shows the average residual norm of the approximate eigenpairs obtained by the DD-FP and DD-PP schemes for a small 2D discretized Laplacian of size $n = 51 \times 50$ in the interval $[\alpha = 1.6, \beta = 1.7]$. In contrast to DD-PP, DD-FP can use a smaller number of quadrature nodes and increase the accuracy of the approximate eigenpairs of the matrix pencil (A, M) by repeating the numerical integration phase and using the most recent approximate eigenvectors as the new set of right-hand sides. Indeed, after four iterations, the DD-FP scheme with $N_c = 4$ quadrature nodes achieves an accuracy close to that of the DD-PP scheme utilizing $N_c = 12$ quadrature nodes.

We note at this point that the above discussion on the accuracy of DD-PP is independent of the number of subdomains p.

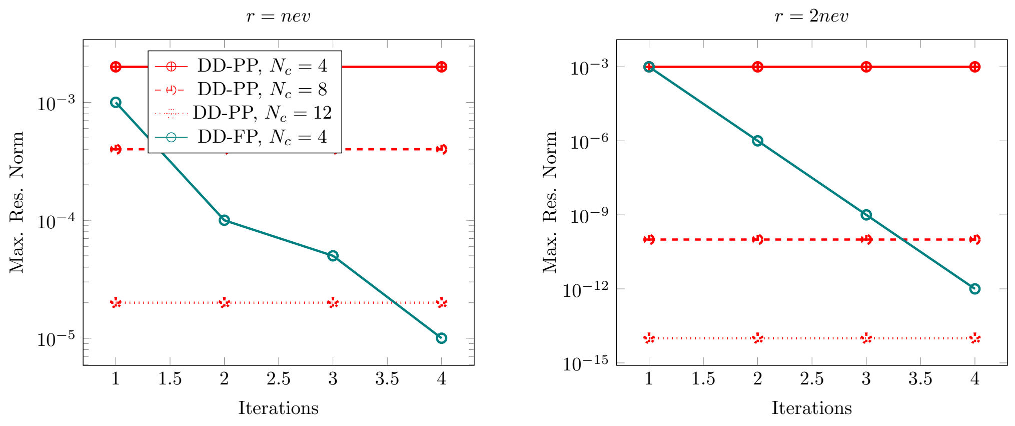


Figure 4.1: Maximum residual norm of the approximation of all eigenpairs in the interval [1.6, 1.7] for a 51 × 50 Finite Difference discretization of the Laplacian operator.

4.2.1 Computational comparison of the DD-FP and DD-PP schemes

From a numerical viewpoint, both the DD-PP and DD-FP schemes can perform similarly, but, from a computational viewpoint, there are some notable differences. DD-PP has a lower computational complexity per quadrature node than DD-FP, since it avoids performing the first two steps of the latter. A straightforward calculation reveals that for each quadrature node, the DD-FP scheme also introduces $n \times r$ more floating point operations than the DD-PP scheme (the block matrix subtractions in Steps 3 and 5 in Algorithm 4.1.1). When accounting for all quadrature nodes together, the DD-FP scheme introduces $N_c \times r \times [\text{cost\_solve}(B_\zeta) + \text{cost\_MV}(E_\zeta) + n]$ additional floating point operations compared to DD-PP. Herein, cost_solve(B_ζ) and cost_MV(E_ζ) denote the costs to multiply B_ζ^{-1} (by solving the linear system) and E_ζ by a single vector, respectively.

4.3 Solving the Schur complement linear systems

The major distributed computational procedure in both Algorithm 4.1.1 and Algorithm 4.2.1 is the solution of linear systems with the Schur complement matrices $S(\zeta_j),\ j = 1, \ldots, N_c$, where each linear system has $r \geq nev$ right-hand sides.

Assuming that each subdomain is assigned to a different MPI process, $S(\zeta)$ is distributed by rows among the different processors and has a natural block structure of the form

$$S(\zeta) = \begin{pmatrix} S_1(\zeta) & S_{12} & \dots & S_{1p} \\ S_{21} & S_2(\zeta) & \dots & S_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \dots & S_p(\zeta) \end{pmatrix}, \tag{4.16}$$

where

$$S_i(\zeta) = C_i - \zeta M_C^{(i)} - \left(\hat{E}_i - \zeta \hat{M}_E^{(i)}\right)^T \left(B_i - \zeta M_B^{(i)}\right)^{-1} \left(\hat{E}_i - \zeta \hat{M}_E^{(i)}\right),\ i = 1, \dots, p,$$

is the "local" Schur complement matrix that corresponds to the ith subdomain, and the off-diagonal blocks $S_{ik},\ i, k = 1, \dots, p,\ i \neq k$, are sparse matrices of size $s_i \times s_k$ that account for the coupling among the different subdomains. The matrix S_{ik} is nonzero if and only if subdomains i and k are adjacent. The matrix $M_C^{(i)}$ denotes the $s_i \times s_i$ on-diagonal block of matrix M_C.

The standard approach to solve the distributed linear system with the Schur complement coefficient matrix in (4.16) is to explicitly form $S(\zeta)$ and compute its LU factorization by a call to a parallel sparse direct solver [81, 83]. For problems issued from discretizations of 2D domains, forming and factorizing $S(\zeta)$ explicitly is an attractive option since the size of the Schur complement is small even when a large number of subdomains is used. On the other hand, Schur complements which originate from discretizations of 3D computational domains typically involve a much higher memory requirement since in the 3D case the size of the Schur complement can become exceedingly large. An alternative discussed next is to solve these linear systems without forming $S(\zeta)$ by exploiting a preconditioned iterative method, e.g. GMRES [84].

4.3.1 Schur complement preconditioners

We consider sparsified approximations of $S(\zeta)$ that are based on sparsity and/or numerical constraints [85, 86, 87], a procedure summarized in Algorithm 4.3.1. Other Schur complement preconditioning approaches can be found in [88, 89, 90].

To form the preconditioner, which we denote by $S_G(\zeta)$, we use two levels of dropping based on numerical constraints. The first level of dropping concerns the LU factorization of $B_i - \zeta M_B^{(i)} =$

$\hat{L}_i\hat{U}_i$ which is performed inexactly, by dropping all entries in the LU factorization whose real or imaginary part is below a threshold value `drop-B`. Then, the ith subdomain forms its local Schur complement

$$\hat{S}_i(\zeta) = C_i - \zeta M_C^{(i)} - \left(\hat{U}_i^{-T}\left(\hat{E}_i - \zeta \hat{M}_E^{(i)}\right)\right)^T \hat{L}_i^{-1}\left(\hat{E}_i - \zeta \hat{M}_E^{(i)}\right), \tag{4.17}$$

while also dropping any entry whose real or imaginary part is below a threshold value `drop-S`.

Overall, the preconditioner takes the form:

$$S_G(\zeta) = \begin{pmatrix} \hat{S}_1(\zeta) & S_{12} & \dots & S_{1p} \\ S_{21} & \hat{S}_2(\zeta) & \dots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ S_{p1} & S_{p2} & \dots & \hat{S}_p(\zeta) \end{pmatrix}. \tag{4.18}$$

Algorithm **4.3.1** *Schur complement preconditioner* $S_G(\zeta)$

```
0.  Given $\zeta \in \mathbb{C}$, drop-B, drop-S
1.  For $i = 1, \dots, p$:
2.      Obtain a factorization $[\hat{L}_i, \hat{U}_i] = B_i - \zeta M_B^{(i)}$ with drop tolerance drop-B
3.      Form $\hat{S}_i(\zeta) = C_i - \zeta M_C^{(i)} - \left(\hat{U}_i^{-T}\left(\hat{E}_i - \zeta \hat{M}_E^{(i)}\right)\right)^T \hat{L}_i^{-1}\left(\hat{E}_i - \zeta \hat{M}_E^{(i)}\right)$ and
-.          drop any entry smaller than drop-S
4.  End
5.  Factorize $S_G(\zeta)$ by a sparse direct solver.
```

Here are a few details regarding Algorithm 4.3.1. When we form $S_G(\zeta)$ we compute $\hat{S}_i(\zeta)$ a few columns at a time and immediately sparsify (for each incomplete factorization of $B_i - \zeta M_B^{(i)}$ we must solve a linear system with s_i sparse right-hand sides). By default we form $\hat{S}_i(\zeta)$ two hundred columns at a time, where all right-hand sides are solved simultaneously using the Pardiso software library (version 5.0.0) [91, 92]. More details will be given in Section 4.4.

4.3.2 Matrix-Vector products with the matrix $S(\zeta)$

The Matrix-Vector (MV) product between $S(\zeta)$ and a vector $v \in \mathbb{C}^s$ can be computed as:

$$S(\zeta)v = C_\zeta v - E_\zeta^T B_\zeta^{-1} E_\zeta v. \tag{4.19}$$

Since the matrices B_ζ and E_ζ are block-diagonal and already distributed among the set of available MPI processes, no communication is required when we perform operations with them. On the other hand, performing operations with C demands communication between MPI processes which handle neighboring subdomains.

In summary, the computations involved in (4.19) are:

1. Compute $E_\zeta^T B_\zeta^{-1} E_\zeta v$ (local),
2. Distribute (exchange) the necessary parts of v and perform $C_\zeta v$ (global),
3. Subtract the vector in 1) from the vector in 2) (local).

Communication in step 2) might overlap with computations in step 1). Using more subdomains (larger values for p) will reduce the computational cost per processor, but, on the other hand, increase communication cost.

4.4 Experiments

The experiments were performed on the `Mesabi` Linux cluster at Minnesota Supercomputing Institute. All numerical schemes were implemented in C/C++ and built on top of the PETSc and MKL scientific libraries. For PETSc, we used a complex build.[5] The source files were compiled with the Intel MPI compiler `mpiicpc`, using the -O3 optimization level.

Regarding DD-FP and DD-PP, the computational domain was partitioned in p non-overlapping subdomains using METIS, and each subdomain was then assigned to a distinct processor group. Communication between different processor groups was achieved by means of MPI. Throughout this section, the number of subdomains p will also denote the number of MPI processes. The LU factorizations and linear system solutions associated with the block-diagonal matrices $B_{\zeta_j},\ j = 1, \ldots, N_c$, were performed by the shared-memory, multi-threaded version of the Pardiso

[5]The complex version of PETSc was built using the option `--with-fortran-kernels=generic`

library. Unless stated otherwise, the default number of threads per MPI process, as denoted by variable τ, will be equal to one.

The quadrature nodes and weights $\zeta_j,\ \hat{\omega}_j,\ j = 1,\ldots,N_c$ were computed by the Gauss-Legendre quadrature rule of order $2N_c$, retaining only the N_c quadrature nodes (and associated weights) with positive imaginary part. While it is possible to utilize block Krylov subspace[6] solvers, e.g., block GMRES [95], throughout the rest of this section, the multiple right-hand sides will be solved one after the other. Whenever we computed an incomplete factorization of the block-diagonal matrices $B_{\zeta_j},\ j = 1,\ldots,N_c$, that was obtained by the UMFPACK [96] library, and the resulting triangular factors were then passed to Pardiso.

Finally, all distributed memory matrix factorizations and triangular substitutions associated with the distributed matrices $A - \zeta_j M$ and $S(\zeta_j),\ j = 1,\ldots,N_c$, were performed by MUMPS.

Throughout the rest of this section we will set $M = I$.

4.4.1 A comparison of the DD-FP and DD-PP schemes for 2D domains

We start by comparing the DD-FP and DD-PP schemes on a set of discretized 2D Laplacian matrices ($n_z = 1$), where the Schur complement matrices $S(\zeta_j),\ j = 1,\ldots,N_c$, were formed and factorized explicitly (`drop-B=drop-S=1e-16`). In order to perform a fair comparison between these two schemes, we allowed only one outer iteration in DD-FP.

The interval of interest was set to $[\alpha,\beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$ and thus included $nev = 200$ eigenvalues. We tried $N_c = 4,\ N_c = 8$ and $N_c = 12$ quadrature nodes, while we also varied the number of right-hand sides, r. Table 4.1 reports the average time spent on a single quadrature node for the case $N_c = 8$. Per quadrature node timings for the rest of the values of N_c were basically identical. DD-PP was always faster than DD-FP, especially as p obtained smaller values, and n and r larger values, respectively. The latter results lie in agreement with the discussion in Section 4.2.1.

Figure 4.2 plots the maximum residual norm of the approximate eigenpairs of the 2D Laplacian of size $n = 1000^2$ for all different combinations of N_c and r reported in Table 4.1. The residual norms of the DD-FP and DD-PP schemes were of the same order of magnitude, therefore we report results only for the DD-PP scheme.

[6]See [93, 94] for details and references

Table 4.1: Average amount of time spent on a single quadrature node in DD-PP and DD-FP to approximate the eigenvalues $\lambda_{1001}, \ldots, \lambda_{1200}$ and associated eigenvectors for three discretized 2D Laplacians.

	$p = 8$		$p = 16$		$p = 32$		$p = 64$	
	DD-PP	DD-FP	DD-PP	DD-FP	DD-PP	DD-FP	DD-PP	DD-FP
$n = 500^2$								
$r = nev + 10$	9.45	13.7	6.77	8.91	5.25	6.34	4.65	5.30
$r = 3nev/2 + 10$	13.5	19.5	9.65	12.7	7.59	9.01	6.64	7.54
$r = 2nev + 10$	18.1	26.0	12.9	16.8	10.0	12.1	8.83	10.1
$n = 1000^2$								
$r = nev + 10$	41.8	62.7	25.3	35.8	17.9	23.1	14.8	19.0
$r = 3nev/2 + 10$	59.7	89.5	36.0	49.9	25.5	33.1	21.1	26.9
$r = 2nev + 10$	79.1	119.3	68.1	68.1	34.1	44.2	28.4	36.3
$n = 1500^2$								
$r = nev + 10$	100.8	140.7	65.2	88.8	39.9	44.2	29.5	37.8
$r = 3nev/2 + 10$	144.2	201.3	93.1	126.4	57.6	63.9	42.6	54.9
$r = 2nev + 10$	192.7	268.6	124.5	168.9	76.0	84.3	56.7	72.7

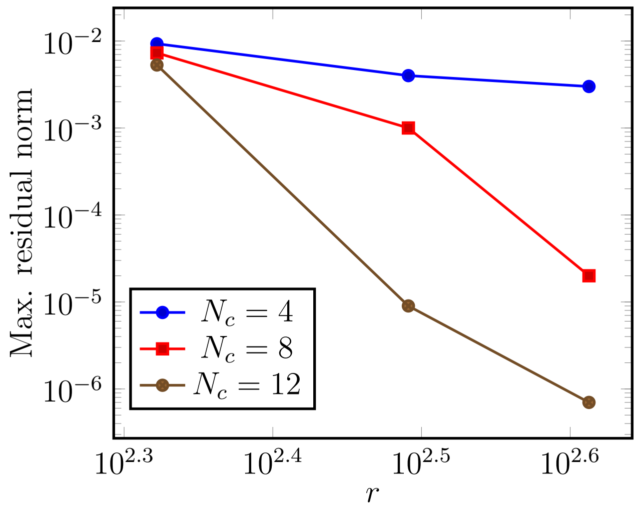


Figure 4.2: Maximum residual norm of the approximation of the eigenpairs inside the interval $[\alpha, \beta] = [(\lambda_{1000}+\lambda_{1001})/2, (\lambda_{1200}+\lambda_{1201})/2]$, when DD-PP is applied to the $n = 1000^2$ Laplacian.

Table 4.2: Wall-clock time to compute eigenvalues $\lambda_{1001}, \ldots, \lambda_{1200}$ and corresponding eigenvectors of the $n = 1500^2$ Laplacian by the CI-M and DD-FP schemes, as the values of N_c and r vary. "Its" denotes the number of outer iterations required by Subspace Iteration.

	Its	$p = 64$		$p = 128$		$p = 256$	
		CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$N_c = 2$							
$r = 3nev/2 + 10$	9	3,922.7	2,280.6	2,624.3	1,242.4	1,911.2	859.5
$r = 2nev + 10$	5	2,863.2	1,764.5	1,877.7	998.5	1,255.5	615.3
$N_c = 4$							
$r = 3nev/2 + 10$	5	4,181.5	2,357.0	2,815.7	1,280.2	1,874.1	877.5
$r = 2nev + 10$	4	4,330.3	2,571.4	2,869.5	1,462.9	2,023.2	1,036.2
$N_c = 6$							
$r = 3nev/2 + 10$	3	3,710.3	2,068.2	2,504.1	1,122.1	1,790.8	766.5
$r = 2nev + 10$	3	4,774.8	2,798.5	3,177.7	1,595.2	2,743.6	1,125.1
$N_c = 8$							
$r = 3nev/2 + 10$	3	4,911.6	2,722.2	3,318.7	1,476.1	2,367.7	1,006.5
$r = 2nev + 10$	2	4,204.7	2,445.2	2,802.1	1,395.4	1,806,6	982.1

The last experiment of this section focuses on a comparison between DD-FP and a PETSc-based implementation of the FEAST algorithm that utilizes MUMPS to factorize and solve the linear systems with matrices $A - \zeta_j M,\ j = 1, \ldots, N_c$. We refer to this scheme as Contour Integration-MUMPS (CI-M). Table 4.2 lists the wall-clock times of DD-FP and CI-M to compute all eigenpairs located inside the interval $[\alpha, \beta] = [(\lambda_{1000} + \lambda_{1001})/2, (\lambda_{1200} + \lambda_{1201})/2]$ for the $n = 1500^2$ Laplacian. Each eigenpair was sought to at least eight digits of accuracy and the variable "Its" denotes the number of outer iterations (same in both schemes). For CI-M, p denotes the number of MPI processes set in MUMPS. Increasing N_c leads to fewer outer iterations, although this does not necessarily imply lower wall-clock times. The performance gap between the DD-FP and CI-M schemes follows a slightly increasing trend as larger values of p are used, mainly because the linear system solution phase scales better in DD-FP than what in CI-M.

4.4.2 A 3D model problem

In this section we consider the solution of an eigenvalue problem where the matrix A originates from a discretization of the Laplacian operator on the unit cube using $n_x = n_y = n_z = 150$ ($n = 3,375,000$). For 3D problems of this size, a direct formation and factorization of the Schur complement matrices can be rather expensive, and, depending on the number of eigenvalues sought, as well as their location in the spectrum, preconditioned iterative solvers might form a

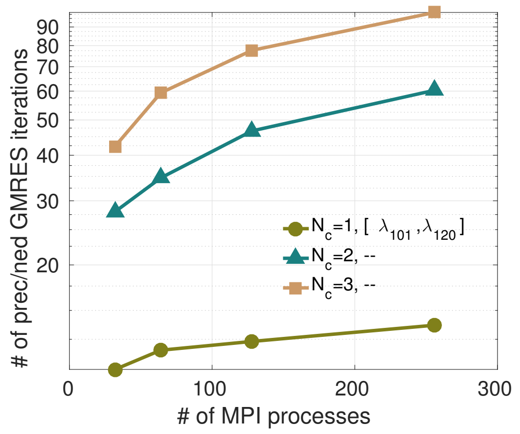


Figure 4.3: Total number of preconditioned GMRES iterations in order to solve a linear system with a single right-hand side for various values of N_c and p.

better alternative.

In contrast with Section 4.4.1, the solution of linear systems with the Schur complement matrices in this section far dominates the overall computational time. We compare the DD-FP and CI-M schemes and consider the problem of computing the smallest nev=20 and all nev=100 eigenvalues (and associated eigenvectors) located inside the intervals: $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$.

DD-FP with preconditioned iterative linear system solvers

Figure 4.3 lists the total number of preconditioned GMRES iterations to compute $\sum_{j=1}^{N_c} S(\zeta_j)^{-1}v$ for a random $v \in \mathbb{C}^s$. Details on the preconditioner used will be given later in this section. The interval of interest was set to $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$. We can observe that as N_c increases, the number of preconditioned GMRES iterations also increases. Indeed, iterative solvers are greatly affected by the location of the quadrature nodes ζ_j, $j = 1, \ldots, N_c$, with ζ_j's which lie closer to the real axis leading to slower convergence [59]. By construction, higher values of N_c will lead to some quadrature nodes being closer to the real axis. Thus, when iterative solvers are exploited, setting N_c to a low value, e.g. $N_c = 1$ or $N_c = 2$, might in practice be a good choice.

Throughout this section, the iterative linear system solver used by DD-FP will be the right preconditioned GMRES, allowing up to 250 iterations per restart. The linear system solution process will terminate as soon as the norm of the residual of the corresponding approximate solution is ten orders of magnitude smaller compared to the initial residual norm.

A comparison of the CI-M and DD-FP schemes

We now compare the wall-clock times of CI-M and DD-FP to compute the smallest $nev = 20$ and all $nev = 100$ eigenpairs located inside the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$. For DD-FP, we kept $p = 32$ fixed, allowing each MPI process to utilize exactly $p/32$ threads.

Table 4.3 lists the best (lowest) wall-clock times achieved by executing CI-M and DD-FP for two different values of N_c, $N_c = 1$ and $N_c = 2$ (setting $N_c > 2$ always led to longer times), and three different values of r. DD-FP was considerably faster than CI-M in all cases where $nev = 20$. When $nev = 100$, DD-FP was faster than CI-M for the interval $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, but considerably slower than CI-M for the interval $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$. In the latter case, the average time required to solve for a right-hand side by MUMPS was much lower than the amount of time required by preconditioned GMRES in DD-FP.

Note that the wall-clock times of DD-FP (especially its scalability) can generally improve, since, for reasons of fair comparison against CI-M, for MUMPS we used only MPI parallelism, and thus only a fraction of the available computer cores were active in DD-FP when we applied the preconditioner $S_G(\zeta_j)$.

Figure 4.4 plots the time breakdown of CI-M if we focus on its two main computational procedures, i.e., the amount of time spent on factorizations and triangular substitutions. Results shown are for the optimal choice r for each different value of N_c shown, and for $p = 128$ MPI processes. The average (per quadrature node) factorization time required by MUMPS was 679.02, 332.11, and 298.43 seconds, for 64, 128, and 256 MPI processes, respectively. Combining the latter with the results in Table 4.3, we observe that for the values of N_c reported in this section, the amount of time spent on linear system solutions generally dominates the wall-clock times. Similar observations also hold for DD-FP.

Table 4.3: Best (lowest) wall-clock times achieved by executing CI-M and DD-FP for $N_c = 1$ and $N_c = 2$. For CI-M we kept $\tau = 1$ fixed, while for DD-FP we kept $p = 32$ fixed. Variable "Its" denotes the total number of outer iterations performed by CI-M and DD-FP.

	Its		$p \times \tau = 64$		$p \times \tau = 128$		$p \times \tau = 256$	
	$N_c = 1$	$N_c = 2$	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$[\alpha, \beta] \equiv [\lambda_{101}, \lambda_{120}]$								
$r = 50$	8	5	1,607.2	324.8	841.4	240.0	685.0	217.9
$r = 100$	6	4	2,073.9	473.1	1,092.1	353.2	875.2	313.8
$r = 39$	8	5	1,420.6	265.5	741.6	194.8	609.1	166.8
$[\alpha, \beta] \equiv [\lambda_{501}, \lambda_{520}]$								
$r = 50$	9	5	1,723.9	1,029.1	904.3	777.4	732.5	685.9
$r = 100$	5	4	1,840.5	1,140.3	966.9	862.3	780.0	781.2
$r = 39$	9	5	1,492.9	808.9	780.4	609.5	638.5	541.6
$[\alpha, \beta] \equiv [\lambda_{101}, \lambda_{200}]$								
$r = 200$	14	5	6,013.9	3,141.9	3,185.4	2,389.6	2,510.1	2,105.4
$r = 300$	9	4	6,942.3	3,030.7	3,662.1	2,304.9	2,814.3	2,072.7
$r = 236$	10	5	6,179.6	2,652.6	3,294.9	1,989.3	2,547.1	1,766.4
$[\alpha, \beta] \equiv [\lambda_{501}, \lambda_{600}]$								
$r = 200$	12	5	6,013.9	13,373.4	3,185.8	10,195.8	2,510.2	9,447.2
$r = 400$	7	3	6,950.2	15,596.3	3,664.1	11,892.2	2,876.2	10,564.3
$r = 166$	13	5	5,220.4	11,954.5	2,759.9	9,114.0	2,178.1	8,444.6

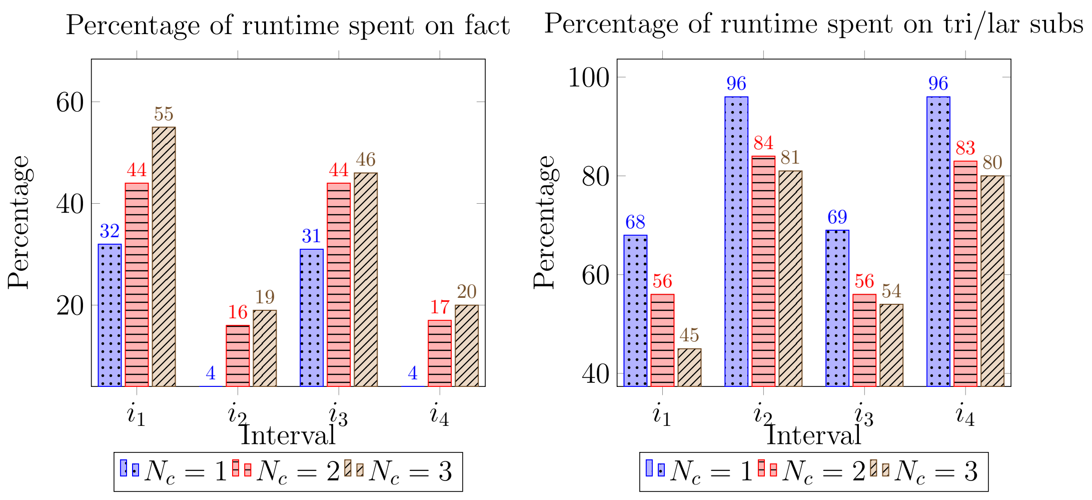


Figure 4.4: Time breakdown of the CI-M scheme (time spent on factorizations and triangular substitutions) for $N_c = 1$, $N_c = 2$ and $N_c = 3$ quadrature nodes, using the optimal choice of r for each different value of N_c, and $p = 128$ MPI processes. For each choice of N_c, we show the breakdown for intervals $i_1 := [(\lambda_{100} + \lambda_{101})/2, (\lambda_{120} + \lambda_{121})/2]$, $i_2 := [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$, $i_3 := [(\lambda_{500} + \lambda_{501})/2, (\lambda_{520} + \lambda_{521})/2]$, and $i_4 := [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$.

Table 4.4: Test matrices obtained by the PARSEC collection. We list the matrix size n, the total number of non-zero entries nnz, the interval of interest $[\alpha, \beta]$, and the number of eigenvalues r located inside $[\alpha, \beta]$.

Matrix	n	nnz	$[\alpha, \beta]$	r
$Ge_{99}H_{100}$	112,985	8,451,295	$[-0.65, -0.0096]$	250
$Si_{41}Ge_{41}H_{72}$	185,639	15,011,265	$[-0.64, -0.0028]$	218
$Si_{87}H_{76}$	240,369	10,661,631	$[-0.66, -0.0300]$	213

4.4.3 The PARSEC matrix collection

Our third set of experiments consists of a few matrices originating from applications in Electronic Structure Calculations. The matrices of interest (Hamiltonians) were generated using the PARSEC software package [64], and can be found in the SuiteSparse Matrix Collection. Details on the size of the matrices, as well as the interval of interest determined by the Density Functional Theory application, are listed in Table 4.4.

The number of nonzero entries of each Hamiltonian is quite large, a consequence of the high-order discretization used, as well as the addition of a (dense) ‘non-local’ term. Together with the 3D nature of the problem, this leads to a large number of interface variables, challenging the practicality of direct linear system solvers. In order to increase the efficiency of contour integration eigensolvers for such problems, we consider the replacement of direct solvers by preconditioned iterative solvers. Throughout this section we will only consider block-Jacobi preconditioners

$$S_{BJ}(\zeta_j) = \texttt{bdiag}(S_1(\zeta_j), \ldots, S_p(\zeta_j)), \qquad j = 1, \ldots, N_c. \tag{4.20}$$

Table 4.5 lists the time elapsed to perform all N_c factorizations of the form $A - \zeta_j M,\ j = 1, \ldots, N_c$ (CI-M) versus the elapsed time to form and factorize the block-Jacobi preconditioner for all $i = 1, \ldots, p$, and $j = 1, \ldots, N_c$ (DD-FP). We report times obtained for a varying number of MPI processes (subdomains). A “X” flag under the CI-M scheme implies that not all N_c matrix factorizations could fit in the memory allocated by each MPI process.

Table 4.6 lists the time elapsed to solve all N_c linear systems by the CI-M and DD-FP schemes for a random right-hand side $v \in \mathbb{C}^n$, i.e, $\sum_{j=1}^{N_c}(A - \zeta_j M)^{-1}v$. For lower values of p, the DD-FP scheme is not competitive, since the cost to apply the block-Jacobi preconditioner is quite high in this case. However, as p increases, the time to solve a linear system by a preconditioned

Table 4.5: Time elapsed to perform the N_c LU matrix factorizations $A - \zeta_j M,\ j = 1, \ldots, N_c$ in CI-M versus time elapsed to form and factorize the block-Jacobi preconditioner in DD-FP.

	$p = 4$		$p = 8$		$p = 16$		$p = 32$	
	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$Ge_{99}H_{100}$								
$N_c = 1$	424.1	27.9	362.8	5.1	155.9	1.36	80.2	0.51
$N_c = 2$	860.9	56.4	714.2	10.7	308.7	2.57	162.3	0.92
$N_c = 3$	1,265.9	86.3	1,089.4	15.5	461.1	4.26	239.5	1.47
$Si_{41}Ge_{41}H_{72}$								
$N_c = 1$	1276.1	38.8	942.6	10.1	486.1	3.31	230.2	1.52
$N_c = 2$	X	74.5	1888.1	19.8	969.3	6.46	452.7	2.81
$N_c = 3$	X	117.4	X	28.8	1,442.5	10.0	691.3	4.40
$Si_{87}H_{76}$								
$N_c = 1$	X	119.8	1726.2	14.5	942.4	1.23	382.1	0.51
$N_c = 2$	X	247.4	X	29.7	1872.8	2.53	758.0	0.94
$N_c = 3$	X	355.1	X	44.6	2,853.9	3.82	1,127.4	1.61

Table 4.6: Time elapsed to perform the computation $\sum_{j=1}^{N_c}(A - \zeta_j M)^{-1}v$ with (DD-FP) and without (CI-M) using the domain decomposition framework. Vector $v \in \mathbb{C}^n$ denotes a random complex vector.

	$p = 4$		$p = 8$		$p = 16$		$p = 32$	
	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP	CI-M	DD-FP
$Ge_{99}H_{100}$								
$N_c = 1$	0.7	5.1	0.7	1.7	0.4	0.6	0.3	0.3
$N_c = 2$	1.5	13.1	1.4	3.2	0.8	1.7	0.7	0.5
$N_c = 3$	2.3	33.3	1.9	11.8	1.1	4.1	1.0	1.2
$Si_{41}Ge_{41}H_{72}$								
$N_c = 1$	1.8	7.5	1.2	3.7	0.7	1.0	0.7	0.5
$N_c = 2$	X	32.8	2.5	12.1	1.4	3.5	1.4	0.8
$N_c = 3$	X	61.3	X	31.2	2.1	8.6	2.1	2.1
$Si_{87}H_{76}$								
$N_c = 1$	X	15.0	1.6	4.3	1.3	0.9	0.9	0.4
$N_c = 2$	X	50.2	X	14.0	2.8	3.3	1.9	0.8
$N_c = 3$	X	120.5	X	34.8	4.0	7.5	2.7	2.0

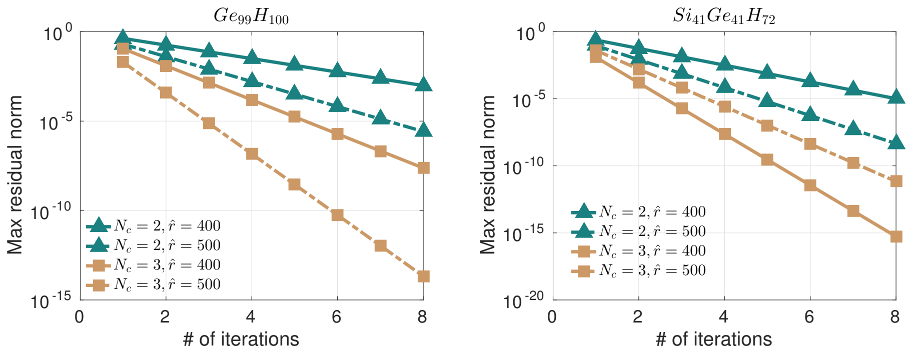


Figure 4.5: Maximum residual norm of the approximation of the eigenpairs inside the interval $[\alpha, \beta]$, as a function of the number of outer iterations performed by DD-FP. Results are shown for $N_c = 2,\ N_c = 3$, and $r = 400,\ r = 500$. Solid lines correspond to $r = 400$, while dashed lines correspond to $r = 500$. Left: $Ge_{99}H_{100}$. Right:$Si_{41}Ge_{41}H_{72}$.

Table 4.7: Wall-clock times of CI-M and DD-FP to compute all eigenpairs located inside the intervals $[\alpha, \beta]$ reported in Table 4.4 (we set $p = 32$). "Its" denotes the total number of outer iterations performed by DD-FP and CI-M.

	$r = 400$			$r = 500$		
	Its	CI-M	DD-FP	Its	CI-M	DD-FP
$Ge_{99}H_{100}$						
$N_c = 2$	22	5,990	4,675	9	4,164	3,198
$N_c = 3$	12	3,787	4,438	5	2,750	3,091
$Si_{41}Ge_{41}H_{72}$						
$N_c = 2$	13	7,651	4,392	8	5,996	3,410
$N_c = 3$	5	4,806	4,287	6	6,865	6,360
$Si_{87}H_{76}$						
$N_c = 2$	15	12,059	4,593	10	10,172	3,852
$N_c = 3$	5	6,467	3,960	6	9,114	5,915

iterative method drops dramatically (we note that the number of iterations is only slightly increased as p increases). Moreover, increasing the value of N_c results in a proportional increase in computational time for the direct solver but to a much more pronounced increase for the case of preconditioned iterative solvers, owed to the fact that iterative solvers are sensitive to the magnitude of the complex part of each quadrature node (see also the discussion in Section 4.4.2).

Figure 4.5 plots the maximum residual norm of the approximation of the eigenpairs located inside the interval $[\alpha, \beta]$ as a function of the number of outer iterations performed by DD-FP.

Finally, Table 4.7 reports the total wall-clock time required by CI-M and DD-FP to compute all sought eigenpairs for the case $p = 32$. DD-FP was faster[7] than CI-M for almost all different combinations of N_c and r tested, due to the avoidance of the costly matrix factorizations in CI-M and its lower timings to perform the required linear system solutions.

4.5 Summary

In this chapter we studied contour integration methods for computing eigenvalues and eigenvectors of sparse matrices using a domain decomposition viewpoint. We discussed two different numerical schemes. The first scheme, abbreviated as DD-FP, is a flexible implementation of the domain decomposition framework in the context of contour integral-based methods. When a direct solver is used for the Schur complement linear systems, DD-FP is equivalent to FEAST combined with a domain decomposition-based direct solver. The second scheme, abbreviated as DD-PP, focuses on approximating the contour integrals only partially by integrating the Schur complement operator along the complex contour. Moreover, we considered the use of domain decomposition in the context of preconditioned iterative solvers as a replacement of the direct solvers. Experiments indicate that this approach can potentially be faster, but that its ultimate effectiveness will be dictated by the performance of the iterative scheme used for solving the linear systems. In particular, the method can be vastly superior than FEAST with a direct solver when computing eigenvalues on both ends of the spectrum but it may encounter difficulties when the eigenvalues to be computed are located deep inside the spectrum.

[7]DD-FP also required far less memory than CI-M.

Chapter 5

Acceleration of rational filtering eigenvalue solvers by exploiting multiple levels of distributed memory parallelism

So far in this dissertation we have considered only one level of distributed memory parallelism; that stemming by applying the domain decomposition framework directly to the eigenvalue equation. On the other hand, one of the main advantages of contour integral eigenvalue solvers is their ability to take advantage of additional levels of MPI parallelism, e.g. by distributing different quadrature nodes and/or right-hand sides to different groups of MPI processes. In this chapter[1] we consider (1) the performance of the DD-FP[2] and FEAST eigenvalue solvers if more than one levels of distributed memory parallelism are exploited, and (2) the benefits of using domain decomposition linear system solvers in FEAST. Our main goal is not to perform an exhaustive analysis but rather to a) suggest a guideline on what the opportunities for additional levels of parallelism are, and b) strategies to allocate the computational resources so that the

[1]This is joint work with James Kestyn and Eric Polizzi (University of Massachusetts, Amherst) and Yousef Saad (University of Minnesota, Twin Cities)

[2]See Chapter 4

lowest possible wall-clock times are achieved.

The organization of this chapter is as follows. Section 5.1 repeats the experiments in Section 4.4.2 for the case where the available computational resources are organized into a 2D grid of MPI processes. Section 5.2 presents a comprehensive examination of the performance of FEAST for a collection of Hamiltonians representing carbon nanotube (CNT) molecules of varying length when two different levels of distributed memory parallelism are considered. Finally, Section 5.3 presents our concluding remarks.

5.1 Combining DD-FP with an additional layer of distributed memory parallelism

By recalling the discussion in Section 4.2.1, each iteration of DD-FP requires $2r$ linear system solutions with matrix B_ζ and r linear system solutions with matrix $S(\zeta)$ for each quadrature node $\zeta \equiv \zeta_j,\ j = 1, \ldots, N_c$.

Let us first focus on the linear system solutions with matrices $S(\zeta_1), \ldots, S(\zeta_{N_c})$. Let $R \in \mathbb{R}^{s \times r}$ as in Chapter 4 and consider the application of the matrix $2\Re e \left\{ \sum_{\ell=1}^{N_c} \hat{\omega}_\ell S(\zeta_\ell)^{-1} \right\}$ onto the matrix R. The first level of distributed memory parallelism is already in-place as the Schur complement matrix $S(\zeta)$ is distributed by blocks of rows among the MPI processes handling the p subdomains. A second level of parallelism can be exploited by organizing the set of available MPI processes into a 2D grid and restricting MPI processes within the same column subgrid to perform computations related to either a fraction of the N_c quadrature nodes, or a fraction of the r right-hand sides. The distribution of the linear system solutions with matrices $B_{\zeta_1}, \ldots, B_{\zeta_{N_c}}$, which are also distributed by blocks among the p subdomains, follows the same pattern with the solutions with matrices $S(\zeta_1), \ldots, S(\zeta_{N_c})$.

5.1.1 Evaluation

We repeated the experiment in Section 4.4.2, this time organizing the MPI processes in various 2D grid formations. As previously, the maximum number of MPI processes was set to 256. For CI-M we tried four different 2D formations; 64×1, 64×2, 64×3, and 64×4. Similarly, for DD-FP, we considered the 2D formations 32×1, 32×2, 32×3, and 32×4, allowing each MPI process to utilize $\tau = 2$ computational threads. For simplicity, we considered only the case $r = 200$.

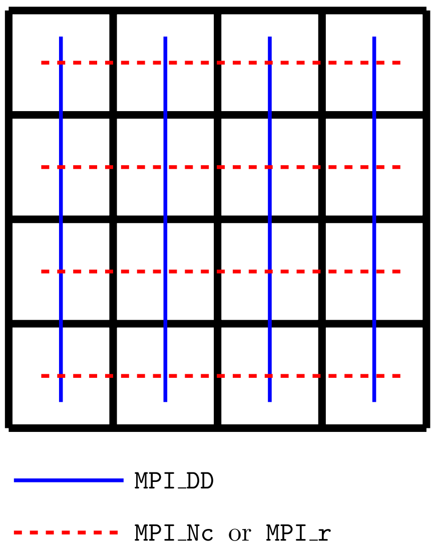


Figure 5.1: Schematic sketch of the 2D grid of MPI processes when two levels of distributed memory parallelism are used in DD-FP. The first level (blue solid lines) are reserved for domain decomposition. The second level (red dashed lines) can be used to distribute either the quadrature nodes or the different right-hand sides to different groups of MPI processes.

For DD-FP, we considered a single quadrature node ($N_c = 1$) and assigned different right-hand sides to different column subgrids of MPI processes (thus each column subgrid of MPI processes was responsible for only a fraction of the r right-hand sides in each iteration in DD-FP). For CI-M, we tested two options. First, we considered CI-M with $N_c = 1$ and $N_c = 2$, and assigned different right-hand sides to different column subgrids of MPI processes (thus, similarly to DD-FP, each column subgrid of MPI processes was responsible for all N_c quadrature nodes but only a fraction of the r right-hand sides at each CI-M iteration). We denote this option by CI-M1. Note that CI-M1 is limited only to scenarios where each column subgrid of MPI processes has enough memory to store all N_c matrix factorizations. Second, we considered CI-M with $N_c = 4$ quadrature nodes and assigned the different quadrature nodes to different column subgrids of MPI processes (thus each separate column subgrid of MPI processes was responsible for all r right-hand sides, but for one/a few of the N_c quadrature node(s) only). We denote this option by CI-M2.

Figure 5.2 plots the wall-clock times of CI-M and DD-FP when two levels of MPI parallelism are considered. Exploiting a 32×4 2D grid, DD-FP computed all $r = 100$ eigenpairs inside the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$ and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} +$

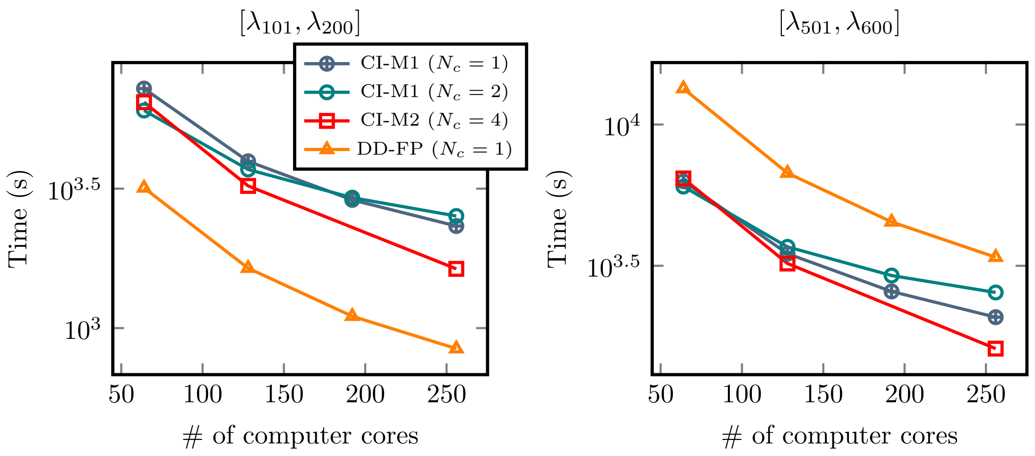


Figure 5.2: Wall-clock time of CI-M and DD-FP to compute all $nev = 100$ eigenpairs inside the intervals $[\alpha, \beta] = [(\lambda_{100} + \lambda_{101})/2, (\lambda_{200} + \lambda_{201})/2]$ and $[\alpha, \beta] = [(\lambda_{500} + \lambda_{501})/2, (\lambda_{600} + \lambda_{601})/2]$ when a 2D grid of MPI processes is exploited. The number of right-hand sides was set to $r = 200$.

$\lambda_{601})/2]$, in less than 900 and 3,500 seconds, respectively. The latter wall-clock times constitute a considerable improvement over those obtained by the 1D grid of MPI processes reported in Table 4.3. For CI-M1, the wall-clock time improvement over the 1D grid of MPI processes was not as pronounced as in DD-FP due to the overhead introduced by the factorizations of matrices $A - \zeta_j M,\ j = 1, \dots, N_c$. On the other hand, increasing N_c and distributing the quadrature nodes, as in CI-M2, seems to be a more efficient choice when a larger number of computational resources becomes available. A more detailed list of comments is following.

- Distributing the r right-hand sides to different processor groups (e.g. CI-M1(N_c)) seems to be the most efficient approach *when the largest portion of the total wall-clock time is spent on the distributed triangular substitutions.* Clearly, this approach is limited only to scenarios where each column subgrid of MPI processes has enough memory to store all N_c matrix factorizations (since each column subgrid of MPI processes must perform all N_c matrix factorizations independently of the other subgrids).

- Distributing the N_c quadrature nodes to different groups of processors, as CI-M2(N_c) does, is the most efficient approach when either a) an increase in the value of N_c leads to (much) faster convergence, or b) not all N_c matrix factorizations can fit in the system memory of a particular group of processors (and thus CI-M1 is inapplicable).

- Increasing the value of N_c and distributing the N_c quadrature nodes to different groups of processors is the most efficient approach when a larger number of computational resources is available, since this technique requires less memory (since each group of MPI processes performs only one/a few of the total N_c matrix factorizations), and also leads to faster convergence.

5.2 The PFEAST scheme

The Parallel FEAST (PFEAST) scheme is a fully parallel implementation of the FEAST library (the algorithmic component of FEAST is described in Section 2.3.2). In particular, PFEAST features three distinct levels of distributed memory parallelism: (**L1**) spectrum slicing, (**L2**) distribution of the N_c different quadrature nodes of the matrix $2\Re e\left\{\sum_{\ell=1}^{N_c}\omega_\ell(A-\zeta_\ell M)^{-1}M\right\}$ to up to N_c different groups of processors, and (**L3**) distributed memory solution of each linear system with the matrix $A-\zeta_\ell M$.

In this section we consider the performance of PFEAST as we vary the number of MPI processes in levels **L2** and **L3**.

5.2.1 Experimental framework

The matrix used for the experiments in this section is a Hamiltonian matrix which represents a CNT (carbon nanotube) molecule system discretized by the Finite Element method using second degree polynomial refinement, and was generated by a 3-3 CNT unit cell.In particular, the Hamiltonian considered in our experiments had 5 unit cells (CNT-5) and 78 atoms in total. The system was terminated with six hydrogen atoms at each end and follows a $(U_h|U_aU_b|\ldots|U_aU_b|U_a|U_h)$ pattern (see the 3 unit cell CNT in Figure 1.1) with U_h representing the six hydrogen atoms and U_a and U_b different configurations of six atom carbon rings. Each carbon atom contains six electrons, which corresponds to three eigenmodes (since the electron spin is not explicitly taken into account). The dimension of the problem was equal to $n = 302,295$ and the number of sought eigenvalues was set to $nev = 226$, where the first six eigenvalues (modes) correspond to the twelve terminating hydrogen atoms, the next eighteen eigenvalues to the U_a carbon ring, and the additional 5×36 eigenvalues to the $|U_aU_b|$ unit cells.

The experiments were performed on the `Mesabi` Linux cluster at Minnesota Supercomputing

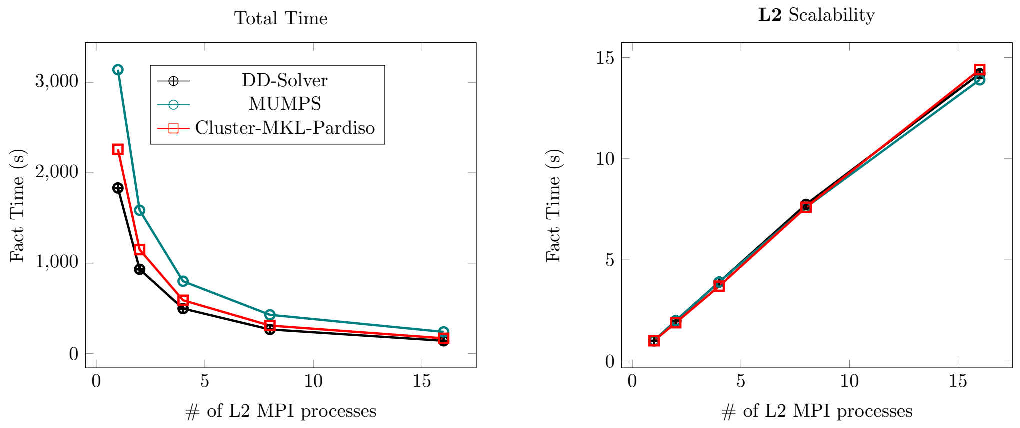


Figure 5.3: Wall-clock times and scalability of the **L2** level of distributed memory parallelism. The number of **L2** MPI processes is increased from 1 to 16 while the number of **L3** MPI processes is kept constant (3 MPI processes per **L2** MPI process).

Institute. Although the PFEAST kernel could be interfaced with an iterative or hybrid solver, we consider only the application of three different sparse direct solvers. Both Cluster-MKL-PARDISO and MUMPS have been tested with the full matrices generated by stitching together the interstitial and atomic meshes within our finite-element electronic structure code. Our application specific domain-decomposition solver (DD-Solver) reorders the matrix based on the number of **L3** MPI processes, in order to reduce communication. For consistency the matrices have been permuted the exact same way for Cluster-MKL-PARDISO, MUMPS and the DD-Solver. All tests have been run with default parameters for the solvers operating in 'in-core' mode.

Throughout the rest of this chapter each MPI process will be utilizing 12 threads. The number of right-hand sides was set to $r = 600$ while that of quadrature nodes was set to $N_c = 16$.

5.2.2 Strong scalability of levels L2 and L3

We start with the **L2** level scalability. The results can be seen in Figure 5.3. Each linear system solution with matrix $A - \zeta_\ell M,\ \ell = 1, \dots, N_c$, exploited 3 MPI processes (for a total of 3 to 48 MPI processes and thus 36 to 576 computer cores). The total time to solve the eigenvalue problem is plotted on the left subfigure. Four PFEAST iterations were needed to reach the default convergence criteria of 10^{-12} on the eigenvector residuals. The right subfigure shows the

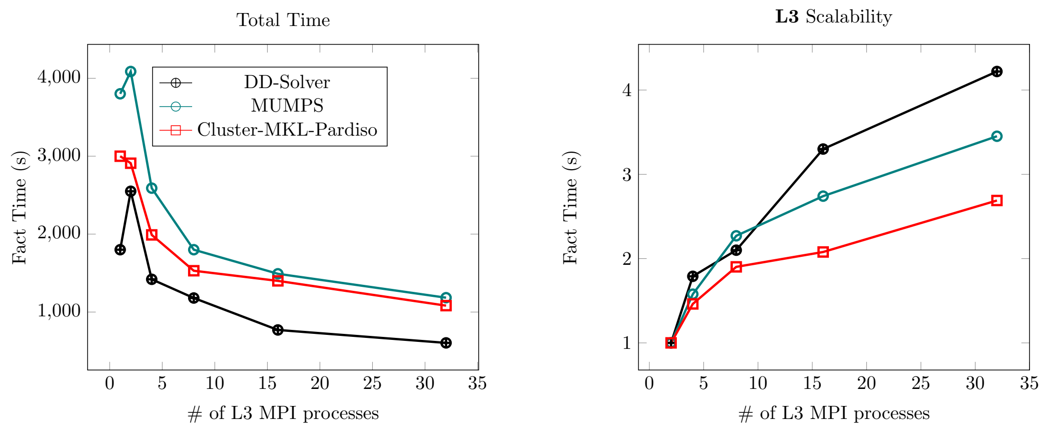


Figure 5.4: Wall-clock times and scalability of the **L3** level of distributed memory parallelism.

observed speedups for each one of the three different solvers used. We see very close to linear scaling at this level. Note that the maximum speedup we can expect at this level of parallelism is bounded by N_c.

Figure 5.4 plots the wall-clock times (left sibfigure) and scalability (right subfigure) of the **L3** level of distributed memory parallelism. The number of **L3** MPI processes was varied from 1 to 32. The speedup on the right subfigure uses the results obtained by using 2 **L3** MPI process as the time of reference. We followed this approach to account for the communication overhead associated with distributed memory solvers. The time should, ideally, drop by half each time the number of MPI processes is doubled. As can be seen in the graph, the efficiency of the **L3** scaling is much worse than that observed for **L2**. In practice, the scalability of the third level of parallelism **L3** is limited by the properties of the eigensystem and linear-system solver. The strong scalability of the DD-Solver, specifically, may be limited by the number atoms. All **L3** MPI processes can work together in the solution to the interstitial matrix regardless of the number of atoms. Note that problems that require a distributed memory solver will usually target nanostructures with many atoms; usually far outnumbering the parallel resources.

5.2.3 Notes on the optimal distribution of parallel resources

Another situation common within the scientific computing community is that where parallel resources are limited to a specific number of processors. Here the three different levels of parallelism within PFEAST can help to optimally utilize all parallel resources and achieve good

Table 5.1: The total number of MPI processes is held constant but is split between the two different levels of distributed memory parallelism.

# L2	# L3	# Rows	DD-Solver	PARDISO	MUMPS
1	64	**6260**	639	982	923
2	32	10553	323	511	643
4	16	19139	172	315	332
8	8	38277	116	171	194
16	4	76554	**104**	**115**	**125**

performance. For our experiment we only consider the second and third levels of parallelism **L2** and **L3**. The question becomes how to divide these resources among these two levels. We consider 64 sockets of `Mesabi` (thus 32 nodes in total) where each socket is equipped with a Intel Xeon Processor E5-2680 v3, i.e., each socket has 12 computer cores. Assuming that $N_c = 16$ quadrature nodes are used, then there are then five ways to distribute the resources across the **L2** and **L3** levels of parallelism. In particular, the possible number of MPI processes used in the **L2** level of distributed memory parallelism are $1, 2, 4, 8$, and 16. In order to fully utilize the computational resources at our disposal we choose the number of **L3** MPI processes as $64, 32, 16, 8$, and 4, respectively. Table 5.1 presents the total time to solve the eigenvalue problem for a fixed number of MPI processes but different distributions of parallel resources over the **L2** and **L3** levels. As we initially predicted, it is optimal to place as many resources as possible at the **L2** level. In paricular, the best performance is obtained when 16 MPI processes are used in the **L2** level. On the other hand, more MPI processes at the **L3** level will reduce the memory required by each MPI process (i.e. the number of rows of the eigenvector and system matrices assigned to each process), and might become a necessity for larger problems.

5.3 Summary

In this chapter we considered the performance of the DD-FP (see Chapter 4) and FEAST eigenvalue solvers if more than one levels of distributed memory parallelism are exploited, as well as the benefits of using domain decomposition linear system solvers in FEAST. For DD-FP, adding one more level of distributed memory parallelism can lead to a significant reduction of the wall-clock time/memory footprint. In particular, distributing different right-hand sides to different groups of processors seems to be a good choice when the Schur complement linear systems are solved by preconditioned iterative solvers. Similarly, the scalability of PFEAST mainly depends on the technique used to solve the complex linear systems as well as the distribution of the parallel resources to the different possible levels of parallelism. If enough resources are available, it is advisable to place as many MPI processes at the first and second levels of parallelism (i.e., **L1** and **L2**) as possible, as these levels lead close to almost ideal linear scaling.

Chapter 6

`Cucheb`: A GPU implementation of FILTLAN

In this chapter we present a Graphics Processing Unit (GPU) implementation of the Lanczos method combined with polynomial filtering. We will refer to such techniques as Filtered Lanczos Procedures (FLP). The proposed GPU implementation will consist of a high-level, open source C/C++ library called `Cucheb` [1] [97] which depends only on the Nvidia CUDA Toolkit [98, 30] and standard C libraries, allowing for easy interface with Nvidia brand GPUs.

A GPU is a single instruction multiple data scalable model which consists of multi-threaded streaming Multiprocessors, each equipped with multiple scalar processor cores (SPs), with each SP performing the same instruction on its local portion of data. While they were initially developed for the purposes of graphics processing, GPUs were adapted in recent years for general purpose computing, and the development of the Compute Unified Device Architecture (CUDA) [30] parallel programming model by Nvidia (an extension of the C/C++ language) provides an easy way for computational scientists to take advantage of the GPU's raw power. Additional information on the programming of of GPUs can be found in [99].

The FLP implemented in the Cucheb libary is similar to FILTLAN presented in Section 2.4. There are, however, a few notable differences. Cucheb features full orthogonalization instead of partial orthogonalization as is the case in FILTLAN. Moreover, Cucheb includes the ability

[1]This is joint work with Jared L. Aurentz (Instituto de Ciencias Matemàticas) and Yousef Saad (University of Minnesota, Twin Cities)

to use block counterparts of the Lanczos method which can be more efficient in the case of multiple or clustered eigenvalues. In addition, FILTLAN uses a more complicated least-squares filter polynomial while Cucheb utilizes the filters described in section 6.2.

While our numerical experiments presented in Section 6.4 consider only the solution of standard symmetric eigenvalue problems, the discussion of the numerical scheme presented throughout this chapter will focus on the more general case of generalized eigenvalue problems. As in the previous chapters, the goal is to compute the $nev \geq 1$ eigenpairs $\left(\lambda_1, x^{(1)}\right), \dots, \left(\lambda_{nev}, x^{(nev)}\right)$ of the matrix pencil (A, M) whose corresponding eigenvalues are located inside the interval $[\alpha, \beta]$.

6.1 Motivation

In the DFT framework the solution of the all-electron Schrödinger equation is replaced by a one-electron Schrödinger equation with an effective potential which leads to a nonlinear eigenvalue problem known as the Kohn-Sham equation [5, 6]:

$$\left[-\frac{\nabla^2}{2} + V_{ion}(r) + V_H(\rho(r), r) + V_{XC}(\rho(r), r)\right] \Psi_i(r) = E_i \Psi_i(r), \tag{6.1}$$

where $\Psi_i(r)$ is a wave function and E_i is a Kohn-Sham eigenvalue. The ionic potential V_{ion} reflects contributions from the core and depends on the position r only. Both the Hartree and the exchange-correlation potentials depend on the charge density:

$$\rho(r) = 2 \sum_{i=1}^{n_{occ}} |\Psi_i(r)|^2, \tag{6.2}$$

where n_{occ} is the number of occupied states (for most systems of interest this is half the number of valence electrons). Since the total potential $V_{total} = V_{ion} + V_H + V_{XC}$ depends on $\rho(r)$ which itself depends on eigenfunctions of the Hamiltonian, Equation (6.1) can be viewed as a nonlinear eigenvalue problem or a *nonlinear eigenvector problem.* The Hartree potential V_H is obtained from ρ by solving the Poisson equation $\nabla^2 V_H(r) = -4\pi\rho(r)$ with appropriate boundary conditions. The exchange-correlation term V_{XC} is the key to the DFT approach and captures the effects of reducing the problem from many particles to a one-electron problem, i.e., from replacing wavefunctions with many coordinates into ones that depend solely on space location r.

Self-consistent iterations for solving the Kohn-Sham equation start with an initial guess of the charge density $\rho(r)$, from which a guess for V_{total} is computed. Then (6.1) is solved for $\Psi_i(r)$'s and a new $\rho(r)$ is obtained from (6.2) and the potentials are updated. Then (6.1) is solved again for a new ρ obtained from the new $\Psi_i(r)$'s, and the process is repeated until the total potential has converged.

A typical electronic structure calculation with many atoms requires the calculation of a large number of eigenvalues, specifically the n_{occ} leftmost ones. In addition, calculations based on time-dependent density functional theory [100, 101], require a substantial number of unoccupied states, states beyond the Fermi level, in addition to the occupied ones. Thus, it is not uncommon to see eigenvalue problems in the size of millions where tens of thousands of eigenvalues may be needed.

Efficient numerical methods that can be easily parallelized in current high-performance computing environments are therefore essential in electronic structure calculations. The high computational power offered by GPUs has increased their presence in the numerical linear algebra community and they are gradually becoming an important tool of scientific codes for solving large-scale, computationally intensive eigenvalue problems. While GPUs are mostly known for their high speedups relative to CPU-bound operations[2], sparse eigenvalue computations can also benefit from hybrid CPU-GPU architectures. Although published literature and scientific codes for the solution of sparse eigenvalue problems on a GPU have not been as common as those that exist for multi-CPU environments, recent studies demonstrated that the combination of polynomial filtering eigenvalue solvers with GPUs can be beneficial [102].

The rest of this chapter is organized as follows. Section 6.2 introduces the concept of polynomial filtering for symmetric eigenvalue problems and provides the basic formulation of the filters used. Section 6.3 provides details on the block Lanczos methods combined with polynomial filtering. Section 6.4 presents computational results with the proposed GPU implementations. Finally, a summary is presented in Section 6.5.

[2]See also the MAGMA project at http://icl.cs.utk.edu/magma/index.html

6.2 Chebyshev polynomial filters

In order to quickly construct a polynomial that is a good approximation to the indicator function $I_{[-1,1]}$[3] defined in (2.1) it is important that we choose a good basis. For functions supported on $[-1,1]$ the obvious choice is Chebyshev polynomials of the first kind. Such representations have already been used successfully for constructing polynomial spectral transformations and for approximating matrix-valued functions in quantum mechanics (see for example [102, 65, 67, 103, 104, 105, 106, 68, 107, 108, 109]).

Recall that the Chebyshev polynomials of the first kind obey the following three-term recurrence

$$T_{i+1}(z) = 2zT_i(z) - T_{i-1}(z),\ i \geq 1. \tag{6.3}$$

starting with $T_0(z) = 1,\ T_1(z) = z$. The Chebyshev polynomials also satisfy the following orthogonality condition and form a complete orthogonal set for the Hilbert space $L^2_\mu([-1,1])$, $d\mu(z) = \left(1 - z^2\right)^{-1/2} dz$:

$$\int_{-1}^{1} \frac{T_i(z)T_j(z)}{\sqrt{1-z^2}} dz = \begin{cases} \pi, & i = j = 0, \\ \frac{\pi}{2}, & i = j > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{6.4}$$

Since $I_{[-1,1]} \in L^2_\mu([-1,1])$ it possesses a convergent Chebyshev series

$$I_{[-1,1]}(z) = \sum_{i=0}^{\infty} b_i T_i(z), \tag{6.5}$$

where the scalars $\{b_i\}_{i=0}^{\infty}$ are defined as follows:

$$b_i = \frac{2 - \delta_{i0}}{\pi} \int_{-1}^{1} \frac{I_{[-1,1]}(z)T_i(z)}{\sqrt{1-z^2}} dz, \tag{6.6}$$

where δ_{ij} represents the Dirac delta symbol. For a given α and β the $\{b_i\}$ are known analytically (see for example [110]),

$$b_i = \begin{cases} \left(\arccos(\alpha) - \arccos(\beta)\right)/\pi, & i = 0, \\ 2\left(\sin\left(i \arccos(\alpha)\right) - \sin\left(i \arccos(\beta)\right)\right)/i\pi, & i > 0. \end{cases} \tag{6.7}$$

[3]Recall that we map the interval $[\alpha, \beta]$ to $[-1,1]$

An obvious choice for constructing an approximation to $I_{[-1,1]}(z)$ is to fix a degree $m \in \mathbb{Z}$ and truncate the Chebyshev series of $I_{[-1,1]}(z)$,

$$p_m(z) = \sum_{i=0}^{m} b_i T_i(z). \tag{6.8}$$

Due to the discontinuities of $I_{[-1,1]}(z)$, $p_m(z)$ does not converge to $I_{[-1,1]}(z)$ uniformly as $m \to \infty$. The lack of uniform convergence is not an issue as long as the filter polynomial separates the wanted and unwanted eigenvalues.

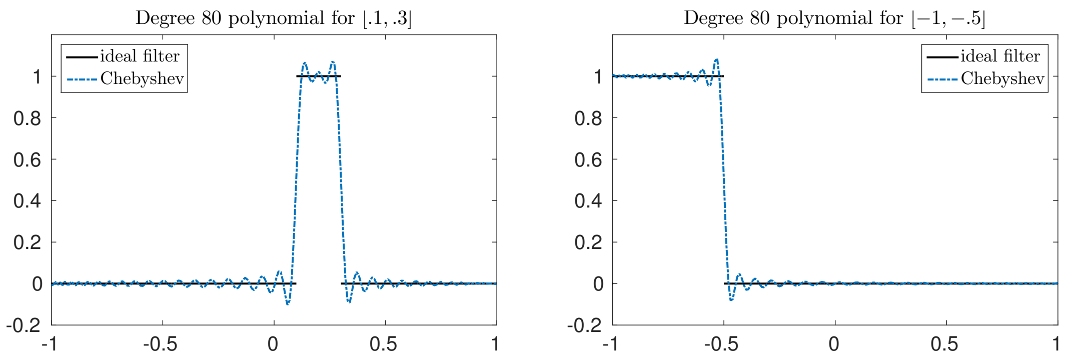


Figure 6.1: Chebyshev approximation of the ideal filter $I_{[-1,1]}(z)$ using a degree 80 polynomial. Left: $[\alpha, \beta] = [.1, .3]$. Right: $[\alpha, \beta] = [-1, -.5]$.

Figure 6.1 shows approximations of the ideal filter $I_{[-1,1]}(z)$ for two different subintervals of $[-1, 1]$, using a fixed degree $m = 80$. In the left subfigure the interval of interest is located around the middle of the spectrum $[\alpha, \beta] = [.1, .3]$, while in the right subfigure the interval of interest is located at the left extreme part $[\alpha, \beta] = [-1, -.5]$. Note that the oscillations near the discontinuities do not prevent the polynomials from separating the spectrum.

Let $M = LL^T$ where $L \in \mathbb{R}^{n \times n}$ is a lower triangular matrix with positive diagonal entries, and let $\overline{L^{-1}AL^{-T}}$ be a scaled version of matrix $L^{-1}AL^{-T}$ such that the spectrum of the latter matrix is mapped to the interval $[-1, 1]$. We can the apply the polynomial filter $p_m(z)$ to $\overline{L^{-1}AL^{-T}}$. Multiplying $p_m(\overline{L^{-1}AL^{-T}})$ by a vector can be done efficiently in parallel using a vectorized version of Clenshaw's algorithm [111] when $p_m(z)$ is represented in a Chebyshev basis. Moreover Clenshaw's algorithm can be run entirely in real arithmetic whenever the Chebyshev coefficients of $p_m(z)$ are real.

6.2.1 Choosing the filter degree m

Assuming that the spectrum of the matrix pencil (A, M) is mapped to $[-1, 1]$, a "good" degree m for $[\alpha, \beta] \subset [-1, 1]$ can be computed using the following formula:

$$m = \min\{m > 0 : ||p_m(z) - I_{[-1,1]}(z)|| < \epsilon ||I_{[-1,1]}(z)||\}, \tag{6.9}$$

where $||f(z)||$ denotes the weighted Chebyshev 2-norm of some function $f(.)$. The tolerance ϵ is a parameter and is chosen experimentally, with the goal of maximizing the separation power of the filter while keeping the polynomial degree and consequently the computation time low.

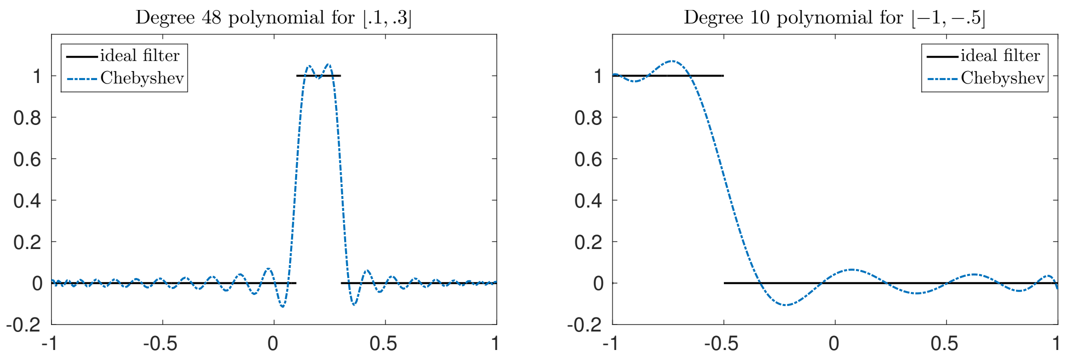


Figure 6.2: Chebyshev and approximation of the ideal filter $I_{[-1,1]}(z)$. Left: $[\alpha, \beta] = [.1, .3]$ with an optimal degree of 48. Right: $[\alpha, \beta] = [-1, -.5]$ with an optimal degree of 10.

Figure 6.2 uses the same ideal filters from Figure 6.1 but this time computes the filter degree based on (6.9). In the left subfigure the interval of interest is located around the middle of the spectrum $[\alpha, \beta] = [.1, .3]$ and the distance between α and β is relatively small, giving a filter degree of 48. In the right subfigure the interval of interest is located at the left extreme part of the spectrum $[\alpha, \beta] = [-1, -.5]$ and the distance α and β is relatively large, giving a filter degree of 10. Although these filters seem like worse approximations than those in Figure 6.1, the lower degrees lead to much shorter computation times.

6.3 Combining polynomial filtering with block Lanczos

Assuming we have constructed a polynomial filter $p_m(z)$, we can approximate eigenvalues of $\overline{L^{-1}AL^{-T}}$ by first approximating eigenvalues and eigenvectors of $p_m(\overline{L^{-1}AL^{-T}})$ using a simple version of the Lanczos method [112]. Many of the matrices arising in practical applications possess repeated eigenvalues, requiring the use of block Lanczos algorithm [113], so we describe the block version of FLP as it contains the standard algorithm as a special case.

Given a block size r and a matrix $Q \in \mathbb{R}^{n\times r}$ with orthonormal columns, the filtered Lanzos procedure iteratively constructs an orthonormal basis for the Krylov subspace generated by $p_m(\overline{L^{-1}AL^{-T}})$ and Q:

$$\mathcal{K}_k\left(p_m(\overline{L^{-1}AL^{-T}}), Q\right) = \textbf{span}\left(\left[Q, p_m(\overline{L^{-1}AL^{-T}})Q, \ldots, p_m(\overline{L^{-1}AL^{-T}})^{k-1}Q\right]\right). \tag{6.10}$$

Let us denote by $Q_k \in \mathbb{R}^{n\times rk}$ the matrix whose columns are generated by $k-1$ steps of the block Lanczos algorithm. Then, for each integer k we have $Q_k^T Q_k = I$ and $\textbf{range}(Q_k) = \textbf{span}\left(\mathcal{K}_k(p_m(\overline{L^{-1}AL^{-T}}), Q)\right)$. Since $p(\overline{L^{-1}AL^{-T}})$ is symmetric the columns of Q_k can be generated using short recurrences. This implies that there exists symmetric $\{D_i\}_{i=1}^k$ and upper-triangular $\{S_i\}_{i=1}^k$, $D_i, S_i \in \mathbb{R}^{r\times r}$, such that

$$p(\overline{L^{-1}AL^{-T}})Q_k = Q_{k+1}\tilde{T}_k, \tag{6.11}$$

where

$$\tilde{T}_k = \begin{bmatrix} T_k \\ S_k E_k^T \end{bmatrix}, \quad T_k = \begin{bmatrix} D_1 & S_1^T & & & \\ S_1 & D_2 & S_2^T & & \\ & S_2 & D_3 & \ddots & \\ & & \ddots & \ddots & S_{k-1}^T \\ & & & S_{k-1} & D_k \end{bmatrix}, \tag{6.12}$$

and $E_k \in \mathbb{R}^{kr\times r}$ denotes the last r columns of the identity matrix of size $kr \times kr$. Left multiplying (6.11) by Q_k^T gives the Rayleigh-Ritz projection

$$Q_k^T p_m(\overline{L^{-1}AL^{-T}})Q_k = T_k. \tag{6.13}$$

The matrix T_k is symmetric and banded, with a semi-bandwidth of size r. The eigenvalues of T_k are the Ritz values of $p_m(\overline{L^{-1}AL^{-T}})$ associated with the subspace spanned by the columns of Q_k and for sufficiently large k the dominant eigenvalues of $p_m(\overline{L^{-1}AL^{-T}})$ will be well approximated by these Ritz values. Of course we aren't actually interested in the eigenvalues of $p_m(\overline{L^{-1}AL^{-T}})$

but those of $\overline{L^{-1}AL^{-T}}$. We can recover these eigenvalues by using the fact that $p_m(\overline{L^{-1}AL^{-T}})$ has the same eigenvectors as $\overline{L^{-1}AL^{-T}}$. Assuming that an eigenvector v of $p_m(\overline{L^{-1}AL^{-T}})$ has been computed accurately we can recover the corresponding eigenvalue λ of $\overline{L^{-1}AL^{-T}}$ from the Rayleigh quotient of v:

$$\lambda = \frac{v^T\overline{L^{-1}AL^{-T}}v}{v^Tv}. \tag{6.14}$$

In practice we will often have only a good approximation $\hat{v}$ of v. The approximate eigenvector $\hat{v}$ will be a Ritz vector of $p_m(\overline{L^{-1}AL^{-T}})$ associated with Q_k. To compute these Ritz vectors we first compute an eigendecomposition of T_k. Since T_k is real and symmetric there exists an orthogonal matrix $W_k \in \mathbb{R}^{rk\times rk}$ and a diagonal matrix $\Lambda_k \in \mathbb{R}^{rk\times rk}$ such that

$$T_kW_k = W_k\Lambda_k. \tag{6.15}$$

Combining (6.11) and (6.15), the Ritz vectors of $p(\overline{L^{-1}AL^{-T}})$ are formed as $\hat{V}_k = Q_kW_k$.

6.4 Experiments

In this section we illustrate the performance of our GPU implementation of the FLP described above. Our test matrices (Hamiltonians) originate from electronic structure calculations. In this setting, one is typically interested in computing a few eigenvalues around the Fermi level of each Hamiltonian. The Hamiltonians were generated using the PARSEC package [64] and can be also found in the SuiteSparse matrix collection. These Hamiltonians are real, symmetric, and have clustered, as well as multiple, eigenvalues. Polynomial filtering has often been reported to be the most efficient numerical method for solving eigenvalue problems with the PARSEC matrix collection [68, 67, 105, 106, 109, 66]. Table 6.1 lists the size n, the total number of non-zero entries nnz, as well as the endpoints of the spectrum of each matrix, i.e., the interval $[\lambda_{\min}, \lambda_{\max}]$. The average number of nonzero entries per row for each Hamiltonian is quite large, a consequence of the high-order discretization and the addition of a (dense) "non-local" term. Figure 6.3 plots the sparsity pattern of matrix "Si41Ge41H72".

All GPU experiments in this section were implemented using the Cucheb library and performed on a machine which has an Intel Xeon E5-2680 v3 2.50GHz processor with 128GB of CPU RAM and two Nvidia K40 GPUs each with 12GB of GPU RAM and 2880 computer cores.

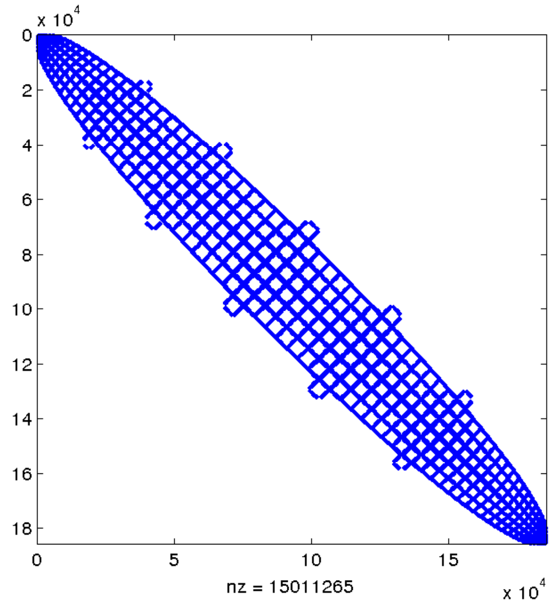


Figure 6.3: Sparsity pattern of matrix Si41Ge41H72.

We made no attempt to access mutliple GPUs and all the experiments were performed using a single K40.

Matrix	n	nnz	nnz/n	Spectral interval
`Ge87H76`	$112,985$	$7,892,195$	69.9	$[-1.21e{+}0,\quad 3.28e{+}1]$
`Ge99H100`	$112,985$	$8,451,395$	74.8	$[-1.23e{+}0,\quad 3.27e{+}1]$
`Si41Ge41H72`	$185,639$	$15,011,265$	80.9	$[-1.21e{+}0,\quad 4.98e{+}1]$
`Si87H76`	$240,369$	$10,661,631$	44.4	$[-1.20e{+}0,\quad 4.31e{+}1]$
`Ga41As41H72`	$268,096$	$18,488,476$	69.0	$[-1.25e{+}0,\quad 1.30e{+}3]$

Table 6.1: A list of the PARSEC matrices used to evaluate our GPU implementation, where n is the dimension of the matrix, nnz is the number of nonzero entries and $[\lambda_{\min}, \lambda_{\max}]$ is the spectral interval.

6.4.1 GPU benchmarking

The results of the GPU experiments are summarized in Table 6.2. The variable ‘interval’ for each Hamiltonian was set so that it included roughly the same number of eigenvalues from the left and right side of the Fermi level, and in total ‘eigs’ eigenvalues. For each matrix and interval $[\alpha, \beta]$ we repeated the same experiment five times, each time using a different degree m for the filter polynomial. The variable 'iters' shows the number of FLP iterations, while 'MV' shows the total number of matrix-vector products (MV) with A, which is computed using the formula 'MV' $= rm \times$ 'iters'. Throughout this section, the block size of the FLP will be equal to $r = 3$. Finally, the variables 'time' and 'residual' show the total compute time and maximum relative residual of the computed eigenpairs. The first four rows for each matrix correspond to executions where the degree m was selected a priori. The fifth row corresponds to an execution where the degree was selected automatically by our implementation, using the mechanism described in (6.9). As expected, using larger values for m leads to faster convergence in terms of total iterations, since higher degree filters are better at separating the wanted and unwanted portions of the spectrum. Although larger degrees lead to less iterations, the amount of work in each filtered Lanczos iteration is also increasing proportionally. This might lead to an increase of the actual computational time, an effect verified for each one of the matrices in Table 6.2.

Table 6.3 compares the percentage of total computation time required by the different subprocesses of the FLP method. We denote the preprocessing time, which consists solely of approximating the upper and lower bounds of the spectrum for A, by 'PREPROC'. We also denote the total amount of time spent in the full reorthogonalization and the total amount of time spent in performing all MV products of the form $p_m(\overline{L^{-1}AL^{-T}})v$ on the GPU, by 'ORTH' and 'MV' respectively. As we can verify, all matrices in this experiment devoted no more than 12% of the total compute time to estimating the spectral interval (i.e. the algebraically smallest/largest eigenvalues $\lambda_{\min}/\lambda_{\max}$ of the matrix pencil (A, M)). For each one of the PARSEC test matrices, the dominant cost came from the MV products, due to their relatively large number of non-zero entries. Note that using a higher degree m will shift the cost more towards the MV products, since the Lanczos procedure will typically converge in fewer outer steps and thus the orthogonalization cost reduces.

We would like to note that the Cucheb software package is capable of running Lanczos without filtering. We originally intended to compare filtered Lanczos with standard Lanczos on the GPU, however for the problems considered in this chapter the number of Lanczos vectors required for convergence exceeded the memory of the K40 GPU. This suggests that for these particular problems filtering is not only beneficial for performance but also necessary if this

Matrix	interval	nev	m	iters	MV	time	residual
Ge87H76	$[-0.645, -0.0053]$	212	50	210	$31,500$	31	$1.7e{-14}$
			100	180	$54,000$	40	$4.0e{-13}$
			150	150	$67,500$	44	$7.4e{-14}$
			200	150	$90,000$	56	$6.3e{-14}$
			49	210	$30,870$	31	$9.0e{-14}$
Ge99H100	$[-0.650, -0.0096]$	250	50	210	$31,500$	32	$6.2e{-13}$
			100	180	$54,000$	41	$8.6e{-13}$
			150	180	$81,000$	56	$5.0e{-13}$
			200	180	$108,000$	70	$1.1e{-13}$
			49	210	$30,870$	32	$3.2e{-13}$
Si41Ge41H72	$[-0.640, -0.0028]$	218	50	210	$31,500$	56	$6.4e{-13}$
			100	180	$54,000$	73	$2.0e{-11}$
			150	180	$81,000$	99	$5.6e{-14}$
			200	150	$90,000$	104	$5.0e{-13}$
			61	180	$32,940$	52	$8.9e{-13}$
Si87H76	$[-0.660, -0.3300]$	107	50	150	$22,500$	38	$3.5e{-14}$
			100	90	$27,000$	35	$4.0e{-15}$
			150	120	$54,000$	63	$9.1e{-15}$
			200	90	$54,000$	60	$1.3e{-13}$
			98	90	$26,460$	35	$1.2e{-14}$
Ga41As41H72	$[-0.640, 0.0000]$	201	200	240	$144,000$	225	$1.5e{-15}$
			300	180	$162,000$	236	$2.1e{-15}$
			400	180	$216,000$	306	$2.5e{-15}$
			500	180	$270,000$	375	$1.0e{-12}$
			308	180	$166,320$	242	$1.5e{-15}$

Table 6.2: Computing the eigenpairs inside an interval using FLP with various filter polynomial degrees. Times listed are in seconds.

particular hardware is used.

6.4.2 CPU-GPU comparison

Figure 6.4 shows the speedup of the GPU FLP implementation over the CPU-based counterpart. The CPU results were obtained by executing the FILTLAN software package on the `Mesabi` linux cluster at University of Minnesota Supercomputing Institute. The FILTLAN package has the

Matrix	m	iters	PREPROC	ORTH	MV
Ge87H76	50	210	7%	22%	52%
	100	180	5%	13%	71%
	150	150	5%	9%	80%
	200	150	4%	7%	84%
	49	210	7%	21%	52%
Ge99H100	50	210	7%	21%	53%
	100	180	5%	13%	71%
	150	180	4%	10%	79%
	200	180	3%	8%	83%
	49	210	7%	21%	53%
Si41Ge41H72	50	210	10%	19%	55%
	100	180	8%	12%	72%
	150	180	6%	9%	80%
	200	150	5%	6%	84%
	61	180	11%	17%	61%
Si87H76	50	150	11%	22%	54%
	100	90	12%	12%	70%
	150	120	7%	10%	78%
	200	90	7%	7%	83%
	98	90	12%	13%	70%
Ga41As41H72	200	240	4%	8%	82%
	300	180	4%	5%	88%
	400	180	3%	4%	91%
	500	180	2%	3%	93%
	308	180	4%	5%	89%

Table 6.3: Percentage of total compute time required by various components of the algorithm. For all these examples the dominant computational cost are the matrix-vector multiplications (MV).

option to link the Intel Math Kernel Library (MKL) when it, as well as a compatible Intel compiler are available. For these experiments we used the Intel compiler `icpc` version 11.3.2.

We have divided the comparison into four parts: a "low degree" situation when $m = 50$ ($m = 200$ for `Ga41As41H72`), and a "high degree" situation when $m = 100$ ($m = 300$ for `Ga41As41H72`) and within each of these we also executed FILTLAN using both 1 thread and 24 threads. The multithreading was handled entirely by the MKL. In the single thread case, the GPU implementation obtains a speedup which ranges between 10 and 14. In the 24 thread case, which corresponds to one thread per core on this machine, the speedups ranged between 2 and 3.

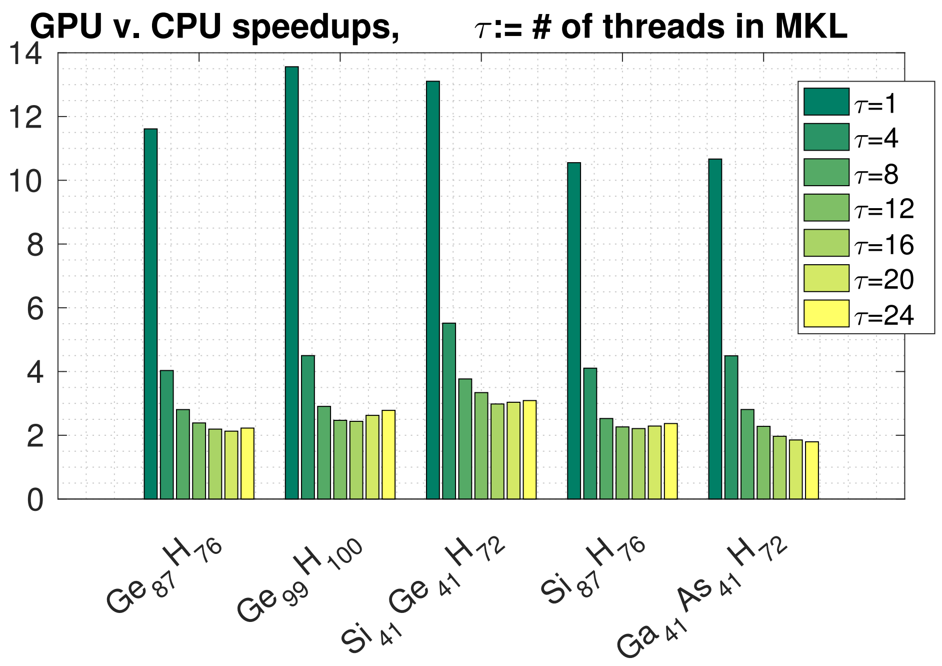


Figure 6.4: Speedup of the GPU FLP implementation over the CPU (FILTLAN) for the PARSEC test matrices.

6.5 Summary

In this chapter we presented a GPU implementation of the filtered Lanczos procedure for solving large and sparse eigenvalue problems such as those that arise from real-space DFT methods in electronic structure calculations. Our experiments indicate that the use of GPU architectures in the context of electronic structure calculations can provide a speedup of at least a factor of 10 over a single core CPU-based implementation, and at least of factor of 2 over a 24 core CPU-based implementation.

Part II

Root-finding techniques

Chapter 7

The method of single linear approximations

This chapter presents domain decomposition algorithms a root-finding technique based on spectral Schur complements.[1] In contrast with filtering techniques described so far, root-finding techniques approximate each sought eigenpair of the matrix pencil (A, M) independently of each other. The main idea is to compute the pair $\left(\lambda_i, y^{(i)}\right)$ of each sought eigenpair $\left(\lambda_i, x^{(i)}\right)$ of (A, M), $x^{(i)} = \left[\left(u^{(i)}\right)^T, \left(y^{(i)}\right)^T\right]^T$, by recasting the interface eigenvalue problem $S(\lambda_i)y(\lambda_i) = 0$ into one of a root-finding. This root-finding problem can be solved by Newton's method. The remaining part of $x^{(i)}$, $u^{(i)}$, is then computed by solving the linear system $B_{\lambda_i}^{-1}u^{(i)} = -E_\lambda y^{(i)}$.

This chapter is organized as follows: Section 7.1 introduces the concept of *eigenbranches*; a parameterization of the eigenvalues of the matrix-valued function $S(\sigma)$. Section 7.2 proposes a Newton-based scheme for the computation of a single eigenpair of the matrix pencil (A, M). Section 7.3 provides insight on the behavior of the eigenbranches as they cross the poles of $S(\sigma)$. Section 7.4 discusses a generalization to the computation of more than one eigenpairs of the matrix-pencil (A, M). Section 7.5 presents numerical experiments on distributed memory environments. Finally, a summary is given in Section 7.6.

[1]This is joint work with Ruipeng Li (Lawrence Livermore National Laboratory) and Yousef Saad (University of Minnesota, Twin Cities)

7.1 Eigenbranches

Consider the following parameterized symmetric eigenvalue problem, stemming by a zero-order approximation of $S(\lambda)$ around a real scalar $\sigma \in \mathbb{R}$:

$$S(\sigma)y_j(\sigma) = \mu_j(\sigma)y_j(\sigma),$$

where $\mu_j(\sigma)$ denotes the jth, $j = 1, \ldots, s$, eigenvalue of the pencil $S(\sigma)$ in a sorted (algebraic) ascending order, and $y_j(\sigma)$ denotes the corresponding eigenvector.

Definition 7.1.1 *The scalar function $\mu_j(\sigma) : \mathbb{R} \rightarrow \mathbb{R}$,*

$$\mu_j(\sigma) = \frac{y_j^T(\sigma)S(\sigma)y_j(\sigma)}{y_j^T(\sigma)y_j(\sigma)}, \tag{7.1}$$

will be referred to as the j'th eigenbranch of $S(\sigma)$, $j = 1, \ldots, s$. Throughout this chapter, the numbering of the eigenbranches will be based on their algebraic value, i.e., $\mu_1(\sigma) \leq \ldots \leq \mu_s(\sigma)$.

Moreover, let $\mu_j(\sigma)$, $j = 1, \ldots, s$ be defined as in (7.1), and consider the following integer:

$$\kappa_\sigma = \arg\min_{1 \leq j \leq s} |\mu_j(\sigma)|. \tag{7.2}$$

We then define the functions $\mu(\sigma) : \mathbb{R} \rightarrow \mathbb{R}$ and $y(\sigma) : \mathbb{R} \rightarrow \mathbb{R}^s$,

$$\mu(\sigma) := \mu_{\kappa_\sigma}(\sigma), \quad y(\sigma) := y_{\kappa_\sigma}(\sigma),$$

i.e., $(\mu(\sigma), y(\sigma))$ denotes the eigenpair associated with the eigenvalue of smallest magnitude of the matrix pencil $S(\sigma)$.

The main idea behind the numerical approach presented in this chapter is that a scalar $\sigma \in \mathbb{R}$ will be an eigenvalue of (A, M) if and only if $\mu(\sigma) = 0$, i.e., $S(\sigma)$ is singular. The question now becomes how to find a σ for which $S(\sigma)$ is singular. One idea, adopted in [17], is to consider the equation $\det(S(\sigma)) = 0$ but this is not practical for large problems. On the other hand, $S(\sigma)$ will be singular exactly when at least one of $\mu_j(\sigma)$, $j = 1, \ldots, s$ is zero. With this, the original eigenvalue problem can be reformulated as that of finding a shift σ such that the eigenvalue of smallest magnitude, $\mu(\sigma)$, of $S(\sigma)$ is zero.

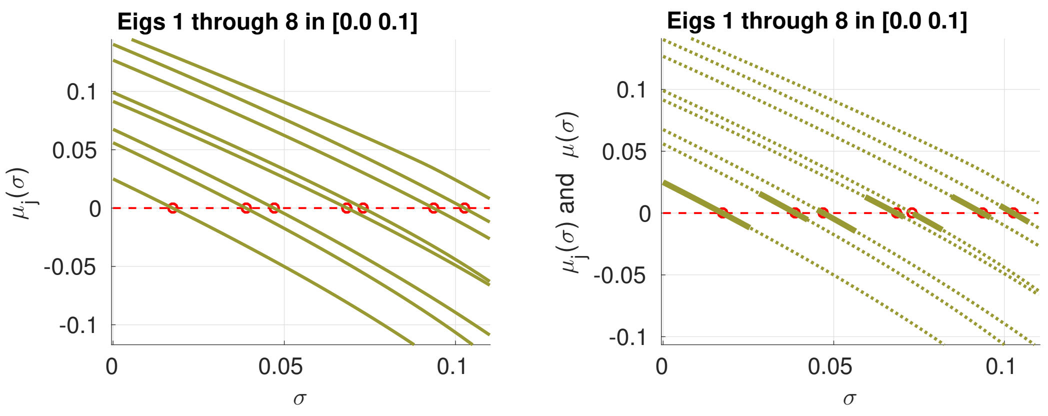


Figure 7.1: Visualization of $\mu_j(\sigma)$ and $\mu_{\kappa_\sigma}(\sigma)$ for a 2D Laplacian matrix. The red circles along the real axis denote eigenvalues of the pencil (A, M). Left: Eigenbranches $\mu_j(\sigma),\ j = 1, \ldots, 8$. Right: Eigenbranches $\mu_j(\sigma),\ j = 1, \ldots, 8$ (dotted lines) and $\mu_{\kappa_\sigma}(\sigma) \equiv \mu(\sigma)$ (solid line). Each eigenvalue of (A, M) is a root of (at least) one eigenbranch, and thus a root of $\mu(\sigma)$ as well.

The advantage of working directly with $\mu(\sigma)$ is that it allows the computation of any eigenvalue λ of (A, M) without the explicit knowledge of the particular index $1 \leq j \leq s$ for which $\delta_j(\lambda) = 0$. In this chapter we consider Newton's method as the root-finding technique [114]. One potential complication with this choice is that $\mu(\sigma)$ is not differentiable across its entire domain of definition. This can be easily verified by Figure 7.1 (right subfigure) where we plot the function $\mu(\sigma)$ (solid line) on top of the first few eigenbranches $\mu_j(\sigma)$ (dotted lines). As we can observe, the index κ_σ defined in (7.2) does not remain constant as σ varies. Nonetheless, Newton's method can still be applied if during its application the integer κ_σ remains fixed. This follows from the analyticity of the eigenbranches.

Proposition 7.1.1 *For any $\sigma \in \mathbb{R}$, the first derivative of $S(\sigma)$, $S'(\sigma)$, is given by*

$$S'(\sigma) = \frac{dS(\sigma)}{d\sigma} = -M_C - E_\sigma^T B_\sigma^{-1} M_B B_\sigma^{-1} E_\sigma + E_\sigma^T B_\sigma^{-1} M_E + M_E^T B_\sigma^{-1} E_\sigma. \tag{7.3}$$

Moreover, the matrix $-S'(\sigma)$ is SPD.

Proof: Let $\omega \in \mathbb{R}$. We can write

$$\begin{aligned} S(\sigma+\omega) &= C_\sigma - \omega M_C - (E_\sigma - \omega M_E)^T (B_\sigma - \omega M_B)^{-1} (E_\sigma - \omega M_E) \\ &= C_{\sigma+\omega} - E_\sigma^T B_{\sigma+\omega}^{-1} E_\sigma + \omega \left(E_\sigma^T B_{\sigma+\omega}^{-1} M_E + M_E^T B_{\sigma+\omega}^{-1} E_\sigma - \omega M_E^T B_{\sigma+\omega}^{-1} M_E \right) \end{aligned} \tag{7.4}$$

Taking the derivative of $S(\sigma+\omega)$ with respect to ω leads to:

$$S'(\sigma+\omega) = -M_C + E_\sigma^T B_{\sigma+\omega}^{-1} M_E + M_E^T B_{\sigma+\omega}^{-1} E_\sigma - E_\sigma^T \left[B_{\sigma+\omega}^{-1}\right]' E_\sigma + \mathcal{R}(\omega),$$

where

$$\mathcal{R}(\omega) = \omega M_E^T \left[B_{\sigma+\omega}^{-1}\right]' E_\sigma + \omega E_\sigma^T \left[B_{\sigma+\omega}^{-1}\right]' M_E - 2\omega M_E^T B_{\sigma+\omega}^{-1} M_E - \omega^2 M_E^T \left[B_{\sigma+\omega}^{-1}\right]' M_E,$$

and

$$B_{\sigma+\omega}^{-1} = B_\sigma^{-1} \sum_{\ell=0} \left[\omega M_B B_\sigma^{-1}\right]^\ell.$$

The result in (7.3) follows by setting $S'(\sigma) = S'(\sigma+\omega)_{\omega=0}$ and noticing that

$$\left[B_{\sigma+\omega}^{-1}\right]'_{\omega=0} = B_\sigma^{-1} M_B B_\sigma^{-1},$$

and $\mathcal{R}(0) = 0$.

To show the second item, let matrix $U_\sigma \in \mathbb{R}^{n\times n}$ be defined as

$$U_\sigma = \begin{pmatrix} I & -B_\sigma^{-1} E_\sigma \\ 0 & I \end{pmatrix}.$$

Then,

$$\begin{aligned} U_\sigma^T M U_\sigma &= \begin{pmatrix} I & 0 \\ E_\sigma^T B_\sigma^{-1} & I \end{pmatrix} \begin{pmatrix} M_B & M_E \\ M_E^T & M_C \end{pmatrix} \begin{pmatrix} I & B_\sigma^{-1} E_\sigma \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} M_B & M_E - M_B B_\sigma^{-1} E_\sigma \\ M_E^T - E_\sigma^T B_\sigma^{-1} M_B & T \end{pmatrix}, \end{aligned}$$

where

$$T = M_C - M_E^T B_\sigma^{-1} E_\sigma - E_\sigma^T B_\sigma^{-1} M_E + E_\sigma^T B_\sigma^{-1} M_B B_\sigma^{-1} E_\sigma$$

is a principal submatrix of $U_\sigma^T M U_\sigma$. Recalling that M is SPD (and thus $U_\sigma^T M U_\sigma$ is also SPD) and noticing that $T = -S'(\sigma)$ concludes the proof. □

Proposition 7.1.2 *The eigenbranches $\mu_j(\sigma),\ j = 1, \ldots, s$ are analytic at any point $\sigma \notin \Lambda(B, M_B)$,* *where $\Lambda(B, M_B)$ denotes the spectrum of (B, M_B). The derivative of each eigenbranch at these points is given by*

$$\mu_j'(\sigma) = \frac{d\mu_j(\sigma)}{d\sigma} = -\frac{y_j^T(\sigma)S'(\sigma)y_j(\sigma)}{y_j^T(\sigma)y_j(\sigma)} \tag{7.5}$$

Proof: Differentiating $S(\sigma)y_j(\sigma) = \mu_j(\sigma)y_j(\sigma)$ with respect to σ leads to

$$S'(\sigma)y_j(\sigma) + S(\sigma)y_j'(\sigma) = \mu_j'(\sigma)y_j(\sigma) + \mu_j(\sigma)y_j'(\sigma),$$

where after collecting terms we get

$$(S(\sigma) - \theta_j(\sigma)I)y_j'(\sigma) = -S'(\sigma)y_j(\sigma) - \mu_j'(\sigma)y_j(\sigma).$$

Multiplying by $y_j^T(\sigma)$ from the left, and noticing that $y_j^T(\sigma)(S(\sigma) - \theta_j(\sigma)I) = 0$, leads to (7.5).

The analyticity of $\mu_j(\sigma)$ and $y_j(\sigma)$ follows directly from [115] (see also [116]) with the results also holding for multiple (semi-simple) eigenvalues. □

Corollary 7.1.2.1 *The eigenbranches $\mu_j(\sigma)$, $j = 1, \ldots, s$, are strictly decreasing for any $\sigma \in \mathbb{R}$.*

7.2 An algorithm for computing a single eigenpair

Algorithm 7.2.1 sketches the main steps of Newton's root-finding applied to the scalar function $\mu(\sigma)$. For any σ, Algorithm 7.2.1 computes the eigenpair ($\mu(\sigma)$,$y(\sigma)$) of the matrix $S(\sigma)$, and evaluates the derivative $\mu'(\sigma)$ to update σ. The algorithm terminates as soon as $|\mu(\sigma)|$ becomes smaller than a threshold tolerance tol, which, together with an initial approximation of the sought eigenvalue, are the only inputs of Algortihm 7.2.1.

Algorithm **7.2.1** *Root-finding eigenvalue solver*

```
0.  Given $\mathrm{tol} \in \mathbb{R}$ and an initial approximation $\sigma \in \mathbb{R}$
1.  Do while (true):
2.      Solve $S(\sigma)y(\sigma) = \mu(\sigma)y(\sigma)$ for the eigenvalue $\mu(\sigma)$
        of smallest magnitude and associated eigenvector $y(\sigma)$
3.      If $|\mu(\sigma)| \leq \mathrm{tol}$, break;
4.      Compute $\mu'(\sigma) = -\frac{y^T(\sigma)S'(\sigma)y(\sigma)}{y^T(\sigma)y(\sigma)}$
5.      Update $\sigma = \sigma - \frac{\mu(\sigma)}{\mu'(\sigma)}$
6.  EndDo
7.  Return $\sigma$ and $\hat{x} = \left[-(B_\sigma^{-1}E_\sigma y(\sigma))^T, y^T(\sigma)\right]^T$
```

Algorithm 7.2.1 does not specify how to extract the eigenpair $(\mu(\sigma), y(\sigma))$ in Step (3) for any σ. All that is needed is the eigenvalue of $S(\sigma)$ closest to zero and its associated eigenvector. This is a perfect setting for a form of inverse iteration [117]. Alternatively, we can use the Lanczos method with partial orthogonalization and perform as many steps as needed to compute $(\mu(\sigma), y(\sigma))$.

Proposition 7.2.1 *Let $\sigma \notin \{\Lambda(A, M) \cup \Lambda(B, M_B)\}$ and let $\mu^{(s)}(\sigma)$ denote the eigenvalue of smallest magnitude of matrix $S_s(\sigma)$ where $1 \leq s < n$ denotes the size of the matrix $S(\sigma)$, i.e., the total number of interface variables. Then*

$$\left|\mu^{(s_2)}(\sigma)\right| \leq \left|\mu^{(s_1)}(\sigma)\right|$$

for any $1 \leq s_1 < s_2 \leq n-1$.

Proof: Recall the identity

$$S(\sigma)^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i - \sigma} y^{(i)} \left(y^{(i)}\right)^T,$$

where $y^{(i)}$ is the bottom $s \times 1$ part of the ith eigenvector $x^{(i)} = \left[\left(u^{(i)}\right)^T, \left(y^{(i)}\right)^T\right]^T$ of the matrix pencil (A, M). The eigenvalue equation $S(\sigma)y(\sigma) = \mu(\sigma)y(\sigma)$ can be written equivalently as $S(\sigma)^{-1}y(\sigma) = \dfrac{1}{\mu(\sigma)}y(\sigma)$. By the Courant-Fischer-Weyl min-max principle we then have either $\dfrac{1}{\mu(\sigma)} = \min_{\|r\|=1} r^T S(\sigma)^{-1} r$ (if $\mu(\sigma) < 0$) or $\dfrac{1}{\mu(\sigma)} = \max_{\|r\|=1} r^T S(\sigma)^{-1} r$ (if $\mu(\sigma) > 0$).

Now, let $S_{s_1}(\sigma)$ and $S_{s_2}(\sigma)$ be the Schur complement matrices associated with two different values for the number of interface variables (i.e., the size of matrix C_σ), and let $1 \leq s_1 < s_2 \leq n-1$. In addition, denote the corresponding vectors $y^{(i)}$ of eigenvector $x^{(i)}$ of the matrix pencil (A, M) as $y_{s_1}^{(i)}$ and $y_{s_2}^{(i)}$, respectively. Then

$$\begin{aligned} S_{s_2}(\sigma)^{-1} &= \sum_{i=1}^{n} \frac{1}{\lambda_i - \sigma} y_{s_2}^{(i)} \left(y_{s_2}^{(i)}\right)^T \\ &= \begin{pmatrix} * & * \\ * & \sum_{i=1}^{n} \dfrac{1}{\lambda_i - \sigma} y_{s_1}^{(i)} \left(y_{s_1}^{(i)}\right)^T \end{pmatrix} \\ &= \begin{pmatrix} * & * \\ * & S_{s_1}(\sigma)^{-1} \end{pmatrix}, \end{aligned}$$

i.e., matrix $S_{s_1}(\sigma)^{-1}$ is a principal submatrix of matrix $S_{s_2}(\sigma)^{-1}$. Therefore, by Cauchy's matrix interlacing theorem we get

$$\begin{aligned} \left|\frac{1}{\mu^{(s_2)}(\sigma)}\right| &\geq \left|\frac{1}{\mu^{(s_1)}(\sigma)}\right| \\ \left|\mu^{(s_2)}(\sigma)\right| &\leq \left|\mu^{(s_1)}(\sigma)\right|. \end{aligned}$$

□

Figure 7.2 plots the values of $\left|\mu^{(s)}(\sigma)\right|$ as σ and s vary for a matrix obtained by a 5-pt stencil discretization of the Laplace operator in $[0,1] \times [0,1]$. Regardless of the value of σ, $\left|\mu^{(s)}(\sigma)\right|$ is a decreasing function of s.

7.2.1 An equivalent update scheme for Newton's method

Since Algorithm 7.2.1 is Newton's method, we expect that if the initial shift σ is "close enough" to an eigenvalue λ of (A, M), Algorithm 7.2.1 will converge quadratically to λ [114]. By Proposition 7.1.2, each eigenbranch is an analytic branch and the first derivative is always non-zero (bounded from above by -1). In addition the second derivative of $\mu(\sigma)$ is finite for any $\sigma \notin \Lambda(B, M_B)$.

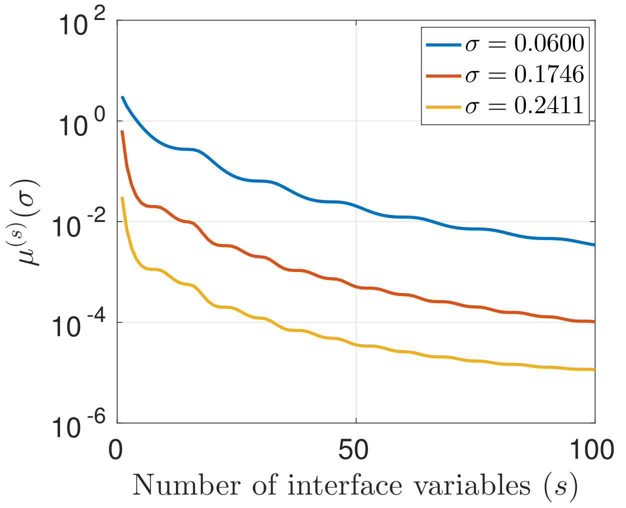


Figure 7.2: Plot of $\left|\mu^{(s)}(\sigma)\right|$ for different values of σ and s.

Therefore, when the scheme converges, it will do so quadratically.

Let

$$\hat{x}(\sigma) = \begin{bmatrix} -B_\sigma^{-1}E_\sigma y(\sigma) \\ y(\sigma) \end{bmatrix}. \tag{7.6}$$

be the approximate eigenvector associated with the approximate eigenvalue σ of the matrix pencil (A, M). The approximate eigenpair $(\sigma, \hat{x}(\sigma))$ has a special residual. Indeed,

$$(A - \sigma M)\hat{x}(\sigma) = \begin{pmatrix} B_\sigma & E_\sigma \\ E_\sigma^T & C_\sigma \end{pmatrix} \begin{pmatrix} -B_\sigma^{-1}E_\sigma y(\sigma) \\ y(\sigma) \end{pmatrix} = \begin{pmatrix} 0 \\ \mu(\sigma)y(\sigma) \end{pmatrix}. \tag{7.7}$$

Proposition 7.2.2 *Let $\sigma_{j+1} = \sigma_j + \mu(\sigma_j)/(1+\eta_j^2)$ be the Newton update at the j'th step of Algorithm 7.2.1, where we set $\eta_j = \|B_{\sigma_j}^{-1}E_{\sigma_j}y(\sigma_j)\|_2$. Then, $\sigma_{j+1} = \rho(A, M, \hat{x}(\sigma_j))$, where $\rho(A, M, \hat{x}(\sigma_j)) = \dfrac{\hat{x}^T(\sigma_j)A\hat{x}(\sigma_j)}{\hat{x}^T(\sigma_j)M\hat{x}(\sigma_j)}$ is the Rayleigh Quotient of (A, M) with respect to the vector $\hat{x}(\sigma_j)$ defined by (7.6).*

Proof: For simplicity, set $\hat{x} \equiv \hat{x}(\sigma_j)$ and assume, without loss of generality, that $\|y(\sigma_j)\| = 1$. We can write $\rho(A, M, \hat{x}) = \sigma_j + \rho(A - \sigma_j M, M, \hat{x})$. The right term of the right-hand is equal to

$$\rho(A - \sigma_j M, \hat{x}) = \frac{\hat{x}^T(A - \sigma_j M)\hat{x}}{\hat{x}^T M\hat{x}}. \tag{7.8}$$

The expressions (7.6) and (7.7) show that $\hat{x}^T(A - \sigma_j M)\hat{x} = \mu(\sigma_j)$ while $\hat{x}^T\hat{x} = 1 + \eta_j^2$. Thus, $\rho(A - \sigma_j M, M, \hat{x}) = \mu(\sigma_j)/(1 + \eta_j^2)$ and so $\rho(A, M, \hat{x}) = \sigma_j + \mu(\sigma_j)/(1 + \eta_j^2) = \sigma_{j+1}$. □

Note that the equivalent update formula discussed in Proposition 7.2.2 is primarily of theoretical interest. In practice, we use the update formula provided in Step 5 of Algorithm 7.2.1.

When $\mu(\sigma) = 0$ then σ is an eigenvalue of the matrix pencil (A, M). In general, we expect that the closer $\mu(\sigma)$ is to zero, the closer the approximate eigenpair $(\sigma, \hat{x}(\sigma))$ should be to an eigenpair (λ, x) of the matrix pencil (A, M).

Proposition 7.2.3 *Let $\hat{x}(\sigma)$ be defined as in (7.6). Then,*

$$\frac{\|A\hat{x}(\sigma) - \sigma M\hat{x}(\sigma)\|}{\|\hat{x}(\sigma)\|} \leq |\mu(\sigma)|.$$

Proof: The relation in (7.7) shows immediately that:

$$\frac{\|A\hat{x}(\sigma) - \sigma M\hat{x}(\sigma)\|}{\|\hat{x}(\sigma)\|} = \frac{|\mu(\sigma)|}{\sqrt{1+\eta^2}}. \tag{7.9}$$

where $\eta = \|B_\sigma^{-1} E_\sigma y(\sigma)\|$. □

7.3 Eigenbranches across the poles

In the following we will refer to the eigenvalues of the matrix pencil (B, M_B) simply as the "poles". An interesting property is revealed when observing the eigenvalues across the poles. While the matrix $S(\sigma)$ is not defined at a pole, the plot reveals that individual eigenvalues may exist and, *quite interestingly, seem to define differentiable branches across the poles.* Informally, we can say that in this situation $S(\sigma)$ has one infinite eigenvalue and $s - 1$ finite ones.

To explain this observation, let δ_1,. . .,δ_d be the eigenvalues of (B, M_B) with associated eigenvectors $v_1, \ldots, v_d$, and consider $S(\sigma)$ written in the following form

$$S(\sigma) = C_\sigma - E_\sigma^T B_\sigma^{-1} E_\sigma = C_\sigma - \sum_{j=1}^{d} \frac{w_j w_j^T}{\theta_j - \sigma}, \quad \text{with} \quad w_j \equiv E_\sigma^T v_j. \tag{7.10}$$

We assume for simplicity that δ_k is a simple eigenvalue in what follows. The operator $S(\sigma)$ is not formally defined when σ equals one eigenvalue δ_k of (B, M_B). However, it can be defined on a restricted subspace, namely the subspace $\{w_k\}^\perp$ and eigenvalues of this restricted operator

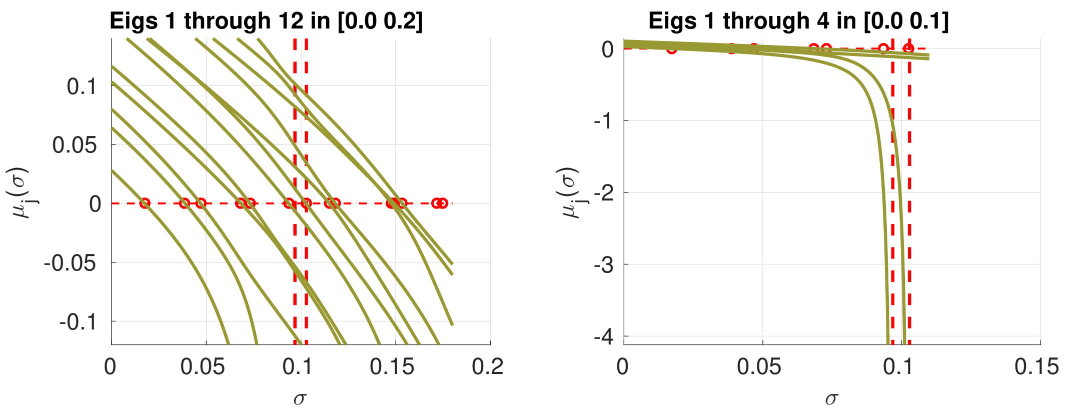


Figure 7.3: Visualization of $\mu_j(\sigma)$ and $\mu_{\kappa_\sigma}(\sigma)$ for a 2D Laplacian matrix. The red circles along the real axis denote eigenvalues of the pencil (A, M). The red dashed vertical lines denote eigenvalues of the pencil (B, M_B). Left: eigenbranches $\mu_j(\sigma)$, $j = 1, \ldots, 12$ chased across panels defined by consecutive eigenvalues of the pencil (B, M_B). Right: focus on eigenbranches $\mu_1(\sigma)$ and $\mu_2(\sigma)$ as σ crosses the two algebraically smallest eigenvalues of the pencil (B, M_B).

are finite.

Accordingly we let $\hat{w}_k = w_k/\|w_k\|$ and define the orthogonal projector

$$P_k = I - \hat{w}_k\hat{w}_k^T \tag{7.11}$$

and

$$S_k(\sigma) \equiv C_\sigma - \sum_{j=1, j\neq k}^{d} \frac{w_j w_j^T}{\delta_j - \sigma}\ , \qquad S_{k,|}(\sigma) \ = \ [P_k S_k(\sigma) P_k]_{|{w_k}^\perp}\ , \tag{7.12}$$

where$[P_k S_k(\sigma) P_k]_{|{w_k}^\perp}$ denotes the restriction of $P_k S_k(\sigma) P_k$ to the subspace orthogonal to w_k.

The operator $S_{k,|}(\sigma)$ defined above is an operator from $\mathbb{R}^{s-1}$ to itself that acts only on vectors of ${w_k}^\perp$. We denote its eigenvalues, also labeled increasingly, by $\mu_j(S_{k,|}(\sigma))$. Apart from an extra zero eigenvalue, the spectrum of $P_k S_k(\sigma) P_k$ is identical with that of $S_{k,|}(\sigma)$. The next theorem examines closely the eigenvalues of $S(\sigma)$ as σ converges to δ_k from the left or the right direction.

Theorem 7.3.1 *When δ_k is a simple eigenvalue then the following equalities hold:*

$$\lim_{\sigma\to\delta_k^-} \mu_j(\sigma) = \begin{cases} -\infty & \text{if} \quad j = 1 \\ \mu_{j-1}(S_{k,|}) & \text{if} \quad j > 1 \end{cases}, \ \lim_{\sigma\to\theta_k^+} \mu_j(\sigma) = \begin{cases} +\infty & \text{if} \quad j = s \\ \mu_j(S_{k,|}) & \text{if} \quad j < s \end{cases}. \tag{7.13}$$

Proof: Consider the first part of the theorem ($\sigma \to \delta_k^-$) and assume that σ is in an interval $[\sigma_0, \delta_k]$ that contains no other poles than δ_k. We define for any nonzero vector r the two Rayleigh quotients:

$$\rho(\sigma, r) = \frac{(S(\sigma)r, r)}{(r, r)}, \qquad \rho_k(\sigma, r) = \frac{(S_k(\sigma)r, r)}{(r, r)} \ . \tag{7.14}$$

Note that from (7.10) we have the following relation for a vector r of unit length:

$$\rho(\sigma, r) = \rho_k(\sigma, r) - \frac{(w_k^T r)^2}{\delta_k - \sigma}. \tag{7.15}$$

For $j = 1$ we have $\mu_1(\sigma) = \min_{r \neq 0} \rho(\sigma, r)$. By taking $r = w_k / \|w_k\|$ in (7.15), the term $-(w_k^T r)^2/(\delta_k - \sigma)$ can be made arbitrarily large and negative when $\sigma \to \delta_k^-$. Hence, the minimum of $\rho(\sigma, r)$ will have a limit of $-\infty$ when $\sigma \to \delta_k^-$. The vector w_k can be viewed as an eigenvector associated with this infinite 'eigenvalue'.

Consider now the situation when $j > 1$. We first show that the limit of $\mu_j(\sigma)$ is finite. For this we invoke the Min-Max theorem [118]:

$$\mu_j(\sigma) = \min_{\mathcal{U}^j, \mathbf{dim}(\mathcal{U}^j) = j} \ \max_{r \in \mathcal{U}^j, \|r\| = 1} \rho(\sigma, r). \tag{7.16}$$

Take any subspace $\mathcal{U}^j$ of dimension j. Since $\mathcal{U}^j$ is of dimension $j > 1$ and $\dim\{w_k\}^\perp = s - 1$, there is a nonzero vector in the intersection $\mathcal{U}^j \cap \{w_k\}^\perp$. Note also that for any $\sigma \in [\sigma_0, \delta_k]$, $S_k(\sigma)$ has no poles and so the term $\rho_k(\sigma, r)$ is bounded from below by a certain (finite) value η for any r of unit length and any $\sigma \in [\sigma_0, \delta_k]$. Therefore,

$$\max_{r \in \mathcal{U}^j, \|r\| = 1} \rho(\sigma, r) \geq \max_{r \in \mathcal{U}^j \cap \ \{w_k\}^\perp, \|r\| = 1} \rho(\sigma, r) = \max_{r \in \mathcal{U}^j \cap \ \{w_k\}^\perp, \|r\| = 1} \rho_k(\sigma, r) \geq \eta.$$

This is true for all $\mathcal{U}^j$ of dimension $j > 1$ and all $\sigma \in [\sigma_0, \delta_k]$. As a result, $\mu_j(\sigma)$ which is the smallest of these maxima over all $\mathcal{U}^j$'s of dimension j, is also $\geq \eta$ and so is its limit as $\sigma \to \delta_k^-$. Thus $\lim_{\sigma \to \delta_k^-} \mu_j(\sigma) \geq \eta$.

Now let $j > 1$ and $y_j(\sigma)$ be a (unit) eigenvector of $S(\sigma)$ associated with the eigenvalue $\mu_j(\sigma)$. For each σ we have

$$\mu_j(\sigma) = \rho(\sigma, y_j(\sigma)) = \rho_k(\sigma, y_j(\sigma)) - \frac{(y_j(\sigma)^T w_k)^2}{\delta_k - \sigma}.$$

Thus, $(y_j(\sigma)^T w_k)^2 = (\delta_k - \sigma)(\rho_k(\sigma, y_j(\sigma)) - \mu_j(\sigma))$, and since $\rho_k(\sigma, y_j(\sigma))$ is bounded for $\sigma \in [\sigma_0, \delta_k]$ and $\mu_j(\sigma) \geq \eta$, we must have $\lim_{\sigma \to \delta_k^-} w_k^T y_j(\sigma) = 0$.

Multiplying the equality $S(\sigma)y_j(\sigma) = \mu_j(\sigma)y_j(\sigma)$ on both sides from the left by P_k and making use of the identities $y_j(\sigma) = P_k y_j(\sigma) + (\hat{w}_k^T y_j(\sigma))\hat{w}_k$, and $P_k S(\sigma) P_k = P_k S_k(\sigma) P_k$, yields the relation:

$$P_k S_k(\sigma) P_k y_j(\sigma) - \mu_j(\sigma) P_k y_j(\sigma) = -P_k S_k(\sigma)(\hat{w}_k^T y_j(\sigma))\hat{w}_k.$$

The above expresses the residual of the approximate eigenpair $(\mu_j(\sigma), P_k y_j(\sigma))$ with respect to $P_k S_k(\sigma) P_k$.[2] When $\sigma \to \delta_k^-$, the operator $P_k S_k(\sigma) P_k$ converges to $P_k S_k(\delta_k) P_k$ which is now well defined. Since $\lim_{\sigma \to \delta_k^-}(\hat{w}_k^T y_j(\sigma)) = 0$, the above residual converges to zero. Therefore, the eigenpair $(\mu_j(\sigma), P_k y_j(\sigma))$ converges to an eigenpair of $P_k S_k(\delta_k) P_k$, which is a trivial extension of $S_{k,|}$.

Now we know that each j-th eigenvalue, with $j > 1$, converges to an eigenvalue of $S_{k,|}$, but it is left to determine to which one. Consider the case $j = 2$, i.e., the eigenpair $(\mu_2(\sigma), y_2(\sigma))$. The eigenvalue $\mu_2(\sigma)$ is the minimum of $\rho(\sigma, r)$ over the set of all vectors r that are orthogonal to $y_1(\sigma)$. Therefore,

$$\mu_2(\sigma) = \min\left\{\rho(\sigma, t)\ |t = \left(I - y_1(\sigma)y_1(\sigma)^T\right) r \neq 0,\ r \in \mathbb{R}^s\right\}. \tag{7.17}$$

Since $\lim_{\sigma \to \delta_k^-} y_1(\sigma) = \hat{w}_k$ (in direction) the limit of the above quantity as $\sigma \to \delta_k^-$ is

$$\lim_{\sigma \to \delta_k^-} \mu_2(\sigma) = \min\{\rho_k(\delta_k, t)\ |t = (I - \hat{w}_k \hat{w}_k^T)r,\ r \in \mathbb{R}^s\}.$$

The Rayleigh quotient $\rho_k(\delta_k, t)$ which can be written as

$$\rho_k(\delta_k, t) = \frac{(P_k r)^T [P_k S_k(\delta_k) P_k](P_k r)}{(P_k r)^T (P_k r)^T}$$

must be minimized over all vectors r such that $P_k r$ be nonzero, i.e., over all vectors $t = P_k r$ that are nonzero members of $\{w_k\}^\perp$. The minimum of this quantity is the smallest eigenvalue of $S_{k,|}$. The proof for the other eigenvalues is similar except that for the jth eigenvalue we now

[2]Note that $P_k y_j(\sigma) = P_k^2 y_j(\sigma)$

need to use cumulative projectors, i.e., t in (7.17), is to be replaced by

$$t = \left(I - y_{j-1}(\sigma)y_{j-1}(\sigma)^T\right) \cdots \left(I - y_2(\sigma)y_2(\sigma)^T\right) \left(I - y_1(\sigma)y_1(\sigma)^T\right) r.$$

The proof for the second part of the theorem is a trivial extension of the above proof, *provided the eigenvalues are labeled decreasingly instead of increasingly* for the proof. Then we would obtain (*for this labeling*) $\lim_{\sigma \to \delta_k^+} \mu_1(\sigma) = +\infty$ and $\lim_{\sigma \to \delta_k^+} \mu_j(\sigma) = \mu_{j-1}(S_{k,|})$ when $j > 1$. Relabeling the eigenvalues *increasingly* yields the result by noting that $S_{k,|}$ has $s - 1$ eigenvalues. $\square$

For simplicity we assumed that $\delta_k(\sigma)$ has multiplicity equal to one, but the result can be easily extended to more general situations.

7.4 A branch-hopping algorithm

The discussion above suggests an algorithm for computing all eigenvalues in an interval $[\alpha, \beta]$ by selecting the shifts carefully. We start with a shift σ equal to α then iterate as in Algorithm 7.2.1 until convergence. Once the first eigenvalue has converged we need to catch the next branch of eigenvalues. Since we are moving from left to right we will just select *the next positive eigenvalue of $S(\sigma)$ after the zero eigenvalue $\mu(\sigma)$ which has just converged.* We would then extract the corresponding eigenvector and compute the next σ by Newton's scheme as in Algorithm 7.2.1. The "Branch-hopping" idea just described above can be formulated as an iterative procedure and is listed as Algorithm 7.4.1.

Algorithm **7.4.1** *Branch hopping Newton-Spectral Schur complement*

```
0.  Given $\alpha, \beta$. Select $\sigma = \alpha$
1.  While $\sigma < \beta$
2.     Until convergence Do:
3.        Compute $\mu(\sigma)$ = Smallest eigenvalue in modulus of $S(\sigma)$
-.        along with the associated unit eigenvector $y(\sigma)$
4.        If $(|\mu(\sigma)| < tol)$
5.           $\sigma$ and associated eigenvector have converged – save them
6.           Obtain $\mu(\sigma)$ = smallest positive eigenvalue of $S(\sigma)$
```

```
-.                  along with the associated unit eigenvector $y(\sigma)$
7.             End
8.             Set $\mu'(\sigma) = -\frac{y^T(\sigma)S'(\sigma)y(\sigma)}{y^T(\sigma)y(\sigma)}$
9.             Set $\sigma := \sigma - \mu(\sigma)/\mu'(\sigma)$
10.        End
11.   End
```

Similarly to Algorithm 7.2.1, Step (3) in Algorithm 7.4.1 is performed using a form of the inverse iteration algorithm with the matrix $S(\sigma)$, in which an iterative method (with or without preconditioning) is invoked to solve the linear systems. Algorithm 7.4.1 can be optionally tied with some form of inverse iteration to obtain a more accurate shift σ for the next target eigenvalue before the Newton scheme is applied. This is the purpose of Step (9+) right after Step (9). If this optional step is used, Algorithm 7.4.1 reverts back to Newton's iteration as soon as the approximate eigenvalue is considered accurate enough. More sophisticated schemes to determine when to switch from inverse iteration back to Newton's iteration can be found in [119]. Regarding the computation of the smallest positive eigenvalue in Step (6) of Algorithm 7.4.1, this step is also computed using inverse iteration, with the difference that the computed eigenvector $y(\sigma)$ that corresponds to the eigenvalue of smallest magnitude in Step (3) is explicitly deflated to avoid repeated convergence. See [120] for a detailed discussion on deflation for symmetric linear systems.

7.5 Experiments

This section reports on numerical results with the Newton schemes proposed in Algorithms 7.2.1 and 7.4.1. All numerical experiments were performed on the `Itasca` Linux cluster at Minnesota Supercomputing Institute.

We implemented and tested three different methods: a) (inexact) Residual Inverse Iteration (RII) applied to (A, M) (each time with the appropriate shift), as discussed in [28], b) Newton's method as described in Algorithms 7.2.1 and 7.4.1, and c) a combination of (inexact) inverse iteration with Algorithms 7.2.1 and 7.4.1 where we first perform a few steps of inverse iteration with (A, M), followed by the Newton's scheme. We chose to compare against RII mainly because of its simplicity and the fact that it represents the most likely contender to our approach. As in the proposed Newton approach, RII approximates an eigenpair "in-place", e.g. without building a subspace.

For Algorithms 7.2.1 and 7.4.1, the number of subdomains used will always equal the number of MPI processes used. The matrix B_σ was factored by the LDL factorization, obtained by the associated routine in the CHOLMOD [121] package. We did not consider any further high-performance computing optimizations. For RII the inner tolerance for each linear system solution was kept fixed. We tried different inner tolerances and report the best results obtained. For Newton's method, used either as a standalone method as in Algorithms 7.2.1 and 7.4.1, or pre-processed by inverse iteration with (A, M), each update of σ was made after approximating $(\mu(\sigma), y(\sigma))$ by solving a linear system with $S(\sigma)$, using a fixed tolerance of tol_ls = 1e-2. For all (iterative) linear system solutions, we used the MINimum RESisudal (MINRES) [122] method as the iterative solver of choice (we used the existing implementation in PETSc). By default we used no preconditioning for the inner linear system solution in any of the methods tested. The residual norm tolerance for accepting an approximate eigenpair as accurate enough was set to tol = 1e-8, although the proposed Newton method almost all the times returned an approximation with residual norm less than 1e-10 or 1e-11. The residual norm was always evaluated directly by using the formula $\|r\| = \|A\hat{x}(\sigma) - \sigma M\hat{x}(\sigma)\|/\sqrt{\hat{x}^T(\sigma)^T M\hat{x}(\sigma)}$. All experiments were repeated multiple times and the average execution time is reported.

Throughout the rest of this section we will set $M = I$. Nonetheless, we will keep referring to the matrix pencil form (A, M).

7.5.1 Results

Table 7.1 shows the actual wall-clock timings when computing $nev = 1$ and $nev = 5$ consecutive eigenpairs next to $\zeta \in \mathbb{R}$ for a set of 2D Laplacians, while Table 7.2 presents similar results for a set of 3D Laplacians. The values of ζ were selected such that the eigenvalue problem was either extremal or slightly interior for both problems. For the 2D problems we chose $\zeta = 0$ and $\zeta = 0.01$ while for the 3D problems we set $\zeta = 0$ and $\zeta = 0.1$. The number in parentheses next to ζ (when $\zeta > 0$) denotes the number of negative eigenvalues of the matrix pencil $(A - \zeta M, M)$. As previously, p denotes the number of subdomains, s denotes the size of the Schur complement matrix $S(\sigma)$, and "It" denotes the total number of Newton steps performed by Algorithm 7.2.1

Table 7.1: Computing $nev = 1$ and $nev = 5$ eigenvalues next to ζ for a set of 2D problems. For the case where $\zeta \neq 0$, the starting shift for each particular eigenpair computation in Newton's scheme was provided by first performing three steps of Inverse Iteration. Times are listed in seconds.

n = 601 × 600

		$\zeta = 0.0$			$\zeta = 0.01$ (269)		
(p, nev)	s	T_{NT}	It	T_{RII}	T_{NT}	It	T_{RII}
(16,1)	7951	2.21	3	3.18	70.2	4	128.2
(16,5)	- -	11.1	15	16.2	363.0	19	615.8
(32,1)	12377	0.89	3	1.63	18.3	3	78.2
(32,5)	- -	5.41	14	9.01	85.2	14	402.5
(64,1)	18495	0.28	3	0.77	13.9	3	58.3
(64,5)	- -	1.94	14	3.63	67.3	14	192.4

n = 801 × 800

		$\zeta = 0.0$			$\zeta = 0.01$ (488)		
(p, nev)	s	T_{NT}	It	T_{RII}	T_{NT}	It	T_{RII}
(64,1)	24945	1.09	3	1.25	37.4	2	156.4
(64,5)	- -	5.95	15	6.98	198.7	12	775.1
(128,1)	36611	0.27	2	0.67	24.0	2	75.4
(128,5)	- -	1.31	9	3.82	125.0	11	382.1
(256,1)	52319	0.22	2	0.48	11.2	2	44.9
(256,5)	- -	1.59	9	2.73	61.3	10	231.6

n = 1001 × 1000

		$\zeta = 0.0$			$\zeta = 0.01$ (764)		
(p, nev)	s	T_{NT}	It	T_{RII}	T_{NT}	It	T_{RII}
(128,1)	46073	0.42	3	1.03	95.3	2	102.1
(128,5)	- -	2.84	15	5.33	482.7	10	532.1
(256,1)	65780	0.27	2	0.64	54.2	2	73.4
(256,5)	- -	1.35	10	3.32	281.3	9	381.4
(512,1)	93440	0.25	2	0.58	49.4	2	58.1
(512,5)	- -	1.42	10	3.21	256.7	10	312.8

(when $nev = 1$) or Algorithm 7.4.1 (when $nev = 5$). We denote the total time spent in Algorithm 7.2.1 or Algorithm 7.4.1 by "T_{NT}" while the time spent in RII applied to (A, M) is denoted as "T_{RI}". All timings are listed in seconds. Because $(\mu(\sigma), y(\sigma))$ is computed only approximately, extra care must be taken in order to avoid divergence when $\zeta \neq 0$. We always performed three steps of (inexact) inverse iteration with (A, M) in order to "lock" the correct eigenpair(s) before we switch to Newton's scheme. In any case, the times reported are total running times and include all phases.

For 2D problems there is a significant advantage of the Newton-based method, for both $\zeta = 0$ and $\zeta = 0.01$ values, as can be seen in Table 7.1. By using $p = 256$ subdomains, we can compute the lowest eigenpair of a $n \approx 10^6$ 2D Laplacian in 0.2 seconds, while the five lowest eigenpairs can be computed in about one second. The severe difference in the runtimes when changing from $\zeta = 0.0$ to $\zeta = 0.01$ is due to the fact that $A - 0.01M$ is now indefinite and has a large number of clustered eigenvalues very close to zero. Results for 3D problems are different from those of the 2D case and residual inverse iteration becomes more competitive relative to the Newton-based approach. The main reason is that now iterating with the spectral Schur complement is more expensive because the number of interface nodes, s, is larger and also the factorization of the B_σ matrix is more expensive. The Newton-based approach becomes faster when we increase p.

As a general comment regarding the results, for both the 2D and 3D problems, the overall cost typically scales linearly with the number of eigenpairs sought. This is a natural consequence of the fact that our method is not a projection method and each eigenpair of (A, M) is computed independently of the rest.

7.5.2 A comparison with ARPACK

This subsection provides a brief comparison between the Newton scheme and ARPACK [123], a broadly used software package based on an implicitly restarted Arnoldi/Lanczos process. By their different nature the two methods are not easy to compare but our goal here is to give a rough idea on how the two methods perform when a small number of eigenvalues are to be computed. ARPACK is expected to be superior to the Newton when a large number of eigenvalues is to be computed. For the Newton scheme we used only one node of Itasca (8 cores). Thus, now each core actually handles multiple subdomains. Under this framework we can also test the performance of the Newton scheme in "serial" environments. For ARPACK, we set the size of

Table 7.2: Computing $nev = 1$ and $nev = 5$ eigenvalues next to ζ for a set of 3D problems. For the case where $\zeta \neq 0$, the starting shift for each particular eigenpair computation in Newton's scheme was provided by first performing three steps of Inverse Iteration. Times are listed in seconds.

$\mathbf{n = 41 \times 40 \times 39}$

		$\zeta = 0.0$			$\zeta = 0.1$ (19)		
(p, nev)	s	T_{NT}	It	T_{RII}	T_{NT}	It	T_{RII}
(16,1)	15423	0.21	3	0.10	1.07	4	1.32
(16,5)	- -	1.39	15	0.62	5.85	19	7.77
(32,1)	20037	0.06	3	0.03	0.27	2	0.90
(32,5)	- -	0.32	14	0.19	1.52	14	4.86
(64,1)	24789	0.09	3	0.04	0.14	3	0.66
(64,5)	- -	0.44	14	0.21	1.01	15	3.51

$\mathbf{n = 71 \times 70 \times 69}$

		$\zeta = 0.0$			$\zeta = 0.1$ (137)		
(p, nev)	s	T_{NT}	It	T_{RII}	T_{NT}	It	T_{RII}
(64,1)	83358	0.80	3	0.61	15.4	2	15.9
(64,5)	- -	4.20	14	3.22	80.4	10	79.9
(128,1)	108508	0.19	3	0.32	3.12	2	8.41
(128,5)	- -	1.25	14	1.71	15.1	10	38.5
(256,1)	136159	0.10	3	0.27	5.99	2	12.7
(256,5)	- -	0.68	13	1.45	25.3	10	51.7

$\mathbf{n = 101 \times 100 \times 99}$

		$\zeta = 0.0$			$\zeta = 0.1$ (439)		
(p, nev)	s	T_{NT}	It	T_{RII}	T_{NT}	It	T_{RII}
(128,1)	230849	2.73	3	2.02	48.1	3	93.3
(128,5)	- -	13.2	15	10.3	233.2	16	472.1
(256,1)	293626	1.10	3	1.61	23.4	3	62.4
(256,5)	- -	5.80	14	8.32	129.2	16	301.5
(512,1)	369663	0.62	2	0.99	32.4	2	75.3
(512,5)	- -	3.01	12	5.71	168.9	12	322.9

Table 7.3: Computing $nev = 1$ and $nev = 5$ eigenvalues next to ζ with the proposed Newton scheme and ARPACK. The discretization selected as $n_x = 71, n_y = 70$, and $n_z = 69$. Times are listed in seconds.

	$\zeta = 0.0$		$\zeta = 0.1$ (137)	
(p, nev)	T_{NT}	T_{ARP}	T_{NT}	T_{ARP}
(64,1)	5.5	35.4	170.0	351.5
(128,1)	3.4	–	105.1	–
(256,1)	5.3	–	122.5	–
(64,5)	28.3	94.1	884.7	416.3
(128,5)	15.3	–	532.3	–
(256,5)	25.9	–	605.3	–

the search subspace equal to twenty and we used only one execution core. The tolerance tol for the requested eigenpairs set again to tol $= 1e - 8$ for both methods. As a demonstration, we used a single test 3D Laplacian with $n_x = 71, n_y = 70$, and $n_z = 69$ discretization points in each dimension. As previously, we selected $\zeta = 0$ and $\zeta = 0.1$.

Table 7.3 compares the execution times obtained of the Newton (T_{NT}) and ARPACK (T_{ARP}) schemes when searching for $nev = 1$ and $nev = 5$ eigenpairs of (A, I), and using a different number of subdomains. Because ARPACK operates directly on (A, I) it is oblivious to the number of subdomains used. On the other hand, the performance of the Newton scheme varies as the number of subdomains changes.

As expected, ARPACK becomes more efficient than the Newton-based method as we increase the number of eigenpairs sought for the specific problem, especially for eigenvalues deeper into the spectrum, since it is a projection method and can obtain simultaneous approximations for multiple eigenpairs. On the other hand, the Newton method approximates each eigenpair separately (the size of the subspace is always one) and the total cost is approximately linear to the number of eigenpairs sought. Note however that when the size of the search subspace in ARPACK was set to less than ten, ARPACK was slower than the proposed Newton-based schemes.

7.6 Summary

The method presented in this chapter combines Newton's method with spectral Schur complements in a domain decomposition framework. The scheme essentially amounts to solving the eigenvalue problem along the interface points only, and exploits the fact that solves with the local subdomains are relatively inexpensive. A parallel implementation was presented and its performance evaluated for model Laplacian problems and general matrices from electronic structure calculations. The proposed method can be quite fast when only one or a very small number of extremal eigenpairs are sought. It can be combined with a few steps of inverse iteration to provide again a fast technique for solving what might be termed moderately interior eigenproblems, i.e., problems with eigenvalues not too deep inside the spectrum. One might compare the proposed approach to a Rayleigh quotient iteration, whereby the consecutive linear systems are handled by an iterative method in a domain decomposition framework. However, the focus on the Schur complement provides additional insights and leads to the Newton procedure presented here.

Chapter 8

The method of mixed linear approximations

The method proposed in this chapter is similar to the technique presented in Chapter 7. This means that in order to compute each sought eigenpair $\left(\lambda_i, x^{(i)}\right)$ of (A, M), where $x^{(i)} = \left[\left(u^{(i)}\right)^T, \left(y^{(i)}\right)^T\right]^T$, we first compute the pair $\left(\lambda_i, y^{(i)}\right)$ by recasting the interface eigenvalue problem $S(\lambda_i)y(\lambda_i) = 0$ into one of a root-finding. Similarly to Chapter 7, this root-finding problem is solved by Newton's method and the remaining part of $x^{(i)}$, $u^{(i)}$, is computed by solving the linear system $B_{\lambda_i}^{-1}u^{(i)} = -E_\lambda y^{(i)}$. However, in the root-finding approach developed in Chapter 7, the scalar function of interest is defined by the eigenbranches of a zeroth-order approximation of the spectral Schur complement. In contrast, in this chapter we exploit a first-order approximation instead. As we show, this choice requires a separate theoretical analysis, while it also leads to faster convergence of the root-finding scheme.

The structure of this chapter is as follows: Section 8.1 discusses the proposed approach and provides some theoretical analysis. Section 8.2 discusses practical details. Section 8.3 presents numerical experiments performed on a set of matrix pencils, and compares the scheme proposed in this chapter against RQI and the root-finding scheme presented in Chapter 7. Finally, Section 8.4 contains our concluding remarks.

8.1 The method of Mixed Linear Approximations

Consider the following parameterized (symmetric) generalized eigenvalue problem, stemming by a first-order approximation of $S(\lambda)$ around a real scalar $\sigma \in \mathbb{R}$:

$$S(\sigma)\hat{y}_j(\sigma) = \theta_j(\sigma)\Big[-S'(\sigma)\Big]\hat{y}_j(\sigma), \tag{8.1}$$

where $S'(\sigma)$ denotes the first derivative of $S(\sigma)$, $\theta_j(\sigma)$ $(j = 1, \ldots, s)$ denotes the jth eigenvalue of the pencil $(S(\sigma), -S'(\sigma))$ in a sorted (algebraic) ascending order, and $\hat{y}_j(\sigma)$ denotes the corresponding eigenvector.

A scalar $\sigma \notin \Lambda(B, M_B)$ will be an eigenvalue of (A, M) if and only if there exists an index $1 \leq j \leq s$ such that $\theta_j(\sigma) = 0$. The rest of this section presents a root-finding approach to compute the eigenpair (λ, x) of (A, M) associated with the eigenvalue λ lying the closest to some user-given initial approximation.

Remark 2 *Except where stated otherwise, throughout the rest of this section we will consider only the case in which:*

(a) the sought eigenvalue λ of the matrix pencil (A, M) is simple,

(b) $\lambda \notin \Lambda(B, M_B)$.

Extensions to the cases where λ is a multiple eigenvalue of the matrix pencil (A, M) or $\lambda \in \Lambda(B, M_B)$ are possible though, and we will comment where deemed necessary.

8.1.1 Eigenbranches

Definition 8.1.1 *The scalar function $\theta_j(\sigma) : \mathbb{R} \rightarrow \mathbb{R}$,*

$$\theta_j(\sigma) = \frac{\hat{y}_j^T(\sigma)S(\sigma)\hat{y}_j(\sigma)}{\hat{y}_j^T(\sigma)[-S'(\sigma)]\hat{y}_j(\sigma)}, \tag{8.2}$$

will be referred to as the jth eigenbranch of $(S(\sigma), -S'(\sigma))$, $j = 1, \ldots, s$. Throughout this chapter, the numbering of the eigenbranches will be based on their algebraic value, i.e., $\theta_1(\sigma) \leq \ldots \leq \theta_s(\sigma)$.

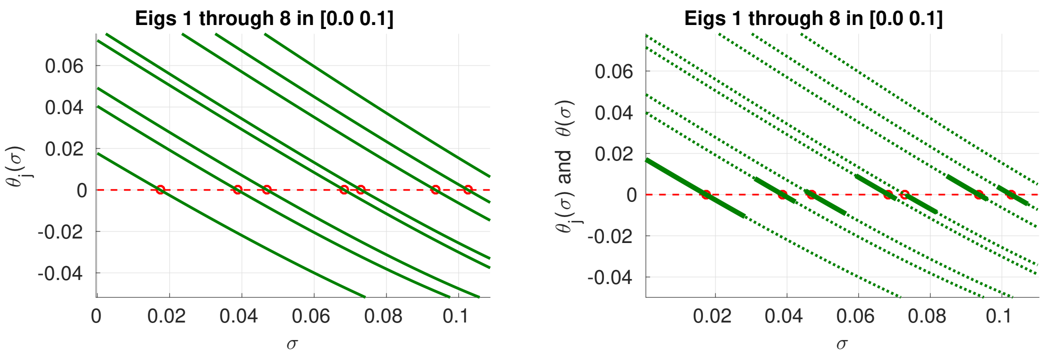


Figure 8.1: Visualization of $\theta_j(\sigma)$ and $\theta_{\kappa_\sigma}(\sigma)$. The red circles along the real axis denote eigenvalues of the pencil (A, M). Left: Eigenbranches $\theta_j(\sigma),\ j = 1, \ldots, 8$. Right: Eigenbranches $\theta_j(\sigma),\ j = 1, \ldots, 8$ (dotted lines) and $\theta_{\kappa_\sigma}(\sigma) \equiv \theta(\sigma)$ (solid line). Each eigenvalue of (A, M) is a root of (at least) one eigenbranch, and thus a root of $\theta(\sigma)$ as well.

Moreover, let $\theta_j(\sigma),\ j = 1, \ldots, s$ be defined as in (8.2), and consider the following integer:

$$\kappa_\sigma = \arg\min_{1 \le j \le s} |\theta_j(\sigma)|. \tag{8.3}$$

We then define the functions $\theta(\sigma) : \mathbb{R} \to \mathbb{R}$ and $\hat{y}(\sigma) : \mathbb{R} \to \mathbb{R}^s$,

$$\theta(\sigma) := \theta_{\kappa_\sigma}(\sigma), \quad \hat{y}(\sigma) := \hat{y}_{\kappa_\sigma}(\sigma), \tag{8.4}$$

i.e., $(\theta(\sigma), \hat{y}(\sigma))$ denotes the eigenpair associated with the eigenvalue of smallest magnitude of the matrix pencil $(S(\sigma), -S'(\sigma))$.

Figure 8.1 visualizes the first few eigenbranches of a disretized Laplacian operator. Each eigenvalue of (A, M) is a root of (at least) one eigenbranch $\theta_j(\sigma),\ 1 \le j \le s$, and thus a root of $\theta(\sigma)$ as well. Thus, the problem of computing an eigenvalue of (A, M) is converted into one of computing the root of the scalar function $\theta(\sigma)$ defined in (8.4) that lies the closest to the initial approximation.

8.1.2 Formulation of a Newton-based procedure

Similarly to Chapter 7 we consider Newton's method as the root-finding technique [114]. While $\theta(\sigma)$ is not differentiable across its entire domain of definition, Newton's method can still be applied if the integer κ_σ defined in (8.3) remains fixed. This follows from the analyticity of the eigenbranches.

Proposition 8.1.1 *The eigenbranches $\theta_j(\sigma),\ j=1,\dots,s$ are analytic for any $\sigma \notin \Lambda(B, M_B)$. The derivative of each eigenbranch is given by*

$$\theta_j'(\sigma) = \frac{d\theta_j(\sigma)}{d\sigma} = -\frac{\hat{y}_j^T(\sigma)\left[S'(\sigma)+\theta_j(\sigma)S''(\sigma)\right]\hat{y}_j(\sigma)}{\hat{y}_j^T(\sigma)S'(\sigma)\hat{y}_j(\sigma)}, \tag{8.5}$$

where

$$\begin{aligned} S''(\sigma) = \frac{d^2S(\sigma)}{d\sigma^2} &= 2E_\sigma^T B_\sigma^{-1} M_B B_\sigma^{-1} M_B B_\sigma^{-1} E_\sigma - 2M_E^T B_\sigma^{-1} M_E + 2M_E^T B_\sigma^{-1} M_B B_\sigma^{-1} E_\sigma \\ &+ 2E_\sigma^T B_\sigma^{-1} M_B B_\sigma^{-1} M_E \end{aligned} \tag{8.6}$$

denotes the second derivative of $S(\sigma)$.

Proof: Differentiating $S(\sigma)\hat{y}_j(\sigma) = -\theta_j(\sigma)S'(\sigma)\hat{y}_j(\sigma)$ with respect to σ leads to

$$S'(\sigma)\hat{y}_j(\sigma) + S(\sigma)\hat{y}_j'(\sigma) = -\theta_j(\sigma)\left[S''(\sigma)\hat{y}_j(\sigma) + S'(\sigma)\hat{y}_j'(\sigma)\right] - \theta_j'(\sigma)S'(\sigma)\hat{y}_j(\sigma),$$

where after collecting terms we get

$$\left[S(\sigma)+\theta_j(\sigma)S'(\sigma)\right]\hat{y}_j'(\sigma) = -\left[S'(\sigma)+\theta_j(\sigma)S''(\sigma)\right]\hat{y}_j(\sigma) - \theta_j'(\sigma)S'(\sigma)\hat{y}_j(\sigma).$$

Multiplying by $\hat{y}_j^T(\sigma)$ from the left, and noticing that $\hat{y}_j^T(\sigma)\left[S(\sigma)+\theta_j(\sigma)S'(\sigma)\right] = 0$, leads to (8.5), which can be further simplified to

$$\theta_j'(\sigma) = -1 - \theta_j(\sigma)\frac{\hat{y}_j^T(\sigma)S''(\sigma)\hat{y}_j(\sigma)}{\hat{y}_j^T(\sigma)S'(\sigma)\hat{y}_j(\sigma)},$$

or $\theta_j'(\sigma) = -1+\theta_j(\sigma)\hat{y}_j^T(\sigma)S''(\sigma)\hat{y}_j(\sigma)$ if the eigenvectors $\hat{y}_j(\sigma)$ are normalized as $-\hat{y}_j^T(\sigma)S'(\sigma)\hat{y}_j(\sigma) = 1$. Differentiating the expression of $S'(\sigma)$ in Proposition 7.1.1 twice leads to

$$\begin{aligned} S''(\sigma+\omega) = &- E_\sigma^T\left[B_{\sigma+\omega}^{-1}\right]'' E_\sigma - 2\left(M_E^T\left(\left[B_{\sigma+\omega}^{-1}\right]' M_E - \left[B_{\sigma+\omega}^{-1}\right]' E_\sigma\right) - E_\sigma^T\left[B_{\sigma+\omega}^{-1}\right]' M_E\right) \\ &+ \mathcal{R}(\omega) \end{aligned}$$

where

$$\mathcal{R}(\omega) = \omega\left(M_E^T[B_{\sigma+\omega}^{-1}]''E_\sigma + E_\sigma^T[B_{\sigma+\omega}^{-1}]''M_E\right) - 2\omega M_E^T\left([B_{\sigma+\omega}^{-1}]' + [B_{\sigma+\omega}^{-1}]''\right)M_E$$
$$- \omega^2 M_E^T[B_{\sigma+\omega}^{-1}]''M_E.$$

The result in (8.6) follows by setting $S''(\sigma) = S''(\sigma+\omega)_{\omega=0}$ and noticing that in addition to $\left[B_{\sigma+\omega}^{-1}\right]'_{\omega=0} = B_\sigma^{-1}M_B B_\sigma^{-1}$, we also have

$$\left[B_{\sigma+\omega}^{-1}\right]''_{\omega=0} = -2B_\sigma^{-1}M_B B_\sigma^{-1}M_B B_\sigma^{-1},$$

and $\mathcal{R}(0) = 0$.

Finally, the analyticity of $\theta_j(\sigma)$ and $\hat{y}_j(\sigma)$ follows directly from [124] with the results also holding for multiple (semi-simple) eigenvalues. $\square$

Remark 3 *When M is the identity matrix, i.e., $M = I$, we have $M_B = M_C = I$ and $M_E = 0$, and the first and second derivatives of the matrix-valued function $S(\sigma)$ are simplified to $S'(\sigma) = -\left(I + E^T(B-\sigma I)^{-2}E\right)$ and $S''(\sigma) = 2E^T(B-\sigma I)^{-3}E$, respectively.*

Algorithm 8.1.1 sketches the main steps of Newton's root-finding applied to the scalar function $\theta(\sigma)$. For any σ, Algorithm 8.1.1 computes the eigenpair $(\theta(\sigma),\hat{y}(\sigma))$ of the pencil $(S(\sigma), -S'(\sigma))$, and evaluates the derivative $\theta'(\sigma)$ to update σ. The algorithm terminates as soon as $|\theta(\sigma)|$ becomes smaller than a threshold tolerance tol $\in \mathbb{R}$, which, together with an initial approximation of the sought eigenvalue, are the only inputs required by Algorithm 8.1.1. Throughout the rest of this paper we will refer to Algorithm 8.1.1 as the method of Mixed Linear Approximations (MLA).

The convergence of MLA can be monitored by computing the residual norm of the approximate eigenpair $(\sigma, \hat{x}(\sigma))$, $\|A\hat{x}(\sigma) - \sigma M\hat{x}(\sigma)\|$. Alternatively, we can skip the Matrix-Vector multiplications with matrices A and M and only monitor $|\theta(\sigma)|$ since

$$\begin{aligned}\|A\hat{x}(\sigma) - \sigma M\hat{x}(\sigma)\| &= \left\| \begin{pmatrix} B_\sigma & E_\sigma \\ E_\sigma^T & C_\sigma \end{pmatrix} \begin{pmatrix} B_\sigma^{-1}E_\sigma\hat{y}(\sigma) \\ \hat{y}(\sigma) \end{pmatrix} \right\| \\ &= \left\| \begin{pmatrix} 0 \\ -\theta(\sigma)S'(\sigma)\hat{y}(\sigma) \end{pmatrix} \right\| \\ &\leq |\theta(\sigma)| \;\; \|\hat{y}(\sigma)\| \;\; \lambda_{\mathbf{max}}\left(-S'(\sigma)\right),\end{aligned}$$

where $\lambda_{\mathbf{max}}(.)$ denotes the largest eigenvalue of the SPD matrix $-S'(\sigma)$.

Algorithm **8.1.1** *MLA*

```
0.  Given $\mathrm{tol} \in \mathbb{R}$ and an initial eigenvalue approximation $\sigma \in \mathbb{R}$
1.  Do while (true):
2.      Solve $S(\sigma)\hat{y}(\sigma) + \theta(\sigma)S'(\sigma)\hat{y}(\sigma) = 0$ for the eigenvalue $\theta(\sigma)$
        of smallest magnitude and associated eigenvector $\hat{y}(\sigma)$
3.      If $|\theta(\sigma)| \leq \mathrm{tol}$, break;
4.      Compute $\theta'(\sigma) = -1 - \theta(\sigma)\dfrac{\hat{y}^T(\sigma)S''(\sigma)\hat{y}(\sigma)}{\hat{y}^T(\sigma)S'(\sigma)\hat{y}(\sigma)}$
5.      Update $\sigma = \sigma - \dfrac{\theta(\sigma)}{\theta'(\sigma)}$
6.  EndDo
7.  Return $\sigma$ and $\hat{x}(\sigma) = [-(B_\sigma^{-1}E_\sigma\hat{y}(\sigma))^T, \hat{y}^T(\sigma)]^T$
```

MLA is essentially Newton's method applied to the scalar function $\theta(\sigma)$. Therefore, when it converges, we expect MLA to do so at a quadratic rate; at least if a sufficiently accurate initial approximation is provided [114]. In practice, the rate of convergence of MLA will depend on the shape of the eigenbranches. As we discuss in Section 8.2.3, the shape of the eigenbranches might be very close to that of a linear function, in which case MLA will converge rapidly.

MLA can be easily extended to the computation of additional eigenpairs of (A, M). The only requirement is to provide an initial approximation of the next sought eigenvalue.

Computation of semi-simple eigenvalues

The MLA scheme can capture the correct multiplicity ρ of a semi-simple eigenvalue λ of the pencil (A, M) as long as $\rho \leq s$.

Proposition 8.1.2 *Suppose that $\lambda \notin \Lambda(B, M_B)$ and let ρ be a non-negative integer such that $1 \leq \rho \leq s$. Then, λ is a root of $\theta(\sigma)$ of multiplicity ρ if and only if λ is an eigenvalue of (A, M) of multiplicity ρ.*

Proof: Let λ be a root of $\theta(\sigma)$ with multiplicity ρ. Then, there exist ρ linearly independent vectors $y^{(1)}, \ldots, y^{(\rho)}$ such that $S(\lambda)y^{(i)} = 0,\ i = 1, \ldots, \rho$. The ρ vectors $x^{(i)} = [-(B_\lambda^{-1}E_\lambda y^{(i)})^T, (y^{(i)})^T]^T$ are then linearly independent and satisfy the equation $(A - \lambda M)x^{(i)} = 0$.

To prove the converse, let $x^{(i)} = \left[-\left(B_\lambda^{-1}E_\lambda y^{(i)}\right)^T, \left(y^{(i)}\right)^T\right]^T,\ i = 1, \ldots, \rho$ be the ρ eigenvectors of (A, M) associated with eigenvalue λ. Stacking these ρ eigenvectors next to each other leads to

$$X = \begin{pmatrix} -B_\lambda^{-1}E_\lambda y^{(1)} & \cdots & -B_\lambda^{-1}E_\lambda y^{(\rho)} \\ y^{(1)} & \cdots & y^{(\rho)} \end{pmatrix} = \begin{pmatrix} -B_\lambda^{-1}E_\lambda \\ I \end{pmatrix} \left[y^{(1)}, \ldots, y^{(p)}\right].$$

Matrix X is of rank ρ and thus vectors $y^{(1)}, \ldots, y^{(\rho)}$ are linearly independent. In addition, the vectors $y^{(1)}, \ldots, y^{(\rho)}$ satisfy the equation $S(\lambda)y^{(i)} = 0,\ i = 1, \ldots, \rho$, and thus $S((\lambda), -S'(\lambda))$ has a zero eigenvalue with multiplicity ρ. $\square$

Remark 4 *In practice, the multiplicity of an eigenvalue λ of (A, M) will not be known. A practical approach is then to compute the second eigenvalue of smallest magnitude of $(S(\lambda), -S'(\lambda))$. If this eigenvalue is non-zero, then, by Proposition 8.1.2, λ is a simple eigenvalue of (A, M). In the opposite case, λ is a semi-simple eigenvalue, and to compute its exact multiplicity we must repeat the same procedure until either the next computed eigenvalue of smallest magnitude of $S((\lambda), -S'(\lambda))$ is non-zero, or all s eigenpairs of $S((\lambda), -S'(\lambda))$ are computed.*

8.1.3 Characterization of the eigenbranches as σ approaches an eigenvalue of (A, M)

We now consider the behavior of the eigenbranches as σ approaches a simple eigenvalue of (A, M).

Theorem 8.1.3 *Let $\lambda_\kappa \notin \Lambda(B, M_B)$, $1 \leq \kappa \leq n$, denote a simple eigenvalue of the pencil (A, M), with an associated eigenvector $x^{(\kappa)} = \left[\left(u^{(\kappa)}\right)^T, \left(y^{(\kappa)}\right)^T\right]^T$. Then,*

$$\mathbf{span}\left\{\mathbf{lim}_{\sigma\to\lambda_\kappa} \hat{y}(\sigma)\right\} \equiv \mathbf{span}\left\{\hat{y}(\lambda_\kappa)\right\} \equiv \mathbf{span}\left\{y^{(\kappa)}\right\}. \tag{8.7}$$

Moreover, for any $j \neq \mathbf{lim}_{\sigma\to\lambda_\kappa} \kappa_\sigma$, we have

$$\mathbf{lim}_{\sigma\to\lambda_\kappa} \theta_j(\sigma) = -\,\mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{\hat{y}_j^T(\sigma)\hat{y}_j(\sigma)}{\sum_{i=1,\ i\neq\kappa}^{n} \frac{\left(\hat{y}_j^T(\sigma)y^{(i)}\right)\left(\hat{y}_j^T(\sigma)S'(\sigma)y^{(i)}\right)}{\lambda_i - \sigma}}\right), \tag{8.8}$$

where $1 \leq \kappa_\sigma \leq s$ denotes the index of the eigenvalue of smallest magnitude of the pencil $(S(\sigma), -S'(\sigma))$, i.e., $(\theta_{\kappa_\sigma}(\sigma), \hat{y}_{\kappa_\sigma}(\sigma)) \equiv (\theta(\sigma), \hat{y}(\sigma))$.

Proof: Equating the (2,2) blocks of the identities

$$(A - \sigma M)^{-1} = \begin{pmatrix} B_\sigma^{-1} + B_\sigma^{-1}E_\sigma S_\sigma^{-1}E_\sigma^T B_\sigma^{-1} & -B_\sigma^{-1}E_\sigma S_\sigma^{-1} \\ -S_\sigma^{-1}E_\sigma^T B_\sigma^{-1} & S_\sigma^{-1} \end{pmatrix},$$

and

$$(A - \sigma M)^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i - \sigma} x^{(i)} \left(x^{(i)}\right)^T = \sum_{i=1}^{n} \frac{1}{\lambda_i - \sigma} \begin{pmatrix} u^{(i)}\left(u^{(i)}\right)^T & u^{(i)}\left(y^{(i)}\right)^T \\ y^{(i)}\left(u^{(i)}\right)^T & y^{(i)}\left(y^{(i)}\right)^T \end{pmatrix},$$

gives

$$S(\sigma)^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i - \sigma} y^{(i)}\left(y^{(i)}\right)^T.$$

Let $\sigma \notin \{\Lambda(B, M_B) \cup \Lambda(A, M)\}$. We can rewrite $S(\sigma)\hat{y}_j(\sigma) = \theta_j(\sigma)[-S'(\sigma)]\hat{y}_j(\sigma)$ for any eigenpair $(\theta_j(\sigma), \hat{y}_j(\sigma))$ as:

$$\begin{aligned} \frac{1}{\theta_j(\sigma)}\hat{y}_j(\sigma) &= -S(\sigma)^{-1}S'(\sigma)\hat{y}_j(\sigma) \\ &= -\sum_{i=1}^{n} \frac{1}{\lambda_i - \sigma} y^{(i)}\left(y^{(i)}\right)^T S'(\sigma)\hat{y}_j(\sigma). \end{aligned} \tag{8.9}$$

Moreover, if we define define the scalar function

$$\chi_{i,j}(\sigma) = \left(y^{(i)}\right)^T S'(\sigma)\hat{y}_j(\sigma),$$

for all $i = 1, \ldots, n$, and $j = 1, \ldots, s$, we can write (8.9) as

$$\frac{1}{\theta_j(\sigma)}\hat{y}_j(\sigma) = -\frac{\chi_{\kappa,j}(\sigma)}{\lambda_\kappa - \sigma}y^{(\kappa)} - \sum_{i=1,\ i\neq\kappa}^{n} \frac{\chi_{i,j}(\sigma)}{\lambda_i - \sigma}y^{(i)}. \tag{8.10}$$

Now, let $j \equiv \kappa_\sigma$, and multiply both sides in (8.10) by $(\lambda_\kappa - \sigma)/\chi_{\kappa,\kappa_\sigma}$. Taking limits leads to

$$\begin{aligned} -\mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{\lambda_\kappa - \sigma}{\chi_{\kappa,\kappa_\sigma}(\sigma)\theta(\sigma)}\hat{y}(\sigma)\right) &= y^{(\kappa)} + \mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\sum_{i=1,\ i\neq\kappa}^{n}\frac{\chi_{i,\kappa_\sigma}(\sigma)(\lambda_\kappa - \sigma)}{\chi_{\kappa,\kappa_\sigma}(\sigma)(\lambda_i - \sigma)}y^{(i)}\right) \\ &= y^{(\kappa)}, \end{aligned} \tag{8.11}$$

where the second equality follows directly from the fact that λ_κ is simple.

Since $\mathbf{lim}_{\sigma\to\lambda_\kappa}\,\theta(\sigma) \to 0$, the limit of each entry of the vector on the left-hand side of (8.11) is of the form 0/0. Let ℓ_η denote the ηth, $1 \le \eta \le s$, entry of vector $\hat{y}(\sigma)$. Applying l'Hôspital's rule (where we differentiate with respect to σ) gives:

$$\mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{-(\lambda_\kappa - \sigma)\ell_\eta}{\chi_{\kappa,\kappa_\sigma}(\sigma)\theta(\sigma)}\right) = \mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{\left[-\left(\lambda_\kappa - \sigma\right)\ell_\eta\right]'}{\left[\chi_{\kappa,\kappa_\sigma}(\sigma)\theta(\sigma)\right]'}\right) = \mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{\ell_\eta}{\chi_{\kappa,\kappa_\sigma}(\sigma)\theta'(\sigma)}\right), \tag{8.12}$$

where we used the identity $\mathbf{lim}_{\sigma\to\lambda_\kappa}\,\theta(\sigma) = \theta(\lambda_k) = 0$. Since both $\chi_{\kappa,\kappa_\sigma}(\sigma)$ and $\theta'(\sigma)$ are non-zero, the above limit is well-defined. Applying (8.12) to all entries of $\hat{y}(\sigma)$ finally gives

$$\mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{1}{\chi_{\kappa,\kappa_\sigma}(\sigma)\theta'(\sigma)}\hat{y}(\sigma)\right) = \mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\frac{-1}{\chi_{\kappa,\kappa_\sigma}(\sigma)}\hat{y}(\sigma)\right) = \frac{-1}{\chi_{\kappa,\kappa_{\lambda_\kappa}}(\lambda_\kappa)}\hat{y}(\lambda_\kappa) = y^{(\kappa)},$$

where we used the identity $\mathbf{lim}_{\sigma\to\lambda_\kappa}\,\theta'(\sigma) = \theta'(\lambda_k) = -1$.

To prove the second item, first notice that the eigenvectors of $(S(\sigma), -S'(\sigma))$ are $S'(\sigma)$-orthogonal, and thus $\hat{y}_j^T(\sigma)S'(\sigma)\hat{y}_{\kappa_\sigma}(\sigma) = 0,\ j \neq \kappa_\sigma$. Exploiting that $\mathbf{span}\{\mathbf{lim}_{\sigma\to\lambda_\kappa}\,\hat{y}_{\kappa_\sigma}(\sigma)\} \equiv \mathbf{span}\left\{y^{(\kappa)}\right\}$ leads to $\mathbf{lim}_{\sigma\to\lambda_\kappa}(\hat{y}_j^T(\sigma)S'(\sigma)y^{(\kappa)}(\sigma)) = 0$ for any $j \neq \kappa_\sigma$. Thus, (8.10) is simplified to

$$\mathbf{lim}_{\sigma\to\lambda_\kappa}\,\frac{1}{\theta_j(\sigma)}\hat{y}_j(\sigma) = -\,\mathbf{lim}_{\sigma\to\lambda_\kappa}\left(\sum_{i=1,\ i\neq\kappa}^{n}\frac{\chi_{i,j}(\sigma)}{\lambda_i - \sigma}y^{(i)}\right), \tag{8.13}$$

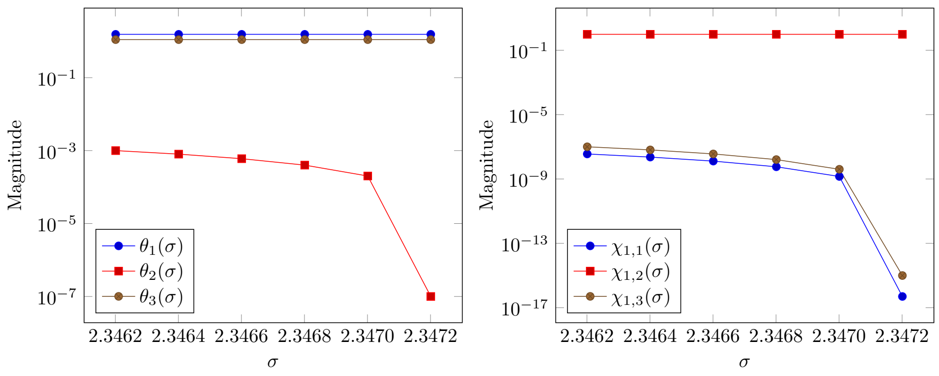


Figure 8.2: Numerical value of $\theta_j(\sigma)$ and $\chi_{3,j}(\sigma)$ as $\sigma \to \lambda_3$.

for any $(\theta_j(\sigma), \hat{y}_j(\sigma))$, $j = 1, \dots, s$, $j \neq \kappa_\sigma$. Multiplying (8.13) by $\hat{y}_j^T(\sigma)$ from the left and re-arranging terms leads to (8.8). $\square$

Theorem 8.1.3 suggests that as σ converges towards the sought (simple) eigenvalue of (A, M), a shift-and-invert-based inner eigenvalue solver will require fewer iterations to compute $(\theta(\sigma), \hat{y}(\sigma))$. We will verify this behavior in Section 8.3. A similar result also holds for multiple eigenvalues λ of (A, M) with algebraic multiplicity μ_λ, i.e., there exist μ_λ eigenvalues $\theta_j(\sigma)$ of $(S(\sigma), -S'(\sigma))$ such that $\lim_{\sigma\to\lambda_\kappa} \theta_j(\sigma) = 0$, with the limit of the rest $s - \mu_\lambda$ ones converging to a nonzero value. Details are omitted.

Example 1 *Let*

$$A = \begin{pmatrix} 2 & 1 & 0 & 1 \\ 1 & 3 & 1 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix}, \quad M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{8.14}$$

and consider $d = 1$, $s = 3$. Figure 8.2 plots $\theta_j(\sigma)$, $j = 1, 2, 3$ (left subfigure), and $\chi_{3,j}(\sigma)$ as $\sigma \to \lambda_3$ ($\lambda_3 = 2.3473$ is simple). As predicted by Theorem 8.1.3, there is exactly one eigenbranch, in this example $\theta_2(\sigma)$, for which $\lim_{\sigma\to\lambda_3} \theta_2(\sigma) = 0$. For this same eigenbranch, we also have $\lim_{\sigma\to\lambda_3} \chi_{2,1}(\sigma) \neq 0$. On the other hand, $\lim_{\sigma\to\lambda_3} \chi_{3,j}(\sigma) = 0$ for any $j \neq 1$.

8.1.4 Characterization of the eigenbranches as σ approaches an eigenvalue of (B, M_B)

Let the ηth eigenvalue-eigenvector pair of (B, M_B) be denoted as $(\delta_\eta, v_\eta),\ \eta = 1, \dots, d$.

If we set

$$w_\eta = E_\sigma^T v_\eta, \qquad f_\eta = M_E^T v_\eta, \tag{8.15}$$

and make use of the identity

$$B_\sigma^{-1} = \sum_{\eta=1}^{d} \frac{v_\eta v_\eta^T}{\delta_\eta - \sigma},$$

we can rewrite

$$S(\sigma) = C_\sigma - \sum_{\eta=1}^{d} \frac{w_\eta w_\eta^T}{\delta_\eta - \sigma}, \tag{8.16}$$

and

$$-S'(\sigma) = M_C + \sum_{\eta=1}^{d} \frac{w_\eta w_\eta^T}{(\delta_\eta - \sigma)^2} - \sum_{\eta=1}^{d} \frac{w_\eta f_\eta^T}{\delta_\eta - \sigma} - \sum_{\eta=1}^{d} \frac{f_\eta w_\eta^T}{\delta_\eta - \sigma}. \tag{8.17}$$

While $S(\sigma)$ and $S'(\sigma)$ can not be formally defined when $\sigma \in \Lambda(B, M_B)$, in practice the eigenbranches remain well-defined regardless of how close σ approximates an eigenvalue of the pencil (B, M_B).

Theorem 8.1.4 *Let $\delta_k \notin \Lambda(A, M),\ 1 \le k \le d$, be a simple eigenvalue of (B, M_B), and define for any $\eta = 1, \dots, d$,*

$$\epsilon_{j,\eta}(\sigma) = w_\eta^T \hat{y}_j(\sigma), \qquad \phi_{j,\eta}(\sigma) = f_\eta^T \hat{y}_j(\sigma), \tag{8.18}$$

where w_η and f_η are defined in (8.15). Then:

- *If $\lim_{\sigma \to \delta_k} \epsilon_{j,k}(\sigma) \neq 0$, then $\lim_{\sigma \to \delta_k} \theta_j(\sigma) = 0$.*
- *If $\lim_{\sigma \to \delta_k} \epsilon_{j,k}(\sigma) = 0$ then*

$$\lim_{\sigma \to \delta_k} \theta_j(\sigma) = \frac{\hat{y}_j^T(\sigma) C_\sigma \hat{y}_j(\sigma) - \sum_{\eta=1, \eta\neq k}^{d} \dfrac{\epsilon_{j,\eta}^2(\sigma)}{\delta_\eta - \sigma}}{\hat{y}_j^T(\sigma) M_c \hat{y}_j(\sigma) + \sum_{\eta=1, \eta\neq k}^{d} \left[\dfrac{\epsilon_{j,\eta}^2(\sigma)}{(\delta_\eta - \sigma)^2} - \dfrac{2\epsilon_{j,\eta}(\sigma)\phi_{j,\eta}(\sigma)}{\delta_\eta - \sigma} \right]}. \tag{8.19}$$

Proof: Making use of Eqs. (8.16) and (8.17), each eigenbranch $\theta_j(\sigma),\ j = 1, \ldots, s$ can be written as:

$$\theta_j(\sigma) = \frac{\hat{y}_j^T(\sigma)C_\sigma\hat{y}_j(\sigma) - \sum_{\eta=1}^{d} \dfrac{\epsilon_{j,\eta}^2(\sigma)}{\delta_\eta - \sigma}}{\hat{y}_j^T(\sigma)M_c\hat{y}_j(\sigma) + \sum_{\eta=1}^{d} \dfrac{\epsilon_{j,\eta}^2(\sigma)}{(\delta_\eta - \sigma)^2} - \sum_{\eta=1}^{d} \dfrac{2\epsilon_{j,\eta}(\sigma)\phi_{j,\eta}(\sigma)}{\delta_\eta - \sigma}}.$$

Multiplying both the numerator and denominator by $(\delta_k - \sigma)^2$ gives:

$$\theta_j(\sigma) = \frac{(\delta_k - \sigma)^2 \left(\hat{y}_j^T(\sigma)C_\sigma\hat{y}_j(\sigma) - \sum_{\eta=1,\eta\neq k}^{d} \dfrac{\epsilon_{j,\eta}^2(\sigma)}{\delta_\eta - \sigma}\right) - (\delta_k - \sigma)\epsilon_{j,\kappa}^2(\sigma)}{(\delta_k - \sigma)^2 \left(\hat{y}_j^T(\sigma)M_c\hat{y}_j(\sigma) + \sum_{\eta=1,\eta\neq k}^{d} \left[\dfrac{\epsilon_{j,\eta}^2(\sigma)}{(\delta_\eta - \sigma)^2} - \dfrac{2\epsilon_{j,\eta}(\sigma)\phi_{j,\eta}(\sigma)}{\delta_\eta - \sigma}\right]\right) + \gamma(j,k,\sigma)} \tag{8.20}$$

where

$$\gamma(j,k,\sigma) = \epsilon_{j,k}^2(\sigma) - 2(\delta_k - \sigma)\epsilon_{j,k}(\sigma)\phi_{j,k}(\sigma).$$

As $\sigma \to \delta_k$, the numerator in (8.20) approaches zero while the denominator approaches $\lim_{\sigma\to\delta_k} \gamma(j,k,\sigma) = \lim_{\sigma\to\delta_k} \epsilon_{j,k}^2(\sigma)$. If $\lim_{\sigma\to\delta_k} \epsilon_{j,k}(\sigma) \neq 0$, then

$$\lim_{\sigma\to\delta_k} \theta_j(\sigma) = \frac{0}{\lim_{\sigma\to\delta_k} \epsilon_{j,k}^2(\sigma)} = 0.$$

In the opposite case, the ratio in (8.20) becomes one of an undetermined form and we need to apply l'Hôspital's rule twice, which then leads to (8.19). Details are omitted. □

Corollary 8.1.4.1 *Under the assumptions stated in Theorem 8.1.4, there exists at least one* $1 \leq j \leq s$ *such that* $\lim_{\sigma\to\delta_k} \theta_j(\sigma) = 0$.

Proof: The eigenvectors of $(S(\sigma), -S'(\sigma))$ are linearly independent, and thus span the entire s-dimensional subspace. Since w_k is nonzero, there must exist at least one eigenvector $\hat{y}_j(\sigma)$ of $(S(\sigma), -S'(\sigma))$ such that $\lim_{\sigma\to\delta_k} \epsilon_{j,k}(\sigma) = \lim_{\sigma\to\delta_k} w_k^T\hat{y}_j(\sigma) \neq 0$, and thus, by Theorem 8.1.4, $\lim_{\sigma\to\delta_k} \theta_j(\sigma) = 0$. □

Example 2 *Consider the same matrices* A *and* M *as in Example 1, and let* $d = 1,\ s = 3$. *Figure 8.3 reports the values of* $\theta_j(\sigma),\ j = 1, 2, 3,$ *and* $\epsilon_{j,1}(\sigma)$ *as* $\sigma \to \delta_1$ *(*$\delta_1 = 2.0$ *is simple).*

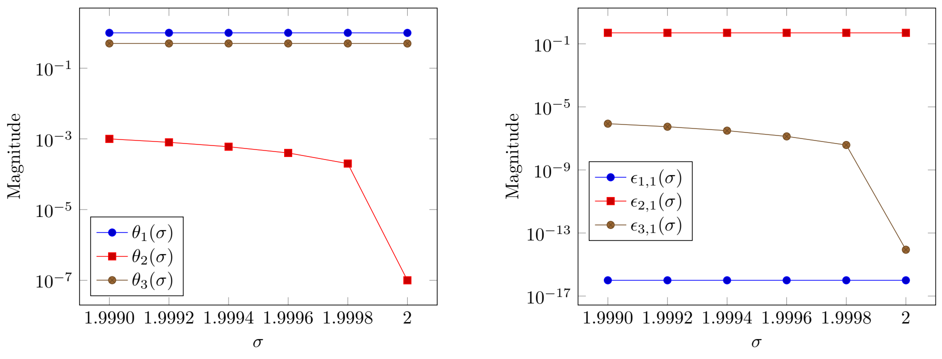


Figure 8.3: Numerical values of $\theta_j(\sigma)$ and $\epsilon_{j,1}(\sigma)$ as $\sigma \to \delta_1$.

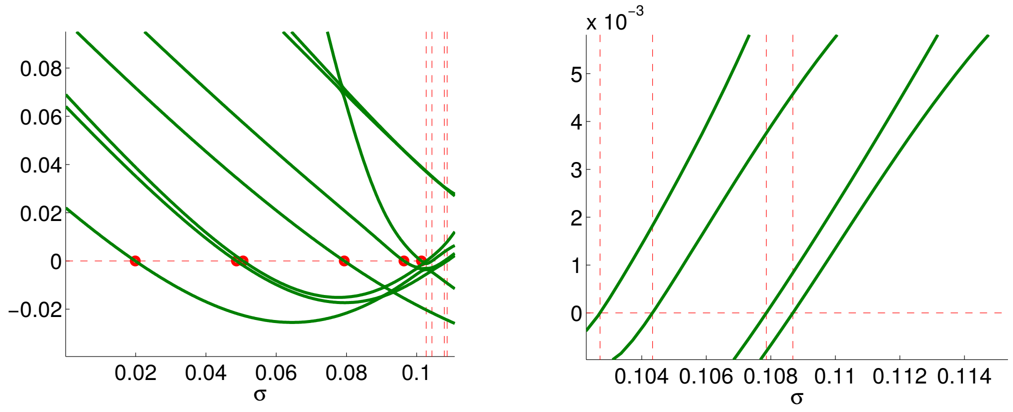


Figure 8.4: Visualization of the first few eigenbranches. Vertical dashed lines denote eigenvalues of (B, M_B). Left: Eigenbranches $\theta_j(\sigma)$ as $\sigma \in [0.0, 0.11]$. Right: Close-up view of the left subfigure focusing in the region around the four algebraically smallest eigenvalues of (B, M_B).

As predicted by Theorem 8.1.4, only the eigenbranch for which $\lim_{\sigma \to \delta_1} \epsilon_{j,1}(\sigma) \neq 0$, *in this example* $\theta_2(\sigma)$, *shifts to zero. On the other hand,* $\lim_{\sigma \to \delta_1} \epsilon_{j,1}(\sigma) = 0$ *for any* $j \neq 2$ *and thus* $\lim_{\sigma \to \delta_1} \theta_j(\sigma) \neq 0$.

Figure 8.4 shows the typical behavior of the eigenbranches as σ approaches an interval that contains one or more simple eigenvalues of (B, M_B). In particular, as we show in Theorem 8.1.5, there exists exactly one eigenbranch which crosses the real axis as σ approaches a simple eigenvalue of (B, M_B). Moreover, this eigenbranch crosses the real axis having a positive slope (see Section 8.3.1).

Definition 8.1.2 *For any non-zero scalar* $\sigma \in \mathbb{R}$ *and vector* $r \in \mathbb{R}^s$ *we define the generalized*

Rayleigh quotient:

$$\pi(\sigma, r) = \frac{r^T S(\sigma) r}{r^T[-S'(\sigma)]r}$$
$$= \frac{r^T(C - \sigma M_C)r - \sum_{\eta=1}^{d} \dfrac{(r^T w_\eta)^2}{\delta_\eta - \sigma}}{r^T M_C r + \sum_{\eta=1}^{d} \dfrac{(r^T w_\eta)^2}{(\delta_\eta - \sigma)^2} - 2\sum_{\eta=1}^{d} \dfrac{(r^T w_\eta)(r^T f_\eta)}{\delta_\eta - \sigma}}.$$

The eigenvalues of the pencil $(S(\sigma), -S'(\sigma))$ can be then characterized by a generalization of the Courant-Fischer minimax theorem as:

$$\theta_j(\sigma) = \min_{\mathbf{dim}(\mathcal{U})=j} \max_{r\in\mathcal{U},\ r\neq 0} \pi(\sigma, r).$$

Theorem 8.1.5 *Let $\delta_k \notin \Lambda(A, M)$, $1 \le k \le d$, be a simple eigenvalue of (B, M_B). Then:*

- *There exists exactly one index $1 \le \zeta \le s$ such that $\mathbf{lim}_{\sigma\to\delta_k} \theta_\zeta(\sigma) = 0$.*
- *For any eigenpair $(\theta_j(\sigma), \hat{y}_j(\sigma))$ that satisfies $\mathbf{lim}_{\sigma\to\delta_k} \theta_j(\sigma) \neq 0$, we have $\mathbf{lim}_{\sigma\to\delta_k} \hat{y}_j^T(\sigma) w_k = 0$.*

Proof: Let $\mathcal{I} = [\delta_k - \sigma_0, \delta_k) \cup (\delta_k, \delta_k + \sigma_0]$ for some positive $\sigma_0 \in \mathbb{R}$ such that $\mathcal{I}$ contains no other eigenvalues of (B, M_B) or any eigenvalues of (A, M), and let $\sigma \in \mathcal{I}$. By Corollary 8.1.4.1, we know that there exists at least one integer $1 \le \zeta \le s$ such that $\mathbf{lim}_{\sigma\to\delta_k} \theta_\zeta(\sigma) = 0$. Let $1 \le \psi \le s$ be the first such integer, i.e.,

$$\mathbf{lim}_{\sigma\to\delta_k} \theta_\psi(\sigma) = \mathbf{lim}_{\sigma\to\delta_k} \left(\min_{\mathbf{dim}(\mathcal{U})=\psi} \max_{r\in\mathcal{U},\ r\neq 0} \pi(\sigma, r) \right) = 0.$$

Since $\mathbf{lim}_{\sigma\to\delta_k} \theta_1(\sigma) \le \mathbf{lim}_{\sigma\to\delta_k} \theta_2(\sigma) \le \ldots \le \mathbf{lim}_{\sigma\to\delta_k} \theta_s(\sigma)$, it follows immediately that when $\psi = s$ the pencil $\mathbf{lim}_{\sigma\to\delta_k}(S(\sigma), -S'(\sigma))$ has exactly one zero eigenvalue. It remains to show the same when $\psi < s$.

Define the subspaces

$$\mathcal{V} \equiv \mathbf{span}\left(\{v | v^T w_k = 0\}\right),$$
$$\mathcal{Z} \equiv \mathbf{span}\left(\{v | v^T [-S'(\sigma)]\, \hat{y}_j(\sigma) = 0,\ j = 1, \ldots, \psi\}\right),$$

and let $r \in \mathcal{V} \cap \mathcal{Z}$. Because $r \in \mathcal{Z}$, we have $\mathbf{lim}_{\sigma\to\delta_k} \pi(\sigma, r) \ge 0$. In addition, because $S(\sigma)$ is non-singular we have $\mathbf{lim}_{\sigma\to\delta_k} \pi(\sigma, r) \neq 0$ for any $r \in \mathcal{V}$. Combining these two observations gives $\mathbf{lim}_{\sigma\to\delta_k}\,\pi(\sigma,r)>0$. What remains is to show that we can always find such a vector r for any $1\le\psi\le s-1$. Indeed, $\mathbf{dim}(\mathcal{Z})=s-\psi$, $\mathbf{dim}(\mathcal{V})=s-1$, and $w_k\notin\mathcal{Z}$. It follows that for any $\mathcal{U}$ such that $\mathbf{dim}(\mathcal{U})=\psi+1$, we have $\mathbf{dim}(\mathcal{U}\cap\mathcal{Z}\cap\mathcal{V})\ge 1$. Therefore

$$\mathbf{lim}_{\sigma\to\delta_k}\,\theta_{\psi+1}(\sigma)=\mathbf{lim}_{\sigma\to\delta_k}\left(\min_{\mathbf{dim}(\mathcal{U})=\psi+1}\;\max_{r\in\mathcal{U},\;r\neq 0}\;\pi(\sigma,r)\right)>0.$$

The same argument also applies to eigenvalues $\mathbf{lim}_{\sigma\to\delta_k}\,\theta_{\psi+2}(\sigma),\ldots,\mathbf{lim}_{\sigma\to\delta_k}\,\theta_s(\sigma)$ which concludes the proof of the first item.

To prove the second item, recall (8.18) and consider the variables $\epsilon_{j,\eta}(\sigma)=w_\eta^T\hat{y}_j(\sigma)$, and $\phi_{j,\eta}(\sigma)=f_\eta^T\hat{y}_j(\sigma)$, $\eta=1,\ldots,d$, where w_η and f_η are defined in (8.15). If we set

$$\chi_{j,k}(\sigma)=(\delta_k-\sigma)^2\left(\hat{y}_j^T(\sigma)C_\sigma\hat{y}_j(\sigma)-\sum_{\eta=1,\eta\neq k}^{d}\frac{\epsilon_{j,\eta}^2(\sigma)}{\delta_\eta-\sigma}\right)$$

and

$$\omega_{j,k}(\sigma)=(\delta_k-\sigma)^2\left(\hat{y}_j^T(\sigma)M_c\hat{y}_j(\sigma)+\sum_{\eta=1,\eta\neq k}^{d}\left[\frac{\epsilon_{j,\eta}^2(\sigma)}{(\delta_\eta-\sigma)^2}-\frac{2\epsilon_{j,\eta}(\sigma)\phi_{j,\eta}(\sigma)}{\delta_\eta-\sigma}\right]\right),$$

we can rewrite (8.20) as

$$\mathbf{lim}_{\sigma\to\delta_k}\,\theta_j(\sigma)=\mathbf{lim}_{\sigma\to\delta_k}\left[\frac{\chi_{j,k}(\sigma)-(\delta_k-\sigma)\epsilon_{j,\kappa}^2(\sigma)}{\omega_{j,k}(\sigma)-(\epsilon_{j,k}^2(\sigma)-2(\delta_k-\sigma)\epsilon_{j,k}(\sigma)\phi_{j,k}(\sigma))}\right].\qquad(8.21)$$

Then, for any $\mathbf{lim}_{\sigma\to\delta_k}\,\theta_j(\sigma)\neq 0$, re-arranging[1] terms in (8.21) leads to

$$\mathbf{lim}_{\sigma\to\delta_k}\,\epsilon_{j,k}^2(\sigma)=-\,\mathbf{lim}_{\sigma\to\delta_k}\left[2(\delta_k-\sigma)\epsilon_{j,k}(\sigma)\phi_{j,k}(\sigma)+\omega_{j,k}(\sigma)+\frac{\chi_{j,k}(\sigma)-(\delta_k-\sigma)\epsilon_{j,\kappa}^2(\sigma)}{\theta_j(\sigma)}\right].\qquad(8.22)$$

Since both $\mathbf{lim}_{\sigma\to\delta_k}\,\omega_{j,k}(\sigma)\to 0$ and $\mathbf{lim}_{\sigma\to\delta_k}\,\chi_{j,k}(\sigma)\to 0$, the right-hand side in (8.22) converges to zero, and thus $\mathbf{lim}_{\sigma\to\delta_k}\,\epsilon_{j,k}^2(\sigma)=\mathbf{lim}_{\sigma\to\delta_k}(w_k^T\hat{y}_j(\sigma))^2=0$. □

Let us now define $\hat{w}_\eta=w_\eta/\|w_\eta\|$, $\eta=1,\ldots,d$, where w_η is defined in (8.15), and the

[1]Note that the limit of the denominator is assumed non-zero and thus $\mathbf{lim}_{\sigma\to\delta_k}\,\dfrac{a}{b}=\dfrac{\mathbf{lim}_{\sigma\to\delta_k}\,a}{\mathbf{lim}_{\sigma\to\delta_k}\,b}$

corresponding orthogonal projector

$$P_\eta = I - \hat{w}_\eta \hat{w}_\eta. \tag{8.23}$$

Moreover, let $1 \leq k \leq d$ and assume that $\delta_k \notin \Lambda(A, M)$ is simple eigenvalue of (B, M_B). We then define the matrices

$$S_k(\sigma) = C_\sigma - \sum_{\eta=1,\eta\neq k}^{d} \frac{w_\eta w_\eta^T}{\delta_\eta - \sigma},$$

and

$$-S_k'(\sigma) = M_C + \sum_{\eta=1,\eta\neq k}^{d} \frac{w_\eta w_\eta^T}{(\delta_\eta - \sigma)^2} - \sum_{\eta=1,\eta\neq k}^{d} \frac{w_\eta f_\eta^T}{\delta_\eta - \sigma} - \sum_{\eta=1,\eta\neq k}^{d} \frac{f_\eta w_\eta^T}{\delta_\eta - \sigma}$$

and the operators

$$S_{k,|}(\sigma) \; = \; [P_k S_k(\sigma) P_k]_{|{w_k}^\perp}, \qquad -S_{k,|}'(\sigma) \; = -\left[P_k S_k'(\sigma) P_k\right]_{|{w_k}^\perp}, \tag{8.24}$$

where $[P_k S_k(\sigma) P_k]_{|{w_k}^\perp}$ and $[-P_k S_k'(\sigma) P_k]_{|{w_k}^\perp}$ are $(s-1) \times (s-1)$ matrices that denote the restriction of $P_k S_k(\sigma) P_k$ and $-P_k S_k'(\sigma) P_k$ to the subspace orthogonal to $\hat{w}_k$, respectively.

Lemma 8.1.6 *Let P_k be defined as in (8.23). Then, the eigenvalues of the matrix pencil $(S_{k,|}(\sigma),\ -S_{k,|}'(\sigma))$ are identical to the non-zero eigenvalues of the matrix pencil $(P_k S_k(\sigma) P_k$, $-P_k S_k'(\sigma) P_k)$.*

Proof: See Lemma 2.2 in [125]. $\square$

We can now prove the following theorem.

Theorem 8.1.7 *Let $\delta_k \notin \Lambda(A, M)$, $1 \leq k \leq d$, be a simple eigenvalue of (B, M_B), and let $\tau_1(\delta_k), \ldots, \tau_{s-1}(\delta_k)$ denote the eigenvalues of the pencil $(S_{k,|}(\delta_k), -S_{k,|}'(\delta_k))$. Then, the $s-1$ nonzero eigenbranches of $\lim_{\sigma \to \delta_k}(S(\sigma), -S'(\sigma))$ satisfy the equation*

$$\lim_{\sigma \to \delta_k} \theta_j(\sigma) = \tau_i(\delta_k), \tag{8.25}$$

where $1 \leq i \leq s-1$ and $j \in \{1, \ldots, s\} - \{\zeta\}$ with ζ determined by Lemma 8.1.5.

Proof: Multiplying both sides of $S(\sigma)\hat{y}_j(\sigma) = \theta_j[-S'(\sigma)]\hat{y}_j(\sigma)$ by P_k from the left, and taking advantage of the identity

$$\hat{y}_j(\sigma) = P_k\hat{y}_j(\sigma) + (\hat{w}_k^T\hat{y}_j(\sigma))\hat{w}_k,$$

leads to

$$\begin{aligned} P_kS(\sigma)\hat{y}_j(\sigma) &= \theta_j[-P_kS'(\sigma)]\hat{y}_j(\sigma) \\ P_kS(\sigma)[P_k\hat{y}_j(\sigma) + (\hat{w}_k^T\hat{y}_j(\sigma))\hat{w}_k] &= \theta_j[-P_kS'(\sigma)][P_k\hat{y}_j(\sigma) + (\hat{w}_k^T\hat{y}_j(\sigma))\hat{w}_k]. \end{aligned} \tag{8.26}$$

Reordering terms in (8.26) and noticing that

$$\begin{aligned} P_kS(\sigma)P_k &= P_kS_k(\sigma)P_k, \\ P_kS'(\sigma)P_k &= P_kS_k'(\sigma)P_k, \end{aligned}$$

finally leads to

$$P_k\left[S_k(\sigma) + \theta_j(\sigma)S_k'(\sigma)\right]P_k(P_k\hat{y}_j(\sigma)) = -[P_kS_k(\sigma) + \theta_j(\sigma)P_kS_k'(\sigma)](\hat{w}_k^T\hat{y}_j(\sigma))\hat{w}_k. \tag{8.27}$$

The right-hand side of (8.27) simply expresses the residual of the approximate eigenpair $(\theta_j(\sigma), P_k\hat{y}_j(\sigma))$ with respect to the matrix pencil $(P_kS_k(\sigma)P_k,\ -P_kS_k'(\sigma)P_k)$, which is now well-defined as $\sigma \to \delta_k$, and converges to $(P_kS_k(\delta_k)P_k, -P_kS_k'(\delta_k)P_k)$. By Lemma 8.1.6, the latter pencil is a trivial extension of the pencil $(S_{k,|}(\delta_k), -S_{k,|}'(\delta_k))$.

To finalize the proof, recall that by Lemma 8.1.5 we have that $\lim_{\sigma\to\delta_k} \hat{w}_k^T\hat{y}_j(\sigma) = 0$ for any $\lim_{\sigma\to\delta_k} \theta_j(\sigma) \neq 0$. Thus each approximate eigenpair $(\theta_j(\sigma), P_k\hat{y}_j(\sigma))$ eventually converges to an actual eigenpair of the pencil $(S_{k,|}(\delta_k), -S_{k,|}'(\delta_k))$. $\square$

For simplicity we assumed that δ_k is a simple eigenvalue of (B, M_B), but the results in Theorem 8.1.7 can be easily extended to the case where δ_k is semi-simple.

Remark 5 *By Proposition 8.1.1, each eigenbranch $\theta_j(\sigma)$ is an analytic (and thus continuous) function of σ in each interval (δ_{k-1}, δ_k), $k = 2, \ldots, d$. Moreover, by Theorem 8.1.7, the limits $\lim_{\sigma\to\delta_k} \theta_j(\sigma)$, $j \neq \kappa_\sigma$, exist and converge to the same value as σ approaches δ_k from any side. It follows that each eigenbranch $\theta_j(\sigma)$, $j = 1, \ldots, s$ is a continuous functions of $\sigma \in \mathbb{R}$ except at those real points that are equal to an eigenvalue δ_k of (B, M_B) and, at the same time, $\lim_{\sigma\to\delta_k} \theta_j(\sigma) \to 0$. In the latter case, the limit exists, but $\theta_j(\delta_k)$ is not formally defined.*

8.2 Practical aspects of MLA

8.2.1 Determining which roots of $\theta(\sigma)$ are eigenvalues of (A, M)

In Section 8.1 we saw that the eigenvalues of both matrix pencils (A, M) and (B, M_B) are roots of $\theta(\sigma)$. In this section we describe a simple mechanism to identify which of these roots are eigenvalues of (A, M).

Definition 8.2.1 *(Sylvester's law of matrix inertia [118]) The inertia of a symmetric matrix $X \in \mathbb{R}^{n\times n}$, is a triplet $[\nu_-(X), \nu_0(X), \nu_+(X)]$ consisting of the numbers of negative, zero, and positive eigenvalues of X, respectively.*

Proposition 8.2.1 *A real interval $[\alpha, \beta]$ contains one or more eigenvalues of (A, M) if and only if $\nu_-(A - \beta M) > \nu_-(A - \alpha M)$. Moreover, it holds that $\nu_-(A - \beta M) = \nu_-(S(\beta)) + \nu_-(B_\beta)$, and $\nu_-(A - \alpha M) = \nu_-(S(\alpha)) + \nu_-(B_\alpha)$.*

Proof: Writing $(A - \sigma M)x = (\lambda - \sigma)Mx$, and noticing that M is SPD, the number of eigenvalues of (A, M) that are less than σ is equal to the number of negative eigenvalues of $A - \sigma M$. Thus, $[\alpha, \beta]$ contains one or more eigenvalues of (A, M) if and only if $\nu_-(A-\beta M) > \nu_-(A-\alpha M)$.

Now, consider the following congruence transformation of $A - \sigma M$:

$$\begin{pmatrix} I & \\ -E_\sigma^T B_\sigma^{-1} & I \end{pmatrix} \begin{pmatrix} B_\sigma & E_\sigma \\ E_\sigma^T & C_\sigma \end{pmatrix} \begin{pmatrix} I & -B_\sigma^{-1} E_\sigma \\ & I \end{pmatrix} = \begin{pmatrix} B_\sigma & \\ & S(\sigma) \end{pmatrix}. \tag{8.28}$$

Since congruence transformations preserve inertias, the inertia of $A - \sigma M$ is equal to the inertia of the block-diagonal matrix on the right-hand side of (8.28), which in turn is equivalent to the sum of the inertias of B_σ and $S(\sigma)$. Thus, $\nu_-(A - \sigma M) = \nu_-(S(\sigma)) + \nu_-(B_\sigma)$. □

Since $-S'(\sigma)$ is SPD, the inertia of $S(\sigma)$ is identical to the inertia of the pencil $(S(\sigma), -S'(\sigma))$. We can now show the following Proposition.

Proposition 8.2.2 *Let $\tau \in \mathbb{R}$ be a simple root of $\theta(\sigma)$ such that either $\tau \in \Lambda(A, M)$ or $\tau \in \Lambda(B, M_B)$. Moreover, let $\tau \in [\tau^-, \tau^+]$ where τ^-, τ^+ are two real scalars located within an infinitesimal distance from τ such that a) the index κ_σ of $\theta(\sigma) \equiv \theta_{\kappa_\sigma}(\sigma)$ remains constant for any $\sigma \in [\tau^-, \tau^+]$, and b) $[\tau^-, \tau^+]$ includes no other roots of $\theta(\sigma)$.*

Then:

- *If $\theta(\tau^-) > 0$ and $\theta(\tau^+) < 0$, $\tau \in \Lambda(A, M)$. The converse also holds.*
- *If $\theta(\tau^-) < 0$ and $\theta(\tau^+) > 0$, $\tau \in \Lambda(B, M_B)$. The converse also holds.*

Proof: By assumption $1 \le \kappa_\sigma \le s$ remains constant for any $\sigma \in [\tau^-, \tau^+]$. In addition, $\theta_j(\sigma),\ j \ne \kappa_\sigma$, are continuous functions of σ in $[\tau^-, \tau^+]$, and thus their sign is fixed (since they do not cross the real axis).

Consider the first item and let $\theta(\tau^-) > 0$ and $\theta(\tau^+) < 0$. We then have $\nu_-(S(\tau^+)) = \nu_-(S(\tau^-)) + 1$. Now, assume that $\tau \in \Lambda(B, M_B)$. Then, we also have $\nu_-(B_{\tau^+}) = \nu_-(B_{\tau^-}) + 1$. Combining the two latter equations leads to $\nu_-(A - \tau^+ M) = \nu_-(A - \tau^- M) + 2$, i.e., τ is also a semi-simple eigenvalue of (A, M). The latter is a contradiction since τ was assumed an (simple) eigenvalue of either (A, M) or (B, M_B). Thus, $\tau \in \Lambda(A, M)$. To show the converse, let $\tau \in \Lambda(A, M)$. Then $\nu_-(A - \tau^+ M) = \nu_-(A - \tau^- M) + 1$. On the other hand, since $\tau \notin \Lambda(B, M_B)$, we have $\nu_-(B_{\tau^+}) = \nu_-(B_{\tau^-})$, which in turn implies $\nu_-(S(\tau^+)) = \nu_-(S(\tau^-)) + 1$. Since $\theta(\sigma)$ is the only eigenbranch that can cross the real axis and change sign in the interval $[\tau^-, \tau^+]$, it follows that $\theta(\tau^-) > 0$ and $\theta(\tau^+) < 0$.

To show the second item, let $\theta(\tau^-) < 0$ and $\theta(\tau^+) > 0$. Following the same reasoning as above, we get $\nu_-(S(\tau^+)) = \nu_-(S(\tau^-)) - 1$. Now, assume that $\tau \in \Lambda(A, M)$. We then have $\nu_-(A - \tau^+ M) = \nu_-(A - \tau^- M) + 1$, which implies that $\nu_-(B_{\tau^+}) = \nu_-(B_{\tau^-}) + 2$, i.e., τ is also a semi-simple eigenvalue of (B, M_B). The latter is a contradiction since τ was assumed an (simple) eigenvalue of either (A, M) or (B, M_B). Thus, $\tau \in \Lambda(B, M_B)$. To show the converse, let $\tau \in \Lambda(B, M_B)$. Then $\nu_-(B_{\tau^+}) = \nu_-(B_{\tau^-}) + 1$. On the other hand, because $\tau \notin \Lambda(A, M)$, we have $\nu_-(A - \tau^- M) = \nu_-(A - \tau^+ M)$, which in turn implies $\nu_-(S(\tau^+)) = \nu_-(S(\tau^-)) - 1$. Since $\theta(\sigma)$ is the only eigenbranch that can cross the real axis and change sign in the interval $[\tau^-, \tau^+]$, it follows that $\theta(\tau^-) < 0$ and $\theta(\tau^+) > 0$. $\square$

In practice, the conditions in Proposition 8.2.2 can be replaced by the conditions $\theta'(\sigma) < 0$, and $\theta'(\sigma) > 0$, respectively, for any σ sufficiently close to the root.

8.2.2 A categorization of the roots

Definition 8.2.2 *We define the "upslope roots" of an eigenbranch $\theta_j(\sigma)$, $1 \le j \le s$, to be those real scalars $\delta \in \Lambda(B, M_B)$ satisfying $\mathbf{lim}_{\sigma \to \delta}\, \theta_j(\sigma) = 0$. Similarly, we define the "regular roots" of an eigenbranch $\theta_j(\sigma)$ to be those real scalars $\lambda \in \Lambda(A, M)$ satisfying $\theta_j(\lambda) = 0$.*

Proposition 8.2.3 *Let n and d be the size of matrix pencils (A, M) and (B, M_B), respectively, and assume that $\Lambda(A, M) \cap \Lambda(B, M_B) = \varnothing$. Then, each eigenbranch $\theta_j(\sigma)$, $1 \leq j \leq s$, $s = n - d$, has:*

- *at least one regular root and at most n regular roots, and*
- *none, one, or at most d upslope roots.*

Proof: Let σ_l and σ_r denote two real scalars chosen such that $[\lambda_1, \lambda_n] \subset [\sigma_l, \sigma_r]$, and recall that $n = d + s$ where d and s denote the size of matrices B_σ and $S(\sigma)$, respectively. By Proposition 8.2.1 we have

$$(\nu_-(S(\sigma_r)) + \nu_-(B_{\sigma_r})) - (\nu_-(S(\sigma_l)) + \nu_-(B_{\sigma_l})) = n.$$

By a generalization of Cauchy's interlacing property we have $[\delta_1, \delta_d] \subseteq [\lambda_1, \lambda_n]$, and thus $\nu_-(B_{\sigma_r}) - \nu_-(B_{\sigma_l}) = d$, leading to $\nu_-(S(\sigma_r)) - \nu_-(S(\sigma_l)) = s$. In addition, $-S'(\sigma_l)$ and $-S'(\sigma_r)$ are both SPD, and thus the inertias of matrices $S(\sigma_l)$ and $S(\sigma_r)$ are identical to those of matrix pencils $(S(\sigma_l), -S'(\sigma_l))$ and $(S(\sigma_r), -S'(\sigma_r))$, respectively. Clearly, $(S(\sigma_l), -S'(\sigma_l))$ has no negative eigenvalues, while $(S(\sigma_r), -S'(\sigma_r))$ has no positive eigenvalues. Thus, all s eigenbranches change their sign from positive to negative at least once within the interval $[\sigma_l, \sigma_r]$. Since we assumed that $\Lambda(B, M_B) \cap \Lambda(A, M) = \varnothing$, Proposition 8.2.2 suggests that the sign of an eigenbranch changes from positive to negative only when it crosses an eigenvalue of (A, M), and thus the proof of the first item is completed.

Regarding the second item, there exist at most d upslope roots overall, and thus this is also the largest number of upslope roots an eigenbranch can have. To see that an eigenbranch can have no upslope roots, simply consider the case where $d < s$. $\square$

Figure 8.5 plots the eigenbranches of the toy matrix pencil shown in (8.14), $A = \left(\begin{smallmatrix} B & E \\ E^T & C \end{smallmatrix}\right) = \left(\begin{smallmatrix} 2 & 1 & 0 & 1 \\ 1 & 3 & 1 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 0 & 2 \end{smallmatrix}\right)$, as the dimension d of the leading principal submatrix B varies (M was chosen as the identity matrix). When $d = n - 1 = 3$, there exists only one eigenbranch and this eigenbranch crosses the real axis at all real points $\sigma \in \Lambda(B, M_B) \cup \Lambda(A, M)$. Note that although $\lambda_1 = \lambda_2 = 1$, only one copy of this semi-simple eigenvalue can be computed. On the other hand, when $d = n - 2 = 2$ we have $s = 2$, and both eigenbranches cross the real axis at $\sigma = \lambda_1 = \lambda_2$. More generally, MLA can compute the correct multiplicity of a semi-simple eigenvalue of (A, M) only if the latter is less or equal than s. In agreement with Proposition 8.2.3, each eigenbranch has

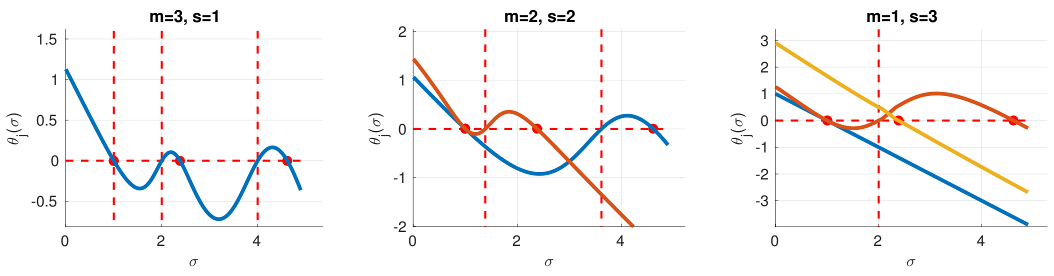


Figure 8.5: Eigenbranches of the matrix appearing in (8.14) for different values of $d \equiv m$ and s. The red circles along the real axis denote eigenvalues of the pencil (A, M). The dashed vertical lines denote eigenvalues of matrix B. Left: $B = A(1:3, 1:3)$. Middle: $B = A(1:2, 1:2)$. Right: $B = A(1:1, 1:1)$.

at least one regular root, and none, one, or at most d upslope roots. For example, when $d = 1$, there exist two eigenbranches that have one regular root but no upslope root.

8.2.3 The impact of the location of the eigenvalues of (B, M_B)

Ideally, we would like the shape of the eigenbranches of $(S(\sigma), -S'(\sigma))$ to be as close to that of a linear function as possible. In particular, we are interested on the shape of the eigenbranch $\theta_j(\sigma)$ of $(S(\sigma), -S'(\sigma))$ that satisfies $\mathbf{lim}_{\sigma \to \lambda} \theta_j(\sigma) = 0$.

Let us first consider the case where $\sigma \approx \lambda$, i.e., σ lies in a small neighborhood around λ. In this scenario, we expect the shape of the eigenbranch $\theta_j(\sigma)$ to be very close to linear since $\theta_j(\sigma) \approx 0$ and thus the derivative $\theta_j'(\sigma) = -(1 + O(\theta(\sigma))$ is almost constant. But what can we tell about the shape of $\theta_j(\sigma)$ when σ is located further away from λ? While a detailed analysis lies outside the scope of this paper, we have observed that the further σ lies from the upslope root(s) of $\theta_j(\sigma)$, the closer to linear the shape of $\theta_j(\sigma)$ is. This observation implies that, ideally, the sought eigenvalue λ should lie as far as possible from the upslope roots of the eigenbranch that satisfies $\mathbf{lim}_{\sigma \to \lambda} \theta_j(\sigma) = 0$.

While it is impossible to either impose any control over the eigenvalues of (B, M_B) or determine a-priori what eigenvalues of (B, M_B) are the upslope roots of a given eigenbranch $\theta_j(\sigma)$, it is possible to reduce the total number of upslope roots by reducing the size of (B, M_B). For example, the eigenvalues of the pencil (B, M_B) in Figure 8.5 are $\{1, 2, 4\}$, $\{1.38, 3.61\}$ and $\{2\}$, when the size of matrices B and M_B is set to $d = 3$, $d = 2$, and $d = 1$, respectively. Therefore, choosing the dimension d of the matrix pencil (B, M_B) to be smaller increases the likelihood

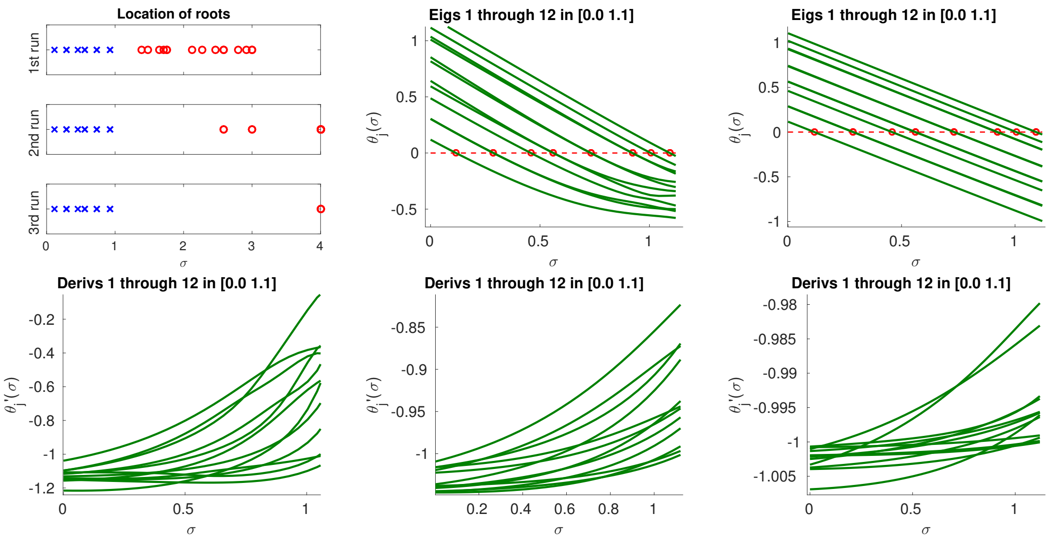


Figure 8.6: Top row (left subfigure): the eigenvalues of (B, M_B) located inside the interval $[0.0, 4.0]$ (marked by "$\circ$") for three different arrangements. The eigenvalues $\lambda_1, \ldots, \lambda_{12}$ of (A, M) (marked by "x") are also shown. Top row (middle and right subfigures): plot of $\theta_j(\sigma),\ 1 \leq j \leq 12$ for the first and second arrangement of the algebraically smallest eigenvalues of (B, M_B). Bottom row: plot of $\theta'_j(\sigma),\ 1 \leq j \leq 12$, for the first (left), second (middle), and third (right) arrangement of the eigenvalues of (B, M_B).

that a regular root $\lambda \in \Lambda(A, M)$ of an eigenbranch $\theta_j(\sigma)$ will be located relatively further away from the upslope roots of the latter. On the other hand, choosing a smaller d increases the size of the eigenvalue problem at each iteration in MLA.

Example 3 *We consider the computation of eigenvalues $\lambda_1, \ldots, \lambda_{12}$ of a SPD pencil (A, M) generated by a 5-pt Finite Difference discretization of the Laplace operator on the unit plane (M is chosen as the identity matrix). Each eigenvalue $\lambda_j,\ j = 1, \ldots, 12$, is a regular root of eigenbranch $\theta_j(\sigma)$, and we focus on the shape of eigenbranches $\theta_j(\sigma),\ j = 1, \ldots, 12$, as their upslope roots located the closest to $[\lambda_1, \lambda_{12}]$ are progressively shifted away. More specifically, we consider three different arrangements of the upslope roots (we only show those located within the interval $[0.0, 4.0]$), each one of them formed by progressively increasing the number of interface variables of the pencil (A, M) (thus reducing the value of d). The exact location of eigenvalues $\lambda_1, \ldots, \lambda_{12}$ (some of these eigenvalues were semi-simple), as well as the location of the algebraically smallest upslope roots of eigenbranches $\theta_j(\sigma),\ j = 1, \ldots, 12$, are shown in Figure 8.6. In the same figure we also plot eigenbranches $\theta_j(\sigma),\ 1 \leq j \leq 12$, for the first and second arrangement of the upslope roots. It is easy to verify that for the second arrangement of the* *upslope roots the shape of the eigenbranches is closer to linear compared to that for the first arrangement. The latter is shown more clearly in the bottom row subfigures in Figure 8.6 where we plot* $\theta_j'(\sigma) = -1 - \theta_j(\sigma)\dfrac{\hat{y}_j^T(\sigma)S''(\sigma)\hat{y}_j(\sigma)}{\hat{y}_j^T(\sigma)S'(\sigma)\hat{y}_j(\sigma)}$*,* $j = 1, \ldots, 12$*, for the first (left), second (middle), and third (right) arrangement of the upslope roots.*

8.3 Numerical Experiments

In this section we consider the performance of MLA. The experiments were performed in a Matlab environment (version R2016a), using 64-bit arithmetic (double precision), on a single core of a computer system equipped with an Intel Haswell[2] E5-2680v3 processor and 32 GB of system memory.

8.3.1 Details on the experimental framework

Throughout the rest of this section we will be comparing three different schemes:

1. The MLA method proposed in Algorithm 8.1.1.

2. The "Branch Hopping" root-finding technique presented in [71]. This method essentially coincides with the MLA method applied to the matrix pencil $(S(\sigma), I)$, and will be abbreviated as "BrH".

3. The Rayleigh Quotient Iteration (RQI), a scheme which is known to converge (asymptotically) cubically for symmetric eigenvalue problems [29]. We note here that no variants of RQI that pre-process the initial eigenvalue approximation by Inverse Iteration so as to improve robustness were considered, e.g. see [119].

Each approximate eigenpair $\left(\tilde{\lambda}, \tilde{x}\right)$ will be signaled as a sufficiently accurate approximation of an eigenpair (λ, x) of (A, M) as soon as the corresponding residual norm satisfies $\|A\tilde{x} - \tilde{\lambda}M\tilde{x}\| \leq 1e-8 \times \|\tilde{x}^T M\tilde{x}\|$. Each approximation $(\tilde{\theta}, \tilde{y})$ of the eigenpair $(\theta(\sigma), \hat{y}(\sigma))$ will be considered sufficiently accurate as soon as the residual norm satisfies $\|S(\sigma)\tilde{y} + \tilde{\theta}S'(\sigma)\tilde{y}\| \leq 1e-10 \times \|\tilde{y}^T S'(\sigma)\tilde{y}\|$.[3] The backslash "(\)" Matlab operator is set as the default linear system solver. Each eigenpair $(\theta(\sigma), \hat{y}(\sigma))$ is computed by our own implementation of Inverse Iteration.

[2]See https://ark.intel.com/

[3]In practice we found that the inner eigenvalue problem can be solved even less accurately (see [126] for more details on the convergence of inexact Newton methods).

Unless mentioned otherwise, $p = 16$ will be the default number of subdomains used throughout this section.

8.3.2 Results

Table 8.1: n: size of A and M, $nnz(.)$: number of nonzero entries.

#	Matrix pencil	n	$nnz(A)/n$	$nnz(M)/n$	s	Application
1.	nos5	468	11.1	1.0	251	Structural
2.	nos3	960	16.5	1.0	310	Structural
3.	bcsst{k,m}27	1,224	45.9	45.9	418	Structural
4.	FEmesh	2,689	6.9	6.8	287	Model problem
5.	saylr4	3,564	6.3	1.0	762	CFD
6.	FDmesh	5,000	6.6	1.0	433	Model problem
7.	{K,M}uu	7,102	47.9	24.0	911	Structural

We consider the application of the numerical schemes listed in Section 8.3.1 to the computation of the ten lowest eigenvalues (and eigenvectors) of the matrix pencils listed in Table 8.1. All sought eigenvalues were simple. The entries under the label 's' denote the number of interface variables for the default choice of $p = 16$ subdomains. All matrix pencils but 4) and 6) can be found in the Suite Sparse Matrix Collection. Matrix "FDmesh" represents a five-point Finite Difference approximation of the Laplace operator on a regular grid with homogeneous Dirichlet boundary conditions on the entire boundary of the domain $[0, 1] \times [0, 1]$ using a mesh size $n_x = 100$ and $n_y = 50$ along the first and second dimension, respectively. Matrix pencil "FEmesh" represents a Finite Elements discretization of the $[-1, 1] \times [-1, 1]$ plane using linear elements with target maximum mesh edge length of $h = 0.05$. Matrix pencils 1), 2), 5) and 6) are of the standard form, i.e., (A, I), where I denotes the identity matrix of appropriate size.

Table 8.2 reports the total number of iterations required by MLA, BrH, and RQI, to compute the ten algebraically smallest eigenvalues and associated eigenvectors of the matrix pencils listed in Table 8.1. The initial eigenvalue approximation σ_0 of each sought eigenvalue λ was set to either $\sigma_0 = \lambda(1+1e-2)$ (left bar), or $\sigma_0 = \lambda(1+1e-3)$ (right bar). The number of subdomains was set to its default value $p = 16$ for both values of σ_0. The values inside the parentheses denote the total number of eigenpairs missed due to misconvergence, i.e., the number of times each scheme converged to a previously computed eigenpair. For the values of p and σ_0 reported[4] in this section, we found RQI to be considerably less robust than the root-based eigensolvers.

[4]For example, when $p = 2$ and $\sigma_0 = \lambda(1+1e-2)$, the root-based eigensolvers also misconverged to a previously computed eigenpair for some of the matrices listed in Table 8.1.

Table 8.2: Total number of iterations required by MLA, BrH, and RQI, to compute the ten algebraically smallest eigenvalues and associated eigenvectors of the matrix pencils listed in Table 8.1.

	$\sigma_0 = \lambda(1+1e-2)$			$\sigma_0 = \lambda(1+1e-3)$		
Matrix	MLA	BrH	RQI	MLA	BrH	RQI
nos5	26	33	36(1)	20	30	30
nos3	24	29	35(1)	20	26	28
bcsst{k,m}27	31	36	44(3)	30	30	35
FEmesh	31	38	40	23	30	31
saylr4	24	29	49(4)	20	26	35(4)
FDmesh	29	35	36(6)	22	30	44(6)
{K,M}uu	30	38	36	22	31	28

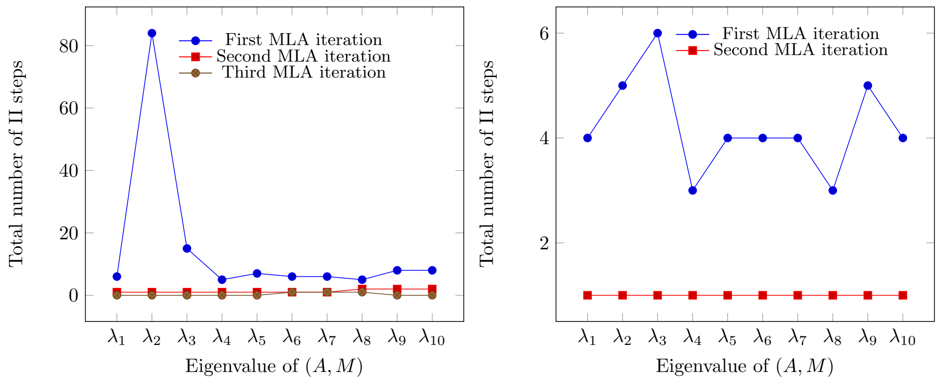


Figure 8.7: Total number of steps required by Inverse Iteration (II) at each individual iteration of MLA when computing the ten algebraically smallest eigenvalues of the matrix "nos3". The initial eigenvalue approximation σ_0 for each sought eigenvalue λ was set to $\sigma_0 = \lambda(1+1e-2)$ (left subfigure) and $\sigma_0 = \lambda(1+1e-3)$ (right subfigure).

More generally, the root-based eigensolvers were found faster (in terms of convergence) and more robust than RQI for virtually all values of p and σ_0 we tested.

Convergence of Inverse Iteration as the inner eigenvalue solver

Figures 8.7 and 8.8 plot the total number of steps required by Inverse Iteration to compute $\theta(\sigma)$ at each individual iteration of MLA when computing the ten algebraically smallest eigenvalues of the matrix "nos3" and "saylr4", respectively. As the iteration index of MLA increases, the number of Inverse Iteration steps per iteration decreases. Indeed, as MLA converges towards the sought simple eigenvalue (root) λ of (A, M), $\theta(\sigma)$ converges to zero, while, at the same time, the rest of the eigenvalues of matrix pencil $(S(\sigma), -S'(\sigma))$ converge to nonzero values. On the

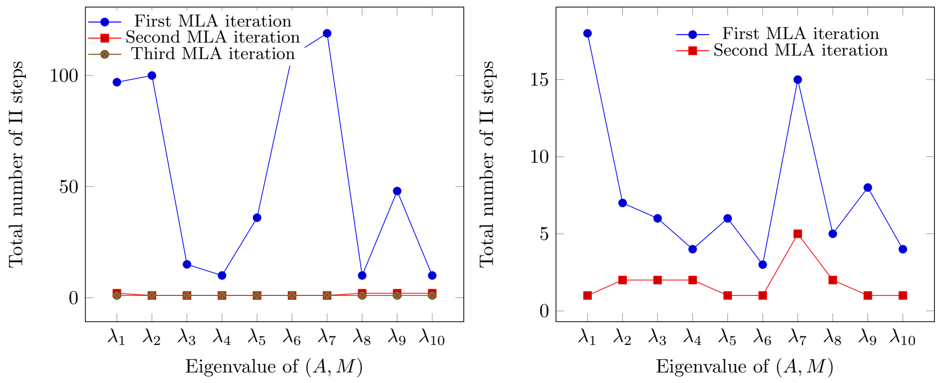


Figure 8.8: Total number of steps required by Inverse Iteration (II) at each individual iteration of MLA when computing the ten algebraically smallest eigenvalues of the matrix “saylr4”. The initial eigenvalue approximation σ_0 for each sought eigenvalue λ was set to $\sigma_0 = \lambda(1 + 1e - 2)$ (left subfigure) and $\sigma_0 = \lambda(1 + 1e - 3)$ (right subfigure).

other hand, a cluster of eigenvalues of (A, M) around σ leads to a cluster of eigenvalues around the origin in matrix $S(\sigma)$. As a result, more Inverse Iteration steps are needed to compute $\theta(\sigma)$. Eventually, σ will come much closer to the sought eigenvalue λ compared to the rest of the eigenvalues of (A, M). As soon as this regime is achieved, one or two steps of Inverse Iteration per MLA iteration will generally be enough.

8.3.3 Increasing the number of interface variables

Increasing the number of interface variables leads to fewer upslope roots. This seems to have a positive effect on the convergence of root-finding eigenvalue solvers as was already hinted in Section 8.2.3. While an exhaustive experimental analysis lies outside the context of this paper, the remaining of this section presents some illustrative results.

Figure 8.9 plots the eigenbranches $\theta_j(\sigma),\ j = 1, \ldots, 10$ (top) and associated derivatives $\theta_j'(\sigma)$ (middle) of the matrix “nos3” as the number of subdomains varies from $p = 16$ (left) to $p = 64$ (right), i.e., as the number of interface variables increases. Similarly to Section 8.2.3, increasing the number of interface variables led to eigenbranches whose shape is closer to that of a linear function. The bottom row plots the relative residual curves of MLA, BrH, and RQI obtained using $p = 16$ (left) and $p = 64$ (right) subdomains. The initial approximation of each sought eigenvalue was set to $\sigma = \lambda(1 + 1e - 2)$. Increasing the number of interface variables has a positive effect on MLA. On the other hand, this has minimal effects on the convergence of RQI.

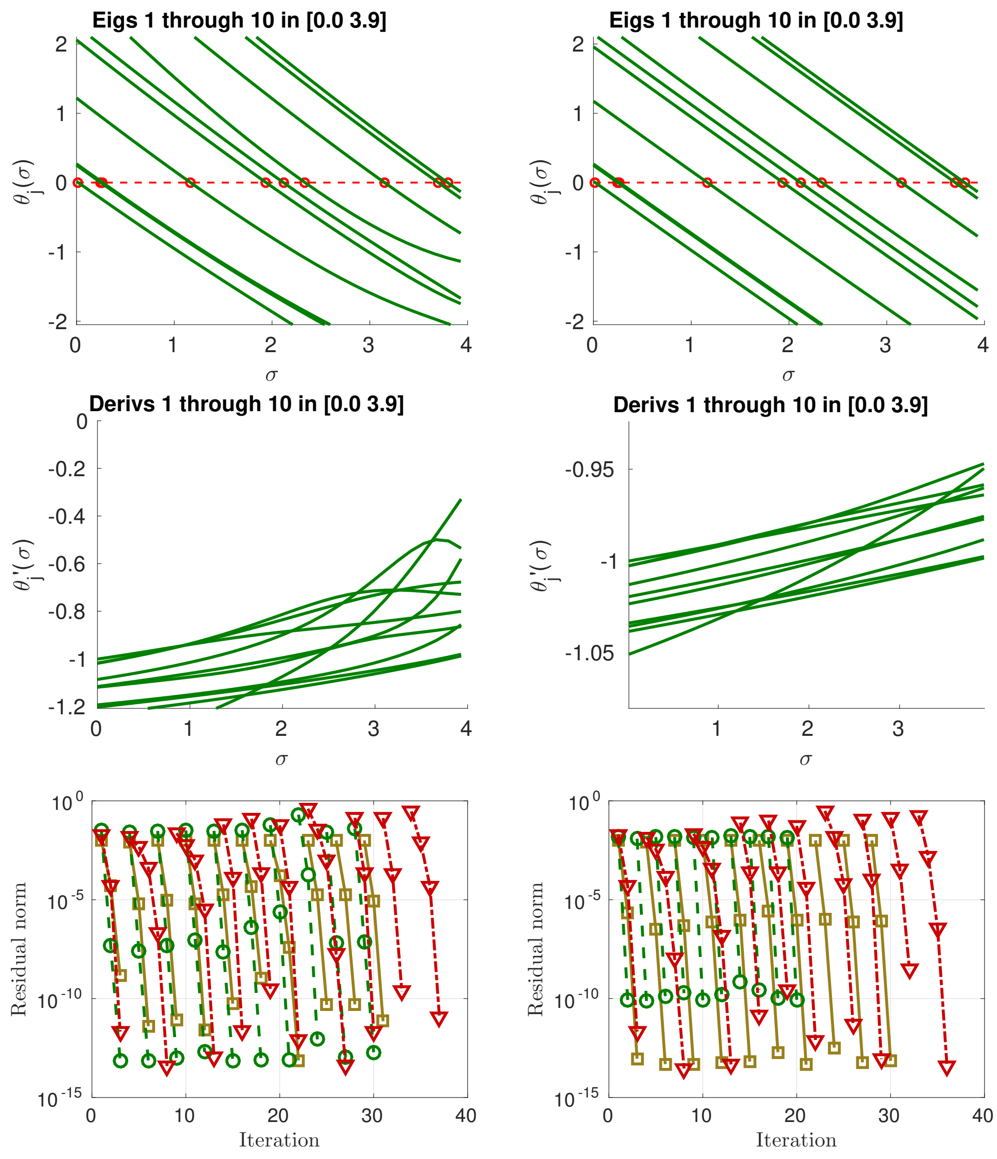


Figure 8.9: Eigenbranches $\theta_j(\sigma)$ (top) and associated derivatives $\theta_j'(\sigma)$ (middle) of the matrix "nos3" as the number of subdomains varies from $p = 16$ (left) to $p = 64$ (right). The bottom figures plot the relative residual curves of MLA ("◯"), BrH ("□") and RQI ("▽") obtained using $p = 16$ (left) and $p = 64$ (right) subdomains.

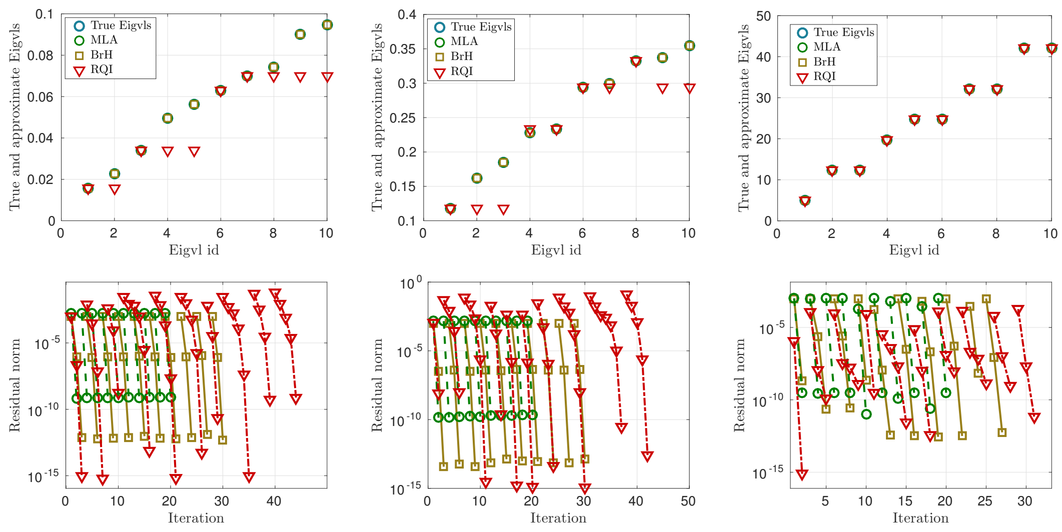


Figure 8.10: A comparison of MLA ("◯"), BrH ("□") and RQI ("▽"). The initial approximation of each sought eigenvalue was determined as $\sigma := \lambda(1+1e-3)$. Left: The ten algebraically smallest eigenvalues of (A, M) and the approximate eigenvalues returned by MLA, BrH, and RQI. Right: Relative residual curves. First column: a 65×26 Dirichlet eigenvalue problem $(M = I)$. Second column: a $25 \times 20 \times 10$ Dirichlet eigenvalue problem $(M = I)$. Third column: "FEmesh".

The latter was expected since RQI only solves a linear system of the form $(A - \sigma M)x = b$, and the solution vector x is not affected by the size of the Schur complement matrix.

Figures 8.10 and 8.11 show a comparison of MLA, BrH, and RQI when each one of these techniques is applied to the computation of the ten algebraically smallest eigenvalues of various matrix pencils. The number of subdomains used in this section was larger than before, in particular we set $p = 64$. For all matrix pencils considered, the root-based techniques proved to be considerably faster than RQI. Moreover, the root-based techniques were considerably more robust than RQI in terms of misconvergence for the first two pencils considered. It is worth mentioning that when $p = 64$ and $\sigma = \lambda(1 + 1e - 3)$ MLA required exactly two iterations per eigenpair to converge to the sought accuracy.[5]

Remark 6 *While MLA might converge faster than RQI, the computational cost per iteration of the former is higher than that of the latter, since MLA requires repeated linear system solutions with matrix $S(\sigma)$ until the eigenpair of smallest magnitude of $(S(\sigma), -S'(\sigma))$ is computed.*

[5]In practice MLA required only one iteration per eigenpair as the second step only verified that the sought accuracy was already reached.

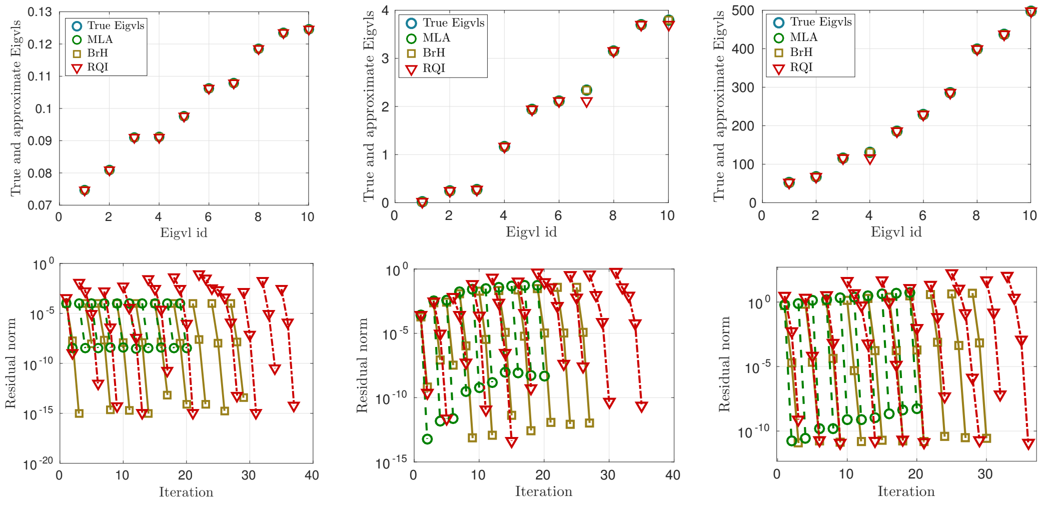


Figure 8.11: A comparison of MLA ("◯"), BrH ("□") and RQI ("▽"). The initial approximation of each sought eigenvalue was determined as $\sigma := \lambda(1+1e-3)$. Left: The ten algebraically smallest eigenvalues of (A, M) and the approximate eigenvalues returned by MLA, BrH, and RQI. Right: Relative residual curves. First column: an artificial generalized $35\times 20\times 5$ Dirichlet eigenvalue problem with $M = \texttt{toeplitz}(2, 1, \text{zeros}(1, n-2))$. Second column: "nos3" ($M = I$). Third column: "nos5" ($M = I$).

In contrast, RQI requires only a single linear system solution with $S(\sigma)$. Assuming that both schemes converge to the same eigenpair, MLA might be a better alternative than RQI when the computational cost to form and factorize $S(\sigma)$ is higher than the cost associated with solving a few linear systems with the matrix $S(\sigma)$.

8.4 Summary

In this chapter we proposed the Mixed Linear Approximations (MLA) scheme, an approach based in domain decomposition and root-finding. MLA recasts the original linear eigenvalue problem into one of computing roots of scalar functions defined by the eigenvalues of a generalized eigenvalue problem stemming by a first-order approximation of the non-linear matrix-valued function associated with the interface variables. To solve this root-finding problem MLA considers Newton's iteration. We discussed several theoretical and implementation details, and demonstrated by experiments that MLA can converge rapidly with a rate which is comparable of even faster than that of the Rayleigh Quotient Iteration, while also being more robust.

Chapter 9

Summary and future work

This dissertation focused on domain decomposition techniques for the solution of symmetric generalized eigenvalue problems. In this chapter we summarize the contributions of the present dissertation and discuss possible future research directions.

9.1 Filtering techniques

The first class of numerical techniques presented in this dissertation applied domain decomposition onto a transformation $\rho(.)$ of the matrix pencil (A, M). This transformation was chosen so that: (1) the *nev* eigenvalues of the matrix pencil (A, M) located inside the interval $[\alpha, \beta]$ become those of largest magnitude in the transformed matrix pencil $\rho(A, M)$, and (2) the eigenvalues of the matrix pencil (A, M) located outside the interval $[\alpha, \beta]$ become approximately zero in the transformed matrix pencil $\rho(A, M)$. The eigenvectors of the transformed matrix pencil $\rho(A, M)$ are identical to those of the matrix pencil (A, M). Both rational and polynomial transformations were considered.

In Chapter 3 we combined domain decomposition with rational filtering and Krylov subspace projection schemes. The proposed technique, abbreviated as RF-DDES, applies the rational filter only to the region defined by the interface variables. This approach leads to advantages such as (1) reduced orthogonalization costs, (2) reduced usage of complex arithmetic, and (3) the ability to achieve convergence in even fewer than *nev* iterations. As part of future work, we aim to extend RF-DDES by considering additional levels of parallelism. In addition to the ability to divide the initial interval $[\alpha, \beta]$ into non-overlapping subintervals and apply RF-DDES to each one of them in parallel, e.g. see [58, 43], we can also assign the complex linear system solutions associated with different quadrature nodes to distinct processor groups. Moreover, these linear systems can be solved by distributed preconditioned Krylov subspace approaches which could be helpful when RF-DDES is applied to the solution of symmetric eigenvalue problems arising from discretization of 3D domains. To this end, one option we are currently exploring is to combine RF-DDES with the distributed memory extension of the work in [127]. Another interesting research direction would also be to explore recursive implementations of RF-DDES. For example, RF-DDES could be applied individually to each matrix pencil $\left(B_\sigma^{(j)}, M_B^{(j)}\right),\ j = 1, \ldots, p$. This could be particularly helpful when either d_j, the number of interior variables of the jth subdomain, or $nev_B^{(j)}$, are large. On the algorithmic side, it would be of interest to develop more efficient criteria to set the value of $nev_B^{(j)},\ j = 1, \ldots, p$ in each subdomain; perhaps by adapting the work in [80].

In Chapter 4 we combined domain decomposition with contour integral eigenvalue solvers. In particular, we presented two numerical schemes. The first scheme, abbreviated as DD-FP, is similar to FEAST with the difference that the Schur complement linear systems are solver by a hybrid iterative solver. The second scheme, abbreviated as DD-PP, integrates the matrix resolvent only partially. Experiments performed in distributed memory environments verified the superiority of the proposed schemes over popular contour integral eigenvalue solvers such as FEAST. One major direction we plan to explore as part of future work is the development of techniques to increase the accuracy of DD-PP. As presented, DD-PP is a one-shot algorithm. However, it is possible to increase its accuracy by considering ideas similar to those in RF-DDES discussed in Chapter 3. More specifically, we can keep applying the matrix $\Re e \left\{ \sum_{\ell=1}^{N_c} \omega_\ell S(\zeta_\ell)^{-1} \right\}$ onto a set of new vectors each time till we capture its range space or the numerical rank of the product matrix stops increasing. The main advantage of this approach over using Lanczos (as RF-DDES does) is that (1) it allows for inexact solves, and (2) its simplicity when block linear system solvers are considered. Other future research directions include the incorporation of a distributed block GMRES linear system solver in DD-FP and DD-PP to solve the complex linear systems with the Schur complement matrices, as well as a further study of the performance of DD-FP and DD-PP when additional levels of distributed and shared memory parallelism are exploited.

In Chapter 5 we considered the performance of the DD-FP and FEAST eigenvalue solvers when more than one levels of distributed memory parallelism are exploited. For FEAST, we also considered the benefits of using domain decomposition linear system solvers. For DD-FP, adding one more level of distributed memory parallelism can lead to a significant reduction of the wall-clock time/memory footprint. In particular, distributing different right-hand sides to different groups of processors seems to be a good choice when the Schur complement linear systems are solved by preconditioned iterative solvers. Similarly, the scalability of parallel FEAST mainly depends on the technique used to solve the complex linear systems as well as the distribution of the parallel resources to the different possible levels of parallelism.

In Chapter 6 we described `Cucheb`, a GPU implementation of the filtered Lanczos procedure for the solution of large sparse symmetric eigenvalue problems. Cucheb was compared against FILTLAN, a similar CPU-based filtered Lanczos procedure, on a set of eigenvalue problems originating from electronic structure calculations. In particular, our experiments indicate that the use of GPU architectures in the context of electronic structure calculations can provide a speedup of at least a factor of 10 over a single core CPU implementation and at least of factor of 2 for a 24 core implementation. Possible future research directions include the utilization of more than one GPU to perform the filtered Lanczos procedure in computing environments with access to multiple GPUs. Each GPU can then be used to either perform the sparse Matrix-Vector products and other operations of the FLP in parallel, or compute all eigenpairs in a sub-interval of the original interval. In the later case the implementation proposed in this chapter can be used without any modifications. Another interesting extension would be the ability to support additional sparse matrix formats. A dense matrix version of the proposed implementation would also be of interest, e.g. to solve sequences of eigenvalue problems as in [128].

9.2 Root-finding techniques

The second part of this dissertation focused on computing each sought eigenpair of the matrix pencil (A, M) independently by converting the eigenvalue problem into one of a root-finding.

In Chapter 7 we presented a spectral Schur complement technique which recasts the interface eigenvalue problem into one of computing the roots of a scalar function. The aforementioned scalar function is defined by parameterizing the eigenvalues of an approximation of the nonlinear matrix-valued function associated with the interface region. The root-finding problem is then solved by Newton's method. A parallel implementation was presented and its performance was evaluated for model Laplacian problems. The proposed method can be quite fast when only a few extremal eigenpairs of the matrix pencil (A, M) are sought. A key issue still requiring further investigation is to find efficient ways to approximate the eigenpair $(\mu(\sigma), y(\sigma))$ of the Schur complement $S(\sigma)$. In this chapter we used inverse iteration implemented with the MINRES iterative method but did not consider any specific preconditioners. Preconditioning will become mandatory when solving interior eigenvalue problems where the desired eigenvalues are deep inside the spectrum. Such problems can be extremely difficult to solve if sparse direct solvers are ruled out as is the case for very large 3D problems. Eigenvectors of previous spectral Schur complements can again be used to accelerate the iterative solution as the shift σ changes.

In Chapter 8 we proposed the scheme of Mixed Linear Approximations (MLA). Similarly to Chapter 7, MLA recasts the original linear eigenvalue problem into one of computing roots of a scalar function. However, in contrast to the approach presented in Chapter 7, the scalar function of interest in MLA is defined by the eigenvalue of smallest magnitude of a generalized eigenvalue problem stemming by a first-order approximation of the non-linear matrix-valued function associated with the interface variables. To solve the root-finding problem MLA considers Newton's iteration. We discussed several theoretical and implementation details, and demonstrated by experiment that MLA can converge rapidly with a rate which is comparable of even faster than that of the Rayleigh Quotient Iteration, while also being more robust. Several practical details remain to be considered. For example, in our experiments we used Inverse Iteration as the inner eigenvalue solver in MLA, however other approaches, e.g. Lanczos of (generalized) Davidson, are possible. Furthermore, this dissertation only considered the use of direct solvers to solve the linear systems with the Schur complement matrices. The trade-off between performing the Newton update and linear system solutions inexactly and their effect in the convergence of of MLA is the subject of ongoing research. Finally, another interesting research direction would be the adaptation of MLA (where possible) to the solution of symmetric (Hermitian) nonlinear eigenvalue problems.

References

[1] R. K. Clough and J. Penzien. *Dynamics of Structures.* McGraw Hill, New York, 1975.

[2] Jin Hwan Ko and Zhaojun Bai. High-frequency response analysis via algebraic substructuring. *International Journal for Numerical Methods in Engineering*, 76(3):295–313, 2008.

[3] Karl Meerbergen and Zhaojun Bai. The Lanczos method for parameterized symmetric linear systems with multiple right-hand sides. *SIAM Journal on Matrix Analysis and Applications*, 31(4):1642–1662, 2010.

[4] Eberhard KU Gross and Reiner M Dreizler. *Density functional theory*, volume 337. Springer Science & Business Media, 2013.

[5] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Physical Review*, 136:B864–B871, Nov 1964.

[6] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140:A1133–A1138, Nov 1965.

[7] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Society for Industrial and Applied Mathematics, 2011.

[8] Edgar Bright Wilson, John Courtney Decius, and Paul C Cross. *Molecular vibrations: the theory of infrared and Raman vibrational spectra.* Courier Corporation, 1955.

[9] A. N. Nikolakopoulos, V. Kalantzis, E. Gallopoulos, and J. D. Garofalakis. Factored proximity models for top-n recommendations. In *Proceedings of the 2017 IEEE International Conference on Big Knowledge (ICBK)*, pages 80–87, Aug 2017.

[10] Athanasios N. Nikolakopoulos, Vassilis Kalantzis, Efstratios Gallopoulos, and John D. Garofalakis. Eigenrec: generalizing puresvd for effective and efficient top-n recommendations. *Knowledge and Information Systems*, May 2018.

[11] Aritra Bose, Vassilis Kalantzis, Eugenia M. Kontopoulou, Mai Elkadi, Peristera Paschou, and Petros Drineas. TeraPCA: a fast and scalable method to study genetic variation in tera-scale genotypes. *Technical Report*, 2018.

[12] Andrea Toselli and Olof Widlund. *Domain decomposition methods: algorithms and theory.* Springer, 2005.

[13] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, New York, NY, USA, 1996.

[14] Tony F. Chan and Tarek P. Mathew. Domain decomposition algorithms. *Acta Numerica*, 3:61–143, 1994.

[15] Azzam Haidar. *On the parallel scalability of hybrid linear solvers for large 3D problems.* PhD thesis, INPT, 2008.

[16] A. Stathopoulos, Y. Saad, and C. Fischer. A Schur complement method for eigenvalue problems. *7th Copper Mountain Conference on Multigrid Methods, NASA Conference Proceedings, NASA*, 1995.

[17] S. H. Lui. Kron's method for symmetric eigenvalue problems. *Journal of Computational and Appied Mathematics*, 98(1):35–48, 1998.

[18] S. H. Lui. Domain decomposition methods for eigenvalue problems. *Journal of Computational and Appied Mathematics*, 117(1):17–34, 2000.

[19] A. Knyazev and A. Skorokhodov. Preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problems. *SIAM Journal on Numerical Analysis*, 31(4):1226–1239, 1994.

[20] B. Philippe and Y. Saad. On correction equations and domain decomposition for computing invariant subspaces. *Computer Methods in Applied Mechanics and Engineering*,

196(8):1471 – 1483, 2007. Domain Decomposition Methods: recent advances and new challenges in engineering.

[21] B. Cockburn, J. Gopalakrishnan, F. Li, N.-C. Nguyen, and J. Peraire. Hybridization and postprocessing techniques for mixed eigenfunctions. *SIAM Journal on Numerical Analysis*, 48(3):857–881, 2010.

[22] J. Gopalakrishnan, F. Li, N.-C. Nguyen, and J. Peraire. Spectral approximations by the HDG method. *Mathematics of Computation*, 84(293):1037–1059, 2015.

[23] Dennis de Klerk, Daniel J Rixen, and SN Voormeeren. General framework for dynamic substructuring: history, review, and classification of techniques. *AIAA journal*, 46(5):1169, 2008.

[24] Klaus-Jrgen Bathe and Jian Dong. Component mode synthesis with subspace iterations for controlled accuracy of frequency and mode shape solutions. *Computers Structures*, 139:28 – 32, 2014.

[25] J. K. Bennighof and R. B. Lehoucq. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM Journal on Scientific Computing*, 25:2084–2106, 2004.

[26] Mervyn CC Bampton and Roy R Craig Jr. Coupling of substructures for dynamic analyses. *AIAA Journal*, 6(7):1313–1319, 1968.

[27] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79:115112, Mar 2009.

[28] A. Neumaier. Residual inverse iteration for the nonlinear eigenvalue problem. *SIAM Journal on Numerical Analysis*, 22(5):914–923, 1985.

[29] B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Upper Saddle River, NJ, USA, 1998.

[30] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide*, v7.0 edition, October 2015.

[31] S. Balay et al. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2015.

[32] S. Balay et al. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

[33] S. Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[34] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core.* MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998.

[35] L Susan Blackford et al. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[36] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: A portable linear algebra library for high-performance computers. In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*, Supercomputing '90, pages 2–11, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[37] *Intel Math Kernel Library. Reference Manual.* Intel Corporation, 2009. Santa Clara, USA. ISBN 630813-054US.

[38] James W. Demmel. *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[39] J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[40] Bahram Nour-Omid, Beresford N Parlett, Thomas Ericsson, and Paul S Jensen. How to implement the spectral transformation. *Mathematics of Computation*, 48(178):663–673, 1987.

[41] H. D. Simon. The Lanczos algorithm with partial reorthogonalization. *Mathematics of Computation*, 42(165):pp. 115–142, 1984.

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

[43] Ruipeng Li, Yuanzhe Xi, Eugene Vecharynski, Chao Yang, and Yousef Saad. A thick-restart Lanczos algorithm with polynomial filtering for hermitian eigenvalue problems. *SIAM Journal on Scientific Computing*, 38(4):A2512–A2534, 2016.

[44] Ruipeng Li, Yuanzhe Xi, Lucas Erlandson, and Yousef Saad. The EigenValues Slicing Library (EVSL): Algorithms, implementation, and software. *arXiv preprint arXiv:1802.05215*, 2018.

[45] V. Kalantzis, Y. Xi, and Y. Saad. Beyond Automated MultiLevel Substructuring: Domain decomposition with rational filtering. *SIAM Journal on Scientific Computing*, 40(4):C477–C502, 2018.

[46] Anthony P. Austin and Lloyd N. Trefethen. Computing eigenvalues of real symmetric matrices with rational filters in real arithmetic. *SIAM Journal on Scientific Computing*, 37(3):A1365–A1387, 2015.

[47] Stefan Güttel, Eric Polizzi, Ping Tak Peter Tang, and Gautier Viaud. Zolotarev quadrature rules and load balancing for the FEAST eigensolver. *SIAM Journal on Scientific Computing*, 37(4):A2100–A2122, 2015.

[48] Yuanzhe Xi and Yousef Saad. Computing partial spectra with least-squares rational filters. *SIAM Journal on Scientific Computing*, 38(5):A3020–A3045, 2016.

[49] Jan Winkelmann and Edoardo Di Napoli. Non-linear least-squares optimization of rational filters for the solution of interior eigenvalue problems. *arXiv preprint arXiv:1704.03255*, 2017.

[50] Milton Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated, 1974.

[51] Ping Tak Peter Tang and Eric Polizzi. FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. *SIAM Journal on Matrix Analysis and Applications*, 35(2):354–390, 2014.

[52] James Kestyn, Eric Polizzi, and Ping Tak Peter Tang. FEAST eigensolver for non-hermitian problems. *SIAM Journal on Scientific Computing*, 38(5):S772–S799, 2016.

[53] Edoardo Di Napoli, Eric Polizzi, and Yousef Saad. Efficient estimation of eigenvalue counts in an interval. *Numerical Linear Algebra with Applications*, 23(4):674–692, 2016.

[54] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, second edition, 2003.

[55] Jörg Liesen and Zdenek Strakos. *Krylov subspace methods : principles and analysis.* Numerical mathematics and scientific computation. Oxford University Press, Oxford, 2013.

[56] Gene Howard Golub and Richard Underwood. The block lanczos method for computing eigenvalues. In *Mathematical software*, pages 361–377. Elsevier, 1977.

[57] Hasan Metin Aktulga, Lin Lin, Christopher Haine, Esmond G. Ng, and Chao Yang. Parallel eigenvalue calculation based on multiple shift - invert Lanczos and contour integral based spectral projection method. *Parallel Computing*, 40(7):195 – 212, 2014. 7th Workshop on Parallel Matrix Algorithms and Applications.

[58] James Kestyn, Vasileios Kalantzis, Eric Polizzi, and Yousef Saad. PFEAST: A high performance sparse eigenvalue solver using distributed-memory linear solvers. In *In Proceedings of the ACM/IEEE Supercomputing Conference (SC16)*, 2016.

[59] Martin Galgon, Lukas Krämer, Jonas Thies, Achim Basermann, and Bruno Lang. On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues. *Parallel Computing*, 49(C):153–163, November 2015.

[60] Vassilis Kalantzis, James Kestyn, Eric Polizzi, and Yousef Saad. Domain decomposition approaches for accelerating contour integration eigenvalue solvers for symmetric eigenvalue problems. *To appear in Numerical Linear Algebra with Applications*, 2018.

[61] Tetsuya Sakurai and Hiroshi Sugiura. A projection method for generalized eigenvalue problems using numerical integration. *Journal of Computational and Applied Mathematics*, 159(1):119 – 128, 2003. 6th Japan-China Joint Seminar on Numerical Mathematics; In Search for the Frontier of Computational and Applied Mathematics toward the 21st Century.

[62] Tetsuya Sakurai and Hiroto Tadano. CIRR: a Rayleigh-Ritz type method with contour integral for generalized eigenvalue problems. *Hokkaido Math Journal*, 36(4):745–757, 11 2007.

[63] Junko Asakura, Tetsuya Sakurai, Hiroto Tadano, Tsutomu Ikegami, and Kinji Kimura. A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Letters*, 1:52–55, 2009.

[64] L. Kronik et al. PARSEC–the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nano-structures. *Physics Status Solidi (B)*, 243(5):1063–1079, 2006.

[65] C. Bekas, E. Kokiopoulou, and Yousef Saad. Computation of large invariant subspaces using polynomial filtered Lanczos iterations with applications in Density Functional Theory. *SIAM Journal on Matrix Analysis and Applications*, 30(1):397–418, April 2008.

[66] Y. Saad, A. Stathopoulos, J. Chelikowsky, K. Wu, and S. Öğüt. Solution of large eigenvalue problems in electronic structure calculations. *BIT*, 36(3):563–578, 1996.

[67] Haw ren Fang and Yousef Saad. A filtered Lanczos procedure for extreme and interior eigenvalue problems. *SIAM Journal on Scientific Computing*, 34(4):A2220–A2246, 2012.

[68] G. Schofield, J. R. Chelikowsky, and Y. Saad. A spectrum slicing method for the Kohn–Sham problem. *Computer Physics Communications*, 183:497 – 505, 2012.

[69] Alexander L. Skorokhodov Andrew V. Knyazev. Preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problems. *SIAM Journal on Numerical Analysis*, 31(4):1226–1239, 1994.

[70] C. Bekas and Y. Saad. Computation of smallest eigenvalues using spectral Schur complements. *SIAM Journal on Scientific Computing*, 27:458–481, 2006.

[71] V. Kalantzis, R. Li, and Y. Saad. Spectral Schur complement techniques for symmetric eigenvalue problems. *Electronic Transactions on Numerical Analysis*, 45:305–329, 2016.

[72] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[73] F. Pellegrini. SCOTCH *and* LIBSCOTCH *5.1 User's Guide*. INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, 2010.

[74] Weiguo Gao, Xiaoye S. Li, Chao Yang, and Zhaojun Bai. An implementation and evaluation of the AMLS method for sparse eigenvalue problems. *ACM Transactions on Mathematical Software*, 34(4):20:1–20:28, July 2008.

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

[76] L. Komzsik and T. Rose. Parallel methods on large-scale structural analysis and physics applications substructuring in MSC/NASTRAN for large scale parallel applications. *Computing Systems in Engineering*, 2(2):167 – 173, 1991.

[77] Jiacong Yin, Heinrich Voss, and Pu Chen. Improving eigenpairs of automated multilevel substructuring with subspace iterations. *Computers Structures*, 119:115 – 124, 2013.

[78] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, April 1973.

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

[80] Chao Yang, Weiguo Gao, Zhaojun Bai, Xiaoye S. Li, Lie-Quan Lee, Parry Husbands, and Esmond Ng. An algebraic substructuring method for large-scale eigenvalue calculation. *SIAM Journal on Scientific Computing*, 27(3):873–892, 2005.

[81] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[82] Wolf-Jürgen Beyn. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra and its Applications*, 436(10):3839 – 3863, 2012.

[83] Xiaoye S. Li and James W. Demmel. Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, June 2003.

[84] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[85] L. M. Carvalho, L. Giraud, and P. Le Tallec. Algebraic two-level preconditioners for the Schur complement method. *SIAM Journal on Scientific Computing*, 22(6):1987–2005, 2001.

[86] Luc Giraud, Azzam Haidar, and Yousef Saad. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D. *Numerical Mathematics*, 3(3):276–294, 2010.

[87] S. Rajamanickam, E.G. Boman, and M.A. Heroux. ShyLU: A hybrid-hybrid solver for multicore platforms. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 631–643, May 2012.

[88] Y. Saad and B. Suchomel. ARMS: an Algebraic Recursive Multilevel Solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9(5):359–378, 2002.

[89] Zhongze Li, Yousef Saad, and Masha Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10(5-6):485–509, 2003.

[90] Yousef Saad and Maria Sosonkina. Distributed Schur complement techniques for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(4):1337–1356, 1999.

[91] Andrey Kuzmin, Mathieu Luisier, and Olaf Schenk. Fast methods for computing selected elements of the Green's function in massively parallel nanoelectronic device simulations. In *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, pages 533–544, Berlin, Heidelberg, 2013. Springer-Verlag.

[92] Cosmin G. Petra, Olaf Schenk, Miles Lubin, and Klaus Gärtner. An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization. *SIAM Journal on Scientific Computing*, 36(2):C139–C162, 2014.

[93] Vassilis Kalantzis, Costantinos Bekas, Alessandro Curioni, and Efstratios Gallopoulos. Accelerating data uncertainty quantification by solving linear systems with multiple right-hand sides. *Numerical Algorithms*, 62(4):637–653, 2013.

[94] Vassilis Kalantzis, A. Cristiano I. Malossi, Costas Bekas, Alessandro Curioni, Efstratios Gallopoulos, and Yousef Saad. A scalable iterative dense linear system solver for multiple right-hand sides in data analytics. *Parallel Computing*, 74:136 – 153, 2018.

[95] V. Simoncini and E. Gallopoulos. Convergence properties of block GMRES and matrix polynomials. *Linear Algebra and its Applications*, 247:97 – 119, 1996.

[96] Timothy A. Davis. Algorithm 832: UMFPACK V4.3—an Unsymmetric-pattern Multi-frontal Method. *ACM Transactions on Mathematical Software*, 30(2):196–199, June 2004.

[97] J. L. Aurentz, V. Kalantzis, and Y. Saad. Cucheb. `https://github.com/jaurentz/cucheb`, 2016.

[98] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.

[99] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: a hands-on approach.* Morgan Kaufmann, 2016.

[100] J. R. Chelikowsky, Y. Saad, and I. Vasiliev. Atoms and clusters. In *Time-Dependent Density Functional Theory*, volume 706 of *Lecture notes in Physics*, chapter 17, pages 259–269. Springer-Verlag, Berlin, Heidelberg, 2006.

[101] W. R. Burdick, Y. Saad, L. Kronik, M. Jain, and J. R. Chelikowsky. Parallel implementations of time-dependent density functional theory. *Computer Physics Communications*, 156:22–42, 2003.

[102] J. L. Aurentz. *GPU accelerated polynomial spectral transformation methods.* PhD thesis, Washington State University, 2014.

[103] L. O. Jay, H. Kim, Y. Saad, and J. R. Chelikowsky. Electronic structure calculations for plane-wave codes without diagonalization. *Computer Physics Communications*, 118(1):21–30, 1999.

[104] Y. Saad. Filtered conjugate residual-type algorithms with applications. *SIAM Journal on Matrix Analysis and Applications*, 28(3):845–870, 2006.

[105] Y. Zhou. A block Chebyshev-Davidson method with inner-outer restart for large eigenvalue problems. *Journal of Computational Physics*, 229(24):9188 – 9200, 2010.

[106] Y. Zhou and Y. Saad. A Chebyshev-Davidson algorithm for large symmetric eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 29(3):954–971, 2007.

[107] R. N. Silver, H. Roeder, A. F. Voter, and J. D. Kress. Kernel polynomial approximations for densities of states and spectral functions. *Journal of Computational Physics*, 124(1):115–130, 1996.

[108] A. Weiße, G. Wellein, A. Alvermann, and H. Fehske. The kernel polynomial method. *Rev. Modern Phys.*, 78(1):275, 2006.

[109] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Self-consistent-field calculations using Chebyshev-filtered subspace iteration. *Journal of Computational Physics*, 219(1):172 – 184, 2006.

[110] D. Jackson. *The Theory of Approximation*, volume 11 of *Colloquium publications*. AMS, New York, NY, USA, 1930.

[111] C. W. Clenshaw. A note on the summation of Chebyshev series. *Mathematics of Computation*, 9:118–120, 1955.

[112] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45:255–282, 1950.

[113] J. Cullum and W. E. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices. In *Decision and Control including the 13th Symposium on Adaptive Processes, 1974 IEEE Conference on*, pages 505–509. IEEE, 1974.

[114] C. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995.

[115] T. Kato. *Perturbation Theory for Linear Operators.* Springer–Verlag, New–York, 1976.

[116] Peter D. Lax. *Linear algebra and its applications.* Pure and applied mathematics. Wiley-Interscience, Hoboken (N.J.), 2007.

[117] Ilse C. F. Ipsen. Computing an eigenvector with inverse iteration. *SIAM Review*, 39(2):254–291, 1997.

[118] G. H. Golub and C. F. Van Loan. *Matrix Computations, 4th edition.* Johns Hopkins University Press, Baltimore, MD, 4th edition, 2013.

[119] Daniel B. Szyld. Criteria for combining inverse and Rayleigh quotient iteration. *SIAM Journal on Numerical Analysis*, 25(6):1369–1375, 1988.

[120] Andr Gaul, Martin H. Gutknecht, Jörg Liesen, and Reinhard Nabben. A framework for deflated and augmented Krylov subspace methods. *SIAM Journal on Matrix Analysis and Applications*, 34(2):495–518, 2013.

[121] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickan. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, 35:22:1–22:14, 2008.

[122] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[123] R. B. Lehoucq, D. C. Sorensen, and C. Yang. ARPACK users guide: Solution of large-scale eigenvalue problems by implicitely restarted Arnoldi methods. *SIAM, Philadelphia, PA, 1998.*

[124] Alan L. Andrew and Roger C. E. Tan. Computation of derivatives of repeated eigenvalues and the corresponding eigenvectors of symmetric matrix pencils. *SIAM Journal on Matrix Analysis and Applications*, 20(1):78–100, 1998.

[125] Haim Avron and Esmond Ng. A generalized Courant-Fischer minimax theorem. 2008.

[126] Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, 1996.

[127] G. Dillon, V. Kalantzis, Y. Xi, and Y. Saad. A hierarchical low rank Schur complement preconditioner for indefinite linear systems. *SIAM Journal on Scientific Computing*, 40(4):A2234–A2252, 2018.

[128] M. Berljafa, D. Wortmann, and E. Di Napoli. An optimized and scalable eigensolver for sequences of eigenvalue problems. *Concurrency and Computation: Practice and Experience*, 27(4):905–922, 2015.