Electronic Transactions on Numerical Analysis.
Volume 45, pp. 305–329, 2016.
Copyright © 2016, Kent State University.
ISSN 1068–9613.

SPECTRAL SCHUR COMPLEMENT TECHNIQUES FOR SYMMETRIC EIGENVALUE PROBLEMS[*]

VASSILIS KALANTZIS[†], RUIPENG LI[†], AND YOUSEF SAAD[†]

Abstract. This paper presents a domain decomposition-type method for solving real symmetric (Hermitian) eigenvalue problems in which we seek all eigenpairs in an interval $[\alpha, \beta]$ or a few eigenpairs next to a given real shift ζ. A Newton-based scheme is described whereby the problem is converted to one that deals with the interface nodes of the computational domain. This approach relies on the fact that the inner solves related to each local subdomain are relatively inexpensive. This Newton scheme exploits spectral Schur complements, and these lead to so-called eigenbranches, which are rational functions whose roots are eigenvalues of the original matrix. Theoretical and practical aspects of domain decomposition techniques for computing eigenvalues and eigenvectors are discussed. A parallel implementation is presented and its performance on distributed computing environments is illustrated by means of a few numerical examples.

Key words. Domain decomposition, spectral Schur complements, eigenvalue problems, Newton's method, parallel computing

AMS subject classifications. 65F15, 15A18, 65F50

1. Introduction.

We are interested in the partial solution of the symmetric eigenvalue problem

$$Ax = \lambda x, \tag{1.1}$$

where A is an $n \times n$ symmetric (Hermitian) matrix, and we assume that it is large and sparse. We assume that the eigenvalues $\lambda_1, \ldots, \lambda_n$, of A are labeled increasingly. By "partial solution" we mean one of the two following scenarios:

- Find all eigenpairs (λ, x) of A where λ belongs to the sub-interval $[\alpha, \beta]$ of the spectrum, $[\alpha, \beta] \subseteq [\lambda_1, \lambda_n]$.
- Given a shift $\zeta \in \mathbb{R}$ and an integer k, find the eigenpairs (λ, x) of A for which λ is one of the k closest eigenvalues to ζ. A similar problem is the computation of the k eigenpairs of A located immediately to the right (or to the left) of the given shift ζ.

The interval $[\alpha, \beta]$ can be located anywhere inside the region $[\lambda_1, \lambda_n]$. When $\alpha := \lambda_1$ or $\beta := \lambda_n$, we will refer to the eigenvalue problem as *extremal*, otherwise we will refer to it as *interior*.

It is typically easier to solve extremal eigenvalue problems than interior ones. Methods such as the Lanczos algorithm [22], and its more sophisticated practical variants such as the Implicitly Restarted Lanczos (IRL) [24], the closely related thick-restart Lanczos [39, 43], the method of trace minimization [35], or the method of Jacobi-Davidson [37], are powerful methods for solving eigenvalue problems associated with extremal eigenvalues. However, these methods become expensive for interior eigenvalue problems typically requiring a large number of matrix-vector products.

A standard approach for solving interior eigenvalue problems is the shift-and-invert technique where A is replaced by $(A - \sigma I)^{-1}$. By this transformation, eigenvalues of A closest to σ become extremal ones for $(A - \sigma I)^{-1}$ and a projection method of choice, be

[*] Received July 22, 2015. Accepted June 27, 2016. Published online on September 2, 2016. Recommended by D. Szyld. The work of V. Kalantzis and Y. Saad was supported by 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 DE-SC0008877. The work of R. Li was supported by the National Science Foundation under grant NSF/DMS-1216366.

[†] Computer Science & Engineering, University of Minnesota, Twin Cities, Minneapolis, US ({kalantzi,rli,saad}@cs.umn.edu).

it subspace iteration or a Krylov-based approach, will converge (much) faster. However, a factorization is now necessary and this seriously limits the size and type of problems that can be efficiently solved by shift-and-invert techniques. For example, matrix problems that stem from discretizations of partial differential equations on 3D computational domains are known to generate a large amount of fill-in. An alternative for avoiding the factorization of $(A - \sigma I)$ is to exploit polynomial filtering, which essentially consists of replacing $(A - \sigma I)^{-1}$ by a polynomial in A, $p(A)$. The goal of the polynomial is to dampen eigenvalues outside the interval of interest. Such polynomial filtering methods can be especially useful for large interior eigenvalue problems where many eigenvalues are needed; see [2, 11]. Their disadvantage is that they are sensitive to uneven distributions of the eigenvalues, and the degree of the polynomial might have to be selected very high in some cases. Recently, contour integration methods, such as the FEAST method [32] or the method of Sakurai-Sugiura [34], have gained popularity. The more robust implementations of these utilize direct solvers to deal with the complex linear systems that come from numerical quadrature, and this again can become expensive for 3D problems. When iterative methods are used instead, then the number of matrix-vector products can be as high as in the case of polynomial filtering.

This paper takes a different perspective from all the approaches listed above by considering a Domain Decomposition (DD) type approach instead. In this framework, the computational domain is partitioned into a number of (non-overlapping) subdomains, which can then be treated independently, along with an interface region that accounts for the coupling among the subdomains. The problem on the interface region is *non-local* in the sense that communication among the subdomains is required. The advantage of domain decomposition-type methods is that they naturally lend themselves to parallelization. Thus, a DD approach starts by determining the part of the solution that lies on the interface nodes. The original eigenvalue problem is then recast into an eigenvalue problem that is to be solved only on the interface nodes, by exploiting spectral Schur complements. This converts the original large eigenvalue problem into a smaller but nonlinear eigenvalue problem. This problem is then solved by a Newton iteration.

The idea of using domain decomposition for eigenvalue problems is not new. Though not formulated in the framework of DD, the paper by Abramov and Chishov (see the discussion in [38]) is the earliest we know that introduced the concept of spectral Schur complements. Other earlier publications describing approaches that share some common features with our work can be found in [9, 15, 20, 26, 27, 31]. The articles [26, 27] establish some theory when the method is viewed from a partial differential equations viewpoint. The paper [20] 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 [9, 15] discuss condensation techniques applied to the Raviart-Thomas approximation and the 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.

A well-known example of a domain decomposition approach for approximating the lowest eigenvalues of a symmetric real matrix is represented by the Automated MultiLevel Substructuring method (AMLS) [5]. The main similarity between our approach and AMLS is that they both exploit Schur complements. Apart from this, the two approaches are quite different. AMLS uses only one shift (e.g., at the origin) and then constructs a good subspace with which to perform a Rayleigh-Ritz projection to compute many eigenpairs with this one shift. The result is often a fast computation but moderate accuracy. Because the Schur complement is constructed explicitly, the method tends to perform quite well for problems that have a 2D geometry. Our approach is at the other extreme: the shift changes at each Newton step, and our method essentially computes one eigenpair at a time. In our case, the spectral Schur complement is never formed explicitly. In addition, the resulting eigenpair is computed with as high accuracy as required.

The primary goal of this paper is to further extend the current understanding of domain decomposition methods for eigenvalue problems as well as to develop practical related algorithms. As pointed out earlier, domain decomposition goes hand-in-hand with a parallel computing viewpoint and we implemented the proposed scheme in distributed computing environments by making use of the PETSc framework [3].

The paper is organized as follows: Section 2 discusses background concepts on domain decomposition and distributed eigenvalue problems. Section 3 proposes a Newton scheme for computing a single eigenpair of A and presents some analysis. Section 4 discusses a generalization to the case of computing multiple eigenpairs of A. Section 5 discusses our parallel implementation within the domain decomposition framework and Section 6 presents numerical experiments on both serial and distributed environments. Finally, a conclusion is given in Section 7.

2. Background: distributed eigenvalue problems. In a domain decomposition framework, we typically begin by subdividing the problem into p parts with the help of a graph partitioner [6, 7, 17, 21, 30, 33]. Generally, this consists of assigning sets of variables to subdomains. Figure 2.1 shows a common way of partitioning a graph, where vertices are assigned to subdomains or partitions. A vertex is a pair equation-unknown (equation number k and unknown number k), and the partitioner subdivides the vertex set into p partitions, i.e., p non-intersecting subsets whose union is equal to the original vertex set. In this situation some edges are cut between domains, and so this way of partitioning a graph is also known as edge-separator partitioning.

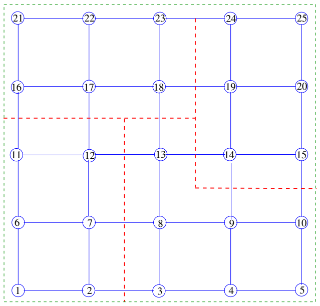

FIG. 2.1. *A classical way of partitioning a graph.*

In this paper we partition the problem using an edge-separator as it is done in the pARMS [25] linear system solver for example. Once the matrix is partitioned, 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. Local points in each subdomain are reordered so that the interface points are listed after the interior points. Thus, each local piece x_i of the eigenvector is split into two parts: the subvector u_i of internal vector components followed by the subvector y_i of local interface vector components; we assume $i = 1, \dots, p$. Let subdomain i have d_i interior variables and s_i interface variables, i.e., the length of vectors u_i and y_i is d_i and s_i, respectively. We denote by $B_i \in \mathbb{R}^{d_i \times d_i}$ the matrix that represents the couplings between the interior variables, i.e., between variables in u_i. Similarly, we denote by $\hat{E}_i \in \mathbb{R}^{d_i \times s_i}$ the matrix that maps interface variables of subdomain i to interior variables of subdomain i and by $C_i \in \mathbb{R}^{s_i \times s_i}$ the matrix that represents the couplings between the interface variables of subdomain i, i.e., between variables in y_i. Finally, we let $E_{ij} \in \mathbb{R}^{s_i \times s_j}$ be the matrix representing the couplings between external interface unknowns from subdomain j and local interface variables of subdomain i. Note that E_{ij} is non-local in that it acts on variables that are external to domain i and produces local (interface) variables.

Then, the equation $(A - \lambda I)x = 0$ can be written locally as

$$\underbrace{\begin{bmatrix} B_i - \lambda I & \hat{E}_i \\ \hat{E}_i^T & C_i - \lambda I \end{bmatrix}}_{A_i} \underbrace{\begin{bmatrix} u_i \\ y_i \end{bmatrix}}_{x_i} + \begin{bmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{bmatrix} = 0, \quad i = 1, \ldots, p. \tag{2.1}$$

Here, N_i is the set of indices for subdomains that are neighbors to the subdomain i. The term $E_{ij}y_j$ is a part of the product that reflects the contribution to the local equation from the neighboring subdomain j. The result of this multiplication affects only local interface equations, which is indicated by a zero in the top part of the second term of the left-hand side of (2.1).

2.1. The interface and spectral Schur complement matrices.

The local equations in (2.1) are naturally split in two terms: the first term (represented by the term $A_i x_i$) involves only local variables and the second contains couplings between these local variables and the external interface variables. The second row of (2.1) is

$$\hat{E}_i^T u_i + (C_i - \lambda I) y_i + \sum_{j \in N_i} E_{ij} y_j = 0.$$

This couples all the interface variables with the local (interior) variable u_i. The action of the operation on the left-hand side of the above equation on the vector of all interface variables, i.e., the vector $y^T = [y_1^T, y_2^T, \ldots, y_p^T]$, can be gathered into the following matrix $C \in \mathbb{R}^{s \times s}$,

$$C = \begin{bmatrix} C_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & C_2 & \ldots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{p1} & E_{p2} & \ldots & C_p \end{bmatrix},$$

where $E_{ij} = 0$ if $j \notin N_i$, and $s = s_1 + s_2 + \cdots + s_p$.

Thus, if we stack all interior variables $u_1, u_2, \ldots, u_p$ into a vector u in this order and reorder the equations so that the u_i's are listed first followed by the y_i's, we obtain a reordered global eigenvalue problem that has the following form:

$$\underbrace{\begin{bmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_p & E_p \\ E_1^T & E_2^T & \ldots & E_p^T & C \end{bmatrix}}_{PAP^T} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{bmatrix}, \tag{2.2}$$

where $P \in \mathbb{R}^{n \times n}$ is the corresponding permutation matrix. The matrix E_i, $i = 1, \ldots, p$, $E_i \in \mathbb{R}^{d_i \times s}$, in (2.2) is an expanded version of the corresponding matrix $\hat{E}_i$ defined earlier and used in (2.1). More specifically, we have $E_i = [0_{d_i,\ell_i}, \hat{E}_i, 0_{d_i,\nu_i}]$, where $\ell_i = \sum_{j=1}^{j<i} s_j$,

$\nu_i = \sum_{j>i}^{j=p} s_j$, where $0_{\chi,\psi}$ denotes the zero matrix of size $\chi \times \psi$. Note in particular that we have $E_i y = \hat{E}_i y_i$. The coefficient matrix of the system (2.2) is of the form

$$A = \begin{bmatrix} B & E \\ E^T & C \end{bmatrix}, \quad E = [E_1^T \; E_2^T \; \dots \; E_p^T]^T, \tag{2.3}$$

where $B \in \mathbb{R}^{d \times d}$, $E \in \mathbb{R}^{d \times s}$, $d = d_1 + d_2 + \cdots + d_p$, and we have kept the same symbol A to represent the unpermuted matrix in (1.1). An illustration of (2.3) for $p = 4$ subdomains is shown in Figure 2.2.

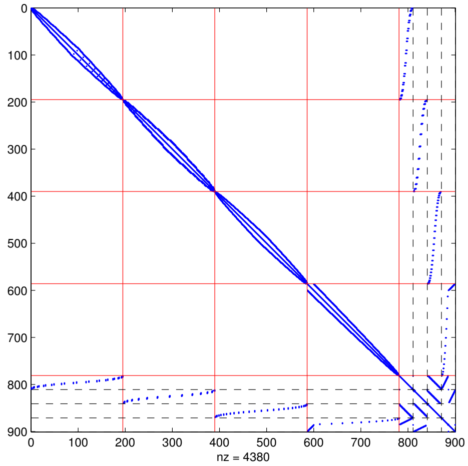

FIG. 2.2. *Laplacian matrix partitioned in* $p = 4$ *subdomains and reordered according to* (2.2).

2.2. Spectral Schur complements. Schur complement techniques eliminate interior variables to yield equations associated with the interface variables only. Specifically, we can eliminate the variable u_i from (2.1), which gives $u_i = -(B_i - \lambda I)^{-1} \hat{E}_i y_i$, and upon substitution in the second equation, we get

$$S_i(\lambda) y_i + \sum_{j \in N_i} E_{ij} y_j = 0, \tag{2.4}$$

where $S_i(\lambda) \in \mathbb{R}^{s_i \times s_i}$ is the "local" spectral Schur complement

$$S_i(\lambda) = C_i - \lambda I - \hat{E}_i^T (B_i - \lambda I)^{-1} \hat{E}_i.$$

The equations (2.4) for all subdomains, $i = 1, \dots, p$, constitute a nonlinear eigenvalue problem; one that involves only the interface unknown vectors y_i. This problem has a block structure:

$$\underbrace{\begin{bmatrix} S_1(\lambda) & E_{12} & \dots & E_{1p} \\ E_{21} & S_2(\lambda) & \dots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{p1} & E_{p2} & \dots & S_p(\lambda) \end{bmatrix}}_{S(\lambda)} \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}}_{y} = 0. \tag{2.5}$$

The diagonal blocks in this system, the local Schur complement matrices $S_i(\lambda)$, are dense in general. The off-diagonal blocks E_{ij}, which are identical with those of the local system (2.1), are sparse. Note that the above spectral Schur complement is nothing but the spectral Schur complement associated with the decomposition (2.3), i.e., we have:

$$S(\lambda) = C - \lambda I - E^T (B - \lambda I)^{-1} E, \tag{2.6}$$

where $S(\lambda) \in \mathbb{R}^{s \times s}$. In other words, the expressions for $S(\lambda)$ in (2.5) and (2.6) are identical: the first being more useful for practical purposes and the second more useful for theoretical derivations. Spectral Schur complements were discussed also in [4, 5].

If we can solve the global Schur complement problem (2.5), then the solution to the global eigenproblem (1.1) can be trivially obtained by substituting the y_i's into the first part of (2.1), i.e., by setting for each subdomain:

$$u_i = -(B_i - \lambda I)^{-1} \hat{E}_i y_i, \quad i = 1, \ldots, p. \tag{2.7}$$

3. Solving the spectral Schur complement problem. Our next focus is on the solution of the spectral Schur complement problem (2.5). The problem we have is to find a pair $(\lambda, y(\lambda))$ satisfying:

$$S(\lambda) y(\lambda) = 0.$$

Then, λ is an eigenvalue of A with $y(\lambda)$ being the bottom part of the associated eigenvector x. For an arbitrary value $\sigma \in \mathbb{R}$ (which we will call a shift), we consider the eigenvalue problem

$$S(\sigma) y(\sigma) = \mu(\sigma) y(\sigma), \tag{3.1}$$

where $\mu(\sigma)$ and $y(\sigma) \in \mathbb{R}^s$ denote the eigenvalue of smallest magnitude of $S(\sigma)$ and the corresponding eigenvector, respectively. The matrix $S(\sigma)$ has s eigenvalues, which will be denoted as $\mu_i(\sigma)$, $i = 1, \ldots, s$, in a sorted (algebraic) ascending order. Each $\mu_i(\sigma)$ is a function of σ, which we refer to as the ith *eigenbranch* of $S(\sigma)$. We will denote the corresponding eigenvectors of $S(\sigma)$ as $y^{(i)}(\sigma) \in \mathbb{R}^s$, $i = 1, \ldots, s$.

The question now becomes how to find a σ for which $S(\sigma)$ is singular. One idea, adopted in [26], is to consider the equation $\det(S(\sigma)) = 0$, but this is not practical for large problems. On the other hand, $S(\sigma)$ is singular exactly when at least one of $\mu_i(\sigma)$, $i = 1, \ldots, s$ is zero. With this, the original eigenvalue problem in (1.1) can be reformulated as that of finding a shift σ such that the eigenvalue of smallest magnitude, $\mu(\sigma)$, of $S(\sigma)$ is zero. Thus, $\mu(\sigma)$ is treated as a non-linear function, and we are interested in obtaining a few of its roots, e.g., those located inside a given interval $[\alpha, \beta]$. To find these roots, we would like to exploit a Newton scheme and for this the derivative of each eigenbranch $\mu_i(\sigma)$, $i = 1, \ldots, s$, is needed.

PROPOSITION 3.1. *The eigenbranches $\mu_i(\sigma)$, $i = 1, \ldots, s$, are analytic at any point $\sigma \notin \Lambda(B)$, where $\Lambda(B)$ denotes the spectrum of B. The derivative of each eigenbranch at these points is given by*

$$\frac{d\mu_i(\sigma)}{d\sigma} = \frac{(S'(\sigma) y^{(i)}(\sigma), y^{(i)}(\sigma))}{(y^{(i)}(\sigma), y^{(i)}(\sigma))} = -1 - \frac{\|(B - \sigma I)^{-1} E y^{(i)}(\sigma)\|_2^2}{\|y^{(i)}(\sigma)\|_2^2}, \tag{3.2}$$

where $S(\sigma) = C - \sigma I - E^T (B - \sigma I)^{-1} E$.

Proof. For each eigenbranch $\mu_i(\sigma)$, $i = 1, \ldots, s$, differentiating (3.1) we get (the dependence on the variable (σ) is omitted throughout the proof):

$$S' y^{(i)} + S y^{(i)'} = \mu_i' y^{(i)} + \mu_i y^{(i)'}.$$

Taking inner products with $y^{(i)}$ yields:

$$(S' y^{(i)}, y^{(i)}) + (S y^{(i)'}, y^{(i)}) = \mu_i' (y^{(i)}, y^{(i)}) + \mu_i (y^{(i)'}, y^{(i)}).$$

Observe that $(Sy^{(i)'}, y^{(i)}) = (y^{(i)'}, Sy^{(i)}) = (y^{(i)'}, \mu_i y^{(i)}) = \mu_i (y^{(i)'}, y^{(i)})$. Then the above equality becomes $(S' y^{(i)}, y^{(i)}) = \mu_i' (y^{(i)}, y^{(i)})$, which gives the first part of (3.2). The second part is obtained by differentiating $S(\sigma)$ with respect to σ:

$$S'(\sigma) = -I - E^T (B - \sigma I)^{-2} E. \tag{3.3}$$

To finalize the proof, we have to show that $S'(\sigma)$, $\mu_i'(\sigma)$ and $y^{(i)'}(\sigma)$ exist. Since $\sigma \notin \Lambda(B)$, $S(\sigma)$ is analytic, and its derivative is well defined. Eigenvalues $\mu_i(\sigma)$ of $S(\sigma)$ define branches. In the neighborhood of a single eigenvalue, $\mu_i(\sigma)$ is analytic as is its associated eigenprojector [18]. From this it also follows that an associated analytic eigenvector 'branch' can be defined; individual eigenvectors are only defined up to scaling—but the eigenprojector $P_i(\sigma)$ is analytic, and so $P_i(\sigma)w$, which is an eigenvector for an arbitrary vector w, is analytic. Because the results established in [18] are valid for semi-simple eigenvalues, and multiple eigenvalues are semi-simple in the Hermitian case, the same statement can be made for multiple eigenvalues. For multiple eigenvalues, additional details can be found in [23]. □

Equation (3.3) shows that the matrix $S'(\sigma)$ is negative definite with all its eigenvalues being finite when $\sigma \notin \Lambda(B)$, i.e., σ is not an eigenvalue of B; the eigenvalues of B will also be referred to as poles [18]. As a result, the derivatives of the eigenbranches $\mu_i(\sigma)$, $i = 1, \ldots, s$, in (3.2) are always negative and the following corollary is immediate.

COROLLARY 3.2. *For any σ located in an interval containing no poles, all eigenbranches $\mu_i(\sigma)$, $i = 1, \ldots, s$, are strictly decreasing.*

In the following, we will drop the index i and will concentrate on $\mu(\sigma)$, which denotes the eigenvalue of smallest magnitude of $S(\sigma)$. Note that $\mu(\sigma)$ is always one of the s eigenvalues of $S(\sigma)$ and satisfies $\mu(\sigma) = \min |\mu_i(\sigma)|$, $i = 1, \ldots, s$. Moreover, we will assume that the eigenvalues of A and B do not overlap.

3.1. An algorithm for computing a single eigenpair.

Assume that we are interested in computing a single eigenpair (λ, x) of A, say the one closest to $\zeta \in \mathbb{R}$. Then, a straightforward algorithm based on Newton's iteration is as follows.

ALGORITHM 3.1. *Newton spectral Schur complement*

```
1. Select $\sigma := \zeta$.
2. Until convergence Do:
3.     Compute $\mu(\sigma)$ = smallest eigenvalue in modulus of $S(\sigma)$
       along with the associated unit eigenvector $y(\sigma)$.
4.     Set $\eta := \|(B - \sigma I)^{-1} E y(\sigma)\|_2$.
5.     Set $\sigma := \sigma + \mu(\sigma)/(1 + \eta^2)$.
6. End
```

Algorithm 3.1 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 [16]. Alternatively, we can use the Lanczos method with partial re-orthogonalization [36] and perform as many steps as needed to compute $(\mu(\sigma), y(\sigma))$. Note that in the latter case, the Lanczos method will operate on vectors of shorter length, equal to s, the number of interface nodes. In this paper we only consider approximating $(\mu(\sigma), y(\sigma))$ by the inverse iteration approach, in which an iterative scheme is used for solving the linear systems of the form $S(\sigma)w = b$. If we were to solve these systems by using an exact factorization of the Schur complement, the whole scheme would be quite similar to a form of Rayleigh Quotient Iteration (RQI) [12] with a DD framework for solving the related linear systems exactly. This is not practical for large 3D problems because Schur complements can be quite large in these cases. More details about using iterative linear solvers in our problem setting will be discussed in Section 6.

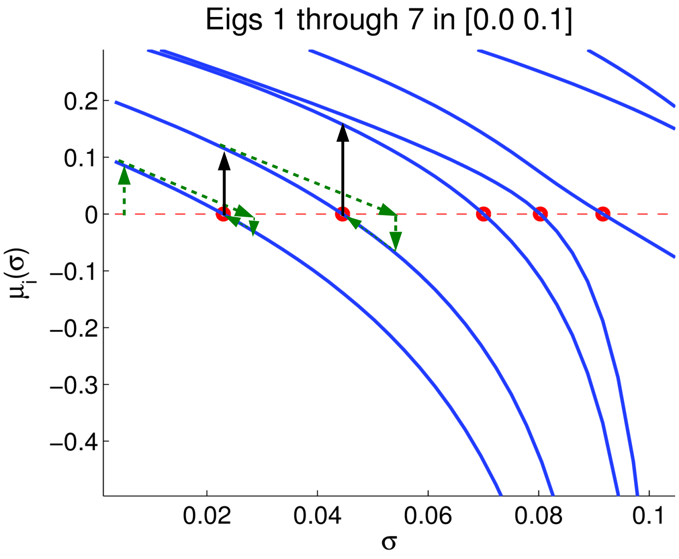

FIG. 3.1. *Eigenbranches $\mu_i(\sigma)$ for $i = 1, \dots, 7$ obtained from a domain decomposition applied to a Laplacian matrix partitioned in 4 subdomains (solid lines). The dashed arrows depict the Newton steps performed by Algorithm 3.1 when computing one of the two algebraically smallest eigenvalues. The black solid arrows indicate a hop to a different eigenbranch when Algorithm 4.1 is used.*

Figure 3.1 visualizes the first few eigenbranches of a 2D Laplacian partitioned in $p = 4$ subdomains in the interval $[0.0, 0.10]$ (solid blue lines). The red circles denote the eigenvalues of A, which are also the roots of the eigenbranches. Figure 3.1 also shows an example of Algorithm 3.1 when it is applied for the computation of one of the two algebraically smallest eigenvalues. The dashed arrows depict the Newton steps as σ is updated. The black solid arrows indicate a hop to a different eigenbranch; see Algorithm 4.1.

3.1.1. An equivalent update scheme for Newton's method. Since Algorithm 3.1 is Newton's method, we expect that if the initial shift σ is "close enough" to an eigenvalue λ, Algorithm 3.1 will converge quadratically to λ [19]. By Proposition 3.1, 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 μ is finite for any $\sigma \notin \Lambda(B)$. Therefore, if the scheme converges, then it will do so quadratically.[1]

It is interesting to link quantities of the algorithm that are related to the Schur complement with those of the original matrix A. We can think of $y(\sigma)$ as the interface part of a global approximate eigenvector. This global approximate eigenvector, which we write in the form $\hat{x}(\sigma) = [u(\sigma)^T, y(\sigma)^T]^T$, can be obtained by substituting $y(\sigma)$ in (2.7) and replacing λ by its approximation σ:

$$\hat{x}(\sigma) = \begin{bmatrix} -(B - \sigma I)^{-1} E y(\sigma) \\ y(\sigma) \end{bmatrix}. \tag{3.4}$$

[1] To be more accurate, the scheme will converge with the same rate as Newton's method does.

The approximate eigenpair $(\sigma, \hat{x}(\sigma))$ has a special residual. Indeed,

$$(3.5)\qquad (A-\sigma I)\hat{x}(\sigma) = \begin{bmatrix} B-\sigma I & E \\ E^T & C-\sigma I \end{bmatrix} \begin{bmatrix} -(B-\sigma I)^{-1}Ey(\sigma) \\ y(\sigma) \end{bmatrix} = \begin{bmatrix} 0 \\ \mu(\sigma)y(\sigma) \end{bmatrix}.$$

In addition, its Rayleigh quotient is equal to the next σ obtained by one Newton step as is stated next.

PROPOSITION 3.3. *Let $\sigma_{j+1} = \sigma_j + \mu(\sigma_j)/(1+\eta_j^2)$ be the Newton update at the jth step of Algorithm 3.1, where we set $\eta_j = \|(B-\sigma_j I)^{-1}Ey(\sigma_j)\|_2$. Then, $\sigma_{j+1} = \rho(A, \hat{x}(\sigma_j))$, where $\rho(A, \hat{x}(\sigma_j))$ is the Rayleigh quotient of A associated with the vector $\hat{x}(\sigma_j)$ defined by* (3.4).

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

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

The expressions (3.4) and (3.5) show that $\hat{x}^T(A-\sigma_j I)\hat{x} = \mu(\sigma_j)$ while $\hat{x}^T\hat{x} = 1+\eta_j^2$. Thus, $\rho(A-\sigma_j I, \hat{x}) = \mu(\sigma_j)/(1+\eta_j^2)$ and so $\rho(A, \hat{x}) = \sigma_j + \mu(\sigma_j)/(1+\eta_j^2) = \sigma_{j+1}$. □

Note that the equivalent update formula discussed in Proposition 3.3 is primarily of theoretical interest. In practice, we use the update formula provided in Step 5 of Algorithm 3.1, which avoids the extra matrix-vector multiplication with the matrix A.

If $\mu(\sigma) = 0$, then σ is an eigenvalue of A. We expect that the closer $\mu(\sigma)$ is to zero, the closer σ should be to an eigenvalue of A. In fact, the relation (3.5) immediately shows that

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

where $\eta = \|(B-\sigma I)^{-1}Ey(\sigma)\|$.

4. Computing eigenpairs in an interval.

The framework developed in Algorithm 3.1 computes a single eigenpair of A. It is possible to extend Algorithm 3.1 for computing more than one eigenpairs, e.g., when searching for all eigenpairs inside a given interval $[\alpha, \beta]$, or a few consecutive eigenpairs next to a given shift $\zeta \in \mathbb{R}$. The main question is how to select a good starting point for the next unconverged eigenpair of A as soon as the current eigenpair has converged. For this, we need to take a closer look at the eigenbranches of the spectral Schur complement.

4.1. Eigenbranches across the poles.

The left subfigure of Figure 4.1 shows a few relevant eigenvalues of $S(\sigma)$ for a sample 2D Laplacian when $\sigma \in [2.358, 2.4]$. The separating dashed spikes are eigenvalues of B, i.e., poles of $S(\sigma)$. An interesting property is revealed when observing the eigenvalues across the borderlines (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. This behavior is illustrated in the right subfigure of Figure 4.1, where the eigenbranches $\mu_1(\sigma)$ and $\mu_2(\sigma)$ are plotted. As σ approaches the pole of $\mu_1(\sigma)$ from the left side, $\mu_1(\sigma)$ descends to $-\infty$ while $\mu_2(\sigma)$ (as well as the rest of the eigenbranches not shown) crosses the pole in a continuous fashion.

To explain this observation, let $\theta_1, \ldots, \theta_d$ be the eigenvalues of B with associated eigenvectors $v_1, \ldots, v_d$, and consider $S(\sigma)$ defined by (2.6), which we write in the following form

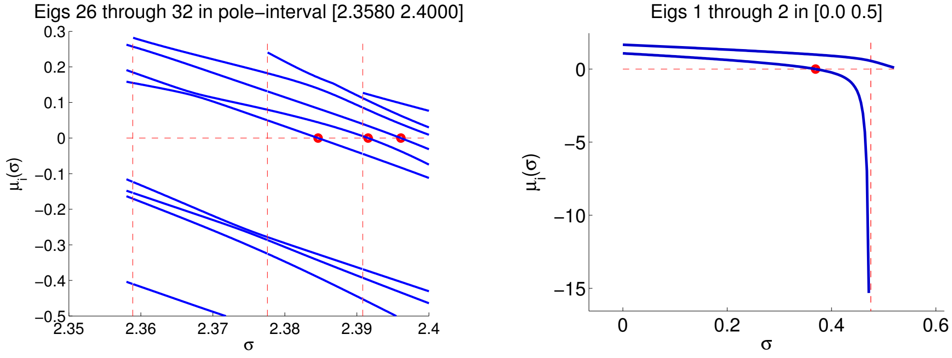

FIG. 4.1. *Left: eigenbranches $\mu_i(\sigma)$ for $i = 26, \ldots, 32$ obtained from a domain decomposition applied to a Laplacian matrix partitioned in $p = 4$ subdomains. The eigenvalues are followed on 3 different panels bordered by poles. There is one eigenvalue of A in the second panel and two in the third. Right: eigenbranches $\mu_1(\sigma)$ and $\mu_2(\sigma)$ as σ approaches the pole of $\mu_1(\sigma)$.*

$$(4.1)\quad S(\sigma) = C - \sigma I - E^T(B-\sigma I)^{-1}E = C - \sigma I - \sum_{j=1}^{d} \frac{w_j w_j^T}{\theta_j - \sigma}\ , \quad \text{with} \quad w_j \equiv E^T v_j .$$

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

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

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

and

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

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 $S_{k,|}(\sigma)$ 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 θ_k from the left or the right direction.

THEOREM 4.1. *If θ_k is a simple eigenvalue, then the following equalities hold:*

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

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

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

Note that from (4.1) we have the following relation for a vector r of unit length

$$\rho(\sigma, r) = \rho_k(\sigma, r) - \frac{(w_k^T r)^2}{\theta_k - \sigma} \,. \tag{4.2}$$

For $j = 1$ we have $\mu_1(\sigma) = \min_{r \neq 0} \rho(\sigma, r)$. By taking $r = w_k/\|w_k\|$ in (4.2), the term $-(w_k^T r)^2/(\theta_k - \sigma)$ can be made arbitrarily large and negative for $\sigma \to \theta_k^-$. Hence, the minimum of $\rho(\sigma, r)$ will have a limit of $-\infty$ when $\sigma \to \theta_k^-$. 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 [14]

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

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 $\sigma \in [\sigma_0, \theta_k]$, $S_k(\sigma)$ 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 $\sigma \in [\sigma_0, \theta_k]$. 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 $\sigma \in [\sigma_0, \theta_k]$. As a result, $\mu_j(\sigma)$, which is the smallest of these maxima over all $\mathcal{U}^j$'s of dimension j, is also greater or equal to η, and so is its limit as $\sigma \to \theta_k^-$. Thus $\lim_{\sigma \to \theta_k^-} \mu_j(\sigma) \geq \eta$.

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

$$\mu_j(\sigma) = \rho(\sigma, y^{(j)}(\sigma)) = \rho_k(\sigma, y^{(j)}(\sigma)) - \frac{(y^{(j)}(\sigma)^T w_k)^2}{\theta_k - \sigma}.$$

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

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 $y^{(j)}(\sigma) = P_k y^{(j)}(\sigma) + (\hat{w}_k^T y^{(j)}(\sigma))\hat{w}_k$ 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 $\sigma \to \theta_k^-$, the operator $P_k S_k(\sigma) P_k$ converges to $P_k S_k(\theta_k) P_k$, which is now well defined. Since $\lim_{\sigma \to \theta_k^-} (\hat{w}_k^T y^{(j)}(\sigma)) = 0$, the above residual converges to zero. Therefore, the eigenpair $(\mu_j(\sigma), P_k y^{(j)}(\sigma))$ converges to an eigenpair of $P_k S_k(\theta_k) P_k$, which is a trivial extension of $S_{k,|}$.

Now we know that each jth 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,

$$\mu_2(\sigma) = \min\{\rho(\sigma, t)\ |t = (I - y^{(1)}(\sigma) y^{(1)}(\sigma)^T) r \neq 0,\ r \in \mathbb{R}^s\}. \tag{4.3}$$

[2] Note that $P_k y^{(j)}(\sigma) = P_k^2 y^{(j)}(\sigma)$.

Since $\lim_{\sigma\to\theta_k^-} y^{(1)}(\sigma) = \hat{w}_k$ (in direction) the limit of the above quantity as $\sigma \to \theta_k^-$ is

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

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

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

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 need to use cumulative projectors, i.e., t in (4.3) is to be replaced by

$$t = (I - y^{(j-1)}(\sigma)y^{(j-1)}(\sigma)^T)\cdots(I - y^{(2)}(\sigma)y^{(2)}(\sigma)^T)(I - y^{(1)}(\sigma)y^{(1)}(\sigma)^T)r.$$

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*) $\lim_{\sigma\to\theta_k^+} \mu_1(\sigma) = +\infty$ and $\lim_{\sigma\to\theta_k^+} \mu_j(\sigma) = \mu_{j-1}(S_{k,|})$ when $j > 1$. Relabeling the eigenvalues *increasingly* yields the result by noting that $S_{k,|}$ has $s - 1$ eigenvalues. □

For simplicity we assumed that θ_k has multiplicity equal to one, but the result can be easily extended to more general situations.

4.2. The inertia theorem in a domain decomposition framework. The tool of choice for counting the number of eigenvalues of a Hermitian matrix in a given interval is Sylvester's inertia theorem [14]. However, this theorem requires the factorization of the whole matrix A, and there are instances when this factorization is too expensive to compute. A sort of distributed version of the theorem can be exploited, and it can be obtained from a block factorization of the matrix. Recall that the inertia of a matrix X is a triplet $[\nu_-(X), \nu_0(X), \nu_+(X)]$ consisting of the numbers of negative, zero, and positive eigenvalues of X, respectively. In the following proposition, the sum of two inertias is defined as the sum of these inertias viewed as vectors of 3 components.

PROPOSITION 4.2. *Let σ be a shift such that $\sigma \notin \Lambda(B)$. Then the inertia of $A - \sigma I$ is the sum of the inertias of $B - \sigma I$ and $S(\sigma)$.*

Proof. The result follows immediately by applying the congruence transformation:

$$\begin{bmatrix} I & \\ -E^T(B-\sigma I)^{-1} & I \end{bmatrix} \begin{bmatrix} B-\sigma I & E \\ E^T & C-\sigma I \end{bmatrix} \begin{bmatrix} I & -(B-\sigma I)^{-1}E \\ & I \end{bmatrix} = \begin{bmatrix} B-\sigma I & \\ & S(\sigma) \end{bmatrix}.$$

Since congruences do not alter inertias, the inertia of A is the same as that of the block-diagonal matrix at the end of the equation shown above. □

The inertia of $S(\sigma)$ can be computed either by explicitly forming and factorizing $S(\sigma)$ or by using the Lanczos algorithm and computing all negative eigenvalues. This result can now be utilized to count the number of eigenvalues of A in the interval $[\alpha, \beta]$.

COROLLARY 4.3. *Assume that neither α nor β are eigenvalues of B, and let ν_a be the number of negative eigenvalues of $S(\alpha)$, ν_β the number of non-positive eigenvalues of $S(\beta)$, and $\mu_{(\alpha,\beta)}(B)$ the number of eigenvalues of B located in (α, β). Then the number of eigenvalues of A in $[\alpha, \beta]$ is equal to $\nu_\beta - \nu_\alpha + \mu_{(\alpha,\beta)}(B)$.*

Proof. Using the previous proposition, the number of eigenvalues of A that are less or equal to β is equal to $\nu_-(S(\beta)) + \nu_-(B - \beta I) + \nu_0(S(\beta)) + \nu_0(B - \beta I)$. By assumption β

is not an eigenvalue of B so $\nu_0(B - \beta I) = 0$. The number of eigenvalues of A that are less than α is equal to $[\nu_-(S(\alpha)) + \nu_-(B - \alpha I)]$. Taking the difference yields,

$$\nu_-(S(\beta)) + \nu_0(S(\beta)) - \nu_-(S(a)) + [\nu_-(B - \beta I) - \nu_-(B - \alpha I)]$$
$$= \nu_\beta - \nu_\alpha + \mu_{(\alpha,\beta)}(B)$$

as desired. □

From a computational point-of-view, the matrix B is block-diagonal, and thus we have $\mu_{(\alpha,\beta)}(B) = \sum_{i=1}^{p} \mu_{(\alpha,\beta)}(B_i)$ with each $\mu_{(\alpha,\beta)}(B_i),\ i = 1,\dots,p$, being computed in parallel. The quantities ν_α and ν_β can be computed either by using a direct factorization for $S(\alpha)$ and $S(\beta)$ or by using the Lanczos method. It is also possible to generalize Corollary 4.3 to the situation where α or β are eigenvalues of B, interpreting $S(\sigma)$ in the way discussed in Section 4.1. The result is omitted.

4.3. 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 3.1 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 3.1. The "Branch-hopping" idea just described above can be formulated as an iterative procedure and is listed as Algorithm 4.1.

ALGORITHM 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 $\eta := \|(B - \sigma I)^{-1} E y(\sigma)\|_2$.
9.          Set $\sigma := \sigma + \mu(\sigma)/(1 + \eta^2)$.
9+.         % If convergence took place at this step, optionally
            % refine $\sigma$ using a few steps of inverse iteration on $(A - \sigma I)$.
10.     End
11. End
```

Similarly to Algorithm 3.1, Step 3 in Algorithm 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 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 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 an inverse iteration back to Newton's iteration can be found in [40]. Regarding the computation of the smallest positive eigenvalue in Step 6 of Algorithm 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 [13] for a detailed discussion on deflation for symmetric linear systems.

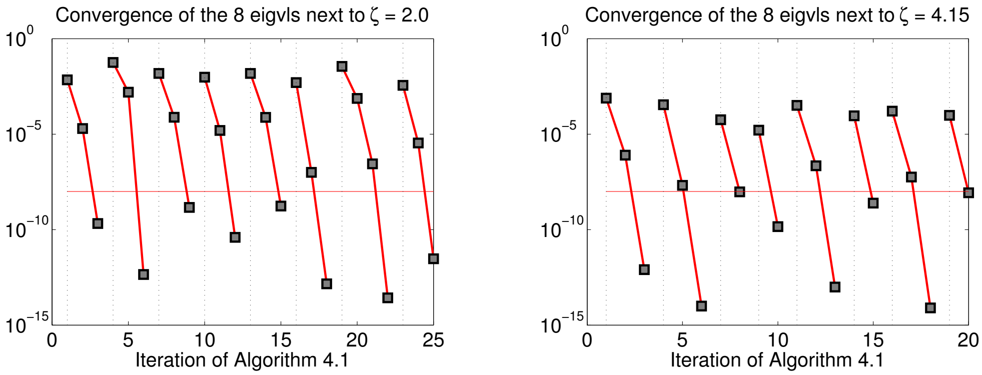

FIG. 4.2. *Residual norm for a few consecutive eigenpairs next to an initial shift* $\zeta \in \mathbb{R}$ *when using Algorithm 4.1. Left:* $\zeta = 2.0$*. Right:* $\zeta = 4.15$.

EXAMPLE 4.4. In Figure 4.2 we plot the eigenpairs' residual norm obtained by Algorithm 4.1 when searching for the $k = 8$ consecutive eigenvalues (from left to right only) next to a pre-selected shift $\zeta \in \mathbb{R}$ for a 2D Laplacian. We used two different shifts, $\zeta = 2.0$ and $\zeta = 4.15$. The horizontal line visualizes the threshold where the norm of the eigenpairs' relative residual is equal to tol= 10^{-8}. The quadratic convergence of the scheme is evident in both plots where the number of correct digits is roughly doubled at each step. Note also that Algorithm 4.1 provides a good starting point for the next targeted eigenpair.

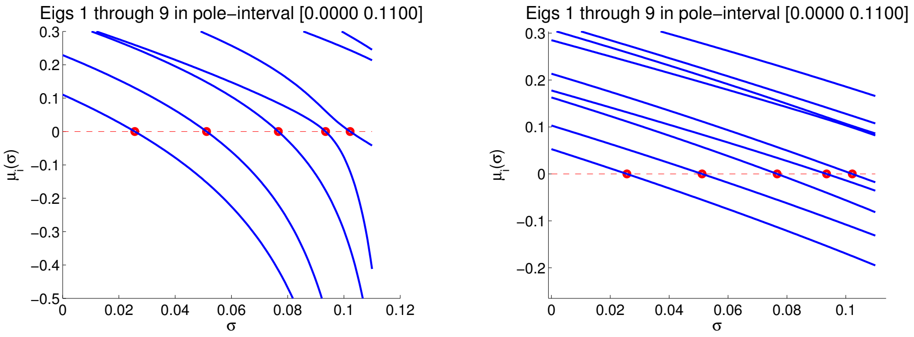

FIG. 4.3. *Eigenbranches* $\mu_1(\sigma), \ldots, \mu_9(\sigma)$ *in the interval* $[0.0, 0.11]$ *for a 2D Laplacian. Left:* $p = 4$*. Right:* $p = 16$.

4.3.1. Influence of the number of subdomains. In a Newton scheme, functions that are nearly linear lead to faster convergence. We often observe that the shape of the eigenbranches tend(s) to become closer to linear as the number p of subdomains increases. This behavior is illustrated in Figure 4.3 with a small example of a 2D discretized Laplacian. We used $p = 4$ and $p = 16$ subdomains and plot the eigenbranches $\mu_1(\sigma), \ldots, \mu_9(\sigma)$ in the interval $[0.0, 0.11]$. Notice the difference in the shape of the eigenbranches: for $p = 16$ the eigenbranches are closer to straight lines while for $p = 4$ the eigenbranches tend to bend more. The explanation behind this phenomenon is similar to the discussion in Section 4.1. Consider a specific eigenbranch

$\mu_i(\sigma)$ that has two poles. When the poles of $\mu_i(\sigma)$ are far apart, the shape of the eigenbranch $\mu_i(\sigma)$ is closer to linear in the middle of the interval defined by its poles. This follows by the expression of the derivative $\mu_i'(\sigma)$, as given by (3.2), in conjunction with Theorem 4.1. The value of $\mu_i'(\sigma)$ changes fast when σ approaches one the poles of $\mu_i(\sigma)$, while, on the other hand, varies slowly when σ lies away the poles.[3] As a consequence, if the root of $\mu_i(\sigma)$ is located far from its poles, e.g., located towards the middle of the interval defined by the two poles of $\mu_i(\sigma)$, we should expect Algorithm 4.1 (Algorithm 3.1) to converge faster.

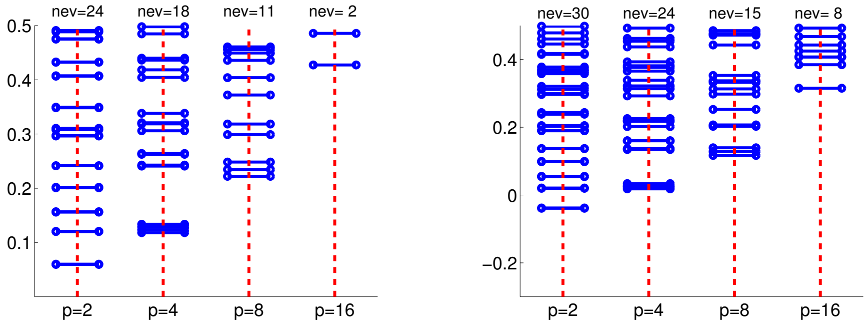

FIG. 4.4. *Eigenvalue distributions for various B's obtained for a discretized 2D Laplacian of dimension $n = 1224$ for different numbers of subdomains. Left: original. Right: with random diagonal perturbation. The y-axis shows the value of the eigenvalues of B which fall within the interval $[0.0, 0.5]$ and $[-0.3, 0.5]$ for the original and perturbed Laplacians, respectively. The x-axis indexes four different values of p. By "nev" the number of eigenvalues within the interval is denoted.*

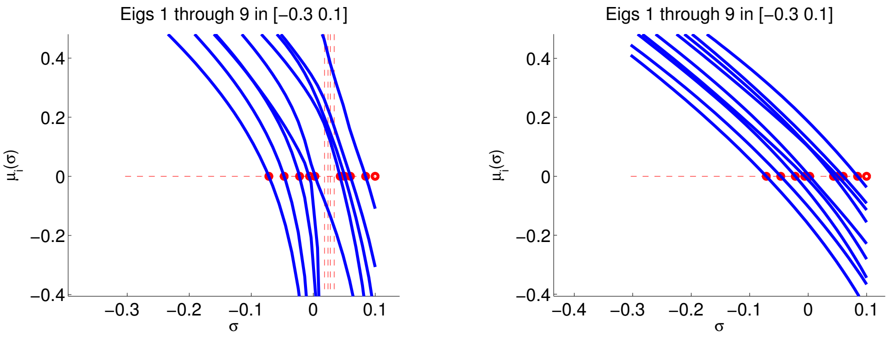

FIG. 4.5. *Eigenbranches $\mu_1(\sigma), \ldots, \mu_9(\sigma)$ in the interval $[-0.3, 0.1]$ for a perturbed 2D Laplacian. Left: $p = 4$. Right: $p = 16$.*

The important question now is whether increasing the number of subdomains p will result in a more favorable distribution of the poles of the eigenbranches under consideration, e.g., the poles of the eigenbranches that have a root inside $[\alpha, \beta]$. For simplicity, assume that each sought eigenvalue λ is simple and that its corresponding eigenbranch has two poles. For a given number of p subdomains, B is a $d \times d$ (leading) principal submatrix of A, and, as a consequence of the Courant-Fischer theorem, the eigenvalues of B will interlace those of A as $\lambda_j \leq \theta_j \leq \lambda_{n+j-d},\ j = 1, \ldots, d$. Larger values of p will result in smaller values for d

[3] In practice, the rate of change of $\mu_i(\sigma)$ depends on the distance $(\sigma - \theta)^2$ with θ the pole of $\mu_i(\sigma)$ lying the closest to σ.

and fewer poles in general, with each interval $[\theta_j, \theta_{j+1}],\ j = 1, \ldots, d-1$, now potentially including more eigenvalues of A. As a result, it is less likely that the poles of the eigenbranch of interest will be located near its root λ.

As an illustration consider again the first few eigenbranches of the same 2D Laplacian used in Figure 4.3. The left subfigure of Figure 4.4 shows the distribution of the eigenvalues of B inside the interval $[0.0, 0.5]$. Notice the difference between $p = 4$ and $p = 16$. In the latter case the few smallest eigenvalues of B, which are also poles of the first few eigenbranches, have shifted towards the interior of the spectrum and lie in a greater distance from the interval $[0.0, 0.1]$. This explains why the shape of the first few eigenbranches in Figure 4.3 is closer to linear in the interval $[0.0, 0.1]$ if $p = 16$. The above experiment is repeated by adding a pseudo-random perturbation to the diagonal of the same 2D Laplacian. The spectral distributions for the few smallest eigenvalues of B as p varies are shown in the right subfigure of Figure 4.4. Note that in this case the perturbation led to negative eigenvalues and the interval of interest for the perturbed case was set to $[-0.3, 0.5]$. We can see that larger values of p force the few smallest eigenvalues of B, which again are the poles of the first few eigenbranches, to move towards the interior of the spectrum. Thus, using $p = 16$ results in the first few eigenbranches having a shape closer to that of a linear function than with $p = 4$ and this is verified in Figure 4.5.

4.4. Other practical considerations. Before we conclude the discussion on the algorithmic scheme developed in Algorithm 4.1, let us comment on a few practical issues. First, Algorithm 4.1 (as well as Algorithm 3.1) can be extended to compute eigenpairs of A which correspond to multiple eigenvalues. A multiple eigenvalue λ of A translates into a multiple (with the same degree of multiplicity) zero eigenvalue of $S(\lambda)$. Each eigenvector of $S(\lambda)$ associated with one of the zero eigenvalues can be used to extract a separate eigenvector of A. The eigenvectors of $S(\lambda)$ can be recovered using a subspace version of inverse iteration.

Furthermore, after hopping to a new eigenbranch, the value of σ might not be close enough to the next targeted eigenvalue. Usually this results to slower convergence of the Newton scheme. In the extreme case, Algorithm 4.1 might miss the targeted eigenvalue and converge to a nearby one. The inertia tool discussed in Section 4.2 can then be run periodically to determine whether the scheme proceeds as expected.

Finally, we should also remark that the Branch-hopping scheme computes each eigenpair in a sequential manner, i.e., one after the other. If too many eigenpairs are sought, then it could be more efficient to use techniques which solve the non-linear symmetric (Hermitian) eigenvalue problem of the form $S(\lambda)y(\lambda) = 0$ for several eigenpairs simultaneously; see for example [41, 42] for some recent developments towards this direction based on preconditioned block methods.

5. The Branch-hopping Newton scheme in a parallel framework.

In a parallel implementation we assume that the matrix A is distributed across a set of p processors, with each processor holding a certain part of its rows as the local system (2.1). The smallest eigenvalues $\mu(\sigma)$ of each spectral Schur complement $S(\sigma)$ can be computed by an inverse iteration, which relies on solving linear systems of the form $S(\sigma)w = b$. The linear solves are performed by an iterative matrix-free method, e.g., a Krylov subspace method, that does not require an explicit construction of $S(\sigma)$. Specifically, each iteration of the iterative linear solver requires a matrix-vector (MV) product with $S(\sigma)$, which can be done partially in parallel as follows.

Conceptually, the global spectral Schur complement $S(\sigma)$ in (2.5) is also distributed across the p processors, with each processor handling a different subdomain. The MV product with $S(\sigma)$ is computed as

$$S(\sigma)x = (C - \sigma I)x - E^T(B - \sigma I)^{-1}Ex. \tag{5.1}$$

An important property from the DD method with edge-separation is that the second term on the right-hand side of (5.1) is entirely local, as can be easily seen from the structure of $S(\sigma)$ in (2.5). In summary, the computations involved in (5.1) are:

1. MV multiplication with $C - \sigma I$ (non-local),
2. MV multiplication with E and E^T (local),
3. system solution with $B - \sigma I$ (local),

where only the first computation requires communication between adjacent subdomains. Hence, the communication cost of the MV product with $S(\sigma)$ is the same as that with C and is point-to-point. Note here that there is a trade-off between the qualitative and computational performance of the algorithms described in this paper. Using a larger value for p will typically bring the shape of the eigenbranches closer to being linear, thus enhancing convergence of Algorithms 3.1 and 4.1, but, from a computational point-of-view, might lead to less scalable MV products with $S(\sigma)$ since now more neighboring subdomains will have to exchange information and might overshadow the gain by reducing the local computations per subdomain.

The other major communication cost when solving linear systems with $S(\sigma)$ is introduced by the vector dot-products used in the projection method of choice and are of the all-reduction kind. The local solve with the block-diagonal $(B - \sigma I)$ is carried out by using a sparse direct method, and $(B - \sigma I)$ is factored once for each shift σ.

6. Numerical experiments. This section reports on numerical results with the proposed Newton schemes listed in Algorithms 3.1 and 4.1. All numerical experiments were performed on 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 numerical experiments are organized in three different sets. In the first set, we assume that each eigenpair $(\mu(\sigma), y(\sigma))$ in Algorithms 3.1 and 4.1 is computed with the highest possible accuracy in order to study the algorithm theoretically. The second set of experiments consists of results obtained in distributed computing environments where we are interested in measuring actual wall-clock timings and $(\mu(\sigma), y(\sigma))$ is only approximately computed. The third set of experiments consists of a brief experimental framework with matrices from electronic structure calculations.

To compute a single eigenpair we use Algorithm 3.1 and to compute a few consecutive eigenpairs, or all eigenpairs inside a given interval, we will use Algorithm 4.1.

6.1. Model problem. The test matrices in this section originate from discretizations of elliptic PDEs on two (and three) dimensional domains. More specifically, we are interested in solving the eigenvalue problem

$$-\Delta u = \lambda u$$

on a rectangular domain with Dirichlet boundary conditions; Δ denotes the Laplacian differential operator. Using second order centered finite differences with n_x, n_y, and n_z discretization points in each corresponding dimension, we obtain a matrix A, the discretized version of Δ, which is of size $n = n_x n_y n_z$.

6.2. Algorithm validation. This section focuses on the numerical behavior of Algorithms 3.1 and 4.1 and so the results described here were produced by using a MATLAB prototype implementation. In addition, the Lanczos algorithm with partial re-orthogonalization is used to compute $(\mu(\sigma), y(\sigma))$.

TABLE 6.1

Number of Newton iterations when computing all eigenvalues inside the interval $[\alpha, \beta]$ *by Algorithm 4.1. Different matrix sizes and number of subdomains are used.*

		$[\alpha, \beta] := [0, 0.5]$		$[\alpha, \beta] := [2, 2.2]$		$[\alpha, \beta] := [4.1, 4.2]$	
		#Eigvls	It	#Eigvls	It	#Eigvls	It
$n = 21 \times 20 \times 9$							
# subdomains (p)	2	14	41	41	85	55	124
	4		26		74		80
	8		32		60		70
	16		32		55		70
$n = 21 \times 20 \times 19$							
# subdomains (p)	2	35	60	82	152	127	360
	4		43		130		172
	8		35		116		152
	16		39		96		148
$n = 41 \times 20 \times 19$							
# subdomains (p)	2	72	210	154	342	209	424
	4		170		292		314
	8		154		273		310
	16		138		241		300
$n = 41 \times 40 \times 20$							
# subdomains (p)	2	160	385	319	703	472	802
	4		354		540		647
	8		296		502		592
	16		270		451		533

Table 6.1 shows the numerical performance of Algorithm 4.1 for a few discretized Laplacians of size $n \approx 4000,\ 8000,\ 16000$, and $n \approx 32000$, when computing all eigenpairs inside a given interval $[\alpha, \beta]$. The intervals were selected as $[\alpha, \beta] = [0, 0.5]$, $[2, 2.2]$, and $[4, 4.1]$. Note that the last two intervals lie well inside the spectrum. For the purposes of this experiment we enforced a strict requirement for convergence by setting tol$= 10^{-12}$. For each different discretization we also used a varying number of subdomains p. For each interval $[\alpha, \beta]$ we list the number of eigenvalues inside the interval (denoted as "#Eigvls") as well as the total number of Newton iterations in Algorithm 4.1 to compute all eigenvalues (denoted as "It"). We observe that, on average, a few Newton iterations per eigenvalue are enough to compute each eigenpair close to machine precision. Moreover, when using larger values for p, we typically need only one or two Newton steps per eigenpair. This was expected from our observations in Section 3 since the shape of the eigenbranches of $S(\cdot)$ is better approximated by a linear function when we use more subdomains, and thus Newton's method converges faster. Note that when using a large number of subdomains, the initial guess for the next unconverged eigenpair is usually much more accurate.

Figure 6.1 shows the convergence behavior of Algorithm 3.1 when searching for a single eigenpair closest to ζ, for a Laplacian discretized as $n_x = 11,\ n_y = 10,\ n_z = 9$. In the preprocessing phase, we applied a few steps of an inverse iteration with $(A - \zeta I)$ to generate initial guesses of varying accuracy for Algorithm 3.1. The dashed line shows the convergence of inverse iteration on $(A - \zeta I)$ while the circled curves shows the convergence of Algorithm 3.1. Note that the leftmost circled curve stands for Algorithm 3.1 with $\sigma := \zeta$ as the starting shift.

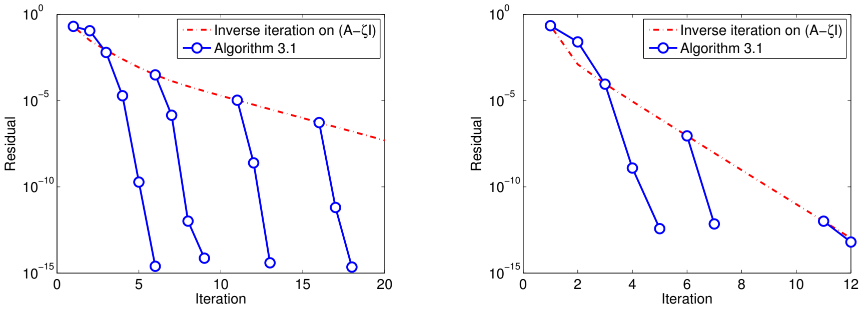

FIG. 6.1. *Convergence behavior for computing the eigenpair closest to* ζ *when combining inverse iteration and Newton's method. In the left subfigure* $\zeta = 0.0$ *while in the right subfigure* $\zeta = 3.0$.

As expected when the initial guess is not very accurate, more iterations of Algorithm 3.1 may be needed for convergence. As the initially provided approximation becomes more accurate, one or two iterations of Algorithm 3.1 are usually enough to reduce the residual norm of the approximate eigenpair below 10^{-10}.

6.3. Distributed computing environments. In this section we present results obtained on distributed computing environments using our parallel implementation of Algorithms 3.1 and 4.1. We focus on parallel execution timings and report results both for extreme and interior eigenvalue problems.

6.3.1. Experimental setup. We implemented and tested three different methods: a) (inexact) residual inverse iteration applied to A (each time with the appropriate shift) as discussed in [28], b) Newton's method as described in Algorithms 3.1 and 4.1, and c) a combination of (inexact) inverse iteration with Algorithms 3.1 and 4.1 where we first perform a few steps of inverse iteration with A followed by the Newton's scheme. We chose to compare against residual inverse iteration mainly because of its simplicity and the fact that it represents the most likely contender to our approach. As in the proposed Newton approach, residual inverse iteration approximates an eigenpair "in-place", e.g., without building a subspace.

All algorithms were implemented in C/C++ linked with the Intel Math Kernel [1] and PETSc [3] libraries, compiled under the Intel MPI compiler using the -O3 optimization level. For Algorithms 3.1 and 4.1, the number of subdomains used will always equal the number of cores used and each distinct subdomain is handled by a single core; distributed-memory model only. The matrix $(B - \sigma I)$ was factored by the LDL factorization, obtained by the associated routine in the CHOLMOD [8] package. We did not consider any further high-performance computing optimizations.

For the inexact residual inverse iteration method applied to A, 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 3.1 and 4.1, or pre-processed by an inverse iteration with A, 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= 10^{-2}. For all (iterative) linear system solutions, we used the MINimum RESisudal (MINRES) [29] method as the iterative solver; we used the existing implementation in PETSc. By default we used no preconditioning for the inner solution in any of the methods tested. The residual norm tolerance for accepting an approximate eigenpair was set to tol= 10^{-8}, although the proposed Newton method almost all the times returned an approximation with residual norm less than 10^{-10} or 10^{-11}. The residual norm was always evaluated directly by

TABLE 6.2

Computing $k = 1$ and $k = 5$ 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.

$n = 601 \times 600$

		$\zeta = 0.0$			$\zeta = 0.01$ (269)		
(p, k)	s	T_{NT}	It	T_{RI}	T_{NT}	It	T_{RI}
(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

$n = 801 \times 800$

		$\zeta = 0.0$			$\zeta = 0.01$ (488)		
(p, k)	s	T_{NT}	It	T_{RI}	T_{NT}	It	T_{RI}
(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

$n = 1001 \times 1000$

		$\zeta = 0.0$			$\zeta = 0.01$ (764)		
(p, k)	s	T_{NT}	It	T_{RI}	T_{NT}	It	T_{RI}
(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

using the formula $\|r\| = \|A\hat{x}(\sigma) - \sigma\hat{x}(\sigma)\|/\|\hat{x}(\sigma)\|$. All experiments were repeated multiple times and the average execution time is reported.

6.3.2. Results. Table 6.2 shows the actual wall-clock timings when computing $k = 1$ and $k = 5$ consecutive eigenpairs next to $\zeta \in \mathbb{R}$ for a set of 2D model problems, while Table 6.3 presents similar results for a set of 3D model problems. 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 $(A - \zeta I)$. As previously, p denotes the number of subdomains, s denotes the size of the Schur complement matrix $S(\cdot)$, and "It" denotes the total number of Newton steps performed by Algorithm 3.1 (when $k = 1$) or Algorithm 4.1; when $k = 5$. We denote the total time spent in Algorithm 3.1 or Algorithm 4.1 by "T_{NT}" while the time spent in residual inverse iteration applied to A is denoted as "T_{RI}". All timings are listed in seconds. Because $(\mu(\sigma), y(\sigma))$ is

TABLE 6.3

Computing $k = 1$ and $k = 5$ 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.

$n = 41 \times 40 \times 39$

		$\zeta = 0.0$			$\zeta = 0.1$ (19)		
(p, k)	s	T_{NT}	It	T_{RI}	T_{NT}	It	T_{RI}
(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

$n = 71 \times 70 \times 69$

		$\zeta = 0.0$			$\zeta = 0.1$ (137)		
(p, k)	s	T_{NT}	It	T_{RI}	T_{NT}	It	T_{RI}
(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

$n = 101 \times 100 \times 99$

		$\zeta = 0.0$			$\zeta = 0.1$ (439)		
(p, k)	s	T_{NT}	It	T_{RI}	T_{NT}	It	T_{RI}
(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

computed only approximately, extra care must be taken in order to avoid divergence when $\zeta \neq 0$. We always performed three steps of an (inexact) inverse iteration with A 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 6.2. 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 $(A - 0.01I)$ 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 $(B - \sigma I)$ matrix is more expensive. The Newton-based approach becomes faster when we use enough subdomains.

TABLE 6.4
Computing $k = 1$ and $k = 5$ 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, k)	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	—

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 is computed on its own. Also, note that increasing the number of subdomains does not lead to a great reduction in the number of total Newton steps (as was the case in Table 6.1) because now $(\mu(\sigma), y(\sigma))$ is computed only approximately.

6.4. A comparison with ARPACK. This section provides a brief comparison between the Newton scheme and ARPACK [24], 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 method 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 the search subspace equal to twenty, and we used only one execution core. The tolerance tol for the requested eigenpairs set again to tol= 10^{-8} for both methods. As a demonstration, we used a single test, a 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 6.4 compares the execution times obtained of the Newton (T_{NT}) and ARPACK (T_{ARP}) schemes when searching for $k = 1$ and $k = 5$ eigenpairs of A and using a different number of subdomains. Because ARPACK operates directly on A 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.

6.5. Matrices from electronic structure calculations. This section discusses a few experiments with matrices from the PARSEC matrix set available from the University of Florida sparse matrix collection [10]. These matrices were originally obtained from the PARSEC

TABLE 6.5

Computing a single eigenpair closest to shifts ζ_1, ζ_2, and ζ_3, for a set of Hamiltonian matrices from the PARSEC matrix group. Time is listed in seconds.

Si10H16, $n = 17077$, $nnz = 875923$						
	$\zeta_1 = 0.29$		$\zeta_2 = 0.51$		$\zeta_3 = 1.11$	
p	T_{NT}	T_{RI}	T_{NT}	T_{RI}	T_{NT}	T_{RI}
4	3.82	0.69	6.9	4.27	13.3	11.1
8	0.45	0.48	1.3	3.11	4.6	8.52
SiO, $n = 33401$, $nnz = 1317655$						
	$\zeta_1 = 0.75$		$\zeta_2 = 1.02$		$\zeta_3 = 2.17$	
p	T_{NT}	T_{RI}	T_{NT}	T_{RI}	T_{NT}	T_{RI}
4	9.91	6.23	30.1	30.9	41.0	59.1
8	2.11	3.78	15.2	20.1	22.4	32.7
H2O, $n = 67024$, $nnz = 2216736$						
	$\zeta_1 = 1.35$		$\zeta_2 = 1.88$		$\zeta_3 = 3.84$	
p	T_{NT}	T_{RI}	T_{NT}	T_{RI}	T_{NT}	T_{RI}
16	15.2	12.2	17.5	33.9	29.3	23.1
32	7.02	10.8	7.39	20.5	11.4	17.4

code for electronic structure calculations using a Density Functional Theory (DFT) approach. The Hamiltonians are sparse and symmetric with multiple (and clustered) eigenvalues. Each Hamiltonian has a number of occupied states, say n_0, which is the number of smallest eigenvalues requested. While extensions are possible, as it is described, the proposed scheme cannot compute multiple eigenvalues. In this set of experiments, we restrict our attention in computing a *single* eigenpair of each Hamiltonian matrix.

The results are shown in Table 6.5. We kept the same notation as in the previous section. Next to each matrix we also list its size n, as well as nnz the total number of non-zero entries. Unlike the model problem, the number of non-zero entries per row for the Hamiltonians is fairly large (57.3, 39.4, and 33.07 non-zeros/row from smallest to largest matrices), and after partitioning the number of interface nodes, s tends to be quite large; especially for higher-order finite difference schemes. These PARSEC matrices are three-dimensional discretizations of Hamiltonian matrices in real-space. To begin with, the number of nonzero entries in the original matrix is quite large, a consequence of the high-order discretization and the addition of a (dense) "non-local" term. Together with the 3D nature of the problem this leads to an unusually large number of interface points—even if a good partitioning is employed. As a result the Schur complement matrix tends to be larger, which leads to a harder problem for the Newton scheme. For each test matrix we used three different shifts $\zeta_1, \zeta_2, \zeta_3$, determined so that the shifted matrix had $40,\ 70$, and 200 negative eigenvalues respectively. Similarly with the results obtained in the previous subsection, raising the number of subdomains typically leads to faster convergence for the Newton scheme. The Newton scheme is generally faster than residual inverse iteration, however, both methods start to deteriorate as we move deeper into the spectrum. We should also note that, as was observed in our experiments, both methods can capture five or six digits of accuracy in a relatively short amount of time, but after this point, convergence experiences a plateau until the requested tolerance tol= 10^{-8} is finally met.

7. Conclusion.

The method presented in this paper for solving symmetric eigenvalue problems 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 iterations 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.

A key issue still requiring further investigation is to find effective ways to approximate the eigenpairs $(\mu(\sigma), y(\sigma))$ of the Schur complement. 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, e.g., toward its middle. 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 solver as the shift changes. Because these ideas require a separate and rather involved study, we opted to leave them for future work.

Acknowledgments. We would like to thank the anonymous referees and the editor for their valuable suggestions and insightful comments which greatly improved the paper. We are grateful to the University of Minnesota Supercomputing Institute for providing us with computational resources and assistance with the computations. We also thank Andreas Stathopoulos for fruitful discussions.

REFERENCES

[1] *Intel Fortran compiler XE2013SP1 (14.0) for Linux.*

[2] J. L. Aurentz, V. Kalantzis, and Y. Saad, *A GPU implementation of the filtered Lanczos procedure*, Tech. Report ys-2015-4, Dept. Computer Science and Engineering, University of Minnesota, Minneapolis 2015.

[3] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang, *Petsc users manual revision 3.2*, Argonne National Lab., Mathematics and Computer Science Division, University of Chicago, Chicago.

[4] C. Bekas and Y. Saad, *Computation of smallest eigenvalues using spectral Schur complements*, SIAM J. Sci. Comput., 27 (2005), pp. 458–481.

[5] J. K. Bennighof and R. B. Lehoucq, *An automated multilevel substructuring method for eigenspace computation in linear elastodynamics*, SIAM J. Sci. Comput., 25 (2004), pp. 2084–2106.

[6] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine, *The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: partitioning, ordering, and coloring*, Sci. Programming, 20 (2012), pp. 129–150.

[7] U. V. Çatalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Systems, 10 (1999), pp. 673–693.

[8] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, *Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Software, 35 (2008), Art. 22, (14 pages).

[9] B. Cockburn, J. Gopalakrishnan, F. Li, N.-C. Nguyen, and J. Peraire, *Hybridization and postprocessing techniques for mixed eigenfunctions*, SIAM J. Numer. Anal., 48 (2010), pp. 857–881.

[10] T. A. Davis, *University of Florida sparse matrix collection*, NA digest, vol. 94, October 16, 1994.

[11] H.-R. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comput., 34 (2012), pp. A2220–A2246.

[12] M. A. Freitag and A. Spence, *Convergence of inexact inverse iteration with application to preconditioned iterative solves*, BIT, 47 (2007), pp. 27–44.

[13] A. Gaul, M. H. Gutknecht, J. Liesen, and R. Nabben, *A framework for deflated and augmented Krylov subspace methods*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 495–518.

[14] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, 2013.

[15] J. Gopalakrishnan, F. Li, N.-C. Nguyen, and J. Peraire, *Spectral approximations by the HDG method*, Math. Comp., 84 (2015), pp. 1037–1059.

[16] I. C. F. Ipsen, *Computing an eigenvector with inverse iteration*, SIAM Rev., 39 (1997), pp. 254–291.

[17] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[18] T. Kato, *Perturbation Theory for Linear Operators*, 2nd ed., Springer, Berlin, 1976.

[19] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995.

[20] A. V. Knyazev and A. L. Skorokhodov, *Preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problems*, SIAM J. Numer. Anal., 31 (1994), pp. 1226–1239.

[21] T. G. Kolda, *Partitioning sparse rectangular matrices for parallel processing*, in Solving Irregularly Structured Problems in Parallel (Berkeley, 1998), A. Ferreira, J. Rolim, H. Simon, and S.-H. Teng, eds., vol. 1457 of Lecture Notes in Comput. Sci., Springer, Berlin, 1998, pp. 68–79.

[22] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Research Nat. Bur. Standards, 45 (1950), pp. 255–282.

[23] P. D. Lax, *Linear Algebra and Its Applications*, 2nd ed., Wiley, Hoboken, 2007.

[24] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide*, SIAM, Philadelphia, 1998.

[25] Z. Li, Y. Saad, and M. Sosonkina, *pARMS: a parallel version of the algebraic recursive multilevel solver*, Numer. Linear Algebra Appl., 10 (2003), pp. 485–509.

[26] S. H. Lui, *Kron's method for symmetric eigenvalue problems*, J. Comput. Appl. Math., 98 (1998), pp. 35–48.

[27] ———, *Domain decomposition methods for eigenvalue problems*, J. Comput. Appl. Math., 117 (2000), pp. 17–34.

[28] A. Neumaier, *Residual inverse iteration for the nonlinear eigenvalue problem*, SIAM J. Numer. Anal., 22 (1985), pp. 914–923.

[29] C. C. Paige and M. A. Saunders, *Solutions of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[30] F. Pellegrini, *Scotch and libScotch 5.1 user's guide*, INRIA Bordeaux Sud-Ouest, IPB & LaBRI, UMR CNRS 5800, Université Bordeaux, 2010.

[31] B. Philippe and Y. Saad, *On correction equations and domain decomposition for computing invariant subspaces*, Comput. Methods Appl. Mech. Engrg., 196 (2007), pp. 1471–1483.

[32] E. Polizzi, *Density-matrix-based algorithms for solving eigenvalue problems*, Phys. Rev. B., 79 (2009), Art. 115112, (6 pages).

[33] A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.

[34] T. Sakurai and H. Sugiura, *A projection method for generalized eigenvalue problems using numerical integration*, J. Comput. Appl. Math. 159 (2003), pp. 119–128.

[35] A. H. Sameh and J. A. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM J. Numer. Anal., 19 (1982), pp. 1243–1259.

[36] H. Simon, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp, 42 (1984), pp. 115–142.

[37] G. Sleijpen and H. Van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM Rev., 42 (2000), pp. 267–293.

[38] A. Stathopoulos, Y. Saad, and C. Fischer, *A Schur complement method for eigenvalue problems*, Preliminary Proceedings of the 7th Copper Mountain Conference on Multigrid Methods, NASA Conference Proceedings, NASA, 1995.

[39] A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods*, SIAM J. Sci. Comput., 19 (1998), pp. 227–245.

[40] D. B. Szyld, *Criteria for combining inverse and Rayleigh quotient iteration*, SIAM J. Numer. Anal., 25 (1988), pp. 1369–1375.

[41] D. B. Szyld, E. Vecharynski, and F. Xue, *Preconditioned eigensolvers for large-scale nonlinear Hermitian eigenproblems with variational characterizations. II. Interior eigenvalues*, SIAM J. Sci. Comput., 37 (2015), pp. A2969–A2997.

[42] D. B. Szyld and F. Xue, *Preconditioned eigensolvers for large-scale nonlinear Hermitian eigenproblems with variational characterizations. I. Conjugate gradient methods*, Tech. Report 14-08-26, Department of Mathematics, Temple University, August 2014.

[43] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.