Proceedings of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN), Vol. 1 (2012) pp. 123-132.

FAST STRUCTURED EIGENSOLVER FOR DISCRETIZED PARTIAL DIFFERENTIAL OPERATORS ON GENERAL MESHES*

JIANLIN XIA†, YUANZHE XI‡, AND MAARTEN V. DE HOOP§

Abstract. In this work, we show a fast structured method for finding the eigenvalues of some discretized PDEs on general meshes, as well as symmetric sparse matrices. A fast structured multifrontal factorization scheme is considered, and the organization and partition of the separators in nested dissection for a general graph is considered for the purpose of efficient structured matrix operations. This structured factorization is used to compute the inertia of the matrix, and is then combined with the bisection scheme to quickly look for the eigenvalues. This method is especially useful in finding selected eigenvalues of discretized operators. Moreover, unlike some rank structured direct solvers, our method does not require related off-diagonal numerical ranks to be small for given tolerances. Numerical experiments show the costs and accuracies.

Key words. sparse eigensolver, inertia, bisection, low-rank structure, structured LDL factorization

1. Introduction. In mathematical and engineering computations, it often needs to find the eigenvalues and eigenvectors of large sparse matrices. In particular, sparse eigensolutions provide important quantitative descriptions of the behaviors of PDEs in practical applications. Classical iterative methods such as Arnoldi type methods can take good advantage of the sparsity. However, the convergence may be slow. Moreover, they generally cannot find selected eigenvalues in given intervals.

Among various eigensolvers, the bisection method [9, 11] has some useful properties. It uses inertia computations which only need to estimate the signs of the eigenvalues. It can be used to find eigenvalues in arbitrary intervals. Sometimes, LDL factorizations are used to estimate the inertia. However, such factorizations are often expensive for sparse matrices.

In this paper, we propose a structured sparse eigensolver using bisection and approximate LDL factorizations. Rank structured matrices are used in a sparse matrix factorizations scheme called the multifrontal method [8] to yield structured multifrontal solvers as in [24, 26, 27]. Under certain conditions, such solvers have costs around $O(n)$ flops in 2D and from about $O(n)$ to $O\left(n^{4/3}\right)$ in 3D. This uses a low-rank property. That is, during the direct factorization of certain discretized matrices, the intermediate dense matrices have small off-diagonal numerical ranks, or the numerical ranks follow certain patterns.

Here, for problems where only finite number of eigenvalues are desired, we further use an idea of aggressive low-rank truncation. That is, even if the intermediate dense matrices have relatively high off-diagonal ranks, we can still aggressively truncate off-diagonal singular values. That is, we can manually set a numerical rank that is related to the number of eigenvalues desired. This is especially useful for different problems and 3D ones where the low-rank property do not exist.

1.1. Preliminaries. The structured matrix we use is called hierarchically semiseparable (HSS) form [4, 5, 19, 29], which is a special case of $\mathcal{H}$- and $\mathcal{H}^2$- matrices [1, 2, 12]. In an HSS matrix, the off-diagonal blocks appear in compressed forms, given by a sequence of smaller matrices, called HSS generators. The generators are defined hierarchically so as to improve efficiency. The operations of

*This work was supported in part by the members, BP, ConocoPhillips, ExxonMobil, StatoilHydro and Total, of the Geo-Mathematical Imaging Group. The research of Jianlin Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024. The research of Maarten V. de Hoop was supported in part by NSF grant DMS-1025318.

†Department of Mathematics, Purdue University, West Lafayette, IN 47907, xiaj@math.purdue.edu

‡Department of Mathematics, Purdue University, West Lafayette, IN 47907, yxi@math.purdue.edu

§Department of Mathematics, Purdue University, West Lafayette, IN 47907, mdehoop@math.purdue.edu

the matrices are then replaced by those on the generators. If the off-diagonal block of the matrix have small (numerical) ranks, many matrix operations can be performed in nearly linear time via the HSS form. Examples include HSS matrix-vector multiplication, HSS matrix factorization, and HSS inversion.

In [27, 26], HSS forms are built into the multifrontal method to represent or approximate dense matrices. The multifrontal method converts a sparse factorization into a sequence of intermediate dense factorizations. This is thus very suitable for parallelization and for structured matrix algorithms.

1.2. Outline. The remaining sections are organized as follows. Section 2 discusses a matrix reordering method for general meshes in structured factorizations. A structured LDL factorization method is then shown in Section 3. Our eigensolver is then presented in Section 4, followed by some numerical tests in Section 5.

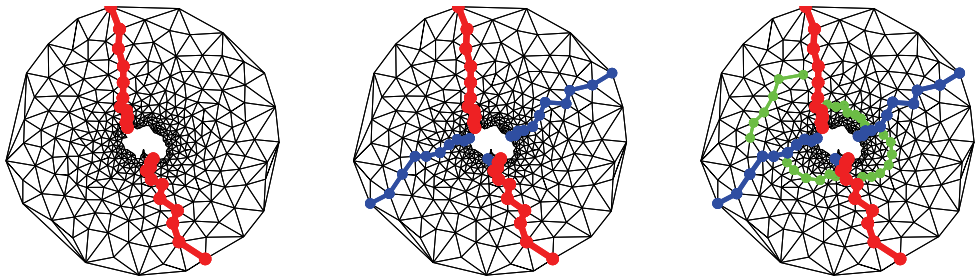
2. Matrix reordering with nested dissection and separator partitioning. In our sparse eigensolver for a symmetric matrix A, we first order with nested dissection [10] so as to reduce fill-in in the factorization of A or A with diagonal shifts.

2.1. Nested dissection for general meshes. Since A is symmetric, we can construct an adjacency graph for it. The following definition is commonly used.

DEFINITION 2.1. *(Adjacency graph) The adjacency graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *for an* $n \times n$ *matrix* A *is defined by a vertex set (or set of grid points)* $\mathbf{V}$ *and an edge set* $\mathbf{E}$ *given as follows:*

$$\mathbf{V} = \{1, 2, \cdots, n\}, \ \mathbf{E} = \{(i,j) | a_{i,j} \neq 0, i, j \in \mathbf{V}.$$

In nested dissection, separators or small sets of vertices in $\mathbf{V}$ are used to recursively partition $\mathbf{G}$ into subgroups. As in [26], we handle general graphs or meshes. Here, we present the detailed strategies. Graph partition algorithms and packages such as METIS [14] can be used to construct a bipartition. We then derive a separator from the partition. See Figure 1 for an example.



(i) One level (ii) Two levels (iii) Three levels

FIG. 1. *Nested dissection ordering of an irregular mesh, where the images are based on a matrix in [7].*

Then a separator tree $\mathbf{T}$ is generated, where each separator corresponds to a node of $\mathbf{T}$. As in [22], we build a binary tree $\mathbf{T}$. The root of $\mathbf{T}$ (denoted root (T)) represents the top-level separator which divides $\mathbf{G}$ into two subgraphs. The child nodes can then similarly be decided. The recursive partition stops when the numbers of nodes in each subgraph are smaller than a given minimum. $\mathbf{T}$ is also called the assembly tree in the multifrontal method we use [8, 18], and can be conveniently represented by a vector. Later, we denote the vertices in a separator (or a node of $\mathbf{T}$) $\mathbf{i}$ by $\mathbf{t_i}$.

2.2. Separator partition and connectivity. As discussed in [27], different ordering of the nodes in the separators and the neighbors of a separator can highly affect the performance of the structured factorizations. Neighbor separators of a separator $\mathbf{i}$ are defined as follows.

DEFINITION 2.2. *(Neighbor separators) For the graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *and the separator tree* $\mathbf{T}$ *of* A*, a separator* $\mathbf{j}$ *is a neighbor of a separator* $\mathbf{i}$ *if*

1. $\mathbf{j}$ *is at the same level as* $\mathbf{i}$ *or at an upper level in* $\mathbf{T}$*.*
2. *There exists some edge in* $\mathbf{E}$ *connecting nodes in* $\mathbf{t_i}$ *and* $\mathbf{t_j}$*.*

In our structured factorizations, we need to partition some upper level separators into pieces that are used to decide the HSS partition of the intermediate dense matrices. Denote the set of all neighbors of $\mathbf{i}$ (including $\mathbf{i}$ itself) by $\mathcal{N}_\mathbf{i}$ and the set of all ascendants of $\mathbf{i}$ in $\mathbf{T}$ by $\mathcal{P}_\mathbf{i}$. Also let $\widehat{\mathbf{t}}_\mathbf{j}^\mathbf{i}$ be a subset of $\mathbf{t_j}$ containing all vertices in $\mathbf{t_j}$ connected to $\mathbf{i}$.

Define

$$\mathbf{S_i} = \bigcup_{\mathbf{j} \in \mathcal{N}_\mathbf{i}} \widehat{\mathbf{t}}_\mathbf{j}^\mathbf{i},$$

and assume the nodes in $\mathbf{S_i}$ are ordered. We call $\widehat{\mathbf{t}}_\mathbf{j}^\mathbf{i}$ a *separator piece* in $\mathbf{t_j}$ and also $\mathbf{S_i}$. The purpose of this step is to decide $\mathbf{S_i}$. Here, we impose the following rules.

DEFINITION 2.3. *(Separator pieces ordering rules) We say* $\mathbf{S_i}$ *is properly ordered if:*

1. *Each piece of a separator contains vertices that are closely connected.*
2. *The pieces of a separator that are close to each other are also close in the ordering.*
3. *Additional levels of partition are applied to the separators for high dimensional problems [6].*

These rules have been shown to be useful in avoiding unnecessarily increasing the intermediate off-diagonal numerical ranks [27]. That is, it is desired to maintain the connectivity of the separators as much as possible in the partitioning of the separators.

Such partition strategy is utilized as follows to find a proper ordering for $\mathbf{S_i}$. Assume $\mathbf{l_s}$ is the starting level in the bottom-up traversal of $\mathbf{T}$ where structured factorizations are used. $\mathbf{l_s}$ is called a *switching level* [27].

If $\mathbf{i}$ is at level $\mathbf{l_s}$, traverse the nodes in $\mathcal{P}_\mathbf{i}$ to find $\mathbf{j} \in \mathcal{N}_\mathbf{i}$ and $\widehat{\mathbf{t}}_\mathbf{j}^\mathbf{i}$. Order all $\widehat{\mathbf{t}}_\mathbf{j}^\mathbf{i}$ following the above rules. Each $\widehat{\mathbf{t}}_\mathbf{j}^\mathbf{i}$ corresponds to a desired separator piece.

If $\mathbf{i}$ is a non-leaf separator, use the partition information $\mathbf{S_{c_1}}$ and $\mathbf{S_{c_1}}$ associated with the children $\mathbf{c}_1$ and $\mathbf{c}_2$ of $\mathbf{i}$, respectively. The procedure involves these steps:

1. Remove vertices in $\mathbf{S_{c_1}} \backslash \mathbf{t_{c_1}}$ and $\mathbf{S_{c_2}} \backslash \mathbf{t_{c_2}}$ which are not connected to $\mathbf{i}$.
2. Find vertices in $\mathbf{t_j}$ for $\mathbf{j} \in \mathcal{N}_\mathbf{i}$ which are not in $\mathbf{S_{c_1}} \backslash \mathbf{t_{c_1}}$ or $\mathbf{S_{c_2}} \backslash \mathbf{t_{c_2}}$.
3. Merge the results.

Notice that we can use this method to handle general separator orientations and connectivity. Figure 2 shows some separators in Figure 1. These separators have arbitrary shapes, and a separator may consist of multiple disjoint pieces.

With the partition information from $\mathbf{S_i}$, we also compute the HSS tree for the local matrix associated with the matrix.

3. Structured sparse LDL factorization. The method in [27] can be modified to compute an approximate LDL factorization

$$A \approx \mathbf{L}\mathbf{\Lambda}\mathbf{L}^T. \tag{3.1}$$

Here we use the supernodal version of the multifrontal method following the traversal of the separator or assembly tree $\mathbf{T}$ obtained above.

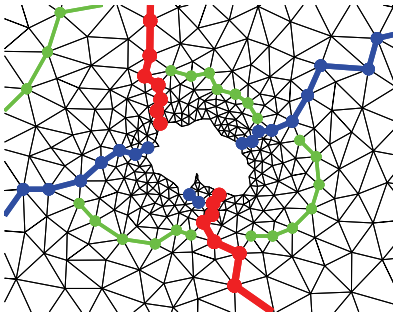


FIG. 2. *Zoomed figure for Figure 1.*

3.1. HSS LDL factorization. We first briefly describe an HSS LDL factorization scheme which is needed in the intermediate steps of the multifrontal factorizations. Given a symmetric HSS matrix F with generators D_i, U_i, R_i, B_i and the corresponding HSS tree T, the structured Cholesky factorization scheme in [5, 29] can be modified to compute an HSS LDL factorization.

Consider the nodes i of T. If i is a leaf, introduce zeros into F_i^- by setting

$$Q_i^T U_i = \begin{pmatrix} 0 \\ \widetilde{U}_i \end{pmatrix}, \tag{3.2}$$

where Q_i is obtained by a QL factorization of U_i. Then compute an LDL^T factorization of $\widehat{D_i} \equiv Q_i^T D_i Q_i$:

$$\widehat{D_i} = \begin{pmatrix} L_i & \\ X_i & I \end{pmatrix} \begin{pmatrix} \Lambda_i & \\ & \widetilde{D}_i \end{pmatrix} \begin{pmatrix} L_i^T & X_i^T \\ & I \end{pmatrix}.$$

If i is a non-leaf node, remove its children c_1 and c_2 from T, so that i becomes a leaf. Redefine

$$D_i = \begin{pmatrix} \widetilde{D}_{c_1} & \widetilde{U}_{c_1} B_{c_1} \widetilde{U}_{c_2}^T \\ \widetilde{U}_{c_2} B_{c_1}^T \widetilde{U}_{c_1}^T & \widetilde{D}_{c_2} \end{pmatrix}, U_i = \begin{pmatrix} \widetilde{U}_{c_1} R_{c_1} \\ \widetilde{U}_{c_2} R_{c_2} \end{pmatrix}.$$

Then the process repeats for i until reaching the root node. For the root node D_i, apply an LDL^T factorization for the entire matrix. At the end, a structured LDL type factorization is computed:

$$F = L \Lambda L^T \tag{3.3}$$

3.2. Structured multifrontal LDL factorization. The multifrontal method can be easily modified to compute an LDL factorization. We use the structured multifrontal methods in [24, 25, 27] to compute an approximate LDL factorization (3.1). We describe the main steps.

In the multifrontal factorization of A, two main types of intermediate dense matrices are involved, called frontal matrices and update matrices. They are formed following the flow of the factorization.

If a node $\mathbf{i}$ of $\mathcal{T}$ is a leaf, define a frontal matrix

$$\mathbf{F_i} \equiv \mathbf{F_i^0} = \begin{pmatrix} A|_{\mathbf{t_i} \times \mathbf{t_i}} & (A|_{\mathbf{S_i} \times \mathbf{t_i}})^T \\ A|_{\mathbf{S_i} \times \mathbf{t_i}} & 0 \end{pmatrix}. \tag{3.4}$$

For convenience, rewrite $\mathbf{F_i}$ as

$$\mathbf{F_i} = \begin{pmatrix} \mathbf{F_{ii}} & \mathbf{F}_{\mathbf{S_i},\mathbf{t_i}}^T \\ \mathbf{F}_{\mathbf{S_i},\mathbf{t_i}} & \mathbf{F}_{\mathbf{S_i},\mathbf{S_i}} \end{pmatrix}, \tag{3.5}$$

where the partition follows that in (3.4). Partially factorize $\mathbf{F_i}$ as

$$\mathbf{F_i} = \begin{pmatrix} \mathbf{L_i} & \\ \mathbf{X_i} & I \end{pmatrix} \begin{pmatrix} \mathbf{\Lambda}_i & \\ & \mathbf{U_i} \end{pmatrix} \begin{pmatrix} \mathbf{L_i}^T & \mathbf{X_i}^T \\ & I \end{pmatrix}, \tag{3.6}$$

where $\mathbf{U_i}$ is the update matrix associated with step $\mathbf{i}$:

$$\mathbf{U_i} = \mathbf{F}_{\mathbf{S_i},\mathbf{S_i}} - \mathbf{F}_{\mathbf{S_i},\mathbf{t_i}} \mathbf{F}_{\mathbf{ii}}^{-1} \mathbf{F}_{\mathbf{S_i},\mathbf{t_i}}^T. \tag{3.7}$$

If $\mathbf{i}$ is a non-leaf node with children $\mathbf{c}_1$ and $\mathbf{c}_2$, then the frontal matrix $\mathbf{F_i}$ is formed by an assembly operation called extend-add operation:

$$\mathbf{F_i} = \mathbf{F_i^0} \boxplus \mathbf{U}_{\mathbf{c}_1} \boxplus \mathbf{U}_{\mathbf{c}_2}, \tag{3.8}$$

where $\mathbf{F_i^0}$ is given in (3.4). This operation aligns the indices of the matrices on the right-hand side of (3.8) following the corresponding indices of the vertices. Then we can partition $\mathbf{F_i}$ as in (3.5) and repeat (3.6).

In the structured factorization, $\mathbf{F_i}$ is an HSS matrix. (This HSS matrix is either constructed from a dense $\mathbf{F_i}$ or is accumulated bottom-up as in [27].) The HSS tree T of $\mathbf{F_i}$ has $q+1$ nodes, and the root $q+1$ has left child k and right child q, so that $T[k]$ and $T[q]$ are the HSS trees for $\mathbf{F_{ii}}$ and $\mathbf{F}_{\mathbf{S_i},\mathbf{S_i}}$, respectively.

The computation of $\mathbf{L_i \Lambda}_i \mathbf{L_i}^T$ is then replaced by (3.6). Similarly, $\mathbf{X_i}$ is a low-rank form obtained by updating the block $\mathbf{F}_{\mathbf{S_i},\mathbf{t_i}}$ of the HSS form of $\mathbf{F_i}$. See [26]. The computation of $\mathbf{U_i}$ can also be quickly done as a low-rank update or an HSS update [26]. For example, it is shown in [26] that

$$\mathbf{U_i} \approx F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} - \Theta_k \widetilde{D}_k^{-1} \Theta_k^T,$$

where $\Theta_k = U_{k+1} B_k^T \widetilde{U}_k^T$ and $\widetilde{U}_k$ is given in (3.2) when the node k is reached.

As shown in [26], under certain conditions, the structured LDL factorization can be computed in about $O(n)$ flops in 2D, and from $O(n)$ to $O\left(n^{4/3}\right)$ in 3D, while the exact factorization costs in 2D and 3D are at least $O\left(n^{1.5}\right)$ and $O\left(n^2\right)$, respectively [13].

4. Structured sparse eigenvalue solution.

4.1. Overall structure. Our sparse eigensolver is based on the bisection method together with inertia estimations. Introductions of such an idea can be founds in [9, 11], which also give a definition of the inertia as follows.

DEFINITION 4.1. *The inertia of a symmetric matrix A is the triple of integers*

$$\text{Inertia}(A) \equiv (n_-, n_0, n_+),$$

where n_-, n_0, and n_+ are numbers of positive, zero and negative eigenvalues of A, respectively.

A basic result we use is Sylvester's Inertia Theorem [21].

THEOREM 4.2. *(Sylvester's inertia theorem) For a symmetric matrix A, the following result holds for any invertible matrix L:*

$$\text{Inertia}(A) = \text{Inertia}(L^T A L).$$

Clearly, once we have the LDL factorization (3.1), we can get Inertia(A) by counting the positive, zero, and negative diagonal entries of $\mathbf{\Lambda}$. Based on the bisection algorithm in [9], we show the overall structure of our method in Algorithm 1, where $n_-(A)$ represents the number of negative eigenvalues of A.

Algorithm 1 Structured bisection for finding all the eigenvalues of a symmetric sparse matrix A inside $[a, b)$

procedure BISECTION(a, b, τ)
  ▷ τ may be decided based on the number of eigenvalues desired (Theorem 4.3)
  $A - aI \approx \mathbf{L}_a\mathbf{\Lambda}_a\mathbf{L}_a^T$, $A - bI \approx \mathbf{L}_b\mathbf{\Lambda}_b\mathbf{L}_b^T$ ▷ Compute structured LDL factorizations
  $n_a = n_-(\mathbf{\Lambda}_a)$, $n_b = n_-(\mathbf{\Lambda}_b)$
  if $n_a = n_b$ **then**
    Stop ▷ There is no eigenvalue in $[a, b)$
  end if
  Push $[a, n_a, b, n_b]$ onto a stack S
  while $S \neq \emptyset$ **do**
    Pop $\left[\widetilde{a}, n_{\widetilde{a}}, \widetilde{b}, n_{\widetilde{b}}\right]$ from the stack S
    if $\widetilde{b} - \widetilde{a} < \tau$ **then** ▷ τ: error tolerance
      There are $n_{\widetilde{b}} - n_{\widetilde{a}}$ eigenvalues in $[\widetilde{a}, \widetilde{b})$
    else ▷ Recursion for subintervals
      $c = \frac{\widetilde{a}+\widetilde{b}}{2}$
      bisect$(\widetilde{a}, c, \tau)$
      bisect$(c, \widetilde{b}, \tau)$
    end if
  end while
end procedure

If the eigenvectors are also needed, we can use the computed eigenvalues as the shifts and apply inverse iterations to find the eigenvectors. In general, the convergence is fast.

4.2. Fast inertia computation and aggressive low-rank truncation. To improve the efficiency of the eigensolution, we use an idea of aggressive low-rank truncation. This is especially useful for problems where the number of eigenvalues desired is small. This idea is first studied in [28] for a dense symmetric positive definite (SPD) matrix F.

THEOREM 4.3. *[28] Assume an SPD matrix F is partitioned into a block 2×2 form as*

$$F = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}.$$

Let $F_{11} = L_1L_1^T$ and $F_{22} = L_2L_2^T$ be the Cholesky factorizations of the diagonal blocks. Assume r largest eigenvalues of F is desired. Approximate F by an approximate factorization

$$F \approx \widetilde{F} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} I & U_1B_1U_2^T \\ U_2B_1^TU_1^T & F_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ & L_{22}^T \end{pmatrix}.$$

Then

$$\text{Inertia}\,(F) = \text{Inertia}\left(\widetilde{F}\right).$$

This theorem indicates that, if only few eigenvalues of F are desired, then it is possible to choose a small numerical rank when truncating appropriate off-diagonal blocks. For the study of general symmetric cases, see [28].

Here, this idea can be roughly applied to the frontal matrices in the structured multifrontal method. Overall, we may manually set a numerical rank which is much smaller than the maximum of the intermediate off-diagonal numerical ranks for a small tolerance τ.

The eigenvector computation may need a smaller τ. However, the number of iterations is usually small.

5. Numerical experiments. We have implemented our algorithm in Matlab. We choose approximate levels of multifrontal factorizations. The computations are done on casey.math.purdue.edu with 32GB DDR3 RAM.

EXAMPLE 1. We first test the efficiency and accuracy of our algorithm with the 2D SEAM (SEG Advanced Modeling) velocity model.

The size of the discretized matrix ranges from 4095 to 259, 852. We compute 3 eigenvalues in a given interval for each matrix size. The storage, number of iterations, and maximum error for the eigenvalues are shown in Table 5.1. The code is in Matlab and the timing is not shown. It is clear that, when the matrix size n quadruples, the storage roughly quadruples. This indicates the benefit of compression in the factorization. We also use a relatively large tolerance for the bisection so as to improve the efficiency. Then we use few steps of inverse iterations to further improve the accuracy of the eigenvalues.

n	Level	Storage	Iteration	Inv iteration	Interval	e
4095	8	$2.92E4$	7	3	$[2.9166, 2.9175]$	$4.99E-7$
16,380	10	$2.06E6$	4	5	$[2.9166, 2.9180]$	$1.73E-13$
64,963	12	$9.57E6$	5	5	$[2.9166, 2.9170]$	$2.63E-9$
259,852	14	$4.13E7$	7	5	$[2.9168, 2.9169]$	$4.01E-8$

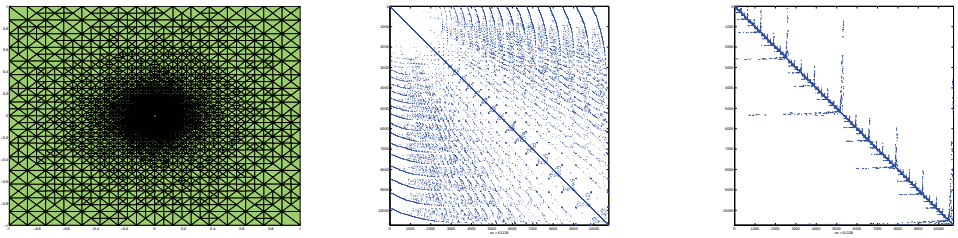
TABLE 5.1

Performance of bisection for finding the eigenvalues of the 2D SEAM model, including the total number of levels in nested dissection, the storage (number of nonzeros in the factors), the number of bisection iterations, the number of inverse iterations, the interval for the eigenvalues, and the maximum error $e = \|Ax - \lambda x\|_2$ for the eigenvalues, where the compression tolerance is $\tau = 10^{-7}$ and the inverse iteration tolerance is 10^{-12}.

EXAMPLE 2. Next, we show the performance of the algorithm for different compression tolerances. Two problems are considered, one originates from a diffusion problem in 2D (denoted `2Dd`), and the other arises from the simulation of the electromagnetic wave propagating across some non-homogeneous media in 3D (denoted `3De`). See Figures 3 and 4.

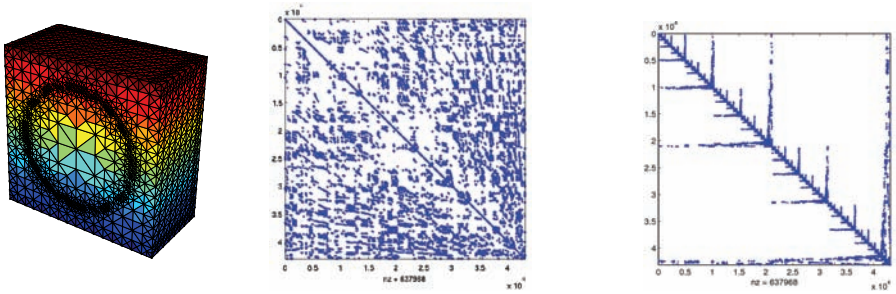
First, we test the convergence of the bisection algorithm for different tolerances. For `2Dd`, we look for the eigenvalues less than 100. For tolerances τ vary from 10^{-9} to 10^{-1}, our algorithm find 5023 eigenvalues, which is equal to the exact number. Similarly, for `3De`, we find the correct number of eigenvalues less than 3 to be 1327.

Then we use our algorithm to compute some interior eigenpairs for the two matrices. Table 5.2 shows the computational cost and errors for the computed eigenpairs. We only compute the eigenvalues within small intervals After finding the approximate eigenvalues with bisection, we use 3 steps of inverse iteration to find the eigenvectors and to further improve the accuracies of the



(i) Mesh (ii) Matrix nonzero pattern (iii) After nested dissection

FIG. 3. *A matrix (2Dd) from a 2D diffusion problem.*



(i) Mesh (ii) Matrix nonzero pattern (iii) After nested dissection

FIG. 4. *A matrix (3De) from a 3D electromagnetic wave propagation problem.*

eigenvalues.

Problem	Interval	Eigenvalues	Iterations	e
2Dd	$[188, 189]$	λ_{5379}	14	$3.16e-9$
		λ_{5380}	14	$3.16e-9$
		λ_{5379}	14	$3.16e-9$
		λ_{5381}	14	$3.16e-9$
		λ_{5382}	14	$3.16e-9$
		λ_{5383}	14	$3.16e-9$
3De	$[256, 257]$	λ_{42987}	14	$2.06e-9$

TABLE 5.2

Performance of bisection for finding the eigenvalues and eigenvectors of 2Dd and 3De, where $e = \|Ax - \lambda x\|_2$ *and the compression tolerance* $\tau = 10^{-4}$.

6. Conclusions. In this work, we propose a fast sparse eigensolver based on a structured multifrontal method. Strategies for handling general meshes are shown, so as to perform nested dissection and decide separator connectivity that are suitable for our factorization method. This method provides an approximate LDL factorization, which is used to decide the inertia of the matrix. Related work show that only modest accuracies are needed to accurately estimate the inertia. Then we combine the inertia estimation with bisection to find the eigenvalues. This eigensolver is especially useful for finding selected eigenvalues for a specified interval. Numerical experiments for several important PDEs are included to illustrate the efficiency and accuracy. Our future work includes

further analysis of the dependence of the accuracies on the tolerances, the implementation of fast codes, and the application to various practical problems.

REFERENCES

[1] S. Börm, *Efficient Numerical Methods for Non-local Operators*, European Mathematical Society, 2010.

[2] S. Börm and W. Hackbusch, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[3] P. R. Amestoy, T.A. Davis, and I.S. Du, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix. Anal. Appl, 17(1996), pp. 886–905.

[4] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2006), pp. 67–81.

[5] S. Chandrasekaran, P. Dewilde, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[6] S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam, *On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2261–2290.

[7] T. A. Davis, Y. Hu, University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/index.html.

[8] I. S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Bans. Math. Software, 9 (1983), pp. 302–325.

[9] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[10] J. A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[11] G. H. Golub and C. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[12] W. Hackbusch, *A Sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[13] A. J. Hoffman, M. S. Martin, and D. J. Rose, *Complexity bounds for regular finite difference and finite element grids*, SIAM J. Numer. Anal., 10 (1973), pp. 364–369.

[14] Karypis, G, *METIS: family of multilevel partitioning algorithms* (1998). http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[15] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[16] R. J. Lipton and R. E. Tarjan, *A seperator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[17] J. W. H. Liu, *Modification of the Minimum-Degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11(1985), 141–153.

[18] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.

[19] W. Lyons, *Fast Algorithms with Applications to PDEs*, PhD thesis, University of California, Santa Barbara, June. 2005.

[20] G. L.MIller, S.–H. Teng, W. Thurston, and S. A. Vavasis, *Geometric separators for finite element meshes*, SIAM J. Sci. Comput., 19, (1998), pp. 364–386.

[21] Sylvester, J. J. "A demonstration of the theorem that every homogeneous quadratic polynomial is reducible by real orthogonal substitutions to the form of a sum of positive and negative squares". Philosophical Magazine (Ser. 4) 4 (1852): 138–142. doi:10.1080/14786445208647087.

[22] S.–H. Teng, *Fast nested dissection for finite element meshes*, SIAM J. Matrix. Anal. Appl., 18, (1997), pp. 552–565.

[23] W.F. Tinney and J.W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. IEEE, 55(1967), pp. 1801–1809.

[24] S. Wang, M.V. de Hoop, and J. Xia, Acoustic inverse scattering via Helmholtz operator factorization and optimization, J. Comput. Phys., 229 (2010), pp. 8445–8462.

[25] S. Wang, M.V. de Hoop, and J. Xia, On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver, Geophys. Prospect., 59 (2011), pp. 857–873.

[26] J. Xia, Efficient structured multifrontal factorization for general large sparse matrices, submitted to SIAM J. Sci. Comput., 2012, http://www.math.purdue.edu/~xiaj/work/mfhss.pdf.

[27] J. Xia, S. Chandrasekaran, M. Gu, X.S. Li, Superfast multifrontal method for large structured linear systems

of equations, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.

[28] J. Xia, Y. Xi, Superfast structured solvers for Toeplitz eigenvalue problems via randomized sampling, SIAM J. Matrix Anal. Appl., to be submitted, 2012.

[29] J. Xia, S.Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Alegera Appl., 17(2010), pp. 953–976.