An Implementation and Evaluation of the AMLS Method for Sparse Eigenvalue Problems

WEIGUO GAO
XIAOYE S. LI
CHAO YANG
ZHAOJUN BAI

We describe an efficient implementation and present a performance study of an Algebraic Multi-Level Sub-structuring (AMLS) method for sparse eigenvalue problems. We assess the time and memory requirements associated with the key steps of the algorithm, and compare it with the shift-and-invert Lanczos algorithm. Our eigenvalue problems come from two very different application areas: the accelerator cavity design and the normal mode vibrational analysis of the polyethylene particles. We show that the AMLS method, when implemented carefully, outperforms the traditional method in broad application areas, when large number of eigenvalues are sought, with relatively low accuracy.

Categories and Subject Descriptors: G.1.3 [**Mathematics of Computing**]: Numerical Linear Algebra—*Eigenvalue and eigenvectors (direct and iterative methods)*

General Terms: Algorithms;Performance

Additional Key Words and Phrases: sparse eigenvalue problems, multi-level sub-structuring, performance evaluation

1. INTRODUCTION

The automated multi-level sub-structuring (AMLS) method [Bennighof 1993; Kaplan 2001; Bennighof and Lehoucq 2004] is a multi-level extension of a simple sub-structuring method called *component mode synthesis* (CMS) [Hurty 1960; Craig and Bampton 1968] originally developed in the 1960s for solving eigenvalue problems arising from structural engineering analysis. The method has recently been

This work was supported by the Director, Office of Advanced Scientific Computing Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098. The work of the first author was partly supported by the Special Funds for Major State Basic Research Projects (2005CB321700) and the National Science Foundation of China under grant number 10571031. Weiguo Gao, wggao@fudan.edu.cn, School of Mathematical Sciences, Fudan University, Shanghai 200433, China. Most of the work was performed while this author was visiting Lawrence Berkeley National Laboratory. Xiaoye S. Li, xsli@lbl.gov, Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720. Chao Yang, cyang@lbl.gov, Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720. Zhaojun Bai, bai@cs.ucdavis.edu, Department of Computer Science, University of California, Davis, CA 95616.

shown to be efficient for performing vibration and acoustic analysis of large scale finite element models of automobile bodies [Kaplan 2001] on a workstation. In particular, it was shown in [Kropp and Heiserer 2003] that AMLS is several times faster than the shift-invert Lanczos (SIL) algorithm commonly used in structural engineering.

The efficiency exhibited by AMLS can be attributed to two main factors: 1) The method, which can be characterized as a projection method, does not explicitly construct or maintain an orthornormal basis of the subspace into which the original eigenvalue problem is projected. This feature makes AMLS extremely appealing when the number of desired eigenvalues is large. 2) The method does not perform a sequence of triangular substitutions required in SIL. Thus, it is suitable for an out-of-core implementation that requires a limited amount of input and output (I/O) traffic.

It is important to recognize that AMLS usually produces approximate solutions that are less accurate than those computed by SIL. Although the lack of accuracy can be tolerated in some applications such as frequency response analysis, it may be a cause for concern in others. There are a number of ways to improve the accuracy of the AMLS algorithm [Bekas and Saad 2005]. However, these schemes are not immediately applicable to the multi-level case, and they tend to increase the runtime of the computation significantly. Therefore, when we measure the performance of AMLS and compare it with other techniques, we should keep the accuracy issue in perspective.

As we will see in Section 2, the basic AMLS algorithm is conceptually easy to describe. However, an efficient implementation of the algorithm would require matrix transformations to be carefully organized to minimize the number of floating point operations and the amount of memory usage. A number of decisions have to be made at different stages of the algorithm to balance the cost and accuracy of the computation. These decisions will obviously affect the outcome and performance of the calculation.

In this paper, we examine a number of practical issues arising from the implementation of the AMLS algorithm. We will evaluate the performance of AMLS through two examples that are not related to structural analysis. Our performance evaluation is based on runtime, memory usage, and accuracy. To the best of our knowledge, this type of empirical analysis of the AMLS method has not been performed outside of structural analysis applications.

We show that AMLS is more efficient than the traditional shift-and-invert Lanczos method when a large number of eigenvalues are wanted and when the desired accuracy is limited to a few digits. Instead of performing the matrix transformations required in AMLS out-of-core, we developed an in-core implementation that requires significantly less memory than a straightforward implementation. The key idea behind such an implementation is to recompute some of the intermediate matrix blocks instead of storing them (either in memory or on a disk). We will show that such an implementation of AMLS results in up to 50% of memory saving while incurring less than 15% increase in runtime.

Our paper is organized as follows. In Section 2, we give a brief overview on the algorithmic ingredients of the AMLS method. We also discuss the accuracy of the algorithm and examine how the selection of sub-structure eigenvectors (modes) affects the quality of the approximate eigenpairs. The implication of mode selection on the implementation will also be discussed. In Section 3, we examine the implementation details. In particular, we illustrate how the matrix operations in AMLS can be organized to reduce both floating point operations and memory usage. Our discussion makes use of a graph-theoretical tool called *seperator tree* that allows one to easily track the data flow and task dependency in the sub-structuring computation. We also present a memory saving technique that requires some intermediate results to be recomputed instead of stored. In Section 4, we evaluate the performance of AMLS on two problems arising from two different application domains. Concluding remarks are in Section 5.

2. THE AMLS METHOD

Before we get into the nitty gritty details of the implementation issues, it is instructive to review the basic idea behind the AMLS algorithm for solving the following algebraic eigenvalue problem

$$Kx = \lambda Mx, \tag{1}$$

where K is symmetric and M is symmetric positive definite, and both are sparse and of dimension n-by-n. They may or may not have the same sparsity pattern. Our description of the method does not make use of any information regarding the geometry or the physical structure on which the original problem is defined. Therefore, it is purely algebraic.

2.1 A single-level algorithm

Let the rows and columns of K and M be permuted so that these matrices can be partitioned as

$$K = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} n_1 & n_2 & n_3 \end{array} \\ \begin{pmatrix} K_{11} & & K_{13} \\ & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix} \end{array} \quad \text{and} \quad M = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} n_1 & n_2 & n_3 \end{array} \\ \begin{pmatrix} M_{11} & & M_{13} \\ & M_{22} & M_{23} \\ M_{13}^T & M_{23}^T & M_{33} \end{pmatrix} \end{array}, \tag{2}$$

where the labels n_1, n_2 and n_3 indicate the dimensions of the submatrix blocks, and $n_1 + n_2 + n_3 = n$. This permutation can be obtained by applying a matrix ordering and partitioning algorithm such as the nested dissection algorithm [George 1973] to the structure of the matrix $|K| + |M|$ (ignoring numerical cancellation).

The pencils (K_{11}, M_{11}) and (K_{22}, M_{22}) now define two algebraic sub-structures that are connected by the third block rows and columns of K and M which is referred to as the *interface* block. We assume that n_3 is much smaller than n_1 and n_2.

In a single-level algebraic sub-structuring algorithm, we proceed by performing a block factorization

$$K = LDL^T, \tag{3}$$

where

$$L = \begin{pmatrix} I_{n_1} & & \\ & I_{n_2} & \\ K_{13}^T K_{11}^{-1} & K_{23}^T K_{22}^{-1} & I_{n_3} \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} K_{11} & & \\ & K_{22} & \\ & & \widehat{K}_{33} \end{pmatrix}.$$

The last diagonal block of D, often known as the *Schur complement*, is defined by

$$\widehat{K}_{33} = K_{33} - K_{13}^T K_{11}^{-1} K_{13} - K_{23}^T K_{22}^{-1} K_{23}.$$

The inverse of the lower triangular factor L defines a congruence transformation that, when applied to the matrix pencil (K, M), yields a new matrix pencil $(\widehat{K}, \widehat{M})$:

$$\widehat{K} = L^{-1} K L^{-T} = D \quad \text{and} \quad \widehat{M} = L^{-1} M L^{-T} = \begin{pmatrix} M_{11} & & \widehat{M}_{13} \\ & M_{22} & \widehat{M}_{23} \\ \widehat{M}_{13}^T & \widehat{M}_{23}^T & \widehat{M}_{33} \end{pmatrix}. \tag{4}$$

The off-diagonal blocks of $\widehat{M}$ satisfy

$$\widehat{M}_{i3} = M_{i3} - M_{ii} K_{ii}^{-1} K_{i3}, \quad \text{for} \quad i = 1, 2.$$

The last diagonal block of $\widehat{M}$ satisfies

$$\widehat{M}_{33} = M_{33} - \sum_{i=1}^{2} (K_{i3}^T K_{ii}^{-1} M_{i3} + M_{i3}^T K_{ii}^{-1} K_{i3} - K_{i3}^T K_{ii}^{-1} M_{ii} K_{ii}^{-1} K_{i3}).$$

The pencil $(\widehat{K}, \widehat{M})$ is often known as the *Craig-Bampton* form [Craig and Bampton 1968] in structural engineering. Note that the eigenvalues of $(\widehat{K}, \widehat{M})$ are identical to those of (K, M), and the corresponding eigenvectors $\widehat{x}$ are related to the eigenvectors of the original problem (1) through $\widehat{x} = L^T x$.

The sub-structuring algorithm constructs a subspace in the form of

$$S = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \end{array} \begin{array}{c} \begin{array}{ccc} k_1 & k_2 & k_3 \end{array} \\ \begin{pmatrix} S_1 & & \\ & S_2 & \\ & & S_3 \end{pmatrix} \end{array} \tag{5}$$

where S_1, S_2 and S_3 consist of k_1, k_2 and k_3 selected eigenvectors of (K_{11}, M_{11}), (K_{22}, M_{22}) and $(\widehat{K}_{33}, \widehat{M}_{33})$ respectively. The eigenvectors associated with (K_{11}, M_{11}) and (K_{22}, M_{22}) will be referred to as the *sub-structure modes*, and those associated with $(\widehat{K}_{33}, \widehat{M}_{33})$ will be referred to as the *coupling modes*. The approximations to the desired eigenpairs of the pencil $(\widehat{K}, \widehat{M})$ are obtained by projecting the pencil $(\widehat{K}, \widehat{M})$ onto the subspace spanned by S, i.e., we seek θ and $q \in \mathbb{R}^{k_1+k_2+k_3}$ such that

$$(S^T \widehat{K} S) q = \theta (S^T \widehat{M} S) q. \tag{6}$$

It follows from the standard Rayleigh-Ritz theory [Parlett 1998, page 236] that θ is the best approximation to an eigenvalue of (K, M) from span$\{S\}$, and the vector formed by $z = L^{-T} S q$ gives an approximation to the corresponding eigenvector.

We make the following remarks. 1) The algorithm is most efficient when k_i can be chosen to be much smaller than n_i. In this case, S_i can be computed by a shift-and-invert Lanczos procedure. The cost of this computation is generally small compared to the rest of the computation, especially when this algorithm is extended to a multi-level scheme. 2) In some applications, n_3 can be much smaller than both n_1 and n_2. In this case, one can retain all the coupling modes by simply setting $k_3 = n_3$ and replacing S_3 with I_{n_3}.

Decisions must be made on how to select eigenvectors from each sub-structure. The selection should be made in such a way that the dimension of the subspace spanned by columns of S is small while a sufficient amount of spectral information of (K, M) is retained. The process of choosing appropriate eigenvectors from each sub-structure is referred to as *mode selection* [Yang et al. 2005]. A few strategies have been proposed in [Elssel and Voss 2004] on how to select eigenvectors from each sub-structure. However, all of these stategies, which are heuritics derived from the the accuracy analysis of the sub-structuring algorithm, tend to be somewhat conservative, i.e. they tend to select more eigenvectors than is necessary. More research is required to develop better strategies to optimize the performance of the sub-structuring algorithm.

2.2 A multi-level extension

The single-level algebraic sub-structuring algorithm can be extended to a multi-level algorithm in a natural way. The matrix reordering and partitioning scheme used to create the block structure (2) can be applied recursively to (K_{11}, M_{11}) and (K_{22}, M_{22}) respectively to produce a multi-level division of (K, M) into smaller submatrices. We will use a two-level example below to illustrate the major steps of the multi-level algorithm.

$$K = \begin{pmatrix} K_{11} & & & & & & \\ & K_{22} & & & & sym. & \\ K_{31} & K_{32} & K_{33} & & & & \\ & & & K_{44} & & & \\ & & & & K_{55} & & \\ & & & K_{64} & K_{65} & K_{66} & \\ K_{71} & K_{72} & K_{73} & K_{74} & K_{75} & K_{76} & K_{77} \end{pmatrix}, \; M = \begin{pmatrix} M_{11} & & & & & & \\ & M_{22} & & & & sym. & \\ M_{31} & M_{32} & M_{33} & & & & \\ & & & M_{44} & & & \\ & & & & M_{55} & & \\ & & & M_{64} & M_{65} & M_{66} & \\ M_{71} & M_{72} & M_{73} & M_{74} & M_{75} & M_{76} & M_{77} \end{pmatrix} \tag{7}$$

Because of symmetry, we store only the non-zero matrix elements in the lower triangular portions of K and M. This multi-level nested structure can be obtained by a graph partitioning software package such as METIS [Karypis and Kumar 1998] which algebraically identifies the separators recursively. In addition to providing an ordering scheme, METIS also returns a *separator tree*, a hierarchical graph in which each node is mapped to a set of columns (or rows) in K and M. In a separator tree such as the one shown in Figure 1(a), the top level node (the rectangular box labeled with 15) corresponds to the first-level separator that partitions K and M into two sub-structures. In Figure 1(b), this separator is shown as shaded rows and columns that intersect at the lower right corner of the matrix marked by 15. Each sub-structure can be further partitioned into two smaller sub-structures. The separators associated with these sub-partitions (marked by 7 and 14 in Figure 1(a) and Figure 1(b)) are connected to the first level separator in Figure 1(a) so that the hierarchical structure of the partition can be identified easily. A sub-structure that is not further partitioned is represented by a *leaf* node that appears at the bottom of the separator tree. In addition to characterizing the nested structure of a matrix partition, the separator tree can also be used to succinctly describe the computational tasks and their dependencies in the AMLS algorithm as we will see below.

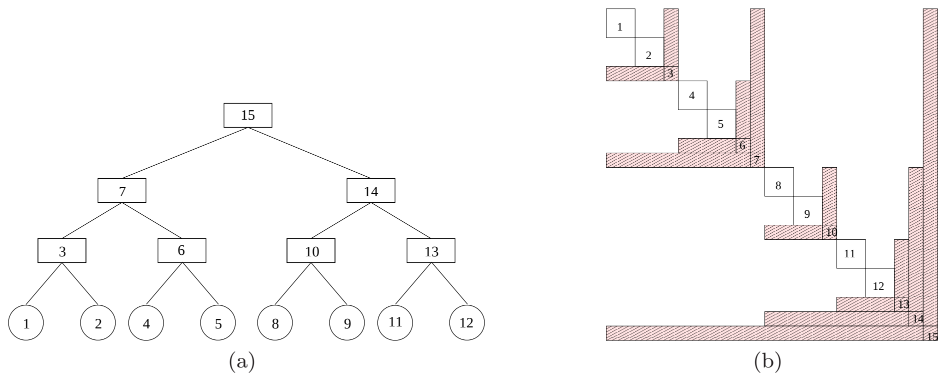


Fig. 1. The separator tree (a) and the reordered matrix (b) for a three-level dissection.

Once the matrix pencil (K, M) has been partitioned and reordered in the form of (7), we can perform Gaussian elimination on K to obtain $K = LDL^T$, where D is block diagonal as follows

$$D = L^{-1}KL^{-T} = diag(K_{11}, K_{22}, \widehat{K}_{33}, K_{44}, K_{55}, \widehat{K}_{66}, \widehat{K}_{77}) \overset{def}{=} \widehat{K} \ . \tag{8}$$

In this elimination process, only the diagonal blocks corresponding to the separators (i.e., K_{33}, K_{66}, and K_{77}) are modified. The diagonal blocks corresponding to the leaves are not altered. Applying the same congruence transform to M yields

$$\widehat{M} = \begin{pmatrix} M_{11} & & & & & & \\ & M_{22} & & & & sym. & \\ \widehat{M}_{31} & \widehat{M}_{32} & \widehat{M}_{33} & & & & \\ & & & M_{44} & & & \\ & & & & M_{55} & & \\ & & & \widehat{M}_{64} & \widehat{M}_{65} & \widehat{M}_{66} & \\ \widehat{M}_{71} & \widehat{M}_{72} & \widehat{M}_{73} & \widehat{M}_{74} & \widehat{M}_{75} & \widehat{M}_{76} & \widehat{M}_{77} \end{pmatrix} \tag{9}$$

Note that $\widehat{M}$ and M have the same block structure. Again, the diagonal blocks associated with the separator tree leaves are not altered, but all other blocks (including all the off-diagonal blocks) are modified. Moreover, the off-diagonal blocks of $\widehat{M}$ typically contain more non-zeros than the corresponding blocks in M.

The congruence tranformation applied to (K, M) is followed by several independent sub-structure calculations in which selected eigenpairs of the smaller pencils (K_{ii}, M_{ii}) are computed for $i = 1, 2, \ldots, N$, where N is the number of leaf nodes in the separator tree. Let S_i be the matrix that contains k_i computed eigenvectors of (K_{ii}, M_{ii}) (or $(\widehat{K}_{ii}, \widehat{M}_{ii})$) as its columns. These eigenvectors are assembled to form an n-by-p matrix

$$S = diag(S_1, S_2, \ldots, S_N) \ , \tag{10}$$

where $p = \sum_{i=1}^{N} k_i$. Approximate eigenpairs of (K, M) can then be obtained by applying the Rayleigh-Ritz procedure within the subspace *span*$\{S\}$. To summarize, we list the major steps of the AMLS algorithm in Algorithm 1.

```
ALGORITHM 1. Algebraic Multi-level Sub-structuring (AMLS)
Input:   A matrix pencil $(K, M)$, where $K$ is symmetric and nonsingular
         and $M$ is symmetric positive definite
Output:  $\theta_j \in R^1$ and $z_j \in R^n$, $(j = 1, 2, ..., k)$ such that $Kz_j \approx \theta_j M z_j$

(1)      Partition and reorder $K$ and $M$ to be in the form of (7)
(2)      Perform block factorization $K = LDL^T$
(3)      Apply the congruence transformation defined by $L^{-1}$ to $(K, M)$ to
         obtain $(\widehat{K}, \widehat{M})$ defined by (8) and (9)
(4)      Compute a subset of the eigenpairs of interest for the subproblems
         $(K_{ii}, M_{ii})$ (and $(\widehat{K}_{ii}, \widehat{M}_{ii})$). Then, form the matrix $S$ in (10)
(5)      Project the matrix pencil $(\widehat{K}, \widehat{M})$ onto the subspace span$\{S\}$
(6)      Compute $k$ desired eigenpairs $(\theta_j, q_j)$ from $(S^T \widehat{K} S)q = \theta(S^T \widehat{M} S)q$,
         and set $z_j = L^{-T} S q_j$ for $j = 1, 2, ..., k$
(7)      Apply the inverse of the permutation used in Step (1) to $z_j$ to
         restore the original order of the solution.
```

2.3 Accuracy

The accuracy of the approximate eigenpairs would depend on how S_1, S_2 and S_3 are constructed in (5). Intuitively, if one increases the dimension of the subspace *span*$\{S\}$ by computing more sub-structure eigenvectors (modes) and including them in S_i, the accuracy of the approximation can be improved. However, the analysis performed in [Yang et al. 2005] indicates that the amount of accuracy improvement may be negligible as a result of simply including more sub-structure modes in S_i while the cost of solving the projected eigenvalue problem

$$(S^T \widehat{K} S)q = \theta(S^T \widehat{M} S)q, \tag{11}$$

can become significantly higher.

Thus, before we begin to discuss the implementation of AMLS, it is worthwhile to take a closer look at the relationship between mode selection and the accuracy of the approximate eigenpairs returned by AMLS. The following dissusion is a summary of the analysis presented in [Yang et al. 2005].

To simplify the discussion, we will look at a single-level partitioning, and work with the matrix pencil $(\widehat{K}, \widehat{M})$, where $\widehat{K}$ and $\widehat{M}$ are defined in (4). As noted earlier, $(\widehat{K}, \widehat{M})$ and (K, M) have the same set of eigenvalues. If $\widehat{x}$ is an eigenvector of $(\widehat{K}, \widehat{M})$, then $x = L^{-T}\widehat{x}$ is an eigenvector of (K, M), where L is the transformation defined in (3).

Let $(\mu_j^{(i)}, v_j^{(i)})$ be the j-th eigenpair of the i-th subproblem $(i = 1, 2)$, and

$(\mu_j^{(3)}, v_j^{(3)})$ be the j-th eigenpair of the coupling subproblem, i.e.,

$$\begin{aligned} K_{ii} v_j^{(i)} &= \mu_j^{(i)} M_{ii} v_j^{(i)}, \text{ for } i = 1, 2 \\ \widehat{K}_{ii} v_j^{(3)} &= \mu_j^{(3)} \widehat{M}_{ii} v_j^{(3)}, \end{aligned}$$

where $(v_j^{(i)})^T M_{ii} v_k^{(i)} = \delta_{j,k}$, $(v_j^{(3)})^T \widehat{M}_{ii} v_k^{(3)} = \delta_{j,k}$, and

$$\delta_{j,k} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

Let $\mu_j^{(i)}$ be ordered such that

$$\mu_1^{(i)} \le \mu_2^{(i)} \le \cdots \le \mu_{n_i}^{(i)}, \tag{12}$$

then we can express $\widehat{x}$ as

$$\widehat{x} = \begin{pmatrix} V_1 & & \\ & V_2 & \\ & & V_3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \tag{13}$$

where $V_i = (v_1^{(i)} \; v_2^{(i)} \; ... \; v_{n_i}^{(i)}), i = 1, 2, 3$, and $y = (y_1^T, y_2^T, y_3^T)^T \neq 0$.

It is easy to verify that y satisfies the following *canonical* generalized eigenvalue problem

$$\begin{pmatrix} \Lambda_1 & & \\ & \Lambda_2 & \\ & & \Lambda_3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \lambda \begin{pmatrix} I_{n_1} & & G_{13} \\ & I_{n_2} & G_{23} \\ G_{13}^T & G_{23}^T & I_{n_3} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \tag{14}$$

where $\Lambda_i = \text{diag}(\mu_1^{(i)}, \mu_2^{(i)}, \ldots, \mu_{n_i}^{(i)})$ for $i = 1, 2, 3$, and $G_{i3} = V_i^T \widehat{M}_{i3} V_3$ for $i = 1, 2$. This pencil is clearly congruent to the pencils $(\widehat{K}, \widehat{M})$ and (K, M). Thus it shares the same set of eigenvalues with that of (K, M). If $\widehat{x}$ can be well approximated by a linear combination of the columns of S, then the vector y_i $(i = 1, 2, 3)$ must contain only a few entries with large magnitude. All the other components of y_i are likely to be small and negligible. Multiplying (14) from the left by $\text{diag}(\Lambda_1, \Lambda_2, \Lambda_3)^{-1}$ yields

$$\begin{aligned} y_i &= \lambda(\Lambda_i - \lambda I_{n_i})^{-1} G_{i3} y_3, \; i = 1, 2 && (15) \\ y_3 &= \lambda(\Lambda_3 - \lambda I_{n_3})^{-1} (G_{13}^T y_1 + G_{23}^T y_2) && (16) \end{aligned}$$

It follows that the j-th component of y_i is given by

$$|e_j^T y_i| = \rho_\lambda(\mu_j^{(i)}) g_j^{(i)}, \tag{17}$$

where,

$$\rho_\lambda(\omega) = |\lambda/(\omega - \lambda)|, \tag{18}$$

$g_j^{(i)} = |e_j^T G_{i3} y_3|$ for $i = 1, 2$, and $g_j^{(3)} = |e_j^T (G_{13}^T y_1 + G_{23}^T y_2)|$.

When $g_j^{(i)}$ can be bounded from above and below by a moderate constant, the magnitude of $|e_j^T y_i|$ is essentially determined by $\rho_\lambda(\mu_j^{(i)})$ which is called a *ρ-factor* in [Yang et al. 2005]. It is easy to see that $\rho_\lambda(\mu_j^{(i)})$ is large when $\mu_j^{(i)}$ is close to

λ, and it is small when $\mu_j^{(i)}$ is away from λ. For the smallest eigenvalue λ_1 of (K, M), it is easy to show that $\rho_{\lambda_1}(\mu_j^{(i)})$ is monotonically decreasing with respect to j. Thus, if λ_1 is the desired eigenvalue, one should naturally choose the matrix S_i in (5) to contain only the leading k_i columns of V_i, for some $k_i \ll n_i$. When more than one eigenpairs are needed, the selection criterion becomes slightly more complicated. It will involve examining the ρ-factors associated with the smallest and largest eigenvalues of interest.

It is shown in [Yang et al. 2005] that a modest upper bound for $g_j^{(i)}$ can be easily derived from the orthonormality constraint on the eigenvector of (14). However, it appears to be difficult to derive a lower bound on $g_j^{(i)}$. The numerical examples presented in [Yang et al. 2005] indicate that $g_j^{(i)}$ can be tiny for some j when $\rho_j^{(i)}$ is relatively large. In this case, one can potentially leave the eigenvector associated with j-th eigenvalue of the i-th sub-structure out of S_i without sacrificing the accuracy of the approximate eigenpair. This would reduce the cost associated with solving the projected problem (11) in the AMLS algorithm. However, since we cannot compute the magnitude of $g_j^{(i)}$ in advance, such a mode selection strategy is not practical. We are currently investigating a number of ways to estimate the magnitude of $g_j^{(i)}$, these estimates may be used to improve the existing mode selection strategy which is based merely on a ρ-factor threshold.

In addition to the truncation error, the accuracy of the AMLS algorithm may be affected by the potential instability introduced by the lack of a global pivoting scheme in the construction of the eliminator L when the matrix K is indefinite. Our implementation of the AMLS algorithm allows local pivoting to be performed on each diagonal block of K (by calling either SuperLU or LAPACK symmetric indefinite factorization routines). However, there is no guarantee that all elements of L are bounded. On the other hand, the use of a global pivoting scheme is difficult because it tends to destroy the desirable sub-structure partition. More research is needed to balance the issue of numerical stability and the efficiency of sub-structuring in this case.

When K is indefinite, it may be tempting to interchange the roles of K and M and use M to construct the eliminator, which guarantees numerical stability. However, this scheme is not appropriate when the eigenvalues of interest are clustered at the low end of the spectrum. The main reason is that the ρ-factors associated with the desired eigenvalues (which, in this case, are the largest eigenvalues of (M, K)) would decrease rapidly due to the relatively large separation between these large eigenvalues. Thus, it would be difficult for the AMLS algorithm to obtain good approximation to more than a few eigenpairs.

3. IMPLEMENTING AMLS

We should emphasize that the major steps outlined in Algorithm 1 constitute only a conceptual description of the AMLS procedure. To develop an efficient implementation, one must carefully organize these steps in a way that would lead to minimal use of memory and floating-point operations (flops).

The complexity of the implementation is magnified by the fact that the congruence transformation $L^{-1}ML^{-T}$ involves more matrix operations than a block

Gaussian elimination. The update of M must be carried out by multiplying M with block elementary eliminators of K (and their transposes) from both the left and the right. Hence, the computation of $L^{-1}ML^{-T}$ cannot be organized as a strictly left-looking or right-looking procedure; it has both right- and left-looking dataflow pattern.

If we were to compute $\widehat{M} = L^{-1}ML^{-T}$ explicitly as suggested by Step (3) of Algorithm 1, the memory usage of the algorithm would become excessive because $\widehat{M}$ is likely to contain a significant number of non-zero fills in each off-diagonal matrix block. However, since computing the desired eigenvalues and the corresponding eigenvectors does not require the presence of $\widehat{M}$ explicitly, a more efficient scheme is to project M into the subspace spanned by columns of $L^{-T}S$ incrementally as L and S are being computed. In other words, we can *interleave* the computations of block elimination (Steps (2) and (3)), the computations of the sub-structure eigenvectors (Step (4)), and the formation of the projected matrices (Step (5)). The pseudo-code in Figure 2 illustrates how specific matrix operations performed in Steps (2)-(5) of Algorithm 1 are interleaved. The sequence in which these matrix operations are performed can also be conveniently described in terms of the traversal of the separator tree shown in Figure 1.

3.1 Interleave congruence transformation and projection calculation

For a node i in a separator tree, we use **Ancestors(i)** to denote the set of nodes on the path from i's parent to the root of the tree. For instance, **Ancestors**$(10) = \{14, 15\}$ in Figure 1. Similarly, we use **Descendants(i)** to denote the set of nodes that belong to the subtree rooted at i (excluding i).

We distinguish the operations performed at leaf nodes from those performed at upper level tree nodes. At a leaf node i, one performs a Cholesky factorization of the diagonal block K_{ii} in order to construct the i-th block column of L. Let m_i be the dimension of K_{ii}. It follows from the standard theory on block Gaussian elimination that computing the i-th block column of L defines an elementary block eliminator

$$L_{e_i} = LE_iE_i^T,$$

where $E_i = (0, ..., I_i, ..., 0)^T$ is an $n \times m_i$ matrix. Note that $L = \Pi_{i=1}^{n_b} L_{e_i}$, where n_b is the number of block columns in L. We say that the i-th leaf node is eliminated when we mulitply K from the left and right by $L_{e_i}^{-1}$ and $L_{e_i}^{-T}$ respectively.

Applying L_{e_i} to K and M amounts to modifying block rows and columns associated with the tree nodes that belong to **Ancestors(i)**. These nodes are shaded in Figure 3(a), for example, when the leaf node being eliminated is the first block column of K and M. The corresponding matrix blocks being updated in K and M are shaded in Figure 3(b). It is important to note that the update of K is "one-sided" (Line 9 of the pseudo-code in Figure 2). That is, because multiplying K from the left by $L_{e_i}^{-1}$ zeros out matrix entries below the i-th diagonal block of the product, post-multiplying $L_{e_i}^{-1}K$ with $L_{e_i}^{-1}$ does not modify the trailing matrix block to the lower right of i-th diagonal block of $L_{e_i}^{-1}K$. However, as we can see from Line 10 of the pseudo-code in Figure 2, the update of M requires multiplications from both sides because $L_{e_i}^{-1}M$ does not zero out matrix entries below the i-th diagonal block.

```
1.  For $i = 1 : N$        % $N$ is the number of block columns (or nodes)
2.     If $i$ is a leaf node
3.        $\mathbf{Factor}(K_{ii})$
4.        $[D_i, Z_i] = \mathbf{Eigs}(K_{ii}, M_{ii})$
5.        For $j \in \mathbf{Ancestors}(i)$
6.           $T_1 = K_{ji}K_{ii}^{-1}$                      % temporary work space
7.           $T_2 = M_{ji} - T_1 M_{ii}$                    % temporary work space
8.           For $k \in \{j\} \cup \mathbf{Ancestors}(j)$     % $(k,i),(i,j) \rightarrow (k,j)$, right-looking
9.              $K_{kj} \leftarrow K_{kj} - K_{ki}T_1^T$
10.             $M_{kj} \leftarrow M_{kj} - M_{ki}T_1^T - \underline{K_{ki}K_{ii}^{-1}}T_2^T$
11.          EndFor
12.          $\mathcal{M}_{ji} = T_2 Z_i$                   % partial projection from the right
13.       EndFor
14.       $\mathbb{K}_{ii} = D_i, \quad \mathbb{M}_{ii} = I$
15.    ElseIf $i$ is separator node
16.       $\mathbf{Factor}(\widehat{K}_{ii})$
17.       $[D_i, Z_i] = \mathbf{Eig}(\widehat{K}_{ii}, \widehat{M}_{ii})$      % Schur complement
18.       For $j \in \mathbf{Ancestors}(i)$
19.          $L_{ji} = K_{ji}\widehat{K}_{ii}^{-1}$                 % stored explicitly
20.          For $k \in \mathbf{Descendants}(i)$          % $(j,i),(i,k) \rightarrow (j,k)$, left-looking
21.             $\mathcal{M}_{jk} = \mathcal{M}_{jk} - L_{ji}\mathcal{M}_{ik}$
22.          EndFor
23.          $T_2 = M_{ji} - L_{ji}M_{ii}$                  % temporary work space
24.          For $k \in \{j\} \cup \mathbf{Ancestors}(j)$     % $(k,i),(i,j) \rightarrow (k,j)$, right-looking
25.             $K_{kj} \leftarrow K_{kj} - K_{ki}L_{ji}^T$
26.             $M_{kj} \leftarrow M_{kj} - M_{ki}L_{ji}^T - L_{ki}T_2^T$
27.          EndFor
28.          $\mathcal{M}_{ji} = T_2 Z_i$                   % partial projection from the right
29.       EndFor
30.       For $k \in \mathbf{Descendants}(i)$
31.          $\mathbb{M}_{ik} = Z_i^T \mathcal{M}_{ik}$                % projection from the left
32.       EndFor
33.       $\mathbb{K}_{ii} = D_i, \quad \mathbb{M}_{ii} = I$
34.    EndIf
35. EndFor
```

Fig. 2. Pseudo-code for Steps (2)-(5) of Algorithm 1.

Also computed at the leaf node i is a partial eigen-decomposition of the pencil (K_{ii}, M_{ii}). This calculation is typically done by using a shift-and-invert Lanczos (SIL) procedure. The computed eigenvector matrix Z_i is used immediately to partially construct the i-th block column of the projected matrix $S^T L^{-1} M L^{-T} S$ in Eq. (6). This partial construction is achieved by replacing M_{ji} with $\mathcal{M}_{ji} = (M_{ji} - K_{ji}K_{ii}^{-1})Z_i$, for $j \in \mathbf{Ancestors}(i)$ (Line 12 of the pseudo-code in Figure 2, illustrated in Figure 4). Since Z_i consists of a small subset of the eigenvectors of (K_{ii}, M_{ii}), $\mathcal{M}_{ji}$ has far fewer columns than M_{ji}. As a result, the calculation of $\mathcal{M}_{ji}$ can be completed with a much lower computational complexity.

At a higher level tree node i, a similar set of operations are performed. What is different at such a node is that the Cholesky factorization of $\widehat{K}_{ii}$ and the partial eigen-decomposition of $(\widehat{K}_{ii}, \widehat{M}_{ii})$ (in Lines 16-17 of the pseudo-code in Figure 2) are performed on the diagonal blocks (of K and M) that have been modified by its

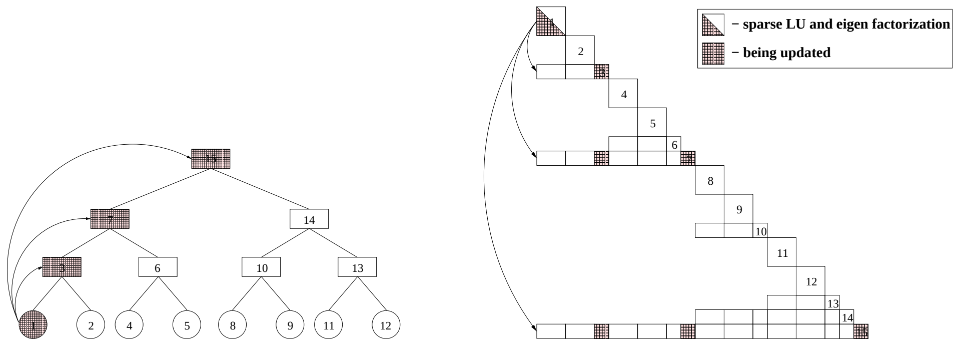


Fig. 3. Elimination of leaf node $i = 1$ and the associated updates to matrix K and M. See lines 8-11 in the code.

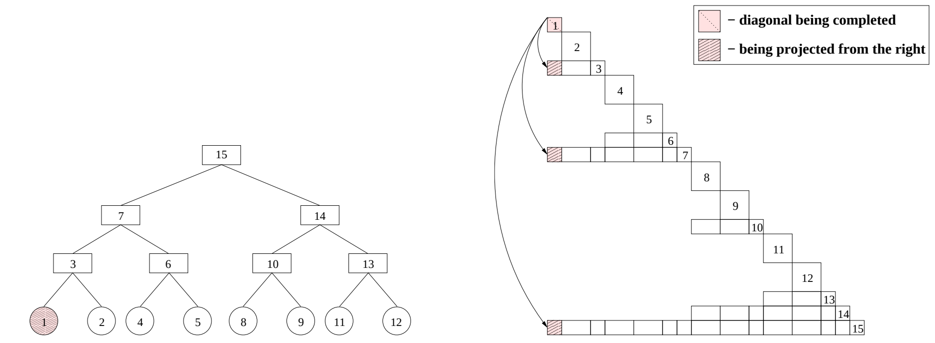


Fig. 4. Projection from the right for leaf node $i = 1$.

descendents. These operations cannot be initiated until the contributions from all its descendent nodes have been included.

In addition, the elementary eleminator constructed at such a node is not only applied to block columns of M that are associated with the ancestors of i, but also to $\mathcal{M}_{jk}$, where $j \in \mathbf{Ancestors}(i)$ and $k \in \mathbf{Desendant}(i)$. This additional step makes the implementation of the AMLS neither a right-looking nor a left-looking algorithm. In Figure 5(a), the dependency of the leaf nodes 1 and 2 on the seperator node 3 is clearly indicated by the direction of the arrows. Figure 5(b) shows that applying the elemintary eliminator constructed at the seperator node 3 to $\mathcal{M}_{jk}$ amounts to subtracting from $\mathcal{M}_{jk}$ a multiple of $\mathcal{M}_{3k}$ for $j \in \mathbf{Ancestors}(3) = \{7, 15\}$ and $k \in \mathbf{Desendant}(3) = \{1, 2\}$. It is important to recall that $\mathcal{M}_{jk}$ typically has far fewer columns than M_{jk}, thus working with $\mathcal{M}_{jk}$ is less costly than working with M_{jk} directly.

Since $\widehat{K}_{ii}$ and $\widehat{M}_{ii}$ are typically dense, and these matrices are much smaller in dimension compared to the diagonal blocks associated with the leaf nodes, the partial eigen-decomposition of $(\widehat{K}_{ii}, \widehat{M}_{ii})$ can usually be computed by an LAPACK routine for dense symmetric eigenvalue problems. All eigenvalues and eigenvectors of $(\widehat{K}_{ii}, \widehat{M}_{ii})$ are computed, but only a selected number of desired eigenvectors are retained and stored in Z_i. The matrix Z_i is used to partially construct the i-th block

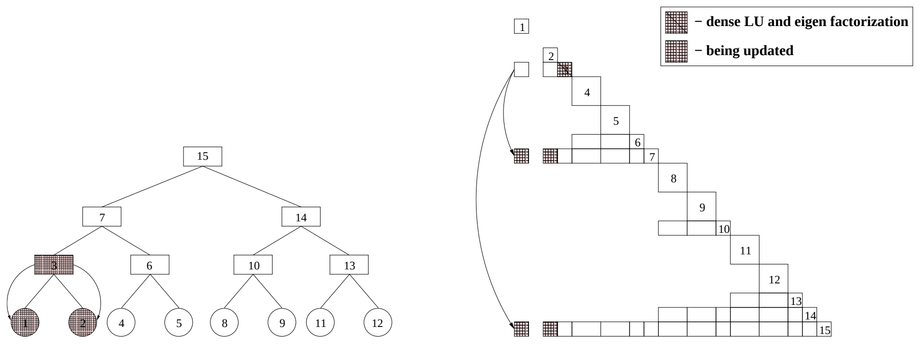


Fig. 5. For seperator node $i = 3$, more congruence updating is needed for the right-projected blocks in the descendent nodes. See lines 20-22 in the code.

column of the projected matrix $S^T L^{-1} M L^{-T} S$. This partial projection produces $\mathcal{M}_{ji}$ for $j \in$ **Ancestors**(i). Moreover, we can now multiply Z_i^T from the left with $\mathcal{M}_{ik}$ for $k \in$ **Descendant**(i) to complete the computation of the i-th block row of $S^T L^{-1} M L^{-T} S$ (shaded blocks in Figure 6 for $i = 3$). In fact, upon the completion of the matrix operations performed at the seperator node i, the leading i block rows and colums of $S^T L^{-1} M L^{-T} S$ become available.

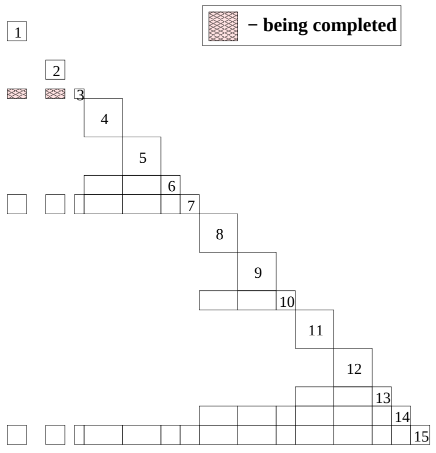


Fig. 6. Left-side projection for separator node $i = 3$. See lines 30-32 in the code.

3.2 A semi-implicit representation of L

When the congruence transformation, the sub-structure mode extraction and the projection calculation (steps (3)-(5) in Figure 1) are interleaved in the way that has been described above, there is no need to store $\widehat{M} = L^{-1} M L^{-T}$. When the i-th separator node is being eleminated, the pseudo-code listed in Figure 2 maintains $\mathcal{M}_{ji}$ for $i \in$ **Descendant**(i). These matrix blocks typically have far fewer columns than M_{ji}. Thus the cost for storing them is significantly lower than that for storing the corresponding blocks in $\widehat{M}$. Applying L_{e_i} to M will introduce some non-zero fills in M_{kj} for $k, j \in$ **Ancestors**(i). However, as we move from the leaf nodes of the separator tree up to the root, the number of these trailing blocks become smaller and smaller. They eventually all vanish as the computation of $S^T\widehat{M}S$ is completed. Therefore, by interleaving Steps (2), (3) and (4) of Algorithm 1, we can reduce the amount of memory required in the calculation of $S^T\widehat{M}S$.

However, what we have not addressed in Section 3.1 is how L is stored. Due to the non-zero fill introduced during the block Cholesky factorization of K, the number of non-zeros in L can potentially be much larger than those in K. Hence it is important to develop a technique for reducing memory usage required for L.

In the AMLS implementation presented in [Kaplan 2001], each block column of L is stored to a disk immediately after it has been used to update K and M. Such an out-of-core implementation reduces the memory requirement of AMLS significantly while incurring a modest amount I/O cost.

In this paper, we present an alternative approach that is equally effective. Note that L has the same block structure as K (see (7)), with $L_{ji} = K_{ji}K_{ii}^{-1}$ for a leaf node i, and $L_{ji} = \widehat{K}_{ji}\widehat{K}_{ii}^{-1}$ for a separator node i. These matrices tend to be denser, thus would require additional storage space to hold the nonzero fill-ins. The basic idea of our approach is to compute and store only the matrix blocks L_{ji} where i is a separator node. All other nonzero matrix blocks in L are not explicitly computed or stored. Whenever L_{ji} is needed, for some leaf node i, in the intermediate matrix-matrix multiplication of the form $B = L_{ji}C$, we apply $K_{ji}K_{ii}^{-1}$ to C directly through a sequence of sparse triangular solves and matrix-matrix multiplications instead of forming $L_{ji} = K_{ji}K_{ii}^{-1}$ first. Such a scheme essentially amounts to recomputing $K_{ji}K_{ii}^{-1}$ whenever it is needed. Because a fully explicit representation of L is avoided in our approach, we call this scheme a *semi-implicit* representation of L.

We underlined in Line 10 of the pseudocode in Figure 2 where the recomputation occurs. It is readily seen that the inner k-loop would require three matrix-matrix multiplications if L is computed and stored explicitly. In our semi-implicit scheme, the inner k-loop requires a triangular solve in addition to three matrix-matrix multiplications. If we assume the cost of a triangular solve is about the same as a matrix-matrix multiplication, the semi-implicit scheme would incur roughly 33% runtime overhead within the k-loop associated with a leaf. However, because constructing the projected pencil $(S^T\widehat{K}S, S^T\widehat{M}S)$ involves more than just these matrix multiplications, the actual overall runtime increase measured in practice is less than 15%, as we will see in Section 4.2.

To give an intuitive idea of how much space might be saved, let us consider a matrix partition that consists of ℓ levels, recomputing the off-diagonal blocks of L instead of storing them explicitly allows us to reduce the total number of stored off-diagonal blocks from $c(\ell)$ to to $c(\ell - 1)$, where

$$c(\omega) = \sum_{j=1}^{\omega} 2^j j. \tag{19}$$

If we assume that each off-diagonal block of L is dense and all off-diagonal blocks are of the same size, the percentage of memory saved for L is roughly

$$\frac{c(l)-c(l-1)}{c(l)} = \frac{2^l\, l}{\sum_{j=1}^{l} 2^j j} \geq \frac{2^l\, l}{l \sum_{j=1}^{l} 2^j} = \frac{2^l}{2(2^l-1)} \approx 50\%.$$

But, since computing $S^T L^{-1} M L^{-T} S$ without using the semi-implicit scheme would require storing not only L but also selected matrix blocks of $L^{-1}ML^{-T}$, the actual percentage of memory saved is lower than the estimate given here. In Section 4.2, we report the actual memory savings for two specific problems.

3.3 Computing Ritz pairs

Since an efficient implmentation of the AMLS algorithm requires one to interleave Steps (2) - (5) outlined in Algorithm 1, we will combine these steps into a single phase of the AMLS implementation. We will later refer to this phase as *phase 1* of the AMLS computation. The rest of the AMLS calculation, which involves solving the projected eigenvalue problem (11) and assembling the Ritz vectors $z = L^{-T}Sq$, will be lumped into *phase 2* of the implementation.

The projected pencil $(S^T\widehat{K}S, S^T\widehat{M}S)$ is generally sparse. In fact, $\widetilde{K} \equiv S^T\widehat{K}S$ is diagonal, and $\widetilde{M} \equiv S^T\widehat{M}S$ typically has the same block nonzero structure as that of M. Furthermore, the diagonal blocks of $\widetilde{M}$ are all diagonals. Thus, it is appropriate to use a SIL procedure to compute the desired eigenpairs of $(\widetilde{K}, \widetilde{M})$ in this case. However, when the dimension of the projected problem is relatively small ($n_{proj} \leq 5000$), and when the number of eigenpairs required is a large fraction of n_{proj}, we observe that it is often more efficient to use a dense eigensolver to compute all eigenpairs of $(\widetilde{K}, \widetilde{M})$.

When n_{proj} becomes very large, SIL can be computationally costly too. To reduce the cost of the Ritz value calculation, Kaplan proposed an alternative, special subspace iteration (SSI) scheme [Kaplan 2001]. Kaplan's scheme first reduces the projected pencil to an even smaller pencil by removing rows and columns of $(\widetilde{K}, \widetilde{M})$ associated with the "undesirable" diagonal entries in $\widetilde{K}$. Eigenpairs of this reduced pencil $(\widetilde{K}_B, \widetilde{M}_B)$ are computed by the SIL algorithm. The n_k Ritz vectors obtained from SIL are used as the starting basis for a block inverse iteration, where n_k is larger than the number of eigenvalues wanted. Denote this starting subspace by $Y^{(0)}$. The jth iteration of the SSI scheme comprises the following steps. First, one step of inverse iteration of the form $X^{(j)} = \widetilde{K}_B^{-1}\widetilde{M}_B Y^{(j-1)}$ produces another subspace $X^{(j)}$ of size n_k. Second, each vector from $X^{(j)}$ is paired with the corresponding Ritz vector in $Y^{(j-1)}$ to form a two-dimensional subspace, from which the best approximation to the lowest eigenpair is computed. Third, the n_k Ritz vectors obtained from the two-dimensional projected problems form a new subspace to project the pencil $(\widetilde{K}_B, \widetilde{M}_B)$, and the n_k approximate eigenvectors computed from this projection spans the subspace $Y^{(j)}$ for the next iterate. The cost of the Kaplan scheme is proportional to the number of block inverse iterations performed. Performing more inverse iterations improves the accuracy of the approximate eigenpairs but increases the runtime. We will compare the cost and accuracy of the Kaplan's scheme with those of SIL in the next section.

4. PERFORMANCE EVALUATION

Since the AMLS algorithm involves only algebraic operations, its implementation does not depend on application specific knowledge. However, the performance of the algorithm may vary from one problem to another due to differences in the sparsity structure and spectral properties of the problems. Up until now, the performance of the algorithm has only been evaluated and analyzed for problems that arise from structural engineering [Kaplan 2001; Kropp and Heiserer 2003]. It has been shown that the performance of AMLS compares favorably with the Lanczos algorithm and the algorithm can reduce the frequency repsonse calculation time by several orders of magnitude on workstations [Kaplan 2001].

In this section, we examine the general performance of AMLS by applying it to eigenvalue problems arising from two other types of applications.

In the first problem, the stiffness and mass matrices are obtained from a finite element discretization of a six-cell damped detuned accelerator structure (DDS) [Ko et al. 2003]. The eigenvalues of interest are related to the cavity resonance frequencies of the accelerator and the corresponding eigenvectors are used to model the electromagnetic accelerating field.

The second problem we include here arises from the normal mode vibrational analysis of the polyethylene (PE) particles [Yang et al. 2001]. In this application, we are interested in the low frequency vibrational modes of the PE molecule, which can be solved by computing the eigenvalues and eigenvectors associated with the Hessian of a potential function that describes the interaction between different atoms. The dimension of the Hessian matrix is three times the number of atoms in the molecule.

Table I summarizes the dimension and sparsity of the test cases. Figure 7 shows the geometric model of the six-cell accelerator cavity structure and the sparsity pattern of the matrix $|K| + |M|$ after K and M are permuted by METIS. Figure 8 shows the molecular structure of the PE3K particle and the sparsity pattern of the Hessian matrix after it is permuted by METIS.

Problem	Dimension	nnz($|K| + |M|$)	Discipline
DDS6_COARSE	5584	95138	accelerator, 6 cells, coarse grid
DDS6_FINE	65730	1455772	accelerator, 6 cells, fine grid
PE3K	9000	3279690	vibrational analysis, 3000 atoms
PE12K	36000	7128473	vibrational analysis, 12000 atoms

Table I. Characteristics of the test problems.

As we will see in the following, the performance of AMLS, which can be measured by runtime, memory requirement and the accuracy of the computed eigenvalues, depends on a number of choices one has to make at runtime. These choices include, for example, the number of partitioning levels (n_{levels}), the number of eigenpairs (n_{modes}) one must compute and choose from each sub-structure, and the method one uses to compute the eigenpairs of the final projected problem.

Although it remains difficult to identify the optimal choices for some of the run time parameters a priori, the following performance evaluation demonstrates the effect of these parameters on the performance of AMLS and the trade-offs between

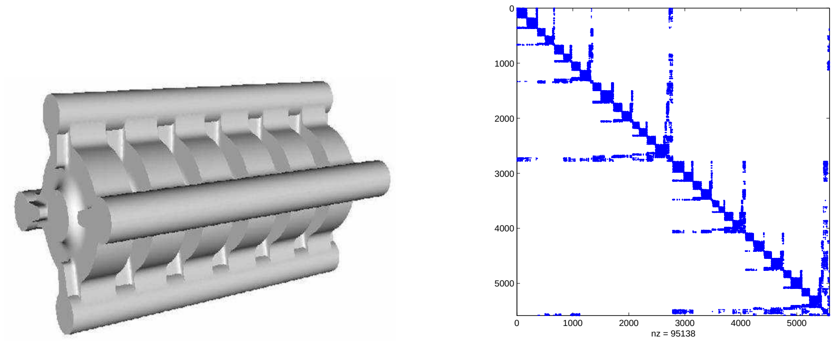


Fig. 7. The accelerator cavity of DDS6_COARSE and the sparsity pattern of $|K| + |M|$ after it is permuted by METIS.

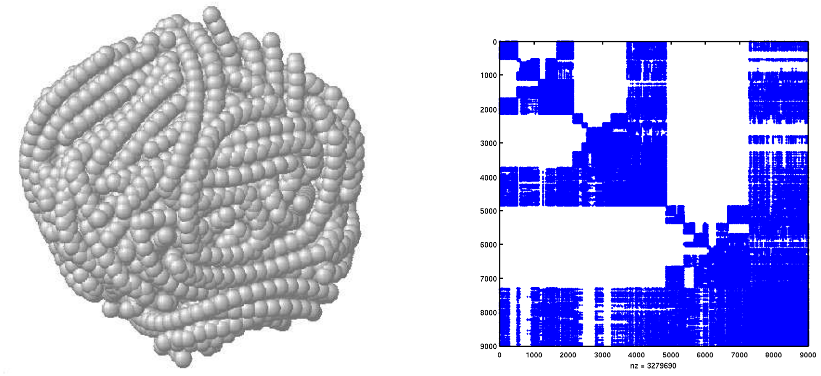


Fig. 8. The molecular structure of PE3K and the sparsity pattern of the Hessian after it is permuted by METIS.

different choices. In addition, the empirical results presented below provide some general guidelines on how to change these paramters in order to achieve better performance.

All computational experiments reported below are carried out on a single IBM Power3 processor with a clock speed of 375Mhz and 2 MB of level-2 cache. The external software packages used are: METIS [Karypis and Kumar 1998], ARPACK [Lehoucq et al. 1998], SuperLDLT [Ng and Peyton 1993] and SuperLU [Demmel et al. 1999]. We compiled all the codes using IBM's `xlc` (C) or `xlf` (Fortran) compilers with `-O3` optimization flag. We use *nev* to denote the number of wanted eigenvalues, n_{levels} to denote the number of partitioning levels, and n_{modes} to denote the number of modes chosen from each sub-structure. The accuracy tolerance for each subproblem is denoted by τ_{sub}, and the accuracy tolerance for the projected problem is denoted by τ_{proj}. In all the experiments, we set $\tau_{sub} = 10^{-10}$ and $\tau_{proj} = 10^{-5}$.

4.1 Variation of runtime with respect to the partitioning levels

In the first set of experiments, we try to assess the variation in the overall runtime of AMLS with respect to the number of partitioning levels (n_{levels}). In general, as one increases the number of levels in algebraic partitioning, the number of spectral components that can be chosen from each sub-structure is reduced. The exact number of spectral components that should be chosen from each sub-structure is problem dependent, and the selection rule should be based on the theory and heuristics established in [Yang et al. 2005]. However, to focus on the effect of partitioning depth, we set aside the accuracy issue here and keep the total number of modes selected from all sub-structures roughly the same for all partitioning schemes. That is, if we double the number of sub-structures by dividing each sub-structuring into two smaller sub-structures, we then reduce n_{modes} associated with each sub-structure by a half.

Table II shows the timing statistics for the DDS6_FINE problem. Since the separators in this problem are small, all the coupling modes are retained in the subspace (10). In addition to the total time, we examined the timing breakdown of the two major phases of the implementation defined in Section 3.3. Recall that in the first phase of the implementation, Steps (2)-(5) of Algorithm 1 are interleaved. In the second phase, we compute the Ritz approximation to the desired eigenpairs. The cost of the first phase depends on n_{levels} and the number of modes selected from the sub-structures (n_{modes}). The cost of the second phase depends on the total number of eigenpairs of (K, M) to be computed (nev) in addition to n_{levels} and n_{modes}.

The total amount of time required to compute the smallest 100 eigenpairs of this problem is listed in Column 5 of the table for different choices of n_{levels}. We observe that the best timing is obtained when $n_{levels} = 3$. As a comparison, computing these eigenpairs by a shift-and-invert Lanczos (SIL) method (using ARPACK and SuperLLT packages [Ng and Peyton 1993] with METIS reordering) would take 407 seconds. Thus when $nev = 100$, AMLS with $n_{levels} = 3$ is competitive with SIL in speed. In Columns 3 and 4 of the table, we show how the runtime is split between the first and the second phases of the AMLS calculation. We observe that the phase 1 calculation becomes less time consuming as we increase n_{levels}. However, increasing n_{levels} leads to a rapid increase in the runtime spent in the second phase of the AMLS calculation. Such an increase is caused by the increased dimension of the projected problem, which is in turn caused by an exponential increase of the number of separators with respect to n_{levels}. We observe that the time spent in the second phase of the AMLS calculation actually exceeds that spent in the first phase for the DDS6_FINE problem when $n_{levels} > 3$.

n_{levels}	n_{modes}	phase 1 (sec)	phase 2 (sec)	total (sec)
2	100	458	33	491
3	50	288	77	365
4	25	220	235	455
5	12	194	292	486
6	6	152	352	504

Table II. Runtime with different n_{levels}. Problem DDS6_FINE, $nev = 100$.

The timing statistics for the PE3K problem shows a similar pattern to that reported in Table II. In this problem, we compute 500 smallest eigenpairs. The separators produced by the METIS nested dissection reordering have relatively large dimensions. We chose roughly $n_{sep}/5$ coupling modes from each separator, where n_{sep} is the size of the separator. As we can see in Table III, the CPU time spent in phase 1 of AMLS also decreases as n_{levels} increases. The increase in phase 2 time is less dramatic than that of the DDS6_FINE problem, because the size of the final projected problem increases slowly for PE3K. The best overall performance is obtained when $n_{levels} = 6$. In this case, AMLS is almost twice as fast as SIL which used 956 seconds to compute all desired eigenpairs.

n_{levels}	n_{modes}	phase 1 (sec)	phase 2 (sec)	total (sec)
2	100	462	55	521
3	50	439	81	523
4	25	404	98	505
5	12	391	105	499
6	6	381	98	483

Table III. Runtime with different n_{levels}. Problem PE3K, $nev = 500$.

4.2 Variation of memory usage with respect to the number of sub-structures

We now assess the variation in AMLS memory usage with respect to n_{levels}. Here, our experimental setup is the same as that in Section 4.1. Table IV shows the memory usuage statistics for the DDS6_FINE problem. Column 3 gives the total memory usage for different choices of n_{levels} when a semi-implicit representation of L is activated. The actual amount of savings gained from this semi-implicit scheme is listed in column 4. In Column 5, we list the estimated percentage of savings based on the formula $\frac{c(l)-c(l-1)}{c(l)}$ given in Section 3.2, where $c(\ell)$ is defined in Eqn. (19).

As we explained in Section 3, a semi-implicit representation of L would incur a slight increase in runtime because the off-diagonal blocks of L that are not stored must be recomputed when they are needed. The extra cost associated with re-computing is reported in Column 6. We see that the actual recomputing overhead reported in Table IV is lower than the rough estimation made in Section 3 (about 33%). This is because our estimation does not account for the time spent in the other part of the algorithm. We observe that up to 50% of the memory usage can be reduced in AMLS for the DDS6_FINE problem with only 10-15% extra runtime. This reduction is very attractive when memory is at a premium.

As a comparison, the memory usage measured in SIL is 308 Megabytes(MB). Thus, our implementation of AMLS appears to be more efficient in both runtime and memory usage than SIL when n_{levels} is set to 3.

For the PE3K problem, the percentage of memory saved using the semi-implicit representation of L is not as impressive as that reported for the DDS6_FINE problem. The reason we observed a smaller amount of saving is that this problem has relatively large separators. Since the off-diagonal blocks of L associated with each separator are kept in memory, the amount of memory that can be saved is limited.

n_{levels}	n_{modes}	total-mem (MB)	mem-saved (MB)	mem-saving estimate	recompute time (sec)
2	100	319	199 (38.4%)	75.0%	9.2 (1.5%)
3	50	263	263 (50.0%)	66.7%	51.5 (11.0%)
4	25	325	248 (43.3%)	62.5%	60.7 (13.3%)
5	12	392	228 (36.8%)	60.0%	64.0 (13.2%)
6	6	480	192 (28.6%)	58.3%	55.3 (10.9%)

Table IV. Memory usage with different n_{levels}. Problem DDS6_FINE, $nev = 100$.

However, even in this case, we observe from Table V that up to 28% memory is saved. The optimal memory usage (265MB) is achieved when $n_{levels} = 6$. This compares favorably with the use of 283MB memory in the SIL calculation.

n_{levels}	n_{modes}	total-mem (MB)	mem-saved (MB)	mem-saving estimate	recompute time (sec)
2	100	289	113 (28.1%)	75.0%	73.9 (14.2%)
3	50	271	96 (26.2%)	66.7%	82.6 (15.8%)
4	25	268	66 (19.8%)	62.5%	57.4 (11.4%)
5	12	270	42 (13.5%)	60.0%	35.9 (7.2%)
6	6	265	34 (11.4%)	58.3%	29.6 (6.1%)

Table V. Memory usage with different n_{levels}. Problem PE3K, $nev = 500$.

4.3 Solving the projected problem

Table II shows that solving the projected eigenvalue problem (6) can become more expensive than the computation required to construct the projected pencil $(S^T\widehat{K}S, S^T\widehat{M}S)$ when the number of partitions is relatively large. Even if we reduce the number of spectral components selected from each sub-structure (by as much as a half) as we did for DDS6_FINE in Table II, the dimension of the projected problem, n_{proj}, is still likely to increase as the number of partitions increases. Such an increase in dimension is caused by the increased number of interface blocks (seperators) produced by the multi-level matrix partitioning. Thus, to reduce the total computational cost of AMLS, one must choose an effcient algorithm to compute the desired eigenpairs of the projected pencil.

Table VI shows the changes in n_{proj} and the CPU time required to compute the smallest $nev = 500$ eigenpairs of $(\widetilde{K}, \widetilde{M})$ obtained by applying AMLS to the DDS6_FINE problem with different choices of n_{levels}. We list the CPU time required by both the SIL and the LAPACK divide-and-conquer eigenvalue calculation routine `DSYEVD`. For this example, n_{modes} is set to 100 when $n_{levels} = 2$, and is reduced by a half when n_{levels} is increased by one. All spectral modes of the interface blocks are selected. Thus, the increase in n_{proj} reflects the increased number of seperators produced by nested dissection. We observe that when $n_{proj} < 5000$, `DSYEVD` is significantly faster than SIL, and SIL is only slightly faster when n_{proj} reaches 6547.

We also experimented with Kaplan's special subspace iteration (SSI) scheme discussed in Section 3.3 using the DDS6_COARSE problem with $n_{levels} = 1$. The

n_{levels}	n_{modes}	n_{proj}	SIL (sec)	`DSYEVD` (sec)
2	100	1289	557	33
3	50	1878	744	77
4	25	2976	986	235
5	12	4432	1206	651
6	6	6547	2036	2064

Table VI. Solving the final projected eigenvalue problem for DDS6_FINE. $nev = 500$.

dimensions of the two sub-structures produced by this one-level partition are 2761 and 2791 respectively. The size of seperator in this case is 32. We set n_{modes} to 796 and 802 on each sub-structure respectively. All coupling modes are included in S. In Figure 9, we plot the relative accuracy of the 50 Ritz values computed by both SIL and SSI. We experimented with performing different numbers of block inverse iterations in SSI, and will denote an SSI run with k block inverse iterations by SSI(k). Figure 9 shows that it takes at least three inverse iterations in SSI to produce Ritz values that are as accurate as those produced by SIL. The CPU time required to perform three block inverse iterations in SSI is comparable to that required to carry out SIL on $(\widetilde{K}, \widetilde{M})$. Therefore, we do not see a clear advantage of using Kaplan's scheme over using SIL for this problem.

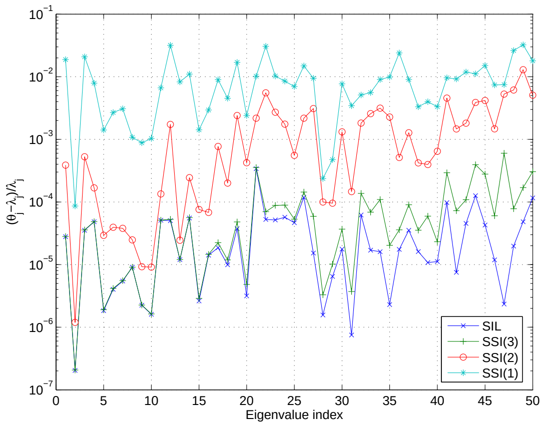


Fig. 9. Accuracy of Kaplan's subspace inverse iteration scheme (SSI) for the projected eigenvalue problem. Problem DDS6_COARSE, $nev = 50$.

4.4 Variation of runtime with respect to nev

Since the computational complexity of AMLS is slightly higher than that associated with a block LDL^T factorization of K, one should not expect a significant performance gain from AMLS over SIL when the number of eigenvalues to be computed (nev) is small (say less than 100). In fact, Table II shows that in the case of DDS6_FINE, the performance of AMLS is slightly worse than SIL when $nev = 100$ and $n_{levels} = 2, 4, 5$ and 6.

However, as nev increases, AMLS becomes more attractive as we can see from Figure 10 where we plotted both the AMLS and SIL runtime as a function of nev. If we assume that a sufficient number of spectral modes have already been selected from each sub-structure so that the projected pencil

$$(\widetilde{K}, \widetilde{M}) \equiv (S^T L^{-1} K L^{-T} S,\ S^T L^{-1} M L^{-T} S)$$

contains accurate approximations to a large number of eigenpairs of (K, M), an increase in nev simply means that we need to compute more Ritz pairs from $(\widetilde{K}, \widetilde{M})$. In this case, the incremental cost of AMLS associated with computing more approximate eigenpairs can be completely accounted for by the additional computation required to solve the projected eigenvalue problem. When the dimension of $(\widetilde{K}, \widetilde{M})$ is much smaller than n, such an incremental cost is relatively small compared to the cost of constructing $(\widetilde{K}, \widetilde{M})$.

In contrast, the incremental cost of SIL can be much higher than the cost of computing the initial block LDL^T factorization. If a single LDL^T factorization is used in SIL, the convergence of eigenvalues far away from the target shift can be very slow, leading to a significant increase in the size of the Lanczos basis required to capture all the desired spectral components of (K, M) or an increase in the number of restarts [Lehoucq et al. 1998] required to filter out the unwanted spectral components from the Krylov subspace constructed by SIL. In either case, the number of sparse triangular substitutions would increase substantially and the cost of maintaining the orthonormality of the Lanczos basis would become higher also. The use of multiple target shifts in SIL has the benefit of accelerating the convergence of the desired eigenpairs, and thus reduces the overall number of triangular substititions. However, because the LDL^T factorization of $K - \sigma M$ must be recomputed for each target shift σ, the cost of SIL typically increases linearly with respect to the number of shifts used. In Figure 10, the estimated runtime of SIL is based on the assumption that a single shift is used to compute at most 100 eigenpairs of (K, M), i.e., when $nev = 500$, five SIL runs with five different target shifts are used to compute all desired eigenvalues. It is easy to see from this figure that AMLS is clearly a more favorable choice than SIL for the DDS6_FINE problem when $nev > 200$.

4.5 Accuracy

The accuracy of AMLS clearly depends on the number of spectral modes (n_{modes}) selected from each sub-structure. For a fixed sub-structure partition, increasing n_{modes} increases the dimension of the subspace $\mathcal{S} \equiv \text{span}\{S\}$ onto which (K, M) is projected. Such an increase naturally results in improved accuracy in the Rayleigh-Ritz approximations to the desired eigenpairs. However, the amount of improvement varies from one problem to another. In Figure 11 we show the accuracy of the smallest 100 eigenvalues of DDS6_FINE computed using the AMLS algorithm. The Y-axis is the relative error of each eigenvalue compared with that computed using SIL. Figure 11(a) shows that, by increasing n_{modes} on all sub-structures of a 5-level partition uniformly from 12 to 100, one improves the accuracy of the leading 100 Ritz values by roughly one digit. Such a modest improvement for this problem can be explained by the slow decay of the ρ-factor (see Eqn. (18)) as a function of eigenvalues of each sub-structure. Since the ρ-factor provides an estimation of

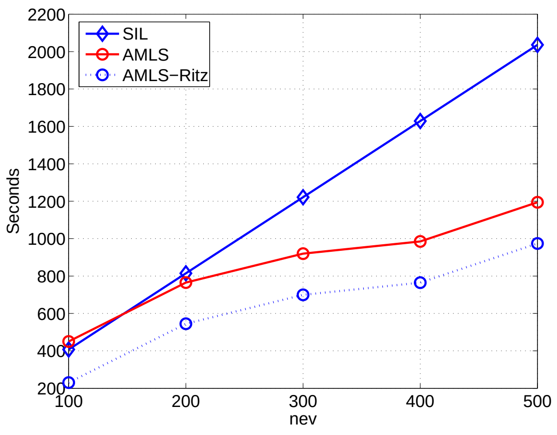


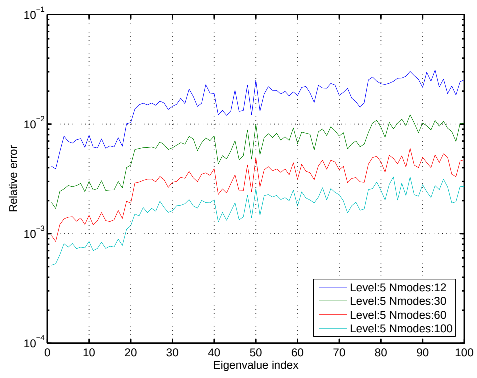
Fig. 10. Runtime of AMLS and SIL with increasing nev. Problem DDS6_FINE, $n_{levels} = 4$, $n_{modes} = 25$. "AMLS-Ritz" is the phase to solve the final projected problem given by (6).

how significant the corresponding sub-structure mode is to the approximate eigenvector [Yang et al. 2005], a slowly decreasing ρ-factor implies that a small change in n_{modes} results in limited change of accuracy in the approximate eigenvector.

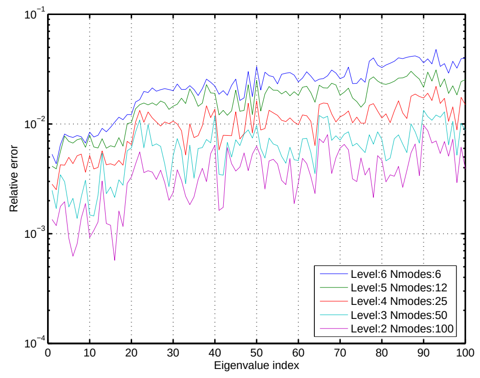
A mode selection scheme based on an appropriate ρ-factor cutoff would guarantee sufficient accuracy of the Rayleigh-Ritz approximation produced by AMLS [Elssel and Voss 2004; Yang et al. 2005]. However, it has been observed that such a selection scheme often retains more modes from each sub-structure than is necessary. It is shown in [Yang et al. 2005] that the contribution of each mode to the approximate eigenvector should be measured by the magnitude of the corresponding element in the y_i vector defined in (14). There are cases in which elements of y_i are tiny (indicating these modes can be discarded) while the corresponding ρ-factors are relatively large. Unfortunately, y_i cannot be computed in advance. Thus, further research is needed to develop additional heuristics for estimating the magnitude of each element of y_i more accurately.

The issue of mode selection versus accuracy becomes more complex when one takes into account the different choices of sub-structure partitions. As n_{levels} increases, each sub-structure contains fewer modes to choose from, thus one would naturally decrease n_{modes} on each sub-structure. Ideally, one would like to decrease n_{modes} such that the dimension of $\mathcal{S}$ (which is the sum of all n_{modes} associated different sub-structures and the interface blocks) remains roughly the same. However, Figure 11(b) shows that such a strategy typically leads to a loss of accuracy in the Ritz values extracted from $\mathcal{S}$. Thus, one may be forced to choose a relatively large n_{modes} on each sub-structure, which would increase the dimension of $\mathcal{S}$. Although increasing the number of sub-structures typically reduces the cost of constructing $S^T L^{-1} M L^{-T} S$, an increase in the dimension of $\mathcal{S}$ makes solving the projected problem more costly. Further research is required to identify the optimal balance.

In Figure 12 we plot the relative residuals for the smallest 100 eigenpairs of DDS6_FINE computed using the AMLS algorithm. The Y-axis is $\|Kz_j - \theta M z_j\|_2 / |\theta_j|$. As can be seen, the residuals are relatively small, with two to three digits accuracy, which usually indicates that the algorithm delivers small backward errors.

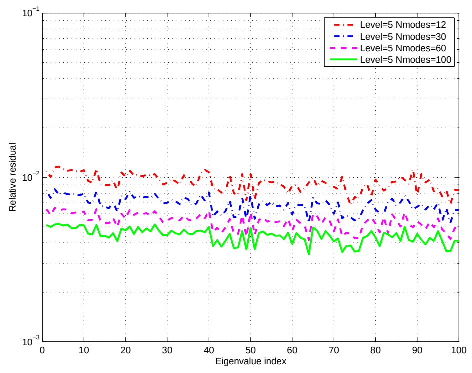


(a) Fixed level, increasing n_{modes}

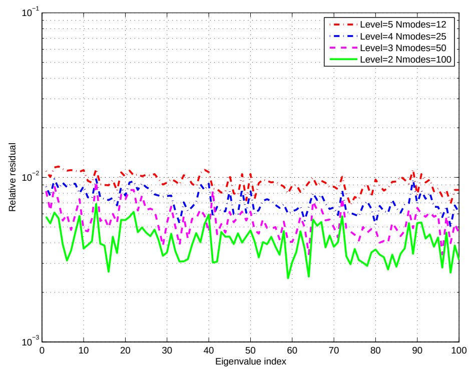


(b) Increasing levels

Fig. 11. Eigenvalue accuracy for DDS6_FINE, compared with the SIL method.



(a) Fixed level, increasing n_{modes}



(b) Increasing levels

Fig. 12. Relative residuals for DDS6_FINE.

For many 3D problems, a multi-level partitioning of $|K|+|M|$ produces relatively large-sized seperators. For these problems, retaining all the spectral components of each seperator in the subspace represented by (5) or (10), as proposed in the CMS method, would result in a large projected pencil. Even when the size of each seperator is modest, the exponential increase of the number of seperators with respect to n_{levels} can make solving the projected problem (6) a computationally intensive task. Thus, in many cases, it is desirable to reduce the number of spectral modes chosen from each seperator so that the dimension of the projected pencil can be reduced.

In [Kaplan 2001], the number of modes selected from each seperator is determined by a cutoff frequency set slightly larger than the highest frequency of interest for the frequency response. For applications that do not arise from frequency reponse analysis, such a cutoff typically does not exist. Even when such a cutoff frequency exists, the error analysis presented in Section 2.3 indicates that using a given cutoff frequency may not lead to an optimal mode selection strategy.

We now illustrate that performing partial mode selection can have a significant impact on the performance of AMLS on the PE3K and PE12K problems. Both of these matrices contain separators of large dimensions. We set n_{modes} to 100 for each sub-structure. With a three-level partitioning, the cumulative size of the seperator in PE3K is roughly one-fourth of dimension of the original problem. For PE12K, the size of the top level seperator is already 4937. Table VII shows the reduction in dimension of the projected pencil $(\widetilde{K}, \widetilde{M})$ when roughly 20% of the modes are selected from each seperator for both PE3K and PE12K. Such a reduction translates into a reduction of the AMLS runtime by at least a half for both PE3K and PE12K as shown in Table VIII. Table VIII also shows that with partial selection, AMLS runtime compares favorably with that of SIL.

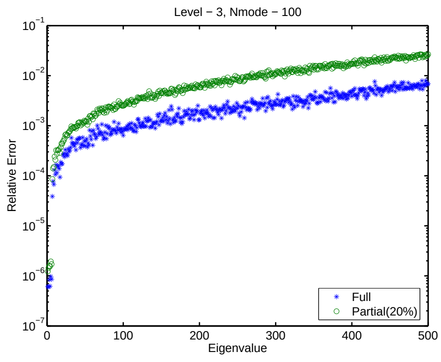
Problem	n_{levels}	partial	full
PE3K	3	1484	5658
PE12K	4	5350	20348

Table VII. Dimension of the projected problem with and without partial selection.

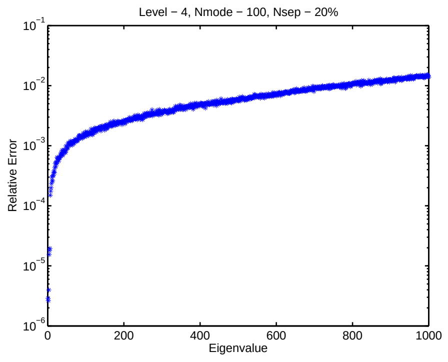
Problem	nev	n_{levels}	partial	full	SIL
PE3K	500	3	581	1776	1061
PE12K	1000	4	15511	> 36000	18000

Table VIII. Effect of partial mode selection on the AMLS runtime (in seconds).

In another experiment, the number of modes selected from each seperator is determined by examining the magnitude of the ρ_{λ_1} factor associated with the top-level seperator. Figure 13 shows that such a selection heuristic allows us to maintain at least two digits of accuracy in the approximation to the smallest eigenvalue of both PE3K and PE12K.



(a) Relative error of the approximation to the smallest 500 eigenvalues of PE3K



(b) Relative error of the approximation to the smallest 1000 eigenvalues of PE12K

Fig. 13. Relative error

5. CONCLUDING REMARKS

An efficient implementation of the AMLS algorithm requires matrix operations performed in the congruence transformation of (K, M), the sub-structure eigenpair calculation, and the projection of (K, M) to be organized in a way that would lead to a minimal floating point operation count and memory usage.

Although the implementation of AMLS bears many similarities to the implementation of a block Cholesky factorization, it differs from block Cholesky in terms of the data flow pattern. In particular, the implementation of AMLS cannot be organized as a strictly left-looking or right-looking procedure. Nonetherless, the major operations in our implementation of AMLS can be conveniently described in terms of the traversal of a separator tree derived from the initial algebraic partitioning of (K, M).

We developed a technique for reducing memory usage in the AMLS calculation. In this technique, only the off-diagonal blocks of L associated with the separator nodes are stored in memory. All other off-diagonal blocks are recomputed whenever they are needed. We demonstrated that such a semi-implicit representation of L can result in up to 50% savings in memory usage while incuring less than 15% increase in runtime.

We evaluated our implementation of AMLS on two different applications outside the area of structural analysis. We observed that AMLS is much faster than SIL when a large number of eigenvalues with a few digits of accuracy are of interest. We also observed that both the runtime and memory requirements of AMLS vary with respect to the number of partitioning levels (n_{levels}). In particular, the first phase of the AMLS computation, which involves interleaving the congruence transformation of (K, M) with the sub-structure calculation and the projection (K, M) into the subspace assembled from the sub-structure eigenvectors, becomes less time consuming as n_{levels} increases. However, increasing n_{levels} tends to increase the dimension of the projection subspace, hence leads to a more costly second phase calculation. More research is needed to identify an optimal choice of n_{levels}.

A sub-structure mode selection scheme has a large effect on both the accuracy and cost of AMLS computation. We illustrated that the accuracy of the Ritz values may not improve if one simply increases n_{modes} on either a sub-structure or a separator. However, such an increase in n_{modes} can lead to a significant increase in AMLS runtime. We asserted that it is more important to select the "important" modes that are associated with the large entries in the eigenvectors (y) of a canonical eigenvalue problem. Although we did not give a recipe for choosing these modes in this paper, our analysis indicates how we might be able to identify these modes by estimating the entries of y.

Acknowledgments

We thank Parry Husbands, Lie-Quan Lee, and Esmond Ng for fruitful discussions. We thank the anonymous reviewers for their careful reading and providing the suggestions to improve the paper.

REFERENCES

Bekas, K. and Saad, Y. 2005. Computation of smallest eigenvalues using spectral Schur complements. *SIAM Journal on Scientific Computing 27,* 2, 458–481.

BENNIGHOF, J. K. November, 1993. Adaptive multi-level substructuring method for acoustic radiation and scattering from complex structures. In *Computational methods for Fluid/Structure Interaction*, A. J. Kalinowski, Ed. Vol. 178. AMSE, New York, 25–38.

BENNIGHOF, J. K. AND LEHOUCQ, R. B. 2004. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM Journal on Scientific Computing 25*, 2084–2106.

CRAIG, R. R. AND BAMPTON, M. C. C. 1968. Coupling of substructures for dynamic analysis. *AIAA Journal 6*, 1313–1319.

DEMMEL, J. W., EISENSTAT, S. C., GILBERT, J. R., LI, X. S., AND LIU, J. W. H. 1999. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20,* 3, 720–755.

ELSSEL, K. AND VOSS, H. 2004. An a priori bound for automated multi-level substructuring. Tech. Rep. 81, Arbeitsbereich Mathematik, TU Hamburg-Harburg, Germany.

GEORGE, A. 1973. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis 10*, 345–363.

HURTY, W. C. 1960. Vibrations of structure systems by component-mode synthesis. *Journal of the Engineering Mechanics Dvision, ASCE 86*, 51–69.

KAPLAN, M. F. 2001. Implementation of automated multilevel substructuring for frequency response analysis of structures. Ph.D. thesis, University of Texas at Austin, Austin, TX.

KARYPIS, G. AND KUMAR, V. 1998. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0.* University of Minnesota.

KO, K., FOLWELL, N., GE, L., GUETZ, A., IVANOV, V., LEE, L., LI, Z., MALIK, I., MI, W., NG, C., AND WOLF, M. 2003. Electromagnetic systems simulation - from simulation to fabrication. SciDAC report, Stanford Linear Accelerator Center, Menlo Park, CA.

KROPP, A. AND HEISERER, D. 2003. Efficient broadband vibro-accoutic analysis of passenger car bodies using an FE-based component mode synthesis approach. *J. Computational Acoustics 11,* 2, 139–157.

LEHOUCQ, R. B., SORENSEN, D. C., AND YANG, C. 1998. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.* SIAM, Philadelphia.

NG, E. G. AND PEYTON, B. W. 1993. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM Journal on Scientific Computing 14,* 5 (September), 1034–1056.

PARLETT, B. N. 1998. *The Symmetric Eigenvalue Problem.* SIAM, Philadelphia.

YANG, C., GAO, W., BAI, Z., LI, X., LEE, L., HUSBANDS, P., AND NG, E. 2005. An algebraic sub-structuring method for large-scale eigenvalue calculation. *SIAM Journal on Scientific Computing 27,* 3, 873–892.

YANG, C., PEYTON, B. W., NOID, D. W., SUMPTER, B. G., AND TUZUN, R. E. 2001. Large-scale normal coordinate analysis for molecular structures. *SIAM Journal on Scientific Computing 23,* 2, 563–582.

xxxx