Block Tridiagonalization of “Effectively” Sparse Symmetric Matrices

Yihua Bai[*§], Wilfried N. Gansterer[*§] and Robert C. Ward[*§]

Technical Report UT-CS-02-492[1]

University of Tennessee

March 2003

Abstract: A block tridiagonalization algorithm is proposed for transforming a sparse (or “effectively” sparse) symmetric matrix into a related block tridiagonal matrix, such that the eigenvalue error remains bounded by some prescribed accuracy tolerance. It is based on a heuristic for imposing a block tridiagonal structure on matrices with a large percentage of zero or “effectively zero” (with respect to the given accuracy tolerance) elements. In the light of a recently developed block tridiagonal divide-and-conquer eigensolver [6], for which block tridiagonalization is needed as a preprocessing step, the algorithm also provides an option for attempting to produce at least a few very small diagonal blocks in the block tridiagonal matrix. This leads to low time complexity of the last merging operation in the block divide-and-conquer method. Numerical experiments are presented and various potential block tridiagonalization strategies are compared.

1. Introduction. There are a number of efficient algorithms for solving real symmetric eigenvalue problems [12]. Depending on the nature of the matrices and desired spectral information, some require tridiagonalization of the original matrix as a preprocessing step while others directly exploit the matrix’s sparsity. For a tridiagonal matrix, the divide-and-conquer method [2,10] is an extremely efficient method for computing eigenpairs. However, transformation to tridiagonal form can be expensive with unfavorable data access patterns and data locality problems for modern deep hierarchical memory computers [4]. The block tridiagonal divide-and-conquer (BD&C) method developed by Gansterer and Ward *et al.* [5,6] can be applied to block tridiagonal matrices, greatly eliminating problems with data access and locality. In addition, their algorithm can be used to very effectively approximate eigenpairs of a block tridiagonal matrix up to a prescribed accuracy tolerance. The rank of the off-diagonal block in the final merging operation (see Section 2) is a major factor in the efficiency of this algorithm. Consequently, it is important to find a “good” block tridiagonal approximation (i.e., one which contains some low-rank off-diagonal blocks) of the matrix whose eigenpairs are to be approximated.

[*]Department of Computer Science, University of Tennessee, 203 Claxton Complex, 1122 Volunteer Blvd., Knoxville, TN 37996-3450.

[§]This work was partially supported by the Academic Strategic Alliances Program of the Accelerated Strategic Computing Initiative (ASCI/ASAP) under subcontract number B341492 of DOE contract W-7505-ENG-48.

[1]Available from: http://www.cs.utk.edu/~library/TechReports.html

1.1 Objectives. The primary objective of our work described in this paper is as follows:

Given an accuracy tolerance τ ($\tau \leq 0.1$) and a (dense or sparse) symmetric matrix A, find a closely related block tridiagonal matrix B with the following properties:

1. Eigenvalues of B differ from those of A by less than the normalized accuracy tolerance $\tau\|\mathrm{A}\|$,
2. B has small bandwidth b relative to the size of the matrix, and
3. At least a few individual off-diagonal blocks of B have either very low dimension or are sufficiently close to a low-rank matrix.

The algorithm we describe in this paper is particularly suited for matrices that we call "effectively sparse", i.e., dense matrices with the property that a large portion of their nonzero entries may be changed to zero without influencing the eigenvalues by more than $\tau\|A\|$. We call matrix entries that may be set to zero without changing the eigenvalues more than $\tau\|A\|$ "effectively zero".

1.2 Applications. Many mathematical models used in quantum mechanical electronic structure computations (e.g., for material design, the computation of optical absorption and emission spectra, x-ray emission and diffraction, vibrational mode and rotational mode in molecules, tribology at electronic level), are built around the time-independent Schroedinger equation. A very popular method for solving this equation involves a basis expansion of the unknown solution. There are various methods for choosing this basis expansion and for truncating it. They all have in common that they lead to a nonlinear finite-dimensional matrix eigenvalue problem (one example being the Hartree-Fock equations). This nonlinear eigenvalue problem is typically solved via a fixed-point iteration approach (called the Self-Consistent Field (SCF) procedure in quantum chemistry), which requires the solution of a linear eigenvalue problem in every iteration [20]. The matrices generated during these iterations are typically real, symmetric and dense and in many important situations they have large elements close to the diagonal and small elements away from the diagonal.

In this paper, we motivate our concepts and illustrate the efficiency of our algorithm with problems from quantum chemistry. Fig. 1 shows a Fock matrix arising when applying the CNDO method [13,14,15,16,17] to a linear $C_{322}H_{646}$ alkane molecule with the absolute value of its elements plotted on a logarithmic scale. The scale bar on the right shows the exponent of the absolute values of the matrix elements.

Because of the inner-outer iterative structure of the SCF procedure for our model problem, it tends to be advantageous to require lower accuracy in the initial iterations and to increase the accuracy as the process converges, especially if there is significant savings in time for lower accuracy requirements. The BD&C algorithm allows the user to specify the desired accuracy of the eigensolution. The block tridiagonalization algorithm described in this paper also has an accuracy tolerance as an input parameter, which is used in an effort to determine the "best" block tridiagonal approximation subject to the constraint that the eigenvalues of B differ less than the specified tolerance from those of A. Therefore, the block tridiagonalization algorithm is a very important preprocessing step for the BD&C eigensolver. Combined, they form an efficient algorithm for the SCF procedure and for symmetric eigenproblems with effectively sparse matrices.

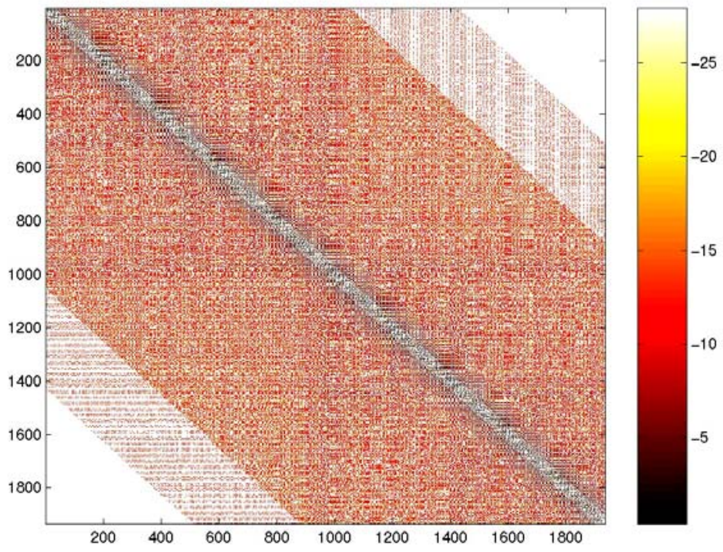


Fig. 1 *log_{10} of absolute value of A (Fock matrix for $C_{322}H_{646}$)*

1.3 Approach. The block tridiagonal structure of a matrix is completely specified by the number p of diagonal blocks and by the sequence of their sizes. Note that the block tridiagonalization of a given sparse symmetric matrix is not unique, though, since smaller blocks can be combined into larger ones to produce a different blocking.

Trivially, every symmetric matrix has block tridiagonal structure with $p = 2$. In general, though, we cannot expect to find a block tridiagonal structure with $p > 2$ that has the desired properties 1 – 3 stated in Section 1.1. However, based on the class of matrices considered in this paper, we can typically find block tridiagonal matrices whose spectrum is sufficiently close to the original spectrum given τ . Using the techniques discussed in Section 3 allows us to find a block tridiagonal structure with more than two blocks in most cases as illustrated below.

$$
\mathrm{A} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{bmatrix}}_{\textit{original symmetric matrix}} \Rightarrow \underbrace{\begin{bmatrix} a_{11} & 0 & a_{13} & 0 & \cdots & 0 \\ 0 & a_{22} & a_{23} & a_{24} & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & a_{3n} \\ 0 & a_{42} & 0 & a_{44} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3} & 0 & \cdots & a_{nn} \end{bmatrix}}_{\textit{sparse symmetric matrix}}
$$

$$\Rightarrow \begin{bmatrix} B_1 & C_1^T & & & \\ C_1 & B_2 & C_2^T & & \\ & C_2 & B_3 & \ddots & \\ & & \ddots & \ddots & C_{p-1}^T \\ & & & C_{p-1} & B_p \end{bmatrix} = \mathrm{B}$$

block tridiagonal matrix

1.4 Synopsis. In this paper we develop a method for finding a block tridiagonal structure using a reasonable heuristic approach. We also provide an option for modifying the block structure in a way that makes it particularly suitable for the BD&C algorithm. In order to motivate this option, the BD&C algorithm is summarized briefly in Section 2. The actual block tridiagonalization algorithm is discussed in Section 3. Section 4 presents numerical results; conclusions and future work are presented in Section 5.

2. The Block Tridiagonal Divide-and-Conquer Algorithm. In this section, we first briefly describe the BD&C algorithm, and then illustrate how the block tridiagonal structure affects the time complexity of the algorithm.

Given a block tridiagonal matrix *M*, the BD&C algorithm computes eigenpairs of *M* to a prescribed accuracy:

$$M = \begin{pmatrix} B_1 & C_1^T & & & \\ C_1 & B_2 & C_2^T & & \\ & C_2 & B_3 & \ddots & \\ & & \ddots & \ddots & C_{p-1}^T \\ & & & C_{p-1} & B_p \end{pmatrix} \approx V\Lambda V^T, \tag{1}$$

where *p* is the number of diagonal blocks, *V* contains approximations to the eigenvectors of *M*, and *Λ* is a diagonal matrix containing approximations to the eigenvalues of *M*.

There are three major steps in this algorithm [6].

Step 1: Subdivision

The off-diagonal blocks C_i are approximated by lower rank matrices using their singular value decompositions:

$$C_i \approx \sum_{j=1}^{r_i} \sigma_j^i u_j^i v_j^{i\,T} = U_i \Sigma_i V_i^T, \tag{2}$$

where r_i is the chosen approximate rank of C_i, and $i = 1,2,\cdots,p-1$.

With the corresponding corrections of the diagonal blocks, the block tridiagonal matrix *M* can now be represented as:

$$M = \widetilde{M} + \sum_{i=1}^{p-1} W_i W_i^T, \tag{3}$$

where $\widetilde{M} = diag\{\widetilde{B}_1, \quad \widetilde{B}_2, \quad \cdots, \quad \widetilde{B}_p\}$,

$$\widetilde{B}_1 = B_1 - V_1\Sigma_1 V_1^T,$$

$$\widetilde{B}_i = B_i - U_{i-1}\Sigma_{i-1}U_{i-1}^T - V_i\Sigma_i V_i^T, \quad \text{for} \quad 2 \le i \le p-1,$$

$$\widetilde{B}_p = B_p - U_{p-1}\Sigma_{p-1}U_{p-1}^T,$$

$$W_1 = \begin{pmatrix} V_1\Sigma_1^{1/2} \\ U_1\Sigma_1^{1/2} \\ 0 \\ 0 \end{pmatrix}, \quad W_i = \begin{pmatrix} 0 \\ V_i\Sigma_i^{1/2} \\ U_i\Sigma_i^{1/2} \\ 0 \end{pmatrix} \text{ for } 2 \le i \le p-2, \text{ and } W_{p-1} = \begin{pmatrix} 0 \\ 0 \\ V_{p-1}\Sigma_{p-1}^{1/2} \\ U_{p-1}\Sigma_{p-1}^{1/2} \end{pmatrix}.$$

Step 2. Solve Subproblems

Each diagonal block $\widetilde{B}_i$ is factorized:

$$\widetilde{B}_i = Q_i D_i Q_i^T, \quad \text{for} \quad i = 1, 2, \cdots, p, \tag{4}$$

from which we obtain

$$\widetilde{M} = QDQ^T, \tag{5}$$

where

$Q = diag\{Q_1, Q_2, \cdots, Q_p\}$ is a block diagonal orthogonal matrix, and

$D = diag\{D_1, D_2, \cdots, D_n\}$ is a diagonal matrix.

Step 3. Synthesis

From (3) and (5) we have:

$$M = Q(D + \sum_{i=1}^{p-1} Y_i Y_i^T)Q^T, \tag{6}$$

where $Y_i = Q^T W_i$.

Denoting $S = D + \sum_{i=1}^{p-1} Y_i Y_i^T$ and $r = \sum_{i=1}^{p-1} r_i$ in the synthesis step, *S* is represented as a sequence of *r* rank-one modifications of *D*. For each rank-one modification, the modified matrix is first decomposed, and the eigenvector matrix from this decomposition is then multiplied onto the accumulated block diagonal eigenvector matrix *Q*. The r_i rank-one modifications corresponding to an off-diagonal block C_i are called one *merging operation*; thus, the algorithm performs a total of *p-1* such merging operations. The accumulation of an intermediate eigenvector matrix for each rank-one modification involves a matrix-matrix multiplication. As the synthesis phase proceeds, the matrices to be multiplied become larger and larger.

The last merging operation involves the largest matrices (see Fig. 2), and its time complexity $T(n, c, r_f)$ is a function of r_f and *c*, where r_f is the rank of the off-diagonal block in the final merging operation, and *c* and *n-c* are the block sizes of the two sub-problems to be merged.

$$\begin{array}{r} \\ c \\ \\ \\ n-c \\ \\ \end{array}\underset{Q}{\begin{pmatrix} \times&\times&\times&&&\\ \times&\times&\times&&&\\ \times&\times&\times&&&\\ &&&\times&\times&\times\\ &&&\times&\times&\times\\ &&&\times&\times&\times \end{pmatrix}} \times \underset{Q_1}{\begin{pmatrix} \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times \end{pmatrix}} \times \cdots\cdots \times \underset{Q_{r_f}}{\begin{pmatrix} \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times\\ \times&\times&\times&\times&\times&\times \end{pmatrix}}$$

Fig. 2 *Eigenvector matrix accumulation for the last merging operation*

$$\begin{aligned} T(n,c,r_f) &= 2n^3 - n^2(4c+1) + 4c^2n + (r_f - 1)(2n^3 - n^2) \\ &= 2r_f n^3 - n^2(r_f + 4c) + 4c^2 n \qquad (7) \end{aligned}$$

Based on the fact that the time complexity for the most unbalanced merging operation is less than that for the most balanced one but with higher rank [6], a block tridiagonal structure is preferred that allows for low rank modifications in the final merging operation. If there are several off-diagonal blocks with the same low rank, the BD&C algorithm chooses the one corresponding to the most balanced merging operation among these [6], i.e., the one closest to the matrix center. Optionally, the block tridiagonalization algorithm described in this paper tries to produce a few small block sizes. Note that the smaller of the dimensions of the two corresponding diagonal blocks is the best (and only) estimate available for the rank of the off-diagonal block.

3. Block Tridiagonalization. The strategy for determining the desired block tridiagonal structure utilizes several algorithmic "tools", which are described in Section 3.1. Our block tridiagonalization algorithm that uses these tools as building blocks is described in Section 3.2.

3.1 Algorithmic Tools. We use five basic algorithmic tools to construct a block tridiagonal matrix with properties 1-3 stated in Section 1.1: (1) global thresholding, (2) target thresholding, (3) sensitivity thresholding, (4) reordering, and (5) covering.
Given a matrix, the underlying goals are to

- reduce the number of nonzeros (tools 1, 2 & 3);
- reduce the bandwidth and concentrate the large nonzeros around the main diagonal (tool 4);
- impose a block tridiagonal structure that contains all the nonzero elements while being as "narrow" as possible (tool 5);
- reduce the size of some of the diagonal blocks (tool 3).

We discuss these tools in Sections 3.1.1 – 3.1.5.

3.1.1 Global Thresholding. A natural approach to decrease the number of nonzero entries in a given matrix M is to apply a threshold to all the matrix elements, i.e., setting to zero every entry m_{ij} for which $|m_{ij}| < \alpha$ with a given tolerance α. We call the resulting matrix M'. It has the property that $|m_{ij}^{'}| \geq \alpha$ for all nonzero $m_{ij}^{'}$.

The following error analysis shows that the resulting absolute eigenvalue error is bounded by $n\alpha$ (independently of the positions i, j):

Due to thresholding with tolerance α

$$M = M' + E, \quad \text{with } |e_{ij}| < \alpha, \text{ for } i, j = 1,2,\dots,n.$$

According to Weyl's theorem (see, for example, [3]), the absolute difference between the eigenvalues λ_i of M and the eigenvalues λ_i' of M' can be bounded by

$$|\lambda_i - \lambda_i'| \leq \|E\|_2.$$

Since E is symmetric, its 2-norm equals the maximum of the absolute values of its eigenvalues, which is smaller than any matrix norm induced by a vector norm. In particular, $\|E\|_2 \leq \|E\|_1$ and $\|E\|_1 \leq n\alpha$. Therefore,

$$|\lambda_i - \lambda_i'| \leq n\alpha.$$

3.1.2 Target Thresholding. Ultimately, we are interested in setting all the elements far away from the main diagonal in a given matrix *M* to zero. As soon as an element m_{ij} is found that can not be safely set to zero, we are less interested in eliminating elements closer to the diagonal since those elements will typically be included in any block tridiagonalization of *M*. Thus, thresholding with a given tolerance α is applied to *M* to produce M' such that $m_{ij}' = 0$ for $|i - j| > k_j$ for some integer k_j, which may be different for each column j and hopefully $k_j << n$ for all *j*, and

$$M = M' + E, \text{ with } \sum_{i=1}^{n} |e_{ij}| < \alpha \text{ for } j = 1,2,\dots,n.$$

Using Weyl's theorem as above, we can now show that the absolute difference between the eigenvalues λ_i of *M* and the eigenvalues λ_i' of M' can be bounded by

$$|\lambda_i - \lambda_i'| \leq \|E\|_1 \leq \alpha.$$

3.1.3 Sensitivity thresholding. With some additional knowledge, it may be possible to eliminate some of the matrix elements whose absolute value is even larger than $\tau\|\mathrm{A}\|$, the specified normalized accuracy tolerance for the eigenvalues, without causing the accumulative error in the eigenvalues to exceed this error. Wilkinson [21] has given a sensitivity analysis that expresses the eigenvalues of a perturbed matrix in terms of the eigenvalues and eigenvectors of the original matrix and of the perturbation:

$$\lambda(M + \varepsilon E) = \lambda(M) + \varepsilon(x^T E x) + O(\varepsilon^2), \tag{8}$$

where *x* denotes the eigenvector corresponding to the eigenvalue $\lambda(M)$ of M.

Based on (8), we can estimate the first order eigenvalue error that results from dropping matrix elements at specific locations (note that the position of an element to be dropped enters into the matrix E).

$$\Delta\lambda := \lambda(M + \varepsilon E) - \lambda(M) \approx \varepsilon(x^T E x) \tag{9}$$

Eliminating a Single Matrix Element. For estimating the eigenvalue error resulting from dropping the element m_{ij} (and, symmetrically, m_{ji}), E is a matrix with entries -1 at the positions (i, j) and (j, i), zero entries at all other positions, and $\varepsilon = m_{ij}$ in Eqn. (9). This yields the estimate

$$\Delta\lambda = 2m_{ij}x_i x_j + O(m_{ij}^2) \qquad (10)$$

for the effect of dropping matrix element m_{ij} on the eigenvalue λ (x_i, x_j are the *i-th* and *j-th* entries, respectively, of the eigenvector *x* corresponding to λ).

Eliminating Several Matrix Elements. An important situation arising frequently in our algorithm is to estimate the accumulative error on the eigenvalues from eliminating a row or a column from an off-diagonal block in a block tridiagonal matrix *M*. For estimating this error, we need to accumulate the error estimates contributed by all matrix elements eliminated for each eigenvalue and then take the maximum over all the eigenvalues. If this maximum error estimate exceeds the accuracy tolerance, then the row or column considered cannot be eliminated.

Error on eigenvalue from eliminating row *i*:

$$\Delta\lambda = 2x_i\left(\sum_{j=col\_start}^{col\_end} m_{ij}x_j\right)$$

Error on eigenvalue from eliminating column j:

$$\Delta\lambda = 2\left(\sum_{i=row\_start}^{row\_end} x_i m_{ij}\right)x_j$$

Note that the cumulative errors do not always grow with each element eliminated since the summation may involve opposite signs.

To illustrate how sensitivity thresholding can help to reduce the size of diagonal blocks, assume that elements in the shaded area of Fig. 3(a) and its symmetric lower part can be dropped. This reduces the size of the third diagonal block as shown in Fig. 3(b).

Further split a block. The reduction of a block size implies the expansion of a neighboring block. An expanded block may be further divided into two sub-blocks if the corresponding sub-divided off-diagonal blocks are able to cover all the nonzero elements of the original off-diagonal block, as illustrated in Fig. 3(c).

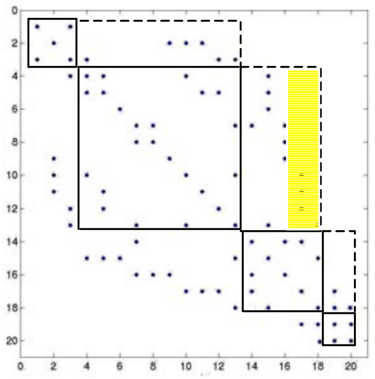


Fig. 3(a) *Original block tridiagonal structure*

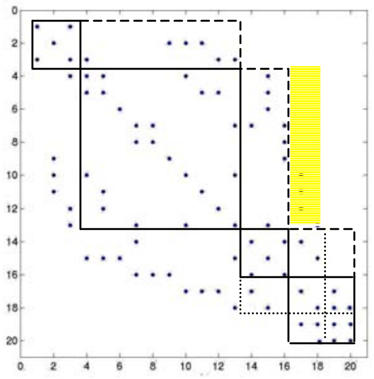


Fig. 3(b) *Block structure after elimination of shaded elements*

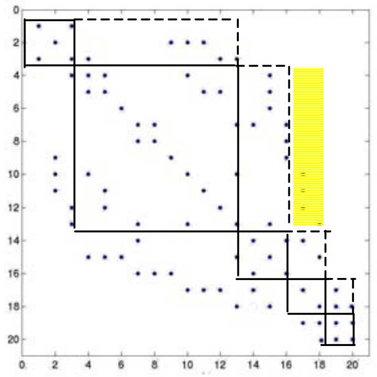


Fig. 3(c) *Block structure after sub-division of expanded block*

3.1.4 Reordering for Bandwidth Reduction. In order to "compress" a matrix M toward its diagonal, we can reorder M , trying to reduce its bandwidth. Several matrix reordering algorithms are available, for example, Cuthill-McKee [7], reverse Cuthill-McKee [7], Gibbs-Poole-Stockmeyer (GPS) [1,9,11], Gibbs-King [8,11] and Sloan [18,19]. All of them are based on level-set orderings to reduce the profile or bandwidth of a symmetric sparse matrix.

From the definitions of the bandwidth *b* and the profile *f* [7] of a matrix

$$b = \max\{|i-j|, \left|m_{ij}\right| \neq 0\},\ 1 \leq i, j \leq n, \tag{11}$$

and

$$f = \sum_{i=1}^{n} i - g(i), \tag{12}$$

where $g(i) = \min\{j \mid m_{ij} \neq 0\}$, $1 \leq i \leq n$, one notices subtle differences between the two. Although a reduction in profile usually leads to a reduction in bandwidth, it is possible that a near minimum profile corresponds to a large bandwidth (for example, if in a single row the last nonzero element is far away from the diagonal). Since our ultimate goal is to generate blocks of small sizes, we focus on reducing bandwidth.

Among the reordering algorithms mentioned above, the GPS algorithm specifically targets bandwidth reduction and, therefore, is our method of choice. The effects of applying this reordering technique to our test matrices are illustrated in Section 3.2.

3.1.5 "Covering" Problem. Given a sparse symmetric matrix M , we want to find a block tridiagonal structure that contains all the nonzero elements of M. We determine initial block sizes $k_1, k_2, \ldots, k_p$ using a straightforward strategy:

- Size of *diagonal block 1*: inspect the first row of M and determine the diagonal block size k_1 from the column index of the last non-zero element in the first row.
- Size of *diagonal block i* for $2 \le i \le p-1$: assume *diagonal block i-1* starts at row *j* and ends at row *k*. Then *diagonal block i* starts at row *k+1*. The column index of the last non-zero element in row *k+1* determines the end of *diagonal block i* and therefore the diagonal block size k_i .
- Size of *diagonal block p*: include all the rows after *diagonal block p-1*.

Sometimes a diagonal block needs to be expanded to ensure that the corresponding off-diagonal blocks cover all the existing non-zero elements or may be reduced and still cover all the necessary non-zeros. An example of this process is shown in Figs. 4(a) – (d). Since we always consider symmetric matrices, we focus on the upper triangular part in the following figures.

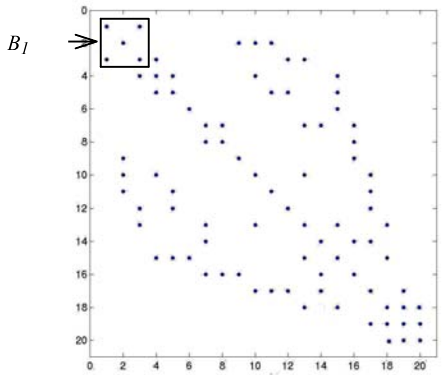


Fig. 4(a) *First diagonal block B_1*

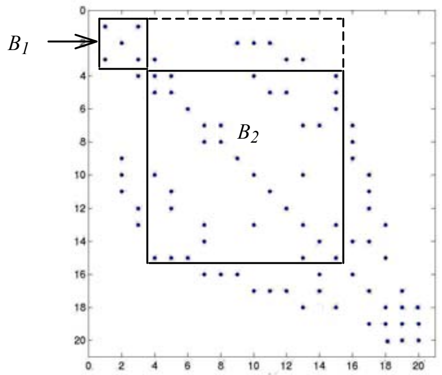


Fig. 4(b) *Second diagonal block B_2*

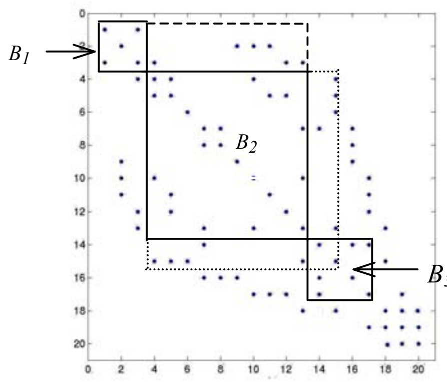


Fig. 4(c) *Reduction of B_2*

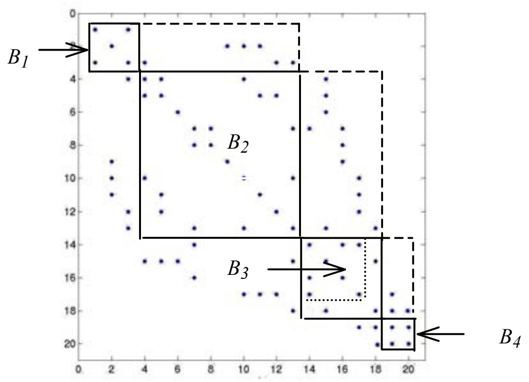


Fig. 4(d) *Expansion of B_3*

3.2 Block Tridiagonalization Algorithm. Our algorithm utilizes the algorithmic tools 3.1.1 to 3.1.5 in a specific order and is heuristic in nature. Given a symmetric matrix A and an accuracy tolerance $\tau < 0.1$, it proceeds in six steps and produces a block tridiagonal matrix with the properties described in Section 1.1. In steps 4 and 6, the accuracy tolerance τ is partitioned as $\tau = \tau_1 + \tau_2$, allowing a portion of the acceptable error to be used for target thresholding and sensitivity thresholding, respectively.

Step 1. Global Threshold A with $\sqrt{\tau}\,\|\mathrm{A}\|$

We start with a threshold $\tau' = \sqrt{\tau}$, larger than permitted by the accuracy requirement and obtain matrix A' by eliminating all elements in A less than $\sqrt{\tau}\,\|\mathrm{A}\|$. Thus, for many matrices arising from applications with strong locality properties, most of the elements will be eliminated. The resultant matrix A' will contain only the largest elements of A and (hopefully) be sparse.

Step 2. Reorder A'

The GPS algorithm (see Section 3.1.4) is used to reorder A' and reduce its bandwidth. Thus, the larger matrix elements of A are moved toward the diagonal. A resultant permutation matrix P is obtained and will be used in Step 3.

Step 3. Reorder original A with the permutation matrix P from Step 2.

The matrix A may have some of its larger elements far from the diagonal, which would yield rather big blocks for the block tridiagonalization. In an effort to move its largest elements toward the diagonal, the permutation matrix computed in Step 2 is applied, resulting in matrix $A'' = P^T AP$.

Based upon the characteristics of A and the accuracy tolerance τ, the effectiveness of the above steps varies. We investigate two types of matrices:

1) The Fock matrix for the linear $C_{322}H_{646}$ alkane molecule (Fig. 1) already has most of its largest elements close to the diagonal, and therefore A' is already a banded matrix.

In this case, the bandwidths of A' before and after the reordering are about the same. Tests on other Fock matrices with similar properties produce, as expected, matrices from Steps 1-2 with equal or nearly equal bandwidths. Application of steps 1-3 to a linear $C_{322}H_{646}$ alkane molecule is demonstrated in Figs. 5(a) – (c).

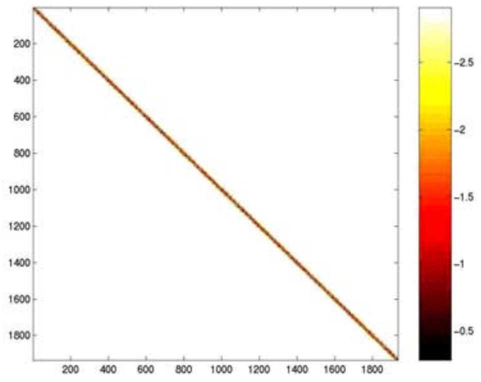


Fig. 5(a) A' *($\tau' = 10^{-3}$, b=24)*

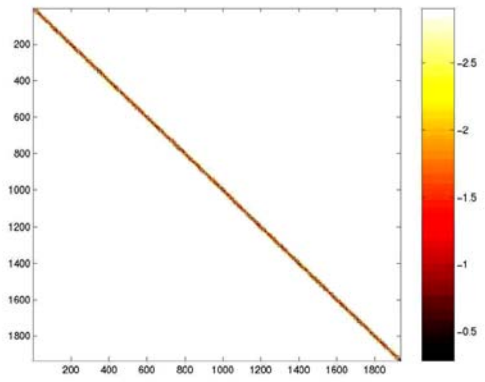


Fig. 5(b) *Reordered* A' *(b=24)*

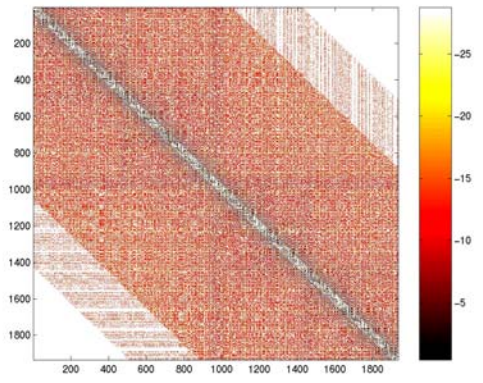


Fig. 5(c) A'' *(Permuted* A *)*

2) Figure 6(a) shows an example of a matrix A whose largest elements are not all close to the diagonal. The matrix was generated by randomly permuting a matrix with large elements close to the diagonal. In this case, the bandwidth of A' after reordering as compared to before reordering is greatly reduced. Matrices A', the reordered A', and A'' are shown in Figs. 6(b) – (d).

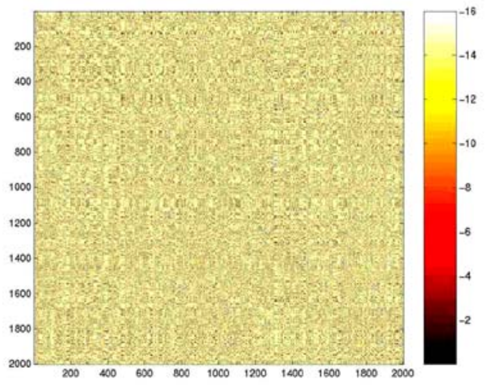


Fig. 6(a) *$\log_{10}$ of absolute value of A (random matrix)*

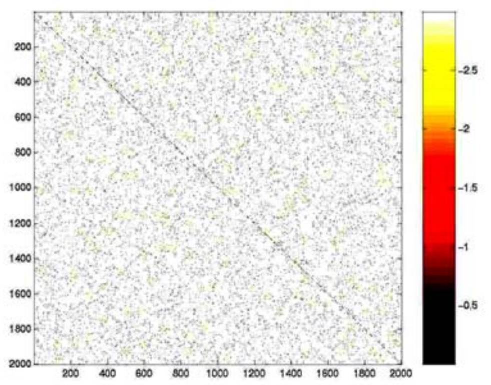


Fig. 6(b) *A' ($\tau' = 10^{-3}$, b=2000)*

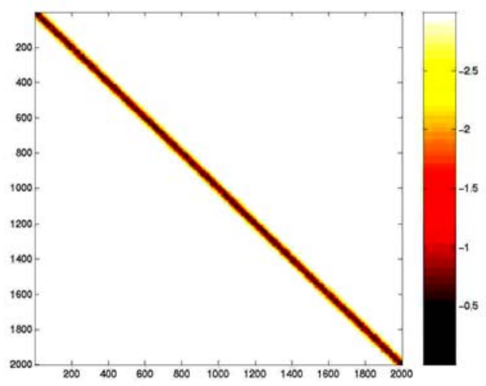


Fig. 6(c) *Reordered A' (b=53)*

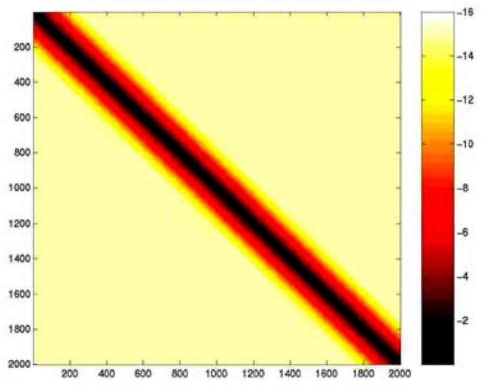


Fig. 6(d) *A'' (Permuted A)*

Step 4. Target Threshold A'' with $\tau_1 \|\mathrm{A}\|$

The goal is to form a matrix A''' by eliminating all elements far away from the diagonal in the permuted full matrix A'' whose influence on the error of any eigenvalue is negligible compared to $\tau_1 \|A\|$. Since the eigenvalue errors are bounded by the 1-norm of the error matrix (see Section 3.1.2), we monitor the accumulative errors by columns as we eliminate elements.

Because our matrix is symmetric, we process the lower triangular part of the matrix. In order to preserve symmetry, dropping an element a_{ij} in the lower triangular part of column *j* implies that its symmetric upper triangular counterpart a_{ji} must also be dropped in column *i*, as illustrated in Fig. 7. Therefore, an element can be dropped only when the accumulative error it incurs is less than $\tau_1 \|A\|$ in *both* columns *j* and *i*.

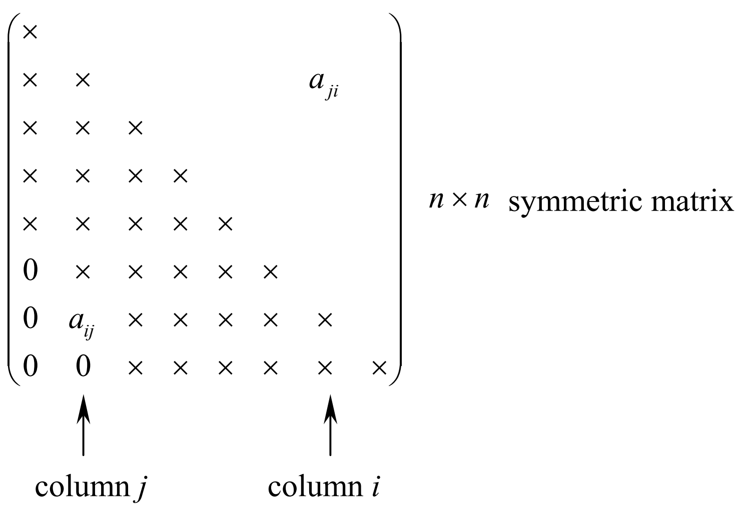


Fig. 7 *Eliminate an element and its symmetric counterpart from* A''

There are several methods one can use to systematically eliminate contiguous small elements of A'' beginning at the (n,1) position. In the following, we present three such algorithms in this paper.

Target Column Thresholding. This is a straightforward algorithm based on the algorithmic tool described in Section 3.1.2. For each consecutive column of A'' beginning with the first, eliminate elements of the column from the bottom toward the diagonal until the accumulative sum of the absolute value of the eliminated elements exceeds $\tau_1 \|A\|$. The eigenvalue errors are then bounded by $\tau_1 \|A\|$ (see Section 3.1.2).

A potential problem with this algorithm is that, for column *j*, dropping a relatively large element a_{ij} ($i > j$) adds error to column *i* and thus may prevent further elimination of elements in column *i*. Two typical resultant anomalous cases are: 1) the banded matrix after thresholding flares out at the bottom, as shown in Fig. 8(a); and 2) the matrix after thresholding has long spikes as shown in Fig. 8(b), which produces a block tridiagonalization with very large - usually sparse - blocks.

In order to avoid this, it is possible to set a separate, more conservative error bound for the upper triangular part of the matrix when dropping an element. In other words, a_{ij} can be dropped only when the sum of the dropped elements for column *j* is less than $\tau_1 \|A\|$, and less than $\tau_1 \|A\| (i-1)/n$ for column *i*. Figures. 8(c) – (d) show the improved matrix structures. However, this approach is not as competitive as the other target thresholding algorithms described below.

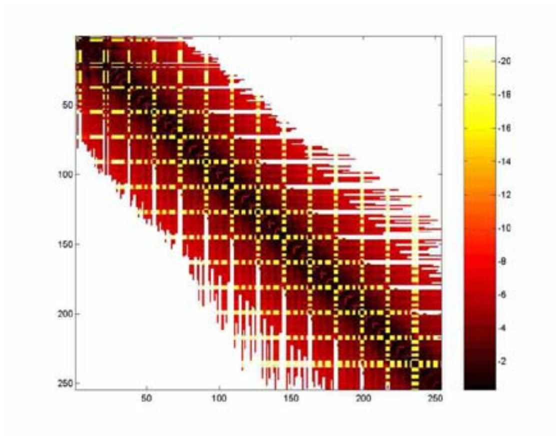


Fig. 8(a) *A''' with flare ($\tau_1 = 10^{-6}$, b=124)*

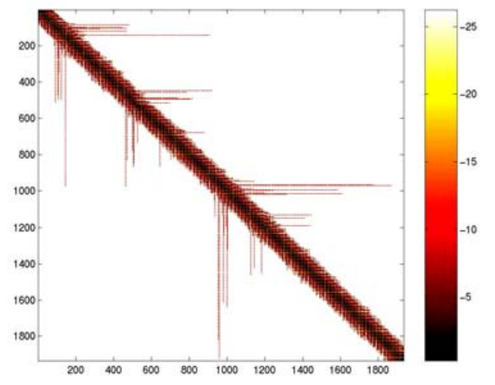


Fig. 8(b) *A''' with spikes ($\tau_1 = 10^{-6}$, b=964)*

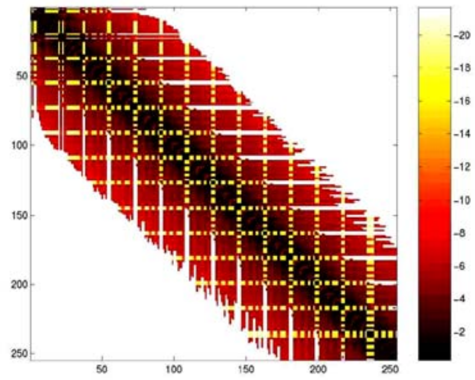


Fig. 8(c) *A''' from Fig. 8(a) with conservative error bound ($\tau_1 = 10^{-6}$, b=90)*

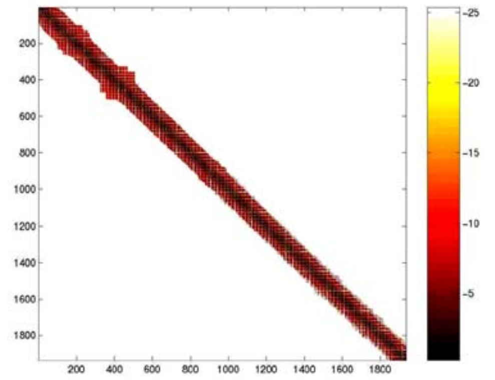


Fig. 8(d) *A''' from Fig. 8(b) with conservative error bound ($\tau_1 = 10^{-6}$, b=147)*

Target Diagonal Thresholding (TDT). As illustrated in Fig. 9(a), this algorithm traverses the matrix elements along the off-diagonals from the end toward the center, and drops elements as long as none of the column-wise sums of absolute values of the dropped elements exceeds $\tau_1\|A\|$. Figure 9(b) shows the matrix from Fig. 1 after thresholding with TDT.

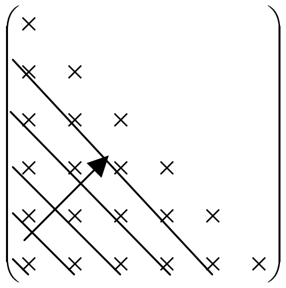


Fig. 9(a) *Traverse elements along matrix off-diagonals*

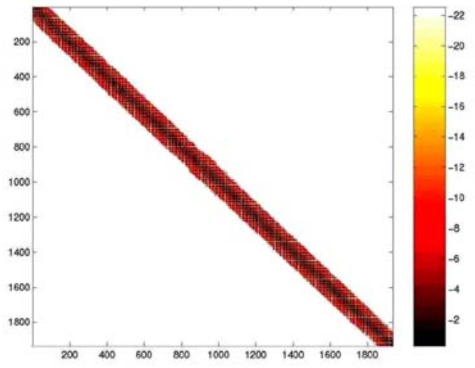


Fig. 9(b) A''' *($\tau_1 = 10^{-6}$, b=104) with TDT*

Target Block Thresholding (TBT). As another variation, one could traverse the matrix elements row-wise and column-wise alternately, as illustrated in Fig. 10(a). The column nonzero pattern produced by this approach tends to be small in the middle and large at both ends. Thus, the block tridiagonal structure typically has fewer blocks with small blocks near the center and fairly large blocks at the ends. The motivation for such a structure is to produce a blocking with a few small blocks (therefore, low ranks in the off-diagonal blocks). Figure 10(b) shows the matrix from Fig. 1 after thresholding with TBT.

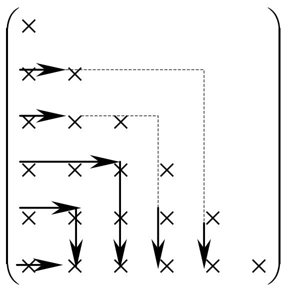


Fig. 10(a) *Traverse elements row- and column-wise alternately*

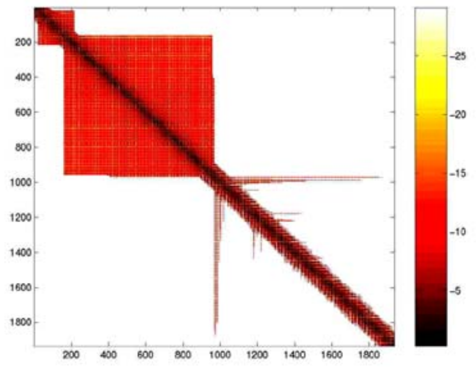


Fig. 10(b) A''' *($\tau_1 = 10^{-6}$, b=932) with TBT*

Step 5. "Cover" A''' (i.e., determine a block tridiagonal structure for A''')

Through a row-wise procedure as described in Section 3.1.5, the diagonal block sizes are determined such that the resulting block tridiagonal matrix contains all the matrix elements that are "effectively nonzero" (i.e., nonzeros in A'''). These are the matrix elements whose effect on the accuracy of the eigenpair approximation may be non-negligible.

The algorithm composed only of Steps 1-5 will be called the *basic algorithm* and used for comparison in later numerical tests.

It can be very beneficial for the BD&C algorithm to have a few very small diagonal blocks (see Section 2). Thus, a Step 6 as described below is added to the basic algorithm to form the *Target Block Reduction algorithm* (TBR) (note that eigenvector approximations will be required for this step). For other eigensolvers, one may use just the basic algorithm or replace Step 6 in the TBR algorithm with a step appropriate for the different eigensolver. For example, if a band eigensolver will be used, one would try to remove the outer-most corners of the largest diagonal blocks by a similar method to the one described below, yielding a matrix with a smaller bandwidth.

Step 6. Reduce block sizes using $\tau_2\|A\|$ if possible

In this last step, remaining elements whose removal may reduce the size of the smaller interior blocks (see the next paragraph for a discussion on why elements in the first and the last off-diagonal blocks are not to be eliminated) are checked individually for elimination.

Apply sensitivity threshold. If the block tridiagonalization is used as a preprocessing step for the BD&C eigensolver, we obviously do not know the eigenvector x needed for the sensitivity thresholding as described in Section 3.1.3 *a priori*, since it is one of the quantities we want to compute. However, we can use an approximative eigenvector instead if we have some information about the approximation quality.

In the context of an iterative method for solving a nonlinear eigenvalue problem (like the SCF method), we have another option: after the first iteration, use the eigenvector from the previous iteration as an approximation of x in the error estimate (9). In this situation, even if we do not have precise information about the accuracy of this eigenvector approximation, underestimating the eigenvalue error and therefore wrongly dropping some elements is typically corrected as the iterative method proceeds.

Starting with a block tridiagonal matrix (take Fig. 3(a) as an example), first select the smallest diagonal block and apply sensitivity thresholding to the corresponding off-diagonal block. If there are several diagonal blocks with the same size, then select the one closest to the middle.

An exception to this rule is that neither the first nor the last block should be a candidate for sensitivity thresholding. Observe that a block at either end of a matrix (i.e., the first or last block) can be arbitrarily reduced until its size reaches zero, which means that its neighboring block is expanded. The result is that the block at the end is combined with its neighbor to form a bigger block, which is contradictory to our goal of generating some small blocks. Therefore, sensitivity thresholding is not applied to the first or last block.

Since each block has two neighbors, to obtain a more balanced block tridiagonal structure, we reduce the side of the block that expands the smaller of the two neighbors first, then the other side. After a block has been reduced, it should not be expanded in later steps. Otherwise, a block could be reduced first and expanded later repeatedly in an oscillating pattern. We repeat the sensitivity thresholding procedure for the next smallest block until all eligible blocks are processed. Fig. 3(b) shows an example of the resultant block tridiagonal structure.

Check for split blocks. All the blocks that have been expanded during the sensitivity thresholding process are checked for the possibility of splitting into multiple blocks. If a block can be sub-divided into two smaller blocks as illustrated earlier by Fig. 3(c), and the smaller diagonal and off-diagonal blocks still cover all the corresponding nonzero elements, then the splitting is implemented by changing the block sizes and increasing the number of blocks by 1. Figures 11(a) – 11(c) with block sizes shown along the x-axes illustrate the split of a block using a matrix generated from modeling an $C_{502}H_{1006}$ alkane molecule.

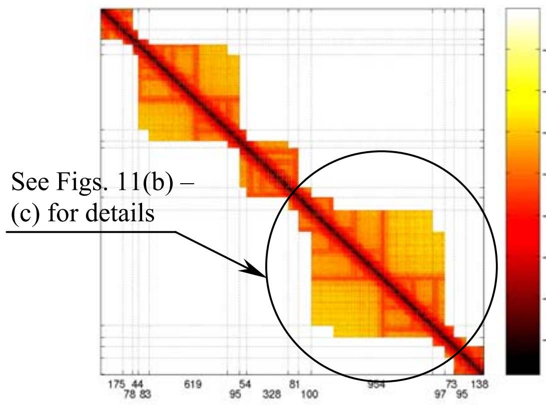
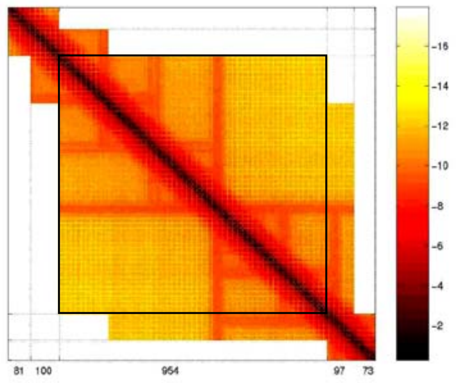


Fig. 11(a) *Split a block*



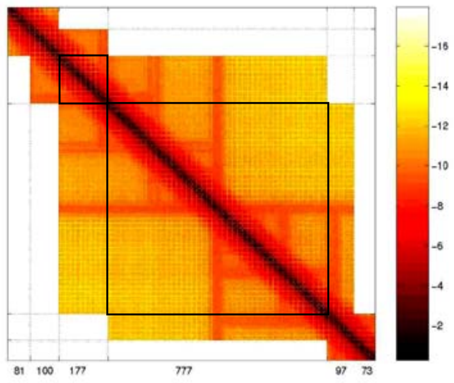


Fig. 11(b) *Local block tridiagonal structure before splitting*

Fig. 11(c) *Local block tridiagonal structure after splitting*

Combine blocks. Since the first and last blocks are never candidates for sensitivity thresholding, they would only be expanded but never reduced. Computational time complexity of the BD&C could increase in the following scenario. The first (or last) block is the smallest block before sensitivity thresholding. After sensitivity thresholding, it is still the smallest one, but its size increases and the same could be true for the rank of the corresponding off-diagonal block. The block is expanded because its neighbor has been reduced. Consequently, the time complexity of the last merging operation becomes higher.

Under certain restrictions, combination of blocks would not increase the total time complexity of all merging operations, although as a general rule one should always try to produce smaller blocks. If the smallest block is at the ends of a matrix, analysis based upon the leading term of the time complexity of a single merging operation [6] shows that, if the following condition

$$r_{f-1} \le \frac{r_f}{1-\left(1-\frac{k_{\min}}{n}\right)^3} - 1, \tag{13}$$

holds, where $k_{\min}$ is the size of the smallest block, n is the size of the symmetric matrix, r_f is the rank in the last merging operation, and r_{f-1} is the rank in the second last merging operation, then the total time complexity of all merging operations decreases through the combination of the first (or last) block with its neighboring block.

Because ranks of the off-diagonal blocks are not available during the block tridiagonalization process, we use the size of the smaller diagonal block as an estimate of the rank of the corresponding off-diagonal block. Thus, if either the first block or the last one is the smallest block after sensitivity thresholding and inequality (13) holds with ranks replaced by block sizes, then we combine it with its neighboring block to form a larger block. By doing this, the total number of blocks p is decremented by 1, and the last merging operation is eliminated.

Final block tridiagonal structure. After block sizes are determined, the diagonal blocks and off-diagonal blocks of the block tridiagonal structure are obtained by filling in blocks with elements from the original matrix *A*. Data defining the permutation matrix *P* computed in Step 2 is also returned by the block tridiagonalization routine to enable proper back-transformation of the eigenvectors computed by the BD&C algorithm. Fig. 12 shows the block tridiagonal matrix resulting from applying the TDT and TBR algorithms to the matrix shown in Fig. 1, with $\tau = 10^{-6}$ and $\tau_1 = \tau_2 = 0.5\tau$. The figure shows log10 of the absolute value of the matrix entries, and the sizes of the diagonal blocks are shown along x-axis.

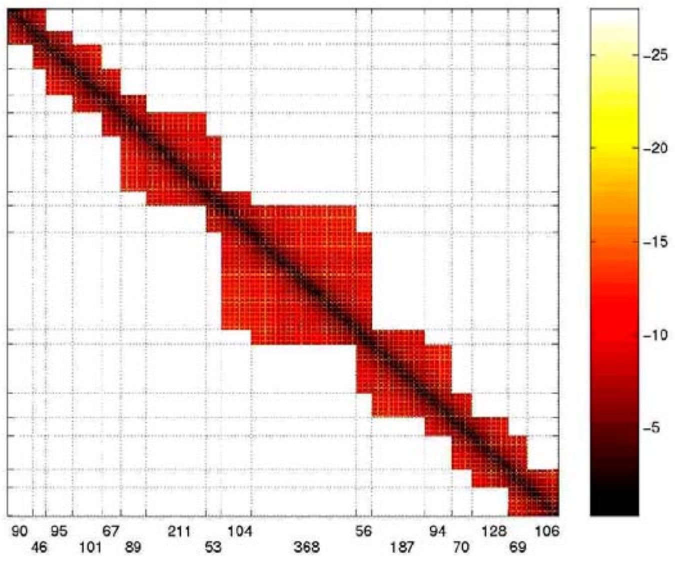


Fig. 12 *log_{10} of block tridiagonal matrix from $C_{322}H_{646}$ alkane with* $\tau = 10^{-6}, \tau_1 = \tau_2 = 0.5\tau$

4. Numerical Tests. The runtimes of the block tridiagonal divide-and-conquer eigensolver on the block tridiagonal structure determined by various potential block tridiagonalization algorithms described in Section 3.2 are compared to illustrate the characteristics of each.

4.1 Effectiveness of sensitivity thresholding. First, we test the effectiveness of Step 6 for the BD&C eigensolver. We compare the BD&C execution time on the block structure determined by the basic algorithm with the time on the structure determined by the TBR algorithm. In addition, the times required for our block tridiagonalization algorithms are measured to evaluate their overhead and to compare with the eigensolver time. With an accuracy tolerance τ, 0.5τ is assigned to τ_1 and τ_2 for target thresholding and sensitivity thresholding, respectively, for the TBR algorithm.

Test matrices originate from solving the Hartree-Fock equation for the modeling of alkane molecules $C_{162}H_{326}$ (both ordered and disordered) and $C_{322}H_{646}$ in quantum chemistry. The matrix sizes are 974, 974 and 1934 respectively. A random matrix of size 2000 as shown in Section 3.2 (Fig. 6(a)) is also tested. The experiments were performed on a 450 MHz UltraSparc-II with 4 MB off-chip cache and 512 MB main memory.

Because solving the Hartree-Fock equation is an iterative process and the eigenvector approximations from the previous iteration are used for sensitivity thresholding, performance is evaluated from iteration 2 to the iteration of convergence (in these test cases, iteration 6). The average execution time of the 5 iterations is provided for different computational accuracies as shown.

Table 1 summarizes this average time for solving eigenproblems with the basic block tridiagonalization algorithm and the TBR algorithm, while Table 2 summarizes the average time for block tridiagonalization. Tolerances τ are set to 10^{-6} and 10^{-9}.

In Table 1, with the TBR algorithm, the average execution time for solving the eigenproblem is less than the basic algorithm in most cases. With TBT, the improvement in performance with TBR is more obvious than with TDT. Two cases of significant degradation in performance are observed in individual iterations for $C_{322}H_{646}$ using a combination of TBR and TDT for $\tau = 10^{-6}$ and $\tau = 10^{-9}$ (see Figs. 13(a) and 13(c)), and we will discuss them in more detail later. Table 2 illustrates that the average time for block tridiagonalization is less than 7% of the average time for the eigensolver.

Table 1.
Average Execution Time of BD&C (in seconds)

		TDT		TBT	
		Basic algorithm	TBR algorithm	Basic algorithm	TBR algorithm
$\tau = 10^{-6}$	$C_{322}H_{646}$	402.24	398.23	551.98	381.43
	$C_{162}H_{326}$	61.99	68.69	56.75	47.54
	$C_{162}H_{326}$ disordered	109.57	104.16	98.93	91.85
	Random	558.99	503.97	580.63	493.48
$\tau = 10^{-9}$	$C_{322}H_{646}$	606.51	819.81	1272.08	609.54
	$C_{162}H_{326}$	84.06	88.18	145.18	70.75
	$C_{162}H_{326}$ disordered	198.85	191.19	189.16	175.77
	Random	1826.98	1644.12	1809.05	1585.93

Table 2.
Average Execution Time of Block Tridiagonalization (in seconds)

		TDT		TBT	
		Basic algorithm	TBR algorithm	Basic algorithm	TBR algorithm
$\tau = 10^{-6}$	$C_{322}H_{646}$	4.74	12.22	10.56	18.37
	$C_{162}H_{326}$	1.26	2.09	1.89	3.13
	$C_{162}H_{326}$ disordered	1.29	2.86	2.03	2.82
	Random	9.09	14.39	8.78	13.95
$\tau = 10^{-9}$	$C_{322}H_{646}$	3.13	5.82	6.97	16.08
	$C_{162}H_{326}$	0.98	1.36	1.78	2.37
	$C_{162}H_{326}$ disordered	1.12	2.15	1.92	2.96
	Random	12.26	15.73	12.70	17.97

To further compare individual iterations, the ratio of the execution time of the BD&C eigensolver using the basic algorithm to the time using the TBR algorithm is plotted in Figs. 13(a) – 13(d). From these figures we can see that performance is increased in most cases, frequently by more than a factor of two.

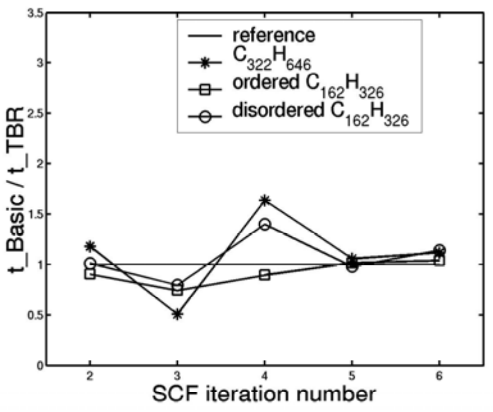


Fig. 13(a) *Ratio of execution times with* $\tau = 10^{-6}$ *and TDT*

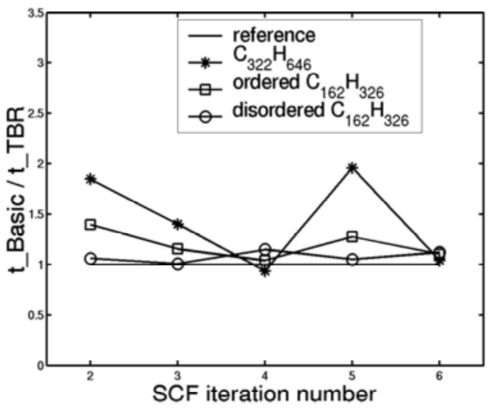


Fig. 13(b) *Ratio of execution times with* $\tau = 10^{-6}$ *and TBT*

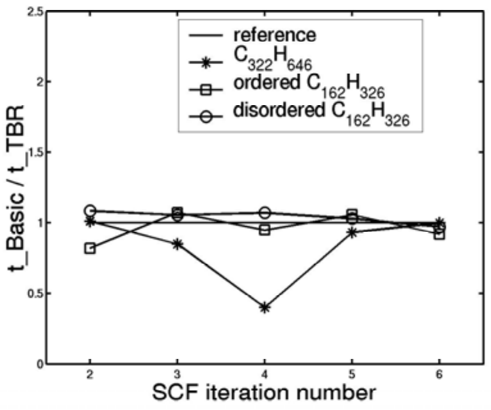


Fig. 13(c) *Ratio of execution times with* $\tau = 10^{-9}$ *and TDT*

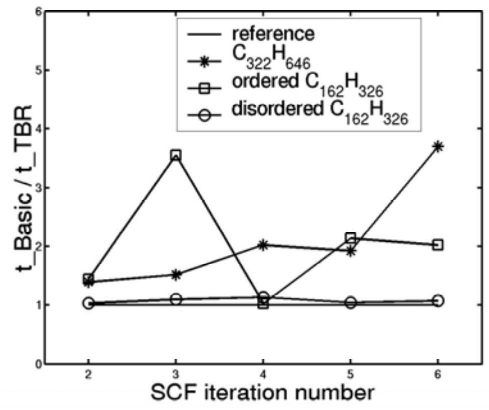


Fig. 13(d) *Ratio of execution times with* $\tau = 10^{-9}$ *and TBT*

There are two cases of significantly degraded performance. The first case is iteration 3 of $C_{322}H_{646}$ alkane with $\tau = 10^{-6}$ in Fig. 13(a). In this case, the performance of the TBR algorithm is worse than that of the basic algorithm, although the rank of the off-diagonal block for the final merging operation remains the same. This phenomenon can be explained by the difference in deflation encountered by the algorithms.

In the BD&C algorithm[6], a merging operation is equivalent to r_i steps of decomposition and accumulation of the rank-one modifications $D + y_i y_i^T$, where the y_i are the vectors that determine the Y_i in Eqn. 6. Deflation happens when there is either a zero (or small) component in y_i or two equal (or close) entries in D. When deflation occurs, we know that the corresponding eigenvector is a unit vector, and therefore no computation is required to compute and accumulate it. Deflation reduces the amount of work in the matrix multiplications for accumulating eigenvectors in the last merging operation.

Without deflation, merging of the off-diagonal block with lower rank will always have lower time complexity. However, with deflation, an off-diagonal block with higher rank might require less work to accumulate eigenvectors than an off-diagonal block with lower rank if the former has more deflation. In the case of iteration 3 of $C_{322}H_{646}$ in discussion, with the basic algorithm, the amount of deflation is 63%, while with SBT, it's only 41%, although both off-diagonal blocks have the same rank.

The second case is iteration 4 of $C_{322}H_{646}$ alkane with $\tau = 10^{-9}$ in Fig. 13(c). It turns out that the TBR algorithm produces a block tridiagonal structure in which the first off-diagonal block has the lowest rank, although neither of the corresponding diagonal blocks is the smallest one. The smallest diagonal block is found at the opposite end of the diagonal - the next to last block. Thus, the TBR algorithm, targeting the diagonal blocks for reduction in priority order by size, was non-optimal for the BD&C algorithm.

In addition and more importantly, the basic algorithm produces a block tridiagonal structure with the off-diagonal block of minimum rank located near the center of the matrix. Given a block tridiagonal structure, the BD&C algorithm always determines the optimal merging order by selecting the off-diagonal block with minimum rank for the final merging operation. Thus, the final merging operation resulting from the TBR algorithm is extremely unbalanced, whereas the final merging operation resulting from the basic algorithm is nearly optimally balanced, which explains the longer runtime for the structure resulting from the TBR algorithm.

Since we cannot predict the amount of deflation and the ranks of off-diagonal blocks beforehand, there is no effective method to correct those problems. Fortunately, such cases do not happen frequently.

4.2 Comparison of Thresholding Algorithms. For TDT and TBT described in Section 3.2, numerical tests show that TDT produces a smaller bandwidth while TBT generally yields quicker execution by the BD&C eigensolver as will be shown below.

Figures 14(a) and (b) plot the ratio of the execution time of the BD&C eigensolver using TBT to that using TDT. Here the TBR algorithm is used for block tridiagonalization. In most cases, TBT outperforms TDT, although performances vary in individual cases.

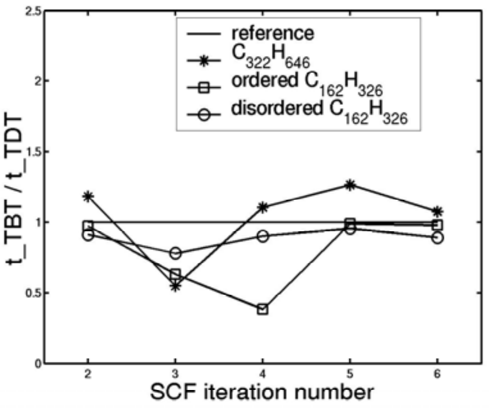


Fig. 14(a) *ratio of execution times using TBR with* $\tau = 10^{-6}$

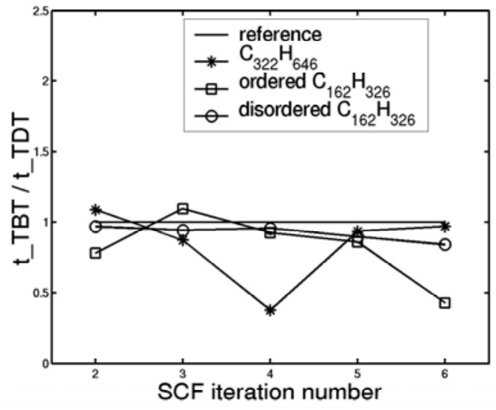


Fig. 14(b) *ratio of execution times using TBR with* $\tau = 10^{-9}$

In Table 3, we list the average memory requirements of iterations 2 – 6 for storing the diagonal and sub-diagonal blocks generated by TDT and TBT. Recall in Section 3.2, matrix A''' produced by TBT tends to flare out toward both ends, that is, produce small blocks in the middle and large blocks at the ends. Since storage requirements for a dense matrix increase quadratically with its dimension, large matrices require proportionally more storage space than small matrices. Consequently, TBT is expected to require more space to store the blocks than TDT. Table 3 shows that the storage space needed on average by the block tridiagonal matrix produced by TBT in our test cases is always more than that needed by TDT. In the worst case, TBT requires almost twice as much space as TDT.

Table 3.
Average Storage Space Required to Store Diagonal and Off-diagonal Blocks (in MB)

		TDT	TBT
$\tau = 10^{-6}$	$C_{322}H_{646}$	7.49	14.65
	$C_{162}H_{326}$	2.25	3.42
	$C_{162}H_{326}$ disordered	2.90	3.63
$\tau = 10^{-9}$	$C_{322}H_{646}$	8.37	13.73
	$C_{162}H_{326}$	3.04	3.65
	$C_{162}H_{326}$ disordered	3.42	4.01

If storage space is not an issue, the TBT algorithm combined with the TBR algorithm is generally more suitable for the BD&C eigensolver because of its better performance. Otherwise, TDT is recommended because it typically produces a block tridiagonalization with fewer variations in block sizes and smaller blocks and requires considerably less memory.

5. Conclusion. A heuristic block tridiagonalization method has been developed for determining a symmetric block tridiagonal matrix that is closely related to a given symmetric matrix in the sense that the eigenvalues of the block tridiagonal matrix differ at most by a prescribed normalized accuracy tolerance $\tau\|A\|$ from the eigenvalues of the given matrix.

Block tridiagonalization is an important preprocessing step for the block tridiagonal divide-and-conquer eigensolver introduced in [5,6]. In this context, it is desirable to have low rank off-diagonal blocks. In general, this may be hard or impossible to achieve (given the accuracy tolerance). However, the block tridiagonalization method described in this paper attempts to make at least one off-diagonal block for the last merging operation as small as possible, assuming that a smaller block tends to have lower rank than a larger block (although this is not always the case). Decreasing the size of a diagonal block, thus hopefully the rank of the related off-diagonal block, has priority over the location of the small block, based on the analysis of time complexity [6]. Numerical experiments show that:

(1) The overhead of block tridiagonalization is negligible compared to the time for solving the eigenproblem.
(2) The non-unique block tridiagonal structure can have a significant influence upon the run time of the BD&C algorithm.
(3) The algorithm with the BD&C algorithm combine to perform very well on problems requiring low accuracy solutions, which appear in important applications, for example, in quantum chemistry problems.

References.

[1] H. L. Crane Jr., N. E. Gibbs, W. G. Poole Jr., and P. K. Stockmeyer, *Matrix Bandwidth and Profile Reduction*, ACM Trans. Math. Softw., 2 (1976), pp. 375-377.

[2] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195.

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

[4] W. N. Gansterer, D. F. Kvasnicka, and C. W. Ueberhuber, *Multi-sweep algorithms for the symmetric eigenproblem*, in VECPAR'98 – Third International Conference for Vector and Parallel Processing, J. M. L. M. Palma, J. J. Dongarra, and V. Hernandez, eds., Lecture Notes in Computer Science, Vol 1573, Springer-Verlag, New York, 1998, pp. 20-28.

[5] W. N. Gansterer, R. C. Ward, and R. P. Muller, *An Extension of the Divide-and-Conquer Method for a Class of Symmetric Block-Tridiagonal Eigenproblems*, ACM Trans. Math. Softw. 28 (2002), pp. 45-58.

[6] W. N. Gansterer, R. C. Ward, R. P. Muller, and W. A. Goddard, III, *Computing Approximate Eigenpairs of Symmetric Block Tridiagonal Matrices*, SIAM J. Sci. Comput., to appear.

[7] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite System*, Prentice-Hall, Inc., 1981.

[8] N. E. Gibbs, *A Hybrid Profile Reduction Algorithm*, ACM Trans. Math. Softw., 2 (1986), pp. 378 – 387.

[9] N. E. Gibbs, W. G. Poole Jr., and P. K. Stockmeyer, *An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix*, SIAM J. Numer. Anal, 13 (1976), pp. 236-250.

[10] M. Gu and S. C. Eisenstat, *A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172-191.

[11] J. G. Lewis, *The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices*, ACM Trans. Math. Softw., 8 (1982), pp. 190-194.

[12] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM Press, Philadelphia, PA, 1998.

[13] J. A. Pople and D. L. Beveridge, *Approximate Molecular Orbital Theory*, 1st ed., McGraw-Hill, New York, 1970.

[14] J. A. Pople, D. L. Beveridge, and P. A. Dobosh, *Approximate Self-consistent Molecular Orbital Theory. V. Intermediate Neglect of Differential Overlap.*, J. Chem. Physics, 47 (1967), p 2026.

[15] J. A. Pople, D. P. Santry, and G. A. Segal, *Approximate Self-consistent Molecular Orbital Theory. I. Invariant Procedures.*, J. Chem. Physics, 43 (1965), p. S129.

[16] J. A. Pople and G. A. Segal, *Approximate Self-consistent Molecular Orbital Theory. II. Calculations with Complete Neglect of Differential Overlap.*, J. Chem. Physics, 43 (1965), p. S136.

[17] J. A. Pople and G. A. Segal, *Approximate Self-consistent Molecular Orbital Theory. III. CNDO Results for AB2 and AB3 Systems.*, J. Chem. Physics, 44 (1966), p. 3829.

[18] S. W. Sloan, *An Algorithm for Profile and Wavefront Reduction of Sparse Matrices,* International Journal for Numerical Methods in Engineering, 23 (1986), pp. 239-251.
[19] S. W. Sloan, *A FORTRAN Program for Profile and Wavefront Reduction*, International Journal for Numerical Methods in Engineering, 28 (1989), pp. 2651-2679.
[20] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry*, Dover Publications, Mineola, NY, 1996.
[21] J. H. Wilkinson, *The Algebraic Eigenvalue Problem,* Oxford University Press, Oxford, 1965.