A Fast Algorithm for the Symmetric Eigenvalue Problem

J.J. Dongarra and D. C. Sorensen

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439

1. Introduction

The symmetric eigenvalue problem is one of the most fundamental problems of computational mathematics. It arises in many applications, and therefore represents an important area for algorithmic research. It is also one of the first eigenvalue problems for which reliable methods have been obtained. It would be surprising therefore, if a new method were to be found that would offer a significant improvement in execution time over the fundamental algorithms available in standard software packages such as EISPACK [7]. However, it is reasonable to expect that eigenvalue calculations might be accelerated through the use of parallel algorithms for parallel computers that are emerging. We shall present such an algorithm in this paper. The algorithm is able to exploit parallelism at all levels of the computation and is well suited to a variety of architectures. However, a pleasant bonus of this research is that the parallel algorithm, even when run in serial mode, is significantly faster than the best sequential algorithm on large problems, and is effective on moderate size (order ≥ 30) problems when run in serial mode.

The problem we consider is the following: Given a real $n \times n$ symmetric matrix A, find all of the eigenvalues and corresponding eigenvectors of A. It is well known [8] that under these assumptions

$$(1.1) \qquad A = QDQ^T \quad , \quad \text{with } Q^TQ = I \; ,$$

so that the columns of the matrix Q are the orthonormal eigenvectors of A and $D = diag(\delta_1, \delta_2, ..., \delta_n)$ is the diagonal matrix of eigenvalues. The standard algorithm for computing this decomposition is to first use a finite algorithm to reduce A to tridiagonal form using a sequence of Householder transformations, and then to apply a version of the QR-algorithm to obtain all the eigenvalues and eigenvectors of the tridiagonal matrix [8]. We shall describe a method for parallelizing the computation of the eigensystem of the tridiagonal matrix.

The method is based upon a divide and conquer algorithm suggested by Cuppen[2]. A fundamental tool used to implement this algorithm is a method that was developed by Bunch, Nielsen, and Sorensen[1] for updating the eigensystem of a symmetric matrix after modification by a rank one change. This rank-one updating method was inspired by some earlier work of Golub[3] on modified eigenvalue problems. The basic idea of the new method is to use rank-one modifications to tear out selected off-diagonal elements of the tridiagonal problem in order to introduce a number of independent subproblems of smaller size. The subproblems are solved at the lowest level using the subroutine TQL2 from EISPACK and then results of these problems are successively glued together using the rank-one modification routine SESUPD that we have developed based upon the ideas presented in [1].

In the following discussion we describe the partitioning of the tridiagonal problem into smaller problems by rank-one tearing. Then we describe the numerical algorithm for gluing the results back together. The organization of the parallel algorithm is laid out, and finally some preliminary computational results are presented.

2. Partitioning by Rank-One Tearing

The crux of the algorithm is to divide a given problem into two smaller subproblems. To do this, we consider the symmetric tridiagonal matrix

$$(2.1) \quad T = \begin{pmatrix} T_1 & \beta e_k e_1^T \\ \beta e_1 e_k^T & T_2 \end{pmatrix}$$

$$= \begin{pmatrix} \hat{T}_1 & 0 \\ 0 & \hat{T}_2 \end{pmatrix} + \beta \begin{pmatrix} \theta e_k \\ \theta^{-1} e_1 \end{pmatrix} (\theta e_k^T, \ \theta^{-1} e_1^T)$$

where $1 \leq k \leq n$ and e_j represents the j-th unit vector of appropriate dimension. The k-th diagonal element of T_1 has been modified to give $\hat{T}_1$ and the first diagonal element of T_2 has been modified to give $\hat{T}_2$. Potential numerical difficulties associated with cancellation are avoided through the appropriate choice of θ. If the diagonal entries to be modified are of the same sign then $\theta = \pm 1$ is chosen so that $-\theta\beta$ has this sign and cancellation is avoided. If the two diagonal entries are of opposite sign, then the sign of θ is chosen so that $-\theta\beta$ has the same sign as one of the elements and the magnitude of θ is chosen to avoid severe loss of significant digits when $\theta^{-1}\beta$ is subtracted from the other. This is perhaps a minor detail, but it does allow the partitioning to be selected solely on the basis of position and without regard to numerical considerations.

Now we have two smaller tridiagonal eigenvalue problems to solve. According to equation (1.1) we compute the two eigensystems

$$\hat{T}_1 = Q_1 D_1 Q_1^T, \quad \hat{T}_2 = Q_2 D_2 Q_2^T$$

This gives

$$(2.2) \quad T = \begin{pmatrix} Q_1 D_1 Q_1^T & 0 \\ 0 & Q_2 D_2 Q_2^T \end{pmatrix} + \beta \begin{pmatrix} \theta e_k \\ \theta^{-1} e_1 \end{pmatrix} (\theta e_k^T, \ \theta^{-1} e_1^T)$$

$$= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left[\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \right.$$

$$\left. + \beta \begin{pmatrix} \theta q_1 \\ \theta^{-1} q_2 \end{pmatrix} (\theta q_1^T, \ \theta^{-1} q_2^T) \right] \begin{pmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{pmatrix}$$

where $q_1 = Q_1^T e_k$ and $q_2 = Q_2^T e_1$. The problem at hand now is to compute the eigensystem of the interior matrix in equation (2.2). A numerical method for solving this problem has been provided in [1] and we shall discuss this method in the next section.

It should be fairly obvious how to proceed from here to exploit parallelism. One simply repeats the tearing on each of the two halves recursively until the original problem has been divided into the desired number of subproblems and then the rank one modification routine may be applied from bottom up to glue the results together again.

3. The Updating Problem

The general problem we are required to solve is that of computing the eigensystem of a matrix of the form

$$(3.1) \quad \hat{Q} \hat{D} \hat{Q}^T = D + \rho z z^T$$

where D is a real $n \times n$ diagonal matrix, α is a scalar, and z is a real vector of order n. It is assumed without loss of generality that z has Euclidean norm 1.

As shown in [1], if $D = diag(\delta_1, \delta_2, \cdots, \delta_n)$ with $\delta_1 < \delta_2 < \cdots < \delta_n$ and no component ς_i of the vector z is zero, then the updated eigenvalues $\hat{\delta}_i$ are roots of the equation

$$(3.2) \quad f(\lambda) \equiv 1 + \rho \sum_{j=1}^{n} \frac{\varsigma_j^2}{\delta_j - \lambda} = 0.$$

Golub[3] refers to this as the secular equation and the behavior of its roots is completely described by the following graph:
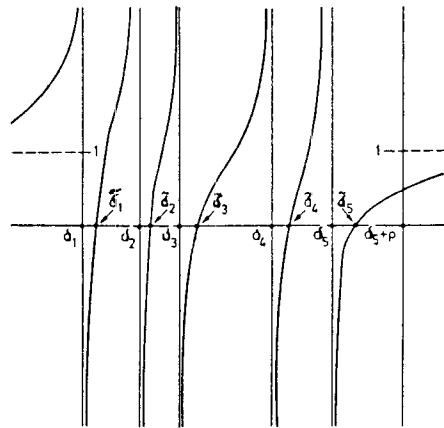

Figure 1. The Secular Equation

Moreover, as shown in [1] the eigenvectors (i.e. the columns of $\hat{Q}$ in (3.1)) are given by the formula

$$(3.3) \quad \hat{q}_i = \gamma_i \Delta_i^{-1} z$$

with γ_i chosen to make $||\hat{q}_i|| = 1$, and with $\Delta_i = diag(\delta_1 - \hat{\delta}_i, \delta_2 - \hat{\delta}_i, \cdots, \delta_n - \hat{\delta}_i)$. Due to this structure, an excellent numerical method may be devised to find the roots of the secular equation and as a by-product to compute the eigenvectors to full accuracy.

In the following discussion we assume that $\rho > 0$ in (3.2). A simple change of variables may always be used to achieve this, so there is no loss of generality. The method we shall describe was inspired by the work of More' [4] and Reinsch[5,6], and relies on the use of simple rational approximations to construct an iterative method for the solution of equation (3.2). Given that we wish to find the i-th root $\hat{\delta}_i$ of the function f in (3.2) we may write this function as

$$f(\lambda) = 1 + \phi(\lambda) + \psi(\lambda)$$

where

$$\psi(\lambda) = \rho \sum_{j=1}^{i} \frac{\varsigma_j^2}{\delta_j - \lambda}$$

and

$$\phi(\lambda) \equiv \rho \sum_{j=i+1}^{n} \frac{\varsigma_j^2}{\delta_j - \lambda}.$$

From the graph in Figure 1 it is seen that the root $\hat{\delta}_i$ lies in the open interval (δ_i, δ_{i+1}) and for λ in this interval all of the terms of ψ are negative and all of the terms of ϕ are positive. We may derive an iterative method for solving the equation

$$-\psi(\lambda) = 1 + \phi(\lambda)$$

by starting with an initial guess λ_0 in the appropriate interval and then constructing simple rational interpolants of the form

$$\frac{p}{q - \lambda}, \quad r + \frac{s}{\delta - \lambda}$$

where the parameters p, q, r, s are defined by the interpolation conditions

$$(3.4) \qquad \frac{p}{q - \lambda_0} = \psi(\lambda_0), \quad r + \frac{s}{\delta - \lambda_0} = \phi(\lambda_0)$$

$$\frac{p}{(q - \lambda_0)^2} = \psi'(\lambda_0), \quad \frac{s}{(\delta - \lambda_0)^2} = \phi'(\lambda_0).$$

The new approximate λ_1 to the root $\hat{\delta}_i$ is then found by solving

$$(3.5) \qquad \frac{-p}{q - \lambda} = 1 + r + \frac{s}{\delta - \lambda}$$

It is possible to construct an initial guess which lies in the open interval $(\delta_i, \hat{\delta}_i)$. A sequence of iterates $\{\lambda_k\}$ may then be constructed as we have just described with λ_{k+1} being derived from λ_k as λ_1 was derived from λ_0 above. The following theorem proved in [1] then shows that this iteration converges quadratically from one side of the root and does not need any safeguarding.

THEOREM (3.6) Let $\rho > 0$ in (3.2). If the initial iterate λ_0 lies in the open interval $(\delta_i, \hat{\delta}_i)$ then the sequence of iterates $\{\lambda_k\}$ as constructed in equations (3.4)–(3.5) are well defined and satisfy $\lambda_k < \lambda_{k+1} < \hat{\delta}_i$ for all $k \geq 0$. Moreover, the sequence converges quadratically to the root $\hat{\delta}_i$

In our implementation of this scheme equation (3.5) is cast in such a way that we solve for the iterative correction $\tau = \lambda_1 - \lambda_0$ The quantities $\delta_j - \lambda_k$ which are used in the eigenvalue calculations are maintained and iterative corrections may be applied directly to them and to the eigenvalue approximation. Cancellation is thus avoided because the corrections become smaller and smaller and are eventually applied to the lowest order bits. These values are then used directly in the calculation of the updated eigenvectors to obtain the highest possible accuracy. The rapid convergence of the iterative method allows the specification of very stringent convergence criteria that will ensure a relative residual and orthogonality of eigenvectors to full machine accuracy. A complete discussion of these stopping criteria is given in [1]. The algorithm did not suffer the effects of nearly equal roots as Cuppen suggests [2] but instead was able to solve such ill conditioned problems as the Wilkinson matrices W_{2k+1}^+ [9 p.308] to full machine precision and with slightly better residual and orthogonality properties than the standard algorithm TQL2 from EISPACK.

At the outset of this discussion we made the assumption that the diagonal elements of D were distinct and that no component of the vector z was zero. These conditions are not satisfied in general, so deflation techniques must be employed to arrange their satisfaction. A deflation technique was suggested in [1] to arrange for distinct eigenvalues which amounts to rotating the basis for the eigenspace corresponding to a multiple eigenvalue so that only one component of the vector z corresponding to this space is nonzero in the new basis. Those terms in (3.2) corresponding to zero components of z may simply be dropped. The eigenvalues and eigenvectors corresponding to these zero components remain static. In finite precision arithmetic the situation becomes more intesting. Terms corresponding to small components of z may be dropped. This can have a very dramatic effect upon the amount of work required in our parallel method. As first observed by Cuppen[2] there can be significant deflation in the updating process as the original matrix is rebuilt from the subproblems.

This has been a brief description of the rank-one updating scheme. Full theoretical details are available in

340

[1]. More on the computational and implementation details will be reported elsewhere. This calculation represents the workhorse of the parallel scheme that we are about to describe.

4. The Parallel Algorithm

Although it is fairly straightforward from Section 2 to see how to obtain a parallel algorithm, certain details are worth discussing further. We shall begin by describing the partitioning phase. This phase amounts to constructing a binary tree with each node representing a rank-one tear and hence a partition into two sub-problems. A tree of level 3 therefore represents a splitting of the original problem into 8 smaller eigenvalue problems. Thus, there are two standard symmetric tridiagonal eigenvalue problems to be solved at each leaf of the tree. Each of these problems may be spawned independently without fear of data conflicts. The tree is then traversed in reverse order with the eigenvalue updating routine SESUPD applied at each node joining the results from the left son and right son calculations. The leaves each define independent rank-one updating problems and again there is no data conflicts between them. The only data dependency at a node is that the left and right son calculations must have been completed. As this condition is satisfied, the results of two adjacent eigenvalue subproblems are ready to be joined through the rank-one updating process and this node may spawn the updating process immediately. Information required at a node to define the problem consists of the index of the element torn out together with the dimension of the left and right son problems. For example, if $n = 50$ with a tree of level 3 we have
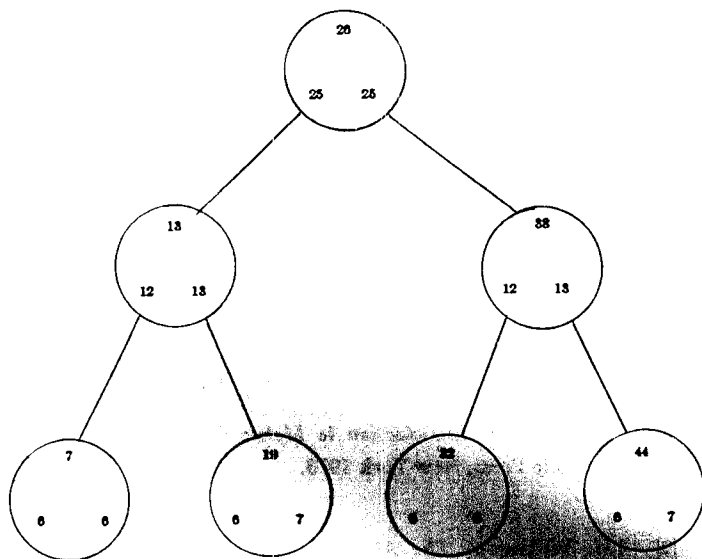

Figure 2. The Computational Tree

This tree defines 8 subproblems at the lowest level. The beginning indices of these problems are 1,7,13,19,26,32,38,44 and the dimension of each of them may be read off from left to right at the lowest level as 6,6,6,7,6,6,6,7 respectively. As soon as the calculation for the problems beginning at indices 1 and 7 have been completed a rank-one update may proceed on the problem beginning at index 1 with dimension 12. The remaining updating problems at this level begin at indices 13,26,38. There are then two updating problems at indices 1 and 26 each of dimension 25 and a final updating problem at index 1 of dimension 50.

Evidently, we lose a degree of large grain parallelism as we move up a level on the tree. However, there is more parallelism to be found at the root finding level and the amount of this increases as we travel up the tree so there is ample opportunity for load balancing in this scheme. The parallelism at the root finding level stems from the fact that each of the root calculations is independent and requires read only access to all but one array. That is the array that contains the diagonal entries of the matrix Δ, described in Section 3. For computational efficiency we may decide on an advantageous number of processes to create at the outset. In the example above that number was 8. Then as we travel up the tree the root-finding procedure is split into 2,4,and finally 8 parallel parts in each node at level 3, 2, 1 respectively. As these computations are roughly equivalent in complexity on a given level it is reasonable to expect to keep all processors devoted to this computation busy throughout.

5. Performance

In this section we present and analyze the results of this algorithm on a number of machines. The same algorithms has been run on a VAX 11/785 and a CRAY X-MP. The algorithm has not yet been implemented on a parallel machine. Those tests are in progress and will be reported elsewhere along with the algorithmic details required to achieve parallelism at all levels. Our implementation uses a tree of level three as shown in Figure 2. We have not yet fully examined splitting the problem into more parts.

We have compared our implementation of the algorithm described in this paper to TQL2 from the EISPACK collection. The table below gives the ratio of execution time for TQL2 from EISPACK and the algorithm presented here when run sequentially.

n	Ratio of time	TQL2 $\|Az - \lambda x\|$	SESUPD $\|Az - \lambda x\|$	TQL2 $\|Q^T Q - I\|$	SESUPD $\|Q^T Q - I\|$
50	.78	1.6×10^{-12}	4.8×10^{-13}	1.9×10^{-12}	1.2×10^{-13}
100	1.26	2.6×10^{-12}	7.6×10^{-13}	3.9×10^{-12}	2.5×10^{-12}
200	1.93	6.1×10^{-12}	1.8×10^{-12}	7.0×10^{-12}	5.8×10^{-11}
300	3.62	1.1×10^{-11}	3.3×10^{-12}	1.1×10^{-11}	2.8×10^{-12}
400	4.70	1.2×10^{-11}	3.8×10^{-12}	1.4×10^{-11}	3.8×10^{-12}

A comparison on the CRAY X-MP for
TQL2 vs the parallel algorithm run sequentially

n	Ratio of time	TQL2 $\|Az - \lambda x\|$	SESUPD $\|Az - \lambda x\|$	TQL2 $\|Q^T Q - I\|$	SESUPD $\|Q^T Q - I\|$
50	1.38	1.2×10^{-15}	5.7×10^{-16}	1.3×10^{-15}	1.9×10^{-16}
100	1.89	2.5×10^{-15}	7.7×10^{-16}	2.1×10^{-15}	1.1×10^{-15}
150	2.69	2.6×10^{-15}	6.1×10^{-15}	2.6×10^{-15}	3.2×10^{-16}

A comparison on the VAX 11/785 for
TQL2 vs the parallel algorithm run sequentially

As can be seen, the performance of the parallel algorithm as implemented to run on a sequential machine is quite impressive. The surprising result here is the observed speed up even in serial mode of execution. This is unusual in a parallel algorithm. Often more work is associated with synchronization and computational overhead required to split the problem into parallel parts. These test problems are a bit misleading, however, because there was considerable deflation involved. Matrices do exist[2] for which this dramatic deflation does not occur. However, an operation count reveals that when the eigenvectors are sought along with the eigenvalues, there should be a factor of 1.33 improvement in performance. The observation of Cuppen that this deflation occurs in many cases was very fortunate. It brought our attention to this algorithm, but we did not really expect the remarkable performance observed here. In the case where many eigenvectors are sought along with the eigenvalues this algorithm seems to be very promising.

6. References

[1] J.R. Bunch, C.P. Nielsen, and D.C. Sorensen, *Rank-One Modification of the Symmetric Eigenproblem*, Numerische Mathematik 31, pp. 31-48, 1978.

[2] J.J.M. Cuppen *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem*, Numerische Mathematik 36, pp. 177-195, 1981.

[3] G.H. Golub, *Some Modified Matrix Eigenvalue Problems*, SIAM Review, 15, pp. 318-334 1973.

[4] J.J. More' *The Levenberg-Marquardt Algorithm: Implementation and Theory*, Proceedings of the Dundee Conference on Numerical Analysis, G.A. Watson ed. Springer-Verlag 1978.

[3] G.H. Golub, *Some Modified Matrix Eigenvalue Problems*, SIAM Review, 15, pp. 318-334 1973.

[4] J.J. More' *The Levenberg-Marquardt Algorithm: Implementation and Theory*, Proceedings of the Dundee Conference on Numerical Analysis, G.A. Watson ed. Springer-Verlag 1978.

[5] C.H. Reinsch, *Smoothing by Spline Functions*, Numerische Mathematik 10, pp. 177-183, 1967.

[6] C.H. Reinsch, *Smoothing by Spline Functions II*, Numerische Mathematik 16, pp. 451-454, 1971.

[7] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, Vol. 6, 2nd edition, Springer-Verlag, Berlin, 1976.

[8] G.W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York 1973.

[9] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford 1965.