. . . . . .

A Fully Parallel 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

In this paper we present a parallel algorithm for the symmetric algebraic eigenvalue problem. The algorithm is based upon a divide and conquer scheme suggested by Cuppen for computing the eigensystem of a symmetric tridiagonal matrix. We extend this idea to obtain a parallel algorithm that retains a number of active parallel processes that is greater than or equal to the initial number throughout the course of the computation. We give a new deflation technique which together with a robust root finding technique will assure computation of an eigensystem to full accuracy in the residuals and in the orthogonality of eigenvectors. A brief analysis of the numerical properties and sensitivity to round off error is presented to indicate where numerical difficulties may occur. The algorithm is able to exploit parallelism at all levels of the computation and is well suited to a variety of architectures.

Computational results are presented for several machines. These results are very encouraging with respect to both accuracy and speedup. A surprising result is that the parallel algorithm, even when run in serial mode, can be significantly faster than the previously best sequential algorithm on large problems, and is effective on moderate size problems when run in serial mode.

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. The problem has received considerable attention in the literature and was probably the first algebraic eigenvalue problem for which reliable methods were 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 [12]. However, it is reasonable to expect that eigenvalue calculations might be accelerated through the use of parallel algorithms for parallel computers that are becoming available. 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.

Work supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under Contracts W-31-109-Eng-38, DE-AC05-840R21400 and DE-FG02-85ER25001.

Presented at the Second SIAM Conference on Vector and Parallel Processing in Scientific Computing, Norfolk, Virginia, November 20, 1985.

However, the surprising result is that the parallel algorithm, even when run in serial mode, is significantly faster than the previously 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 [14] that under these assumptions

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

so that the columns of the matrix Q are the orthonormal eigenvectors of A and $D = diag(\delta_1, \delta_2, \ldots, \delta_n)$ is the diagonal matrix of eigenvalues. The standard algorithm for computing this decomposition is first to 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[14]. The primary purpose of this paper is to describe a method for parallelizing the computation of the eigensystem of the tridiagonal matrix. However, the method we present is intended to be used in conjunction with the initial reduction to tridiagonal form to compute the complete eigensystem of the original matrix A. We, therefore, briefly touch upon the issues involved in parallelizing this initial reduction and suggest a way to combine the parallel initial reduction with the tridiagonal algorithm to obtain a fully parallel algorithm for the symmetric eigenvalue problem.

The method is based upon a divide and conquer algorithm suggested by Cuppen[3]. A fundamental tool used to implement this algorithm is a method that was developed by Bunch, Nielsen, and Sorensen[2] 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[4] 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 [2].

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 computational results are presented. Throughout this paper we adhere to the convention that capital Roman letters represent matrices, lower case Roman letters represent column vectors, and lower case Greek letters represent scalars. A superscript T denotes transpose. All matrices and vectors are real, but the results are easily extended to matrices over the complex field.

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

$$(2.1)\qquad T = \begin{bmatrix} T_1 & \beta e_k e_1^T \\ \beta e_1 e_k^T & T_2 \end{bmatrix} = \begin{bmatrix} \hat{T}_1 & 0 \\ 0 & \hat{T}_2 \end{bmatrix} + \theta\beta \begin{bmatrix} e_k \\ \theta^{-1} e_1 \end{bmatrix} (e_k^T, \ \theta^{-1} e_1^T)$$

where $1 \le k \le n$ 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 may be 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

$$(2.2)\qquad T = \begin{bmatrix} Q_1 D_1 Q_1^T & 0 \\ 0 & Q_2 D_2 Q_2^T \end{bmatrix} + \theta\beta \begin{bmatrix} e_k \\ \theta^{-1} e_1 \end{bmatrix} (e_k^T, \ \theta^{-1} e_1^T)$$
$$= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} + \theta\beta \begin{bmatrix} q_1 \\ \theta^{-1} q_2 \end{bmatrix} (q_1^T, \ \theta^{-1} q_2^T) \right) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}$$

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 [2] 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

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

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

We seek a formula for an eigen-pair for the matrix on the right hand side of (3.1) . Let us assume for the moment that $D = diag(\delta_1, \delta_2, \cdots, \delta_n)$ with $\delta_1 < \delta_2 < \cdots < \delta_n$ and that no component ζ_i of the vector z is zero. In Section 4 we discuss how this may always be arranged. If q, λ is such an eigen-pair then

$$(D + \rho z z^T) q = \lambda q$$

and a simple rearrangement of terms gives

$$(D - \lambda I) q = -\rho (z^T q) z \ .$$

If $\lambda = \delta_i$ for some i then our assumption that the $i-th$ component of z is nonzero implies $z^T q = 0$. This together with the assumption of distinct eigenvalues implies that q is the eigenvector e_i of D , but then $0 = z^T e_i$ would contradict the original assumption. Thus, multiplying on the left by $z^T (D - \lambda I)^{-1}$ is valid and gives

$$z^T q = -\rho (z^T q) z^T (D - \lambda I)^{-1} z \ .$$

Our assumptions again imply that $z^T q \neq 0$, hence

$$1 + \rho z^T (D - \lambda I)^{-1} z = 0$$

must be satisfied by λ. Starting with a λ that is a root to this last equation and putting

$$q = \theta (D - \lambda I)^{-1} z$$

for some scalar θ, one may easily verify that q, λ is an eigen-pair for $D + \rho z z^T$.

If we write this equation in terms of the components ζ_i of z then λ must be a root of the equation

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

Golub[4] refers to this as the secular equation and the behavior of its roots is pictorially described by the following graph:

Figure 1. The Secular Equation

Moreover, as shown in [2] the eigenvectors (i.e. the columns of $\hat{Q}$ in (3.1)) are given by the formula

$$\hat{q}_i = \gamma_i \Delta_i^{-1} z \tag{3.3}$$

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. It must be stressed, however, that great care must be exercised in the numerical method used to solve the secular equation and to construct the eigenvectors from formula (3.3).

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′ [9] and Reinsch[10,11], 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

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

and

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

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

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

where δ is fixed at the current value of $\hat{\delta}_{i+1}$, and the parameters p, q, r, s are defined by the interpolation conditions

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

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

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

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

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 [2] 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 later in the eigenvector calculations are maintained and these iterative corrections are applied directly to them as well as to the eigenvalue approximation. Cancellation in the computation of these differences 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. The stopping criterion used is

$$(i)\ |f(\lambda)| \leq \eta\, max(|\delta_1|, |\delta_2|) \ \text{and}\ (ii)\ |\tau| \leq \eta \min(|\delta_i - \lambda|, |\delta_i - \lambda|),$$

where λ is the current iterate and τ is the last iterative correction that was computed. The condition on τ is very stringent. The purpose of such a stringent stopping criteria will be clarified in the next section when we discuss orthogonality of computed eigenvectors. Let it suffice at this point to say that condition (i) assures a small residual and that (ii) assures orthogonal eigenvectors.

In most problems these criteria are easily satisfied. However, there are pathological situations involving nearly equal eigenvalues that make (ii) very difficult to satisfy. It is here that the basic root finding method described above must be modified. The problem stems from the fact that the iterative corrections cease to modify the value of f due to round off error. When working in single precision, this situation can be rectified through the use of extended precision accumulation of inner products in the evaluation of f and its derivative. However, this becomes less attractive when working in 64-bit arithmetic as is done in most scientific calculations. Whether or not we shall be able to provide a root finder that will satisfy these stringent requirements in 64-bit arithmetic for very pathological cases remains to be seen.

This has been a brief description of the rank-one updating scheme. Full theoretical details are available in [2]. This calculation represents the workhorse of the parallel algorithm. The method seems to be very robust in practice and exhibits high accuracy. It does not seem to suffer the effects of nearly equal roots as Cuppen suggests [3] but instead was able to solve such ill conditioned problems as the Wilkinson matrices W_{2k+1} [15 p.308] to full machine precision and with slightly better residual and orthogonality properties than the standard algorithm TQL2 from EISPACK. In the next section we offer some reasons for this behavior.

4. Deflation and Orthogonality of Eigenvectors

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 ensure their satisfaction. A deflation technique was suggested in [2] to provide 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 when represented 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 interesting. 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[3], there can be significant deflation in the updating process as the original matrix is rebuilt from the subproblems.

This deflation can occur in two ways in exact arithmetic, either through zero components of z or through multiple eigenvalues. In order to obtain an algorithm suitable for finite precision arithmetic, we must refine these notions to include "nearly zero" components of z and "nearly equal" eigenvalues. Analysis of the first situation is straightforward. The second situation can be quite delicate in certain pathological cases, however, so we shall discuss it here in some detail. It turns out that the case of nearly equal eigenvalues may be reduced to the case of small components of z.

It is straightforward to see that when $z^T e_i = 0$ the $i-th$ eigenvalue and corresponding eigenvector of D will be an eigen-pair for $D + \rho z z^T$. Let us ask the question "when is an eigen-pair of D a good approximation to an eigen-pair for the modified matrix?" This question is easily answered. Recall that $\|z\| = 1$, so

$$\|(D + \rho z z^T)e_i - \delta_i e_i\| = |\rho \zeta_i| \, \|z\| = |\rho \zeta_i|.$$

Thus we may accept this eigen-pair as an eigen-pair for the modified matrix whenever

$$|\rho \zeta_i| \le tol$$

where $tol > 0$ is our error tolerance. Typically $tol = macheps\,(\eta \|A\|)$, where $macheps$ is machine precision and η is a constant of order unity. However, at each stage of the updating process we may use

$$tol = macheps\ \eta\,(\max(|\delta_1|, |\delta_n|) + |\rho|)$$

since at every stage this will represent a bound on the spectral radius of a principal submatrix of A. Let us now suppose there are two eigenvalues of D separated by ε so that $\varepsilon = \delta_{i+1} - \delta_i$. Consider the 2×2 submatrix

$$\begin{bmatrix} \delta_i & 0 \\ 0 & \delta_{i+1} \end{bmatrix} + \rho \begin{bmatrix} \zeta_i \\ \zeta_{i+1} \end{bmatrix} (\zeta_i\,,\ \zeta_{i+1})$$

of $D + \rho z z^T$ and let us construct a Givens transformation to introduce a zero in one of the two corresponding components in z. Then

$$\begin{bmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{bmatrix} \left[\begin{bmatrix} \delta_i & 0 \\ 0 & \delta_{i+1} \end{bmatrix} + \rho \begin{bmatrix} \zeta_i \\ \zeta_{i+1} \end{bmatrix} (\zeta_i\,,\ \zeta_{i+1}) \right] \begin{bmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{bmatrix}$$
$$= \begin{bmatrix} \hat{\delta}_i & 0 \\ 0 & \hat{\delta}_{i+1} \end{bmatrix} + \rho \begin{bmatrix} \tau \\ 0 \end{bmatrix} (\tau, 0) + \varepsilon \sigma \gamma \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

where $\gamma^2 + \sigma^2 = 1$, $\hat{\delta}_i = \delta_i \gamma^2 + \delta_{i+1} \sigma^2$, $\hat{\delta}_{i+1} = \delta_i \sigma^2 + \delta_{i+1} \gamma^2$ and $\tau^2 = \zeta_i^2 + \zeta_{i+1}^2$. Now, if we put G_i equal to the $n \times n$ Givens transformation constructed from the identity by replacing the appropriate diagonal block with the 2×2 rotation just constructed, we have

$$\text{(4.1)} \qquad G_i\,(D + \rho z z^T)\,G_i^T = \hat{D} + \rho \hat{z} \hat{z}^T + E_i$$

with $e_i^T \hat{z} = \tau$ and $e_{i+1}^T \hat{z} = 0$, $e_i^T \hat{D} e_i = \hat{\delta}$, $e_{i+1}^T \hat{D} e_{i+1} = \hat{\delta}$, and

$$\|E_i\| = |\varepsilon \sigma \gamma| \ .$$

If we choose, instead, to zero out the $i-th$ component of z then we simply apply G_i on the right and its transpose on the left in equation(4.1) to obtain the desired similar result. The only exception to the previous result is that the sign of the matrix E_i is reversed. Of course this deflation is only done when

$$|\varepsilon \gamma \sigma| \le tol$$

is satisfied.

The result of applying all of the deflations is to replace the updating problem (3.1) with one of smaller size. When appropriate, this is accomplished by applying similarity transformations consisting of several Givens transformations. If G represents the product of these transformations the result is

$$G(D + \rho zz^T)G^T = \begin{bmatrix} D_1 - \rho z_1 z_1^T & 0 \\ 0 & D_2 \end{bmatrix} + E ,$$

where

$$\| E \| \le tol\ \eta_2$$

with η_2 of order unity . The cumulative effect of such errors is additive and thus the final computed eigensystem $\hat{Q}\hat{D}\hat{Q}^T$ which satisfies

$$\| A - \hat{Q}\hat{D}\hat{Q}^T \| \le \eta_3 tol$$

where η_3 is again of order 1 in magnitude. The reduction in size of $D_1 - \rho z_1 z_1^T$ over the original rank 1 modification can be spectacular in certain cases. The effects of such deflation can be dramatic, for the amount of computation required to perform the updating is greatly reduced.

Let us now consider the possible limitations on orthogonality of eigenvectors due to nearly equal roots. Our first result is a perturbation lemma that will indicate the inherent difficulty associated with nearly equal roots. *LEMMA* (4.2) *Let*

(4.3) $$q_\lambda^T \equiv \left(\frac{\zeta_1}{\delta_1 - \lambda}, \frac{\zeta_2}{\delta_2 - \lambda}, \cdots \frac{\zeta_n}{\delta_n - \lambda} \right) \left[\frac{\rho}{f'(\lambda)} \right]^{1/2},$$

where f is defined by formula (3.2). Then for any $\lambda, \mu \notin \{ \delta_i : i = 1,...,n \}$

(4.4) $$| q_\lambda^T q_\mu | = \frac{| f(\lambda) - f(\mu) |}{| \lambda - \mu |} \left[\frac{1}{f'(\lambda) f'(\mu)} \right]^{1/2} .$$

PROOF : Note that

(4.5) $$q_\lambda^T q_\mu = \left[\rho \sum_j \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} \right] \left[\frac{1}{f'(\lambda) f'(\mu)} \right]^{1/2} .$$

But,

$$\frac{1}{(\delta_j - \lambda)(\delta_j - \mu)} = \frac{1}{\lambda - \mu}\left[\frac{1}{(\delta_j - \lambda)} - \frac{1}{(\delta_j - \mu)} \right].$$

Thus

$$(\lambda - \mu) \rho \sum_j \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} = \rho \sum_j \frac{\zeta_j^2}{\delta_j - \lambda} - \rho \sum_j \frac{\zeta_j^2}{\delta_j - \mu} = f(\lambda) - f(\mu)$$

and the result follows . □

Note that in (4.3) q_λ is always a vector of unit length, and the set of n vectors selected by setting λ equal to the roots of the secular equation is the set of eigenvectors for $D + \rho zz^T$. Moreover, equation (4.4) shows that those eigenvectors are mutually orthogonal whenever λ and μ are set to distinct roots of f . Finally, the term $|\lambda - \mu|$ appearing in the denominator of (4.4) sends up a warning that it may be difficult to attain orthogonal eigenvectors when the roots λ and μ are close. We wish to examine this situation now. We will show that, as a result of the deflation process, the $\{\delta_i\}$ are sufficiently separated, and the weights ζ_i are uniformly large enough that the roots of f are bounded away from each other. This statement is made explicit in the following *LEMMA* (4.6) *Let λ be the root of f in the $i-th$ sub interval (δ_i, δ_{i+1}). If the deflation test is satisfied then either*

(i) $$|\delta_{i+1}-\lambda| \ge ½|\delta_{i+1}-\delta_i| \text{ and } |\delta_i-\lambda| \ge \frac{tol^2}{|\rho(\delta_{i+1}-\delta_i)| + 2\rho^2},$$

or

(ii) $$|\delta_i-\lambda| \ge ½|\delta_{i+1}-\delta_i| \text{ and } |\delta_{i+1}-\lambda| \ge \frac{tol^2}{2\rho^2}(\delta_{i+1}-\delta_i)$$

PROOF : In case (i) the fact that $f(\lambda)=0$ provides

$$\rho \sum_{j=1}^{i} \frac{\zeta_j^2}{\lambda-\delta_j} = 1 + \rho \sum_{j=i+1}^{n} \frac{\zeta_j^2}{\delta_j-\lambda}.$$

From this equation we find that

$$|\rho| \frac{\zeta_i^2}{\lambda-\delta_i} \le 1+\frac{|\rho|}{\delta_{i+1}-\lambda} \le 1+\frac{2|\rho|}{\delta_{i+1}-\delta_i},$$

and it follows readily that

$$|\delta_{i+1}-\delta_i| \left[\frac{|\rho|\zeta_i^2}{|\delta_{i+1}-\delta_i| + 2|\rho|}\right] \le |\delta_i-\lambda|$$

The deflation rules assure us that $|\rho|\zeta_i^2 \ge tol^2/|\rho|$ and the result follows. Case (ii) is similar. □

The form of the result given in (4.6) will have importance in the following lemma which gives better insight to the quality of numerical orthogonality attainable with this scheme. The bounds obtained are certainly less than one would hope for. Moreover, it is unfortunate that only the magnitude of the weights enter into the estimate. This obviously does not take into account the deflation due to nearly equal eigenvalues. Unfortunately, we have not been able to improve these estimates by other means and are left with this crude bound. *LEMMA* (4.7) *Suppose that $\hat{\lambda}$ and $\hat{\mu}$ are numerical approximations to exact roots λ and μ of f. Assume that these roots are distinct and let the relative errors for the quantities $\delta_i-\lambda$ and $\delta_i-\mu$ be denoted by θ_i and η_i respectively. That is, the computed quantities*

(4.8) $$\delta_i-\hat{\lambda} = (\delta_i-\lambda)(1+\theta_i) \text{ and } \delta_i-\hat{\mu} = (\delta_i-\mu)(1+\eta_i),$$

for $i=1,2,...,n$. Let $q_{\hat{\lambda}}$ and $q_{\hat{\mu}}$ be defined according to formula (4.3) using the computed quantities given in (4.8). If $|\theta_i|, |\eta_i| \le \varepsilon \ll 1$, then

$$|q_{\hat{\lambda}}^T q_{\hat{\mu}}| = |q_{\lambda}^T E q_{\mu}| \le \varepsilon(2+\varepsilon)\left[\frac{1}{1-\varepsilon}\right]^2,$$

with E a diagonal matrix whose $i-th$ diagonal element is

$$E_{ii} = \frac{\theta_i+\eta_i+\theta_i\eta_i}{(1+\theta_i)(1+\eta_i)} \left[\frac{f'(\hat{\lambda})f'(\hat{\mu})}{f'(\lambda)f'(\mu)}\right]^{½}.$$

PROOF : From formula (4.3) we have

$$-q_{\hat{\lambda}}^T q_{\hat{\mu}} = -\left[\sum_{j=1}^{n} \frac{\zeta_j^2}{(\delta_j-\lambda)(\delta_j-\mu)(1+\theta_j)(1+\eta_j)}\right] \Big/ \left[f'(\hat{\lambda}) f'(\hat{\mu})\right]^{½}$$

$$= \left[\sum_j \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} - \sum_j \frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)(1 + \theta_j)(1 + \eta_j)} \right] \left[f'(\hat{\lambda}) f'(\hat{\mu}) \right]^{-1/2}$$

due to the orthogonality of the exact vectors. Thus,

$$|\hat{q}_{\lambda}^T \hat{q}_{\mu}| = \left| \sum_j \left[\frac{\zeta_j^2}{(\delta_j - \lambda)(\delta_j - \mu)} \right] \left[1 - \frac{1}{(1 + \theta_j)(1 + \eta_j)} \right] \left[f'(\hat{\lambda}) f'(\hat{\mu}) \right]^{-1/2} \right|.$$

Since the quantity

$$\frac{f'(\lambda)}{f'(\hat{\lambda})} = \frac{\sum_j \zeta_j^2 / (\delta_j - \lambda)^2}{\sum_j \zeta_j^2 / ((\delta_j - \lambda)^2 (1 + \theta_j)^2)} \le (1 + \varepsilon)^2$$

and since

$$\max E_{ii} \le \frac{\varepsilon(2 + \varepsilon)}{(1 + \varepsilon)^2}$$

the result follows. □

This shows that orthogonality can be assured whenever it is possible to provide small relative errors when computing the differences $\delta_i - \lambda$. Since, as we mentioned in Section 3, these quantities are updated with the iterative corrections to λ and since deflation has guaranteed that the quantities in Lemma (4.6) are of bounded away from zero, it will be possible in theory to provide small relative errors. Obviously, the analysis given here, simple as it may be, could form the core of a rigorous error analysis of the method. As yet we lack a root finder that could be proven to satisfy numerically the relative error requirements. The one described above will do so in finite precision but may require extended precision accumulation of inner products in pathological cases. However, in practice we have not experienced difficulty with the exception of contrived examples. Cuppen [3] has shown that the computed eigenvectors may be safely reorthogonalized when needed. We hope to avoid this alternative though.

5. Reduction to Tridiagonal Form and Related Issues

This algorithm is designed to work in conjunction with Householders reduction of a symmetric matrix to tridiagonal form . In this standard technique a sequence of $n-1$ Householder transformations is applied to form

$$A_{k+1} = (I - \alpha_k w_k w_k^T) A_k (I - \alpha_k w_k w_k^T) ,$$

with

$$A_k = \begin{bmatrix} T_k & 0 \\ 0 & \hat{A}_k \end{bmatrix} ,$$

where T_k is a tridiagonal matrix of order $k-1$. See [12] for details. We note here that we can form $(I - \alpha w w^T) A (I - \alpha w w^T)$ using the following algorithm.

Algorithm 5.1
1. $v = Aw$
2. $y^T = v^T - \alpha(w^T v) w^T$
3. Replace A by $A - \alpha v w^T - \alpha w y^T$

Steps 1 and 3 may all be parallelized because A may be partitioned into blocks of columns $A = (A_1, A_2, \cdots, A_j)$ and each of the contributions corresponding to these columns may be carried out independently in steps 1 and 3. For example in step 3,

$$A_i \leftarrow A_i - \alpha v w_i^T - \alpha w y_i^T$$

where the vectors w and y have been partitioned in a corresponding manner. In step 2 the partial results will have to be stored in temporary locations until all are completed and then they may be added together to obtain the final result. Note also that when the calculations are arranged this way advantage may be taken of vector operations when they are available.

Algorithm 5.1 has some disadvantages when incorporated into the reduction of a symmetric matrix to tridiagonal form. At the k-th stage of the reduction we have

$$\begin{bmatrix} T^{(k)} & \beta_k e_k e_1^T \\ \beta_k e_1 e_k^T & A^{(k)} \end{bmatrix}$$

The reduction is advanced one step through the application of Algorithm 5.1 to the submatrix $A^{(k)}$. First, a fork-join synchronization construct is imposed since the matrix-vector product requires the entire matrix $A^{(k)}$ to be in place before this product can be completed, so that no portion of Step 2 may begin until Step 1 is finished. Second, due to symmetry, only the lower triangle of $A^{(k)}$ need be computed and this implies that vector lengths shorten during the computation.

When used in conjunction with the rank one tearing scheme, these drawbacks may be overcome. Suppose the final result of this decomposition is T and that this matrix would be partitioned into $(T_1, T_2, \ldots, T_m)$ by the rank one tearing if it were known. It is not necessary to wait until the entire reduction is completed, for $T^{(k)}$ represents a leading principal submatrix of T. Thus, as soon as the first sub-tridiagonal matrix T_1 is exposed the process of computing its eigen system may be initiated. Similarly, as soon as T_2 is exposed its eigensystem may be computed. Then a rank one update may occur, and so on. In this way a number of independent processes may be spawned early on and the number of such processes ready to execute will remain above a reasonable level throughout the course of the computation. An efficient implementation of this scheme is difficult and will be the subject of a subsequent paper. Issues such as proper level of partitioning will have added importance due to the desire to have parallel processes set in motion as soon as possible.

When we do not wish to find eigenvectors there is no reason to store the product Q of these Householder transformations. Nor is it necessary to accumulate the product of the successive eigenvector transformations resulting from the updating problem. That is, we do not need to overwrite Q with

$$Q \leftarrow Q \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \hat{Q}$$

where Q_1, Q_2 and $\hat{Q}$ are the matrices appearing in equations (2.2) and (3.1) above. Instead, we may simply discard Q. Then the vector q_1 may be formed as $\hat{T}_1$ is transformed to D_1 in (2.2) by accumulating the products of the transformations constructed in TQL2 that make up Q_1 against the vector e_k. If there is more than one division then Q_1 will have been calculated with the updating scheme. In this case we do not calculate all of Q_1 but instead use the component-wise formula for the eigenvectors to pick off the appropriate components needed to form q_1.

This algorithm can be generalized to handle band matrices as well. Instead of performing a rank-one tearing to split the matrix into two independent subproblems, we proceed by making $(m+1)m/2$ rank-one changes designed to split the matrix, (m is the half bandwidth, $m=1$ for tridiagonal matrices).

$$
\begin{array}{l ccccc|ccccc}
 & . & . & . & & & & & & & \\
 & x & x & x & x & x & & & & & \\
 & & x & x & x & x & x & & & & \\
i_{th} \text{ row} & & & x & x & x & x & x & & & \\
\cline{5-8}
i+1_{th} \text{ row} & & & & x & x & x & x & x & & \\
 & & & & & x & x & x & x & x & \\
 & & & & & & x & x & x & x & x \\
 & & & & & & & & . & . & .
\end{array}
$$

The three rank-one changes for this band matrix, ($m=2$), involve the following elements:

$$
\begin{array}{ll}
a_{i-1,i-1},\ a_{i-1,i+1}, & \\
a_{i+1,i-1},\ a_{i+1,i+1}, & \\
 & \\
a_{i,i},\ a_{i,i+1}, & \\
a_{i+1,i},\ a_{i+1,i+1}, & \text{and} \\
 & \\
a_{i,i},\ a_{i,i+2}, & \\
a_{i+2,i},\ a_{i+2,i+2}. &
\end{array}
$$

The order in which they are applied does not matter in exact arithmetic. We have not studied the numerical properties of this scheme however.

6. 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 height 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 are no data conflicts between them. The only data dependency at a node is that the left and right son calculations must have been completed. When 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 height 3 we have

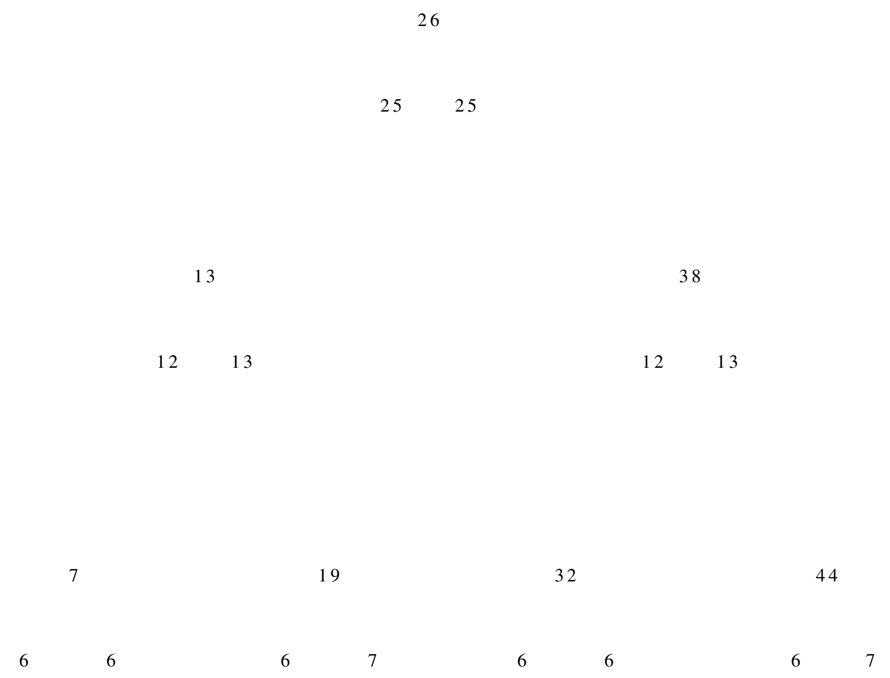

Figure 2. The Computational Tree

This tree defines 8 subproblems at the lowest level. There are two calls to TQL2 required at each leaf of the tree in Figure 2 to solve tridiagonal eigenvalue problems. 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 only read access to all but one array. That is the array that contains the diagonal entries of the matrix Δ_i 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.

7. Implementation and Library Issues

The implementation described here is for computers with a shared memory architecture. The algorithm itself is not limited to this memory model however, Jessop [5] has an implementation of a similar algorithm for a hypercube that is also based upon Cuppen's divide and conquer scheme.

Obviously, the implementation of this scheme is not so straightforward as simply parallelizing a loop using a fork-join construct. The synchronization mechanism requires more sophistication since it must be able to spawn processes at the root finding level based upon computation that has taken place. This process allocation problem is dynamic rather than static since, due to the possibility of deflation, it will not be known in advance how many roots will have to be calculated at a given level. If there are only a few roots to be found, then the desire for reasonable granularity will dictate fewer processes to be spawned.

In addition to requiring a certain level of sophistication in the synchronization scheme, we would like to adhere as much as possible to the principles of transportability. This algorithm is obviously a candidate for a library subroutine and potentially will find use on a wide variety of machines. When designing library subroutines, one wishes to conceal machine dependencies as much as possible from the user. These important considerations seem to be difficult to accommodate if we are to invoke parallelism at the level described above. It would appear that the user must be conscious of the number of parallel processes required by the library subroutines throughout his program. Should the library routines be called from multiple branches of a user's parallel program, then he could inadvertently attempt to create many more processes than allowed due to physical limitations.

We believe there is hope for implementing this algorithm while adhering to the goals set out in the previous two paragraphs. It may be accomplished by adopting a programming technique that is inspired by work of Lusk and Overbeek [7,8] and by Babb [1] on methodologies for implementing transportable parallel codes. We use a package called SCHEDULE [13] that we have been developing while this algorithm was being devised and tested. SCHEDULE is a package of Fortran subroutines designed to aid in programming explicitly parallel algorithms for numerical calculations. The design goal of SCHEDULE is to aid a programmer familiar with a Fortran programming environment to implement a parallel algorithm in a style of Fortran that will lend itself to transporting the resulting program across a wide variety of parallel machines. An important part of this package is the provision of a mechanism for dynamically spawning processes even when such a capability is not present within the parallel language extensions provided for a given machine. The Alliant computer is an example. A detailed discussion of this package is beyond the scope of this paper and will be presented elsewhere. A description of the package and its capabilities is available in [13].

8. Performance

In this section we present and analyze the results of this algorithm on a number of machines. The same algorithm has been run on a VAX 11/785 , Denelcor HEP, Alliant FX/8, and a CRAY X-MP-4.

We have compared our implementation of the algorithm described in this paper to TQL2 from the EISPACK collection. Table 8.1 gives the ratio of execution time for TQL2 from EISPACK run sequentially and the algorithm presented here run in parallel on the same machine. In all cases, the TQL2 code used to test against and the TQL2 code used at lowest level in our algorithm were exactly the same code. Both the parallel algorithm and the serial version of TQL2 were called by a common driver test routine during the same computer run. This assures that both routines were compiled under the same compiler options and ran in the same computing environment to produce this timing data. The timings for TQL2 were obtained by executing it as a single process. By this we mean that the code for TQL2 executed on a single processor of the Alliant and CRAY machines and as the only created process executing on the HEP machine. In all cases the computations were carried out as though the tridiagonal matrix had come from Householder's reduction of a dense symmetric matrix to tridiagonal form. The identity was passed in place of the orthogonal basis that would have been provided by this reduction, but the arithmetic operations performed were the same as those that would have been required to transform that basis into the eigenvectors of the original symmetric matrix.

Table 8.1

n	Ratio of time TQL2/SESUPD	TQL2 $\|Ax - \lambda x\|$	SESUPD $\|Ax - \lambda x\|$	TQL2 $\|X^T X - I\|$	SESUPD $\|X^T X - I\|$
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

As can be seen in Table 8.2, 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.

Table 8.2

	VAX 785/FPA	*Denelcor HEP*	*Alliant FX/8*	*CRAY X-MP-1*	*CRAY X-MP-4*
random matrix	*2.6*	*12*	*15.2*	*1.8*	*4.5*

order = 150

Ratio of execution time $\frac{TQL2\ time}{parallel\ time}$

These results are remarkable because in all cases speedups greater than the number of physical processors were obtained. The gain is due to the numerical properties of the deflation portion of the parallel algorithm. We do not full understand why the algorithm performs as much deflation as is apparent by the comparisons. In all cases the word length was 64 bits and the same level of accuracy was achieved by both methods. The measurement of accuracy used was the maximum 2-norm of the residuals $Tq - \lambda q$ and of the columns of $Q^T Q - I$. The results in Table 8.1 are typical of the performance of this algorithm on random problems with speedups becoming more dramatic as the matrix order increases. In problems of order 500 speedups of 15 have been observed on the CRAY-XMP-4 and speedups over 50 have been observed on the Alliant FX/8, which are 4 and 8 processor machines respectively. The CRAY results can actually be improved because parallelism at the root finding level was not exploited in the implementation run on the CRAY but was fully exploited on the Alliant.

In Table 8.3 and 8.4 we show a more complete set of runs on the Alliant/FX8

Table 8.3

n	Ratio of time TQL2/SESUPD	TQL2 $\|\|Ax - \lambda x\|\|$	SESUPD $\|\|Ax - \lambda x\|\|$	TQL2 $\|\|X^T X - I\|\|$	SESUPD $\|\|X^T X - I\|\|$
100	9.4	9.5×10^{-15}	1.9×10^{-15}	7.8×10^{-15}	5.5×10^{-16}
200	15.4	1.7×10^{-14}	2.7×10^{-15}	1.1×10^{-14}	2.2×10^{-15}
300	17.7	1.9×10^{-14}	3.2×10^{-15}	1.7×10^{-14}	2.6×10^{-15}
400	20.0	2.9×10^{-14}	4.0×10^{-15}	2.0×10^{-14}	9.2×10^{-15}

A comparison on the Alliant FX/8 for TQL2 vs the parallel algorithm on (1,2,1) matrix

Table 8.4

n	Ratio of time TQL2/SESUPD	TQL2 $\|Ax - \lambda x\|$	SESUPD $\|Ax - \lambda x\|$	TQL2 $\|X^T X - I\|$	SESUPD $\|X^T X - I\|$
100	12.1	1.6×10^{-14}	1.9×10^{-13}	1.6×10^{-14}	2.4×10^{-15}
200	19.5	2.1×10^{-14}	2.2×10^{-13}	2.8×10^{-14}	2.3×10^{-15}
300	38.8	4.2×10^{-14}	8.8×10^{-13}	3.7×10^{-14}	5.2×10^{-15}
400	60.7	3.5×10^{-14}	8.2×10^{-13}	3.7×10^{-14}	4.6×10^{-14}

A comparison on the Alliant FX/8 for TQL2 vs the parallel algorithm on a random matrix

These test problems show two types of behavior. In the (1,2,1) matrix not very much deflation takes place. On the random matrix considerable deflation takes place. 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. However, as we mentioned in Section 5 it is not necessary to compute the eigenvectors if they are not needed. We cannot recommend this algorithm in the case where only a few eigenvalues and eigenvectors are sought. A multisectioning algorithm such as the one developed by Lo, Philippe and Sameh [6] is to be preferred in this case.

9. References

[1] R. G. Babb II, *Parallel Processing with Large Grain Data Flow Techniques,* IEEE Computer, Vol. 17, No. 7 , pp. 55-61, July 1984.

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

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

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

[5] L. Jessop, Yale Computer Science Research Report, *"Solving the Symmetric Tridiagonal Eigenvalue Problem on the Hypercube"* in preparation.

[6] S. Lo, B. Philippe, A. Sameh, *A Parallel Algorithm for the Real Symmetric Tridiagonal Eigenvalue Problem,* Center for Supercomputing Research and Development Report,

University of Illinois, Urbana Illinois, Nov. 1985.

[7] E. Lusk and R. Overbeek, "Implementation of Monitors with Macros: A Programming Aid for the HEP and Other Parallel Processors", ANL-83-97, (1983).

[8] E. Lusk and R. Overbeek, "An Approach to Programming Multiprocessing Algorithms on the Denelcor HEP", ANL-83-96, (1983).

[9] 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.

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

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

[12] 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.

[13] D. Sorensen, *SCHEDULE Users Guide,* ANL-MCS-TM-76, Argonne National Laboratory, May, 1986.

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

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

References