ANL-80-84

ANL-80-84

IMPROVING THE ACCURACY OF COMPUTED MATRIX EIGENVALUES

by

Jack J. Dongarra

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

Prepared for the U. S. DEPARTMENT OF ENERGY
under Contract W-31-109-Eng-38

Distribution Category:
Mathematics and Computers
(UC-32)

ANL-80-84

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

IMPROVING THE ACCURACY OF COMPUTED MATRIX EIGENVALUES

by

Jack J. Dongarra

Applied Mathematics Division

Based on a thesis prepared for
the degree Doctor of Philosophy in Mathematics
from the University of New Mexico

August 1980

TABLE OF CONTENTS

IMPROVING THE ACCURACY OF

COMPUTED MATRIX EIGENVALUES

Jack J. Dongarra

ABSTRACT

This dissertation describes a computational method for improving the accuracy of a given eigenvalue and its associated eigenvector, arrived at through a computation in a lower precision. The method to be described will increase the accuracy of the pair and do so at a relatively low cost.

The technique used is similar to iterative refinement for the solution of a linear system. That is, using the factorization from the low-precision computation, an iterative algorithm is applied to increase the accuracy of the eigenpair. Extended precision arithmetic is used at critical points in the algorithm. The iterative algorithm requires $O(n^2)$ operations for each iteration.

Summary of Results

Wilkinson [24] has proposed an algorithm to determine rigorous error bounds for a single computed eigenvalue and the corresponding eigenvector. The method can also be used to improve the accuracy of a given eigenvalue, λ, and its associated eigenvector, x. (We will refer to λ and x as an eigenpair.) Wilkinson describes the conditions under which the method converges, and discusses the behavior of the method when there are multiple roots corresponding to a well conditioned eigenproblem.

In this dissertation we extend the method, as described by Wilkinson, to use information generated during the initial eigenvalue calculation in order to reduce the operational cost of the method. We will show the relationship between the improvement method and Newton's Method. We then extend the improvement algorithm to calculate the eigenvector, given an initial eigenvalue and the decomposition that produced the eigenvalue. Finally, we discuss a way to improve the invariant subspace for the ill-conditioned eigenvalue problem.

CHAPTER 2

Background

An extensive review of the eigenvalue problem can be found in Wilkinson [22], chapters 1-3. Our intent here is not to review all the methods, but to provide some reference on methods that compute eigenvalues and/or eigenvectors, given some initial approximation. The eigenvalue problem must, by its nature, be solved in an iterative fashion. There are existing algorithms that, once the matrix is reduced to a suitable form by a direct approach, apply an iterative technique to expose the eigenvalues and compute the eigenvectors. Some of these methods compute an eigenvalue or eigenvector without prior information as to where it lies, while other approaches require some initial information such as an approximate eigenvector or approximate eigenvalue to carry out the iteration. We will briefly describe some of these methods. A more detailed review can be found in [22].

The power method has been in general use for quite a number of years. It may be used to find the eigenvector corresponding to the dominant eigenvalue. The convergence properties are governed by the degree of dominance of the eigenvalue λ_1, which corresponds to the eigenvector we wish to find, as measured by the ratio $|\lambda_1/\lambda_i|$ where $i=2,\ldots,n$. A drawback of the method is that convergence can be very slow or fail when the eigenvalue associated with the eigenvector is not dominant, that is when the ratio is near one.

The inverse power method or inverse iteration is an attempt to circumvent the dominance problems with the power method. Here the matrix is transformed to have the given eigenvalue as the largest one,

and a few iterations with the power method usually converges to the eigenvector. Inverse iteration is the most widely used technique for computing eigenvectors for selected eigenvalues. Its wide usage is a result of the accuracy gained after only one or two iterations. For the inverse power method, an approximate eigenvalue is needed.

If on the other hand we have a fairly good approximate eigenvector, but no information about the eigenvalue, the Rayleigh quotient can be used to determine the eigenvalue. If the inverse power method is then applied with this approximate eigenvalue from the Rayleigh quotient, there results a new approximate eigenvector. The process may be iterated by computing a new Rayleigh quotient and then applying the inverse power method.

A method that at first sight appears unrelated to inverse iteration is Newton's Method applied to the eigenvalue problem. With Newton's Method both the eigenvalue and the eigenvector are corrected on each iteration. That is, starting with some initial approximation to the eigenvalue and eigenvector, Newton's Method will improve the pair.

With this brief background, we will now turn to the central idea of this paper: improving the accuracy of a given eigenvalue-eigenvector pair.

Basic Method

Wilkinson [24] has described a method for determining rigorous error bounds for a simple eigenvalue and its associated eigenvector. The algorithm has the pleasing feature of providing an improved eigenpair as a by-product. Wilkinson's approach assumes that an eigenpair is given. No assumptions are made about how that eigenpair was found, whether through some knowledge of the physical problem, an initial eigenvalue decomposition in a lower precision or a clever guess.

We are interested in improving the accuracy of an eigenvalue-eigenvector pair. Consider the eigenvalue problem $Ax = \lambda x$, where λ and x have been found by some means. Because they were arrived at by some calculation on a computer with finite precision or by some insight into the problem, they are, in general, not the true eigenvalue and eigenvector, but an approximation. We know, however, that there exist μ and $\tilde{y}$ such that

$$A(x+\tilde{y}) = (\lambda+\mu)(x+\tilde{y}) \tag{1}$$

is the exact solution to the eigenvalue problem, where μ and $\tilde{y}$ are the corrections to the computed λ and x.

We will normalize x such that $\|x\|_\infty = 1$ and say $x_s = 1$, where the s^{th} component of x is the largest. This can be done because we have one degree of freedom in our choice of the components for x. We will assume that the s^{th} component of x is exact and no correction is needed. This determines the value of $\tilde{y}_s$, which is the correction to x_s. Because x_s

is exact, the value of $\tilde{y}_s$ is zero. This also determines the degree of freedom in the corrected vector, $x+\tilde{y}$, through the relationship between x and $\tilde{y}$, namely $(x+\tilde{y})_s = 1$.

We can rewrite equation (1) as

$$(A-\lambda I)\tilde{y} - \mu x = \lambda x - Ax + \mu\tilde{y} \ . \tag{2}$$

Note that $\lambda x - Ax$ is the residual for the computed eigenvalue and eigenvector. If we look more closely at the product $(A-\lambda I)\tilde{y}$, we discover that because $\tilde{y}_s = 0$, the s^{th} column of $A-\lambda I$ does not participate in the product with $\tilde{y}$. In the formulation of $(A-\lambda I)\tilde{y}-\mu x$, we can replace the s component of $\tilde{y}$, which is zero, by the value μ and the s column of $A-\lambda I$ by $-x$ to arrive at $(A-\lambda I)\tilde{y}-\mu x$.

We will define y by $y \equiv \tilde{y}+\mu e_s$, where e_s is the s^{th} column of the identity matrix. So the s^{th} component of the newly defined y has the value μ; i.e., $y_s = \mu$. We will also define the matrix B as the matrix $A-\lambda I$ with the s^{th} column replaced by $-x$. Thus we can rewrite (2) as

$$By = r + y_s\tilde{y} \ , \tag{3}$$

where $r = \lambda x - Ax$.

Another way to view the equation (3) is to consider the matrix $\overline{B}$ as the $(n+1)\times(n+1)$ matrix of the form

$$\overline{B} = \begin{pmatrix} A-\lambda I & -x \\ e_s^T & 0 \end{pmatrix} \ .$$

Then (3) can be derived from (2) by one step of elimination on (2). We would like to solve

$$\bar{B}\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix} = \begin{pmatrix} r + y_s\tilde{y} \\ 0\end{pmatrix} .$$

To solve this, we can apply a permutation, P_s, which will interchange the s^{th} and the $n+1^{th}$ columns of the matrix $\bar{B}$, so that

$$\bar{B}_s P_s^{-1}\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix} = \begin{pmatrix} r + y_s\tilde{y} \\ 0\end{pmatrix} .$$

When the permutation is applied to $\bar{B}$, it has the effect of interchanging the s^{th} and $n+1^{th}$ columns and swapping the s^{th} and $n+1^{th}$ elements of $\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix}$ so that

$$\bar{B}P_s = \begin{pmatrix} a_1 & a_2 & \cdots & a_{s-1} & -x & a_{s+1} & \cdots & a_n & a_s \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & & 1 \end{pmatrix}$$

where a_j is the j^{th} column of $A-\lambda I$ and

$$P_s^{-1}\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{s-1} \\ \mu \\ y_{s+1} \\ \vdots \\ y_n \\ 0 \end{pmatrix} .$$

Because the n+1 element of the solution vector, $P_s^{-1}\begin{pmatrix}\tilde{y}\\ \mu\end{pmatrix}$, is known, we will solve with the truncated form of $\overline{B}$, truncated so the n+1 row and n+1 column are no longer present. This truncation can be done because we know the solution vector has a zero in the $n+1^{th}$ position.

Equation (3) is a nonlinear equation defining the correction y. Wilkinson has investigated the following iterative method for solving (3),

$$\begin{aligned} By^{(p+1)} &= r + y_s^{(p)}\tilde{y}^{(p)} \quad , \quad \text{where} \\ \tilde{y}^{(p)} &= y^{(p)} - y_s^{(p)}e_s \quad . \end{aligned} \tag{4}$$

We will show in chapter 7 that, when certain conditions are satisfied, the iteration described above is a contraction and therefore converges. Considering the previous iterate to p+1 in (4), we have

$$By^{(p)} = r + y_s^{(p-1)}\tilde{y}^{(p-1)} \quad .$$

When the two iterates are subtracted, we get

$$B\delta^{(p)} = y_s^{(p)}\tilde{y}^{(p)} - y_s^{(p-1)}\tilde{y}^{(p-1)} \quad , \tag{5}$$

where $\delta^{(p)} = y^{(p+1)} - y^{(p)}$.

We could solve (4) in an iterative fashion and in the absence of floating point arithmetic we would do so. The right hand side of equation (4) contains two quantities, a fixed part, the residual, and the term $y_s\tilde{y}$ which changes from iteration to iteration. As the process converges on a computer with floating point arithmetic the term $y_s\tilde{y}$ will rapidly become negligible compared to the residual. This may happen, in fact, before the attainable precision is reached. In order to prevent

this premature termination we will not calculate the correction y directly, but instead compute a correction δ, which is a correction to y.

Another source of computational difficulty may be that the accuracy of $y_s^{(p)}\tilde{y}^{(p)} - y_s^{(p-1)}\tilde{y}^{(p-1)}$ may suffer from cancellation when subtraction occurs in the presence of floating point arithmetic. This problem is easily avoided by expanding $y_s^{(p)}\tilde{y}^{(p)}$ in terms of $y^{(p-1)}$ and $\delta^{(p-1)}$. We have

$$y_s^{(p)}\tilde{y}^{(p)} = \left(y_s^{(p-1)}+\delta_s^{(p-1)}\right)\left(\tilde{y}^{(p-1)}+\tilde{\delta}^{(p-1)}\right) ,$$

which leads to

$$\begin{aligned} y_s^{(p)}\tilde{y}^{(p)} - y_s^{(p-1)}\tilde{y}^{(p-1)} &= y_s^{(p-1)}\tilde{y}^{(p-1)} + y_s^{(p-1)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{y}^{(p-1)} \\ &\quad + \delta_s^{(p-1)}\tilde{\delta}^{(p-1)} - y_s^{(p-1)}\tilde{y}^{(p-1)} \\ &= y_s^{(p-1)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{y}^{(p-1)} + \delta_s^{(p-1)}\tilde{\delta}^{(p-1)} \\ &= y_s^{(p)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{y}^{(p-1)} . \end{aligned}$$

This quantity avoids the severe cancellation which may occur and is used in the programs which implement the algorithm. The iteration thus becomes:

$$\begin{aligned} B\delta^{(0)} &= r = \lambda x - Ax & y^{(1)} &= \delta^{(0)} \\ B\delta^{(1)} &= y_s^{(1)}\tilde{y}^{(1)} & y^{(2)} &= y^{(1)} + \delta^{(1)} \\ B\delta^{(2)} &= y_s^{(2)}\tilde{\delta}^{(1)} + \delta_s^{(1)}\tilde{y}^{(1)} & y^{(3)} &= y^{(2)} + \delta^{(2)} \\ &\vdots & &\vdots \\ B\delta^{(p)} &= y_s^{(p)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{y}^{(p-1)} & y^{(p+1)} &= y^{(p)} + \delta^{(p)} \end{aligned}$$

We see that $\delta^{(p)}$ is a correction to $y^{(p)}$ and that the final y contains the correction to the eigenvalue and eigenvector. It should be noted here that the residual calculation, $r = \lambda x - Ax$, is a critical part of the method's performance. In order to guarantee that rounding errors do not invalidate the analysis, we will assume that the residual is calculated using extended precision. The error analysis described in chapter 8 details the reasons for demanding this.

The iterative method given above requires one to have an approximation to the eigenvalue λ and the eigenvector x in its first step. Next, a system of equations is solved to obtain a correction to the pair; this procedure is repeated until convergence. Each iteration involves the same matrix B, which is $A-\lambda I$ with the s column replaced by $-x$, but with the right hand side changed. Every iteration gives rise to a correction to a correction, and, as we will show, this process converges rapidly under certain conditions. It is important to point out the fact that we do not solve for a correction to (λ,x) directly, but solve for $\delta^{(p)}$ which is the correction to $y^{(p)}$. At each step, $y^{(p+1)}$ is determined from $y^{(p)}$ and $\delta^{(p)}$. When the process terminates, the final y contains the correction to (λ,x).

The iterative method that arises from (5) can be stated in algorithmic form

Algorithm 1: The Improvement Algorithm (Wilkinson)

Given A, λ and x; $\|x\|_\infty = 1$ and $x_s = 1$

1: $r \leftarrow \lambda x - Ax$; (extended precision used)

form B; ($A-\lambda I$ with s column replaced by $-x$)

2: factor B into L and U;

3: $B\delta^{(0)} = r$; (solve for $\delta^{(0)}$ using L and U)

$y^{(0)} \leftarrow 0$;

$y^{(1)} \leftarrow \delta^{(0)}$;

$p \leftarrow 0$;

Do until convergence

$p \leftarrow p+1$;

4: $B\delta^{(p)} = y_s^{(p)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{y}^{(p-1)}$; (solve for $\delta^{(p)}$ using L and U)

$y^{(p+1)} \leftarrow y^{(p)} + \delta^{(p)}$;

test for convergence

iterate end;

$\lambda \leftarrow \lambda + y_s^{(p+1)}$;

$x \leftarrow x + \tilde{y}^{(p+1)}$;

Operation Count 1: n^2

2: $1/3\ n^3$

3: n^2

4: $p(n^2)$

Total Count: $1/3\ n^3 + (p+2)n^2$

In the operation count here and in later ones, n refers to the order of the matrix and p refers to the number of iterations needed to converge. Operation counts of O(n) are ignored.

The method outlined above requires a system of equations to be solved p times, where p is the number of iterations required to converge. The initial decomposition of the matrix B used to solve a system of equations involves $O(n^3)$ operations and $O(n^2)$ operations in solving for a specific right hand side. The matrix B is not affected by the correction, so the decomposition need be performed only once, but the right hand side does change at each iteration.

To gain a more accurate solution, $\delta^{(p)}$, at each step, one might attempt iterative refinement in the solution to (5). We can incorporate one step of iterative refinement of $\delta^{(p-1)}$ in the same step in which we determine $\delta^{(p)}$. To do so, we simply augment the right-hand side of (5) by the residual of the previous system corresponding to the computed $\delta^{(p-1)}$.

The first step of the process is to solve $B\delta^{(0)} = r = \lambda x - Ax$. Round-off errors are introduced so $B\delta^{(0)}$ is not exactly equal to r. One iterative step would give

$$Bz^{(0)} = B\delta^{(0)} - (\lambda x - Ax) ,$$

then

$$\bar{\delta}^{(0)} = \delta^{(0)} + z^{(0)} .$$

The next step would look like

$$B\bar{\delta}^{(1)} = y_s^{(1)} \tilde{y}^{(1)} .$$

This process requires two systems to be solved at each step, one for δ and the other for the refinement step. The next refinement is of the form

$$Bz^{(1)} = y_s^{(1)} \tilde{y}^{(1)} - B\bar{\delta}^{(1)} = r^{(1)}$$

and

$$\bar{\delta}^{(2)} = \bar{\delta}^{(1)} + z^{(1)} .$$

Note that the final correction desired is $y^{(p+1)}$, which is

$$y^{(p+1)} = y^{(p)} + \delta^{(p)}$$

or, expanded, looks like

$$y^{(p+1)} = \sum_{i=0}^{p} \delta^{(i)} .$$

The refinement gives $\bar{\delta}$ of the form

$$\begin{aligned} \bar{\delta}^{(0)} &= \delta^{(0)} + z^{(0)} \\ \bar{\delta}^{(1)} &= \delta^{(1)} + z^{(1)} \\ &\vdots \\ \bar{\delta}^{(p-1)} &= \delta^{(p-1)} + z^{(p-1)} \end{aligned}$$

so that $y^{(p+1)}$, if refined at each step is

$$\begin{aligned} y^{(p+1)} &= \sum_{i=0}^{p} \bar{\delta}^{(i)} \\ &= \sum_{i=0}^{p} \left(\delta^{(i)} + z^{(i)}\right) . \end{aligned}$$

We can reduce the number of systems to be solved by taking advantage of the additive nature for the refinement and the correction. This can be done by solving a system based on the current correction $\delta^{(p)}$ and the previous refinement $z^{(p-1)}$, so that

$$B\bar{\delta}^{(p)} = \left[r^{(p-1)} - B\bar{\delta}^{(p-1)}\right] + y_s^{(p)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{y}^{(p-1)} = r^{(p)} \qquad (6)$$

where $r^{(0)} = \lambda x - Ax$ and the $\bar{\delta}$ contains the following:

$$\begin{aligned}
\bar{\delta}^{(0)} &= \delta^{(0)} \\
\bar{\delta}^{(1)} &= \delta^{(1)} + z^{(0)} \\
&\vdots \\
\bar{\delta}^{(p)} &= \delta^{(p)} + z^{(p-1)}
\end{aligned}$$

Then the final correction $y^{(p+1)}$ will have incorporated in it the advantage gained by performing one step of iterative refinement at each step except the last; no additional system of equations need be solved, thereby saving n^2 operations each iteration. On the last iteration the answer will have converged so no refinement is necessary.

The iterative method which arises from (6) can be stated in algorithmic form as:

Algorithm 1.1: The Improvement Algorithm with One Step of Iterative Improvement (Wilkinson)

Given A, λ and x; $\|x\|_\infty = 1$ and $x_s = 1$

1: $r^{(0)} \leftarrow \lambda x - Ax$; (extended precision used)

form B; (A-λI with s column replaced by -x)

2: factor B into L and U;

3: $B\delta^{(0)} = r^{(0)}$; (solve for $\delta^{(0)}$ using L and U)

$y^{(0)} \leftarrow 0$;

$y^{(1)} \leftarrow \delta^{(0)}$;

$p \leftarrow 0$;

Do until convergence

$p \leftarrow p+1$;

4: $z \leftarrow r^{(p-1)} - B\delta^{(p-1)}$; (extended precision used)

$r^{(p)} \leftarrow z + y_s^{(p)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{\delta}^{(p-1)}$;

5: $B\delta^{(p)} = r^{(p)}$; (solve for $\delta^{(p)}$ using L and U)

$y^{(p+1)} \leftarrow y^{(p)} + \delta^{(p)}$;

test for convergence

iterate end;

$\lambda \leftarrow \lambda + y_s^{(p+1)}$;

$x \leftarrow x + \tilde{y}^{(p+1)}$;

Operation Count 1: n^2
2: $1/3\ n^3$
3: n^2
4: $p(n^2)$
5: $p(n^2)$

Total Count $1/3\ n^3 + (2p+2)n^2$

Method Using Updates on the Factors

In his discussion of the method, Wilkinson assumes that the approximate eigenpair (λ,x) is known. Then the improvement algorithm is used to gain more accuracy in the pair. The algorithm requires $1/3\ n^3 + (2p+2)n^2$ operations to perform p steps of improvement. Most of the work in the algorithm, $O(n^3)$ term, is consumed by factoring a matrix. We will assume that an LU factorization is performed, although any other stable factorization could be used. The factors are then used to repeatedly solve systems of equations, $O(n^2)$ term. In many applications users may have no knowledge of the eigenpair a priori, but they may want a very accurate representation of (λ,x). In such cases they must resort to an initial eigenvalue-eigenvector calculation, then use the improvement algorithm to gain more accuracy. The initial eigenvalue-eigenvector calculation, if done by the QR algorithm, requires $O(n^3)$ operations, most of which goes into factoring the original matrix. The QR algorithm will factor a matrix into the form $A = QTQ^T$, where Q is orthogonal and T is either triangular or quasi-triangular depending on the nature of the eigenvalues. The improvement algorithm also requires $O(n^3)$ operations, most of which also goes into factoring a slightly different matrix. Although the coefficient of the n^3 term for the improvement algorithm is much smaller than that of the eigenvalue-eigenvector calculation, information generated during the decomposition to find (λ,x) is not being used to its full potential. The algorithm to be discussed in this chapter assumes the QR algorithm has been used to determine the initial eigenvalue and eigenvector approximation and uses

the factorization to reduce the number of operation for the improvement algorithm from $O(n^3)$ to $O(n^2)$.

During the eigenvalue calculation the original matrix, A, is reduced to triangular or quasi-triangular form [20]. If we could use this reduced form of the matrix in the improvement algorithm, the operation count could effectively be dropped to $O(n^2)$, thereby making the improvement calculation very attractive.

We start by looking at

$$B\delta = b \ ,$$

where B is $A-\lambda I$ with the s column replaced by $-x$, s is the index of the largest component of x in magnitude, and $b = y_s^{(p)}\tilde{\delta}^{(p-1)} + \delta_s^{(p-1)}\tilde{\delta}^{(p-1)}$. We can express B as

$$B = A_\lambda + ce_s^T \ , \tag{7}$$

where $A_\lambda = A-\lambda I$, $c = -x-a_{\lambda s}$ and $a_{\lambda s}$ is the s^{th} column of A_λ. Thus matrix B is obtained by applying a rank-one correction to A_λ.

As a first approach, one might consider using the Sherman-Morrison-Woodbury (S-M-W) formula [10, § 2.24] to find the inverse of a matrix with a rank-one correction. The S-M-W update has the form

$$(A_\lambda - uv^T)^{-1} = A_\lambda^{-1} + \alpha A_\lambda^{-1}uv^TA_\lambda^{-1} \ ,$$
$$\text{where } \alpha = \frac{1}{1-v^TA_\lambda^{-1}u} \ .$$

We would be interested in solving a system of the form $(A_\lambda - uv^T)x = b$ so that

$$x = A_\lambda^{-1}b + \alpha A_\lambda^{-1}uv^TA_\lambda^{-1}b \ .$$

For the problem we are dealing with, the matrix A_λ is singular and the S-M-W formula breaks down in the computation of $A_\lambda^{-1}b$ and $A_\lambda^{-1}u$. Thus we see that the S-M-W formula cannot be used, and we look for an alternative.

We have from the QR algorithm's initial eigenvalue calculation a factorization for A of the form $A = QTQ^T$. Here we have assumed that A has all real eigenvalues, T is a triangular matrix, and Q is orthogonal. Later the solution will be discussed for T quasi-triangular and Q orthogonal. We will perform a rank-one update to A_λ which incorporates ce_s^T. So

$$A_\lambda + ce_s^T = QT^T + ce_s^T$$

or $$Q^TA_\lambda Q + Q^Tce_s^TQ = T_\lambda + Q^Tce_s^TQ \ , \quad \text{where } T_\lambda = T-\lambda I,$$

which can be written as

$$Q^T(A_\lambda+ce_s^T)Q = T_\lambda + df^T \qquad \text{where } d = Q^Tc \text{ and } f = Q^Te_s \ .$$

We would like to bring $T_\lambda+df^T$ back to triangular form. Choose the orthogonal matrix Q_1 such that

$$Q_1d = [P_2P_3\cdots P_n]d = \gamma\, e_1 \quad \text{where} \quad \gamma^2 = \|d\|_2^2 \ .$$

P_i is a Givens transformation which eliminates the i^{th} component of a particular vector by taking linear combinations of rows i-1 and i. When Q_1 is applied to $T_\lambda+df^T$, we have

$$\begin{aligned} Q_1(T_\lambda+df^T) &= Q_1T_\lambda + Q_1df^T \\ &= Q_1T_\lambda + \gamma\, e_1f^T \\ &= \begin{pmatrix} u_1^T \\ U_1 \end{pmatrix} + \gamma e_1f^T \ . \end{aligned}$$

The matrix we are left with is in upper Hessenberg form:

$$Q_1(T_\lambda + df^T) = \begin{pmatrix} \overline{u}_1^T \\ U_1 \end{pmatrix}$$

$$= H \quad ,$$

where $\overline{u}_1^T = u_1^T + \gamma f^T$ and H is an upper Hessenberg matrix. This is the first sweep. To summarize, Givens transformations are chosen to eliminate elements from the vector d starting in the n^{th} position and continuing to the second position. These transformations are applied to T_λ, and transform the triangular matrix to Hessenberg form.

Next, the triangular form is restored by applying Givens transformations to the Hessenberg matrix. This defines the second sweep. An orthogonal matrix Q_2 is chosen such that Q_2H is restored to upper triangular form. We have

$$Q_2H = Q_2 \begin{pmatrix} \overline{u}_1^T \\ U_1 \end{pmatrix} = U_2$$

where $Q_2 = [P_n' \ldots P_3' P_2']$ and U_2 is upper triangular. P_i' is a Givens transformation which eliminates the $h_{i,i-1}$ element of H. To summarize, we can work with the triangular form produced by the initial eigenvalue decomposition. This is done by applying a rank one update to the triangular matrix and transforming that updated matrix back to triangular form in $O(n^2)$ operations. Once the matrix is in triangular form, the system of equations can be solved in $O(n^2)$ operations.

The method that arises from adding a rank-one update and the orthogonal reduction to triangular form can be stated in algorithmic form:

Algorithm 2: Improvement Algorithm Using Factors from the Eigenvalue Decomposition

Given A, λ and x; Q and T such that $A = QTQ^T$; $\|x\|_\infty = 1$ and $x_s = 1$

1: $r \leftarrow \lambda x - Ax$; (extended precision used)

$T_\lambda \leftarrow T - \lambda I$;

$c \leftarrow -x - a_{\lambda s}$; ($a_{\lambda s}$ is s column of A_λ)

$f \leftarrow Q^T e_s$; (e_s is s column of I)

2: $d \leftarrow Q^T c$;

Compute Q_1 such that $Q_1 d = \gamma e_1$;

Compute Q_2 such that $Q_2(Q_1(T_\lambda + df^T))$ is triangular;

3: $\bar{T}_\lambda \leftarrow Q_2 Q_1 (T_\lambda + df^T)$;

$\bar{y}^{(0)} \leftarrow 0$;

4: $\bar{y}^{(1)} \leftarrow Q_2 Q_1 Q^T r$;

$p \leftarrow 0$;

Do until convergence

$p \leftarrow p+1$;

5: $\bar{T}_\lambda \delta^{(p)} = \bar{y}_s^{(p)} \tilde{\delta}^{(p-1)} + \delta_s^{(p-1)} \tilde{\bar{y}}^{(p-1)}$; (solve for $\delta^{(p)}$)

$\bar{y}^{(p+1)} \leftarrow \bar{y}^{(p)} + Q_2 Q_1 \delta^{(p)}$;

test for convergence

iterate end;

6: $y^{(p+1)} \leftarrow Q Q_1^T Q_2^T \bar{y}^{(p+1)}$;

$\lambda \leftarrow \lambda + y_s^{(p+1)}$;

$x \leftarrow x + \tilde{y}^{(p+1)}$;

Operation Count:

1:	n^2	4:	n^2
2:	n^2	5.	$p(1/2\ n^2)$
3:	n^2	6:	n^2

Total Count: $(5 + p/2)n^2$

Algorithm 2 requires the factors Q and T generated when the QR algorithm is applied to the matrix A. The QR algorithm will also produce the approximation to the eigenpair, which will be used to start the improvement process. For Algorithm 1, no assumption is made as to where the initial eigenpair originated, so no factorization is known for the matrix A. This is the main difference between the two algorithms.

One of the main applications of Algorithm 2 is when more precision is required for a computed eigenpair. Algorithm 1, on the other hand, has applications when one has an approximate eigenpair by some knowledge of the problem other than an actual eigenvalue calculation, and wishes to improve the approximation.

Method Using Repeated Updates on the Factors

The methods developed in the earlier chapters can be further extended by updating the matrix B and recomputing the residual with the corrected eigenpair at each iteration. During the course of the iteration, corrections for the pair (λ,x) are generated and used to construct the right-hand side of the equation. The matrix B and residual being used to solve the system depend upon the approximate eigenpair (λ,x), but in the earlier algorithms the initial values $(\lambda^{(0)},x^{(0)})$ were used in defining these quantities, and thus were held fixed during the iteration. If the value of $(\lambda^{(p)},x^{(p)})$ were used during the course of the iteration to correct B, the following iteration would result:

$$B^{(p)}\delta^{(p)} = r^{(p)} \tag{8}$$

$$\text{where} \quad B^{(p)} = (A-\lambda^{(p)}I)+c^{(p)}e_s^T$$

$$c^{(p)} = -x^{(p)}-a_{\lambda s}^{(p)}$$

$$a_{\lambda s}^{(p)} = (A-\lambda^{(p)}I)e_s^T \quad .$$

Another way to view the process is to restart the algorithm with the corrected eigenpair $(\lambda^{(p)},x^{(p)})$ at each iteration. Thus at each iteration a new residual is calculated,

$$r^{(p)} = \lambda^{(p)}x^{(p)}-A\cdot\cdot(p) \quad ,$$

a new matrix and update constructed from the eigenvalue decomposition and the improved pair

$$T_{\lambda}^{(p)} = T - \lambda^{(p)} I \ , \qquad c^{(p)} = -x^{(p)} - a_{\lambda_s} \ ,$$

and a new correction $\delta^{(p)}$ calculated based on this corrected information.

The method can be stated in algorithmic form as:

Algorithm 3: The Improvement Algorithm Using Factors from the Eigenvalue Decomposition with Updates to λ and x at Each Stage

Given A, λ and x; Q and T such that $A = QTQ^T$; $\|x\|_\infty = 1$ and $x_s = 1$

$p \leftarrow 0$;

$\lambda^{(1)} \leftarrow \lambda$;

$x^{(1)} \leftarrow x$;

Do until convergence

$p \leftarrow p+1$;

1: $r^{(p)} \leftarrow \lambda^{(p)}x^{(p)} - Ax^{(p)}$; (extended precision used)

$T_\lambda^{(p)} \leftarrow T - \lambda^{(p)}I$;

$c \leftarrow -x^{(p)} - a_{\lambda s}$; ($a_{\lambda s}$ is s column of $A-\lambda^{(p)}I$)

2: $d \leftarrow Q^T c$;

$f \leftarrow Q^T e_s$; (e_s is s column of I)

compute $Q_1^{(p)}$ such that $Q_1^{(p)}d = \gamma e_1$;

compute $Q_2^{(p)}$ such that $Q_2^{(p)}(Q_1^{(p)}(T_\lambda^{(p)}+df^T))$

is triangular;

3: $\overline{T}_\lambda^{(p)} \leftarrow Q_2^{(p)}Q_1^{(p)}(T_\lambda+df^T)$;

4: $z \leftarrow Q_2^{(p)}Q_1^{(p)}r^{(p)}$;

5: $\overline{T}_\lambda^{(p)}\delta^{(p)} = z$; (solve for $\delta^{(p)}$)

6: $\overline{\delta}^{(p)} \leftarrow QQ_1^{(p)}T_{Q_2}^{(p)}T_\delta^{(p)}$;

$\lambda^{(p+1)} \leftarrow \lambda^{(p)} + \overline{\delta}_s^{(p)}$;

$x^{(p+1)} \leftarrow x^{(p)} + \overline{\delta}^{(p)}$;

test for convergence

iterate end;

Operation Count:	1: $p(n^2)$	4: $p(n^2)$
	2: $p(n^2)$	5: $p(1/2\ n^2)$
	3: $p(n^2)$	6: $p(n^2)$

Total Count: $11/2\ pn^2$

A natural extension to the method just described is to develop a hybrid-like approach that incorporates features from Algorithm 2 and Algorithm 3. That is, perform a few iterations based upon a given approximate pair (λ,x), then correct the pair, update the matrix and iterate again. We have found that Algorithm 3 by itself converges very quickly and the inner iteration is not really needed. By using Algorithm 3 we will do slightly more work per iteration but fewer iterations.

Relationship to Newton's Method

The methods described in the previous chapters resemble Newton's Method for the eigenvalue problem. The eigenvalue problem can be written as

$$(A-\lambda I)x = 0 \quad \text{with} \quad e_s^T x = 1 \quad \text{for some fixed } s.$$

If we let $v = \begin{pmatrix} x \\ \lambda \end{pmatrix}$ and define a function f mapping $\mathbb{R}^{n+1}$ to $\mathbb{R}^{n+1}$ as $f(v) = \begin{pmatrix} Ax-\lambda x \\ e_s^T x-1 \end{pmatrix}$, the eigenvalue problem can be stated as finding the zeros of $f(v)$. Newton's Method applied to this problem is

$$f'(v_i)(v_{i+1}-v_i) = -f(v_i)$$

$$\text{where } v_i = \begin{pmatrix} x_i \\ \lambda_i \end{pmatrix}.$$

The derivative of $f(v_i)$ is

$$f'(v_i) = \begin{pmatrix} A-\lambda_i I & -x_i \\ e_s^T & 0 \end{pmatrix}.$$

The above method expressed in matrix notation is

$$\begin{pmatrix} A-\lambda_i I & -x_i \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} \delta x_i \\ \delta \lambda_i \end{pmatrix} = \begin{pmatrix} r_i \\ 0 \end{pmatrix} \tag{9}$$

$$\text{where } r_i = \lambda_i x_i - Ax_i .$$

This method can be restated as follows.

$$(A-\lambda_i I)\delta x_i - \delta\lambda_i x_i = r_i \quad \text{where} \quad e_s^T \delta x_i = 0 \text{ and } e_s^T x_i = 1 \ . \qquad (10)$$

A common variant of Newton's Method, referred to as simplified Newton's Method [15], holds the Jacobian fixed over the entire iteration, i.e., $f'(v_i)$ is replaced by $f'(v_0)$.

For the simplified Newton's Method we have

$$\begin{pmatrix} A-\lambda_0 I & -x_0 \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} \delta x_i \\ \delta\lambda_i \end{pmatrix} = \begin{pmatrix} r_i \\ 0 \end{pmatrix}$$

or $$A\delta x_i - \lambda_0 \delta x_i - x_0 \delta\lambda_i = \lambda_i x_i - Ax_i \qquad (11)$$

$$\text{with } e_s^T \delta x_i = 0 \ ,$$

where $x_{i+1} = x_i + \delta x_i$ $\qquad$ $\lambda_{i+1} = \lambda_i + \delta\lambda_i$

$$x_{i+1} = x_0 + \sum_{k=0}^{i} \delta x_k \qquad \lambda_{i+1} = \lambda_0 + \sum_{k=0}^{i} \delta\lambda_k \ .$$

We will define

$$\Delta x_i = \sum_{k=0}^{i} \delta x_k \quad \text{and} \quad \Delta\lambda_i = \sum_{k=0}^{i} \delta\lambda_k \qquad \text{for } i = 0,1,2,\ldots$$

so that $x_{i+1} = x_0 + \Delta x_i$ and $\lambda_{i+1} = \lambda_0 + \Delta\lambda_i$.

The fundamental difference between the simplified Newton's Method and the improvement algorithm is that the simplified Newton method gives a correction, δx_{i+1}, which is added to a previously improved solution; i.e., $x_{i+1} = x_i + \delta x_i$ and $\lambda_{i+1} = \lambda_i + \delta\lambda_i$. Thus at every iteration we are building up the corrected solution. In the improvement algorithm, $x_{i+1} = x_0 + \tilde{y}^{(i+1)}$ and $\lambda_{i+1} = \lambda_0 + y_s^{(i+1)}$. Here the improvement comes in one

quantity, $y^{(i+1)}$, at the end of the process and is then added directly to x_0 and λ_0 to get the final corrected solution; we are not computing an accumulated corrected pair at each stage directly. This is the basic difference between the two methods.

We will now turn to the Improvement Method to see the similarity between it and the simplified Newton Method.

Theorem: In the absence of round-off errors, the simplified Newton Method applied to the eigenvalue problem and Algorithm 1 (Improvement Method) produce the same final iterate, (λ_i, x_i).

Proof: The simplified Newton method for the eigenvalue problem can be stated as

$$\begin{pmatrix} A-\lambda_0 I & -x_0 \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} \delta x_i \\ \delta\lambda_i \end{pmatrix} = \begin{pmatrix} r_i \\ 0 \end{pmatrix} = \begin{pmatrix} \lambda_i x_i - Ax_i \\ 0 \end{pmatrix}, \tag{12}$$

$$\text{where} \quad x_i = x_0 + \Delta x_{i-1} \quad \text{and}$$

$$\lambda_i = \lambda_0 + \Delta\lambda_{i-1} .$$

We will expand $\lambda_i x_i - Ax_i$ in the terms above:

$$\begin{aligned} &(\lambda_0 + \Delta\lambda_{i-1})(x_0 + \Delta x_{i-1}) - A(x_0 + \Delta x_{i-1}) \\ &= \lambda_0 x_0 + \lambda_0 \Delta x_{i-1} + \Delta\lambda_{i-1} x_0 \\ &\quad + \Delta\lambda_{i-1}\Delta x_{i-1} - Ax_0 - A\Delta x_{i-1} \\ &= \lambda_0 \Delta x_{i-1} + \Delta\lambda_{i-1} x_0 - A\Delta x_{i-1} + (\lambda_0 x_0 - Ax_0) + \Delta\lambda_{i-1}\Delta x_{i-1} . \end{aligned}$$

Equation (12) can then be written as

$$(A-\lambda_0 I)\delta x_i - \delta\lambda_i x_0 = \lambda_0 \Delta x_{i-1} + \Delta\lambda_{i-1} x_0 - A\Delta x_{i-1} + r_0 + \Delta\lambda_{i-1}\Delta x_{i-1}$$

where $r_0 = \lambda_0 x_0 - Ax_0$.

Rearranging terms in $\Delta\lambda_i$ and Δx_i on the right gives

$$(A-\lambda_0 I)\Delta x_i - \Delta\lambda_i x_0 = r_0 + \Delta\lambda_{i-1}\Delta x_{i-1} . \tag{13}$$

The improvement iterate from Algorithm 1 is

$$B\delta^{(i)} = y_s^{(i)}\tilde{y}^{(i)} - y_s^{(i-1)}\tilde{y}^{(i-1)} = z^{(i+1)} .$$

If $z^{(i)} - B\delta^{(i-1)}$ is added to the right side, it then becomes

$$B\delta^{(i)} = z^{(i)} - B\delta^{(i-1)} + y_s^{(i)}\tilde{y}^{(i)} - y_s^{(i-1)}\tilde{y}^{(i-1)} = z^{(i+1)} .$$

If $z^{(i)}$ is expanded, we have

$$z^{(i)} = z^{(i-1)} - B\delta^{(i-2)} + y_s^{(i-1)}\tilde{y}^{(i-1)} - y_s^{(i-2)}\tilde{y}^{(i-2)}$$

where $z^{(0)} = r_0$.

Substituting the fully expanded expression for $z^{(i)}$ we arrive at

$$B\delta^{(i)} = r_0 - B\delta^{(1)} - B\delta^{(2)} - \ldots - B\delta^{(i-1)} + y_s^{(i)}\tilde{y}^{(i)} .$$

Bringing terms in B to the right and using the definition of

$$y^{(i+1)} = \sum_{k=0}^{i} \delta^{(k)}$$

we arrive at

$$By^{(i+1)} = r_0 + y_s^{(i)} \tilde{y}^{(i)} \quad .$$

By using the definition of B and $y^{(i+1)}$, we have

$$(A-\lambda_0 I)\tilde{y}^{(i+1)} - y_s^{(i+1)} x_0 = r_0 + y_s^{(i)} \tilde{y}^{(i)} \quad ,$$

which is the same as (13) derived from the simplified Newton Method. So the two methods are the same and produce the same final iterate.

Q.E.D.

Because the two methods are the same, we will be assured of convergence for the Improvement Method, whenever the convergence conditions are satisfied for the simplified Newton method in exact arithmetic.

If in the Improvement Method the corrected values for λ_i and x_i are used in constructing the B matrix at every iteration, then the method begins to resemble the full Newton method applied to the eigenvalue problem. We can accomplish this quite easily by restarting the Improvement Method at each iteration. We then have the following theorem.

Theorem. In the absence of roundoff errors, Newton's Method and Algorithm 3 (the Improvement Method restarted every iteration) produce the same final iterate (λ_i, x_i).

Proof. Newton's Method for the eigenvalue problem is $(A-\lambda_i I)\delta x_i - \delta\lambda_i x_i = r_i$ where $e_s^T \delta x_i = 0$. The Improvement Method which is restarted every iteration from equation (8) is $B^{(i)}\delta^{(i)} = r^{(i)}$. If $B^{(i)}$ is expressed in terms of its original parts we have

$$B^{(i)}\delta^{(i)} = \left[(A-\lambda^{(i)}I) + (-x^{(i)}-a_{\lambda s}^{(i)})e_s^T\right]\delta^{(i)}$$

$$\text{where} \quad \delta^{(i)} = \tilde{\delta}^{(i)} + \delta_s^{(i)} e_s$$

$$B^{(i)}\delta^{(i)} = (A-\lambda^{(i)}I)\tilde{\delta}^{(i)} + \delta_s^{(i)} a_{\lambda s} - \delta_s^{(i)} x^{(i)} - \delta_s^{(i)} a_{\lambda s}^{(i)}$$

$$= (A-\lambda^{(i)}I)\tilde{\delta}^{(i)} - \delta_s^{(i)} x^{(i)} = r^{(i)} .$$

But this is the same as Newton's Method because $\tilde{\delta}^{(i)} = \delta x_i$ and $\delta_s^{(i)} = \delta\lambda_i$ when the method is restarted every iteration. Thus, they produce the same final iterate.

Q.E.D.

We will now turn to an algorithm, inverse iteration, which at first sight appears to be completely unrelated to Newton's Method. Inverse iteration is the most widely used method for computing eigenvectors corresponding to selected precomputed eigenvalues. If λ is a good approximation to some eigenvalue, then consider the sequence of vectors x_i derived from:

$$(A-\lambda I)x_{i+1} = k_i x_i ,$$

where x_0 is almost arbitrary and k_i is a scale factor chosen so x_{i+1} is normalized. If A has a full set of eigenvectors, v_j, then x_0 can be expressed as

$$x_0 = \sum_{j=1}^{n} \alpha_j v_j ,$$

consequently

$$x_{i+1} = (k_1 k_2 \ldots k_i) \sum_{j=1}^{n} \alpha_j v_j/(\lambda_j-\lambda)^{i+1} .$$

Let λ_ℓ be the eigenvalue closest to λ. If $|\lambda-\lambda_\ell| << |\lambda-\lambda_j|$ for $j = 1,2,\ldots,n$, $\ell \neq j$ and $\alpha_\ell \neq 0$ then v_ℓ become increasingly dominant. In fact if λ is very accurate, say to working precision, the first iterate x_1 will probably be an accurate eigenvector, possibly correct to working accuracy [17]. The fact that $A-\lambda I$ is near singular is not troublesome, since we are interested in an eigenvector or some multiple of it; i.e. we are interested only in the direction of the eigenvalue being computed. Wilkinson has pointed out the following theorem.

Theorem (Wilkinson). In the absence of roundoff errors, Newton's Method applied to the eigenproblem and inverse iteration produce the same iterates.

Proof. For inverse iteration we have

$$(A-\lambda I)x_{i+1} = k_i x_i ,$$

where $x_{i+1} = x_i+\delta x_i$, δx_i is the correction to go to the next iteration, and k_i is some scale factor. We will assume that $\|x_i\|_\infty = 1$ with $x_{i,s} = 1$, then $\delta x_{i,s} = 0$.

$$\begin{aligned} &(A-\lambda I)(x_i+\delta x_i) = k_i x_i \\ &(A-\lambda I)\delta x_i - k_i x_i = (\lambda_i I-A)x_i = r_i \end{aligned} \qquad (14)$$

If we now go back to equation (10), which is Newton's Method for the eigenvalue problem, we see that k_i, the scale factor for inverse iteration, is the correction to the eigenvalue $\delta\lambda_i$ from Newton's Method. Thus the two methods produce the same iterate.

Q.E.D.

Wilkinson [17] points out that if the initial λ is quite accurate it is not necessary to correct λ. If one does, the matrix $A-\lambda I$, stored in factored form, will change with each iteration. But in the improvement algorithm, the matrix used to solve systems is triangular with a rank-one change, so no additional cost is incurred since we do not have to refactor.

Keeping in mind the relationships between Newton's Method applied to the eigenvalue problem, inverse iteration and the improvement algorithm, we can extend the Improvement Method one step further; namely, given an approximate eigenvalue and the decomposition which gave the approximation, compute the eigenvector from some initial random guess. This is the way in which inverse iteration is used to construct the eigenvector.

The algorithm to carry out the improvement of λ_0 and to compute the eigenvector is almost identical to Algorithm 3 given in the previous chapter, except that $x^{(1)}$ is chosen randomly.

Algorithm 4: The Improvement Algorithm Using Factors from the Eigenvalue Decomposition, with NO Eigenvector Approximation

Given A and λ, Q and T such that $A = QTQ^T$;

$p \leftarrow 0$;

$\lambda^{(p)} \leftarrow \lambda$;

choose $x^{(1)}$ randomly;

Do until convergence

$p \leftarrow p+1$;

normalize $x^{(p)}$ such that $\|x^{(p)}\|_\infty = 1$ and $x_s^{(p)} = 1$;

1: $r^{(p)} \leftarrow \lambda^{(p)}x^{(p)} - Ax^{(p)}$; (extended precision used)

$T_\lambda^{(p)} \leftarrow T - \lambda^{(p)}I$;

$c \leftarrow -x^{(p)} - a_{\lambda s}$; ($a_{\lambda s}$ is s column of $A-\lambda^{(p)}I$)

2: $d \leftarrow Q^T c$;

$f \leftarrow Q^T e_s$; (e_s is s column of I)

compute $Q_1^{(p)}$ such that $Q_1^{(p)} d = \gamma e_1$;

compute $Q_2^{(p)}$ such that $Q_2^{(p)}(Q_1^{(p)}(T_\lambda^{(p)} + df^T))$

is triangular;

3: $\overline{T}_\lambda \leftarrow Q_2^{(p)}Q_1^{(p)}(T_\lambda^{(p)} + df^T)$;

4: $z \leftarrow Q_2^{(p)}Q_1^{(p)}Q^T r^{(p)}$;

5: $\overline{T}_\lambda^{(p)}\delta^{(p+1)} = z$; (solve for $\delta^{(p)}$)

6: $\overline{\delta}^{(p)} \leftarrow QQ_1^{(p)T}Q_2^{(p)T}\delta^{(p)}$;

$\lambda^{(p+1)} \leftarrow \lambda^{(p)} + \overline{\delta}^{(p)}$;

$x^{(p+1)} \leftarrow x^{(p)} + \overset{\sim}{\overline{\delta}}^{s(p)}$;

test for convergence

iterate end;

Operation Count:

1:	$p(n^2)$	4:	$p(n^2)$
2:	$p(n^2)$	5:	$p(1/2\ n^2)$
3:	$p(n^2)$	6:	$p(n^2)$

Total Count: $11/2\ pn^2$

The basic difference between Algorithm 3 and Algorithm 4 is that Algorithm 4 chooses as the initial approximate eigenvector some random vector. This vector is then corrected and renormalization is performed within the iteration. Whereas in Algorithm 3, the approximate eigenvector was accurate to working precision to start, and no renormalization is performed during the iteration.

When x is updated with the correction during the iteration, the largest component of x may change. This will cause the updated x to be renormalized and the value s to change.

We have found that Algorithm 4 works remarkably well for improving the initial eigenvalue and constructing the eigenvector given some initial random eigenvector. Typically Algorithm 4 takes three or four iterations to converge, even when the original eigenvector guess is completely orthogonal to the true eigenvector.

Convergence Results

We would like to show that various versions of the Improvement Method described in earlier chapters will converge. That is, we would like to analyze the conditions under which $\delta^{(p)} \to 0$, and hence $y^{(p)} \to y$. Wilkinson [24] proves one theorem on convergence for the improvement method. The classical Kantorovich theorem on Newton's method is also applicable. In this chapter we will show that the two approaches are equivalent.

We start by considering

$$By = r + y_s\tilde{y} \quad \text{or}$$
$$y = B^{-1}r + B^{-1}y_s\tilde{y} .$$

By a simple scaling of the program we can assume $\|B\|_\infty = 1$. Let $\varepsilon b = B^{-1}r$, where $\|b\|_\infty = 1$ and $\varepsilon = \|B^{-1}r\|_\infty$. Consider the iteration

$$y^{(p+1)} = \varepsilon b + B^{-1}y_s^{(p)}\tilde{y}^{(p)}$$

where the first few iterates are

$$y^{(0)} = 0$$
$$y^{(1)} = \varepsilon b$$
$$y^{(2)} = \varepsilon b + \varepsilon^2 B^{-1}b_s\tilde{b} .$$

We have that

$$|b_s| \leq \|b\|_\infty = 1$$
$$\|\tilde{b}\|_\infty \leq \|b\|_\infty = 1 ,$$

which implies

$$\|y^{(1)} - \varepsilon b\|_\infty = 0$$

$$\|y^{(2)} - \varepsilon b\|_\infty \leq \varepsilon^2 \kappa \quad \text{where} \quad \kappa = \|B^{-1}\|_\infty .$$

Since we have assumed $\|B\|_\infty = 1$, κ is the condition number of B. We will show that if $\varepsilon\kappa < 1/4$ then

$$\|y^{(p)} - \varepsilon b\|_\infty \leq \alpha\varepsilon^2\kappa$$

$$\|y^{(p)}\|_\infty \leq \varepsilon + \alpha\varepsilon^2\kappa ,$$

where α is the small root of $\alpha = (1 + \alpha\varepsilon\kappa)^2$ which is

$$\alpha = \frac{2}{(1 - 2\varepsilon\kappa) + (1 - 4\varepsilon\kappa)^{1/2}} .$$

If $\varepsilon\kappa < 1/4$, the α's are real and the positive sign gives the smaller α; clearly, $1 \leq \alpha < 4$. When $\varepsilon\kappa = 0$, $\alpha = 1$, and when $4\varepsilon\kappa = 1$, $\alpha < 4$. For small values of $\varepsilon\kappa$, $\alpha \approx 1 + 2\varepsilon\kappa$.

Lemma (Wilkinson): If $\varepsilon\kappa < 1/4$, then

$$\|y^{(p)} - \varepsilon b\|_\infty \leq \alpha\varepsilon^2\kappa \quad \text{and} \tag{15}$$

$$\|y^{(p)}\|_\infty \leq \varepsilon + \alpha\varepsilon^2\kappa .$$

Proof: The proof will be by induction on p. By definition

$$\|y^{(1)}-\varepsilon b\|_\infty = 0 \ .$$

So the inequality holds for p = 1.

Now assume (15) is true for p, then

$$y^{(p+1)} = \varepsilon b + B^{-1} y_s^{(p)} \tilde{y}^{(p)}$$

$$\|y^{(p+1)}-\varepsilon b\| \leq \|B^{-1}\|_\infty |y_s^{(p)}| \|\tilde{y}^{(p)}\|_\infty \ .$$

From the inductive hypothesis

$$\begin{aligned} \|y^{(p+1)}-\varepsilon b\| &\leq \kappa(\varepsilon + \alpha\varepsilon^2\kappa)^2 \\ &= \varepsilon^2\kappa(1 + \alpha\varepsilon\kappa)^2 \qquad \text{(by the definition of } \alpha \text{ and} \\ &\qquad\qquad\qquad\qquad \text{using the fact that } \varepsilon\kappa < 1/4) \\ &= \alpha\varepsilon^2\kappa \end{aligned}$$

which establishes the result. Hence it is true for all p. The second inequality

$$\|y^{(p)}\|_\infty \leq \varepsilon+\alpha\varepsilon^2\kappa$$

is a direct consequence of above because $\|b\| = 1$.

Q.E.D.

We will now show that if B is not too badly conditioned the corrections become smaller at each step.

Theorem (Wilkinson): If $\varepsilon\kappa < 1/4$, then $\delta^{(p)} \to 0$ and $y^{(p)} \to y$.

Proof: We start by considering two iterates,

$$y^{(p+1)} = \varepsilon b + B^{-1} y_s^{(p)} \tilde{y}^{(p)}$$
$$y^{(p)} = \varepsilon b + B^{-1} y_s^{(p-1)} \tilde{y}^{(p-1)} .$$

Writing $y^{(p+1)} - y^{(p)} = \delta^{(p)}$ we have

$$\delta^{(p)} = B^{-1} \left| y_s^{(p)} \tilde{y}^{(p)} - y_s^{(p-1)} \tilde{y}^{(p-1)} \right| .$$

Adding and subtracting $y_s^{(p-1)} \tilde{y}^{(p)}$, we have

$$\delta^{(p)} = B^{-1} \left| y_s^{(p)} \tilde{y}^{(p)} - y_s^{(p-1)} \tilde{y}^{(p)} + y_s^{(p-1)} \tilde{y}^{(p)} - y_s^{(p-1)} \tilde{y}^{(p-1)} \right| .$$

Combining terms, we get

$$\delta^{(p)} = B^{-1} \left| (y_s^{(p)} - y_s^{(p-1)}) \tilde{y}^{(p)} + y_s^{(p-1)} (\tilde{y}^{(p)} - \tilde{y}^{(p-1)}) \right|$$
$$\delta^{(p)} = B^{-1} \left| \delta_s^{(p-1)} \tilde{y}^{(p)} + y_s^{(p-1)} \tilde{\delta}^{(p-1)} \right| .$$

Hence

$$\begin{aligned} \|\delta^{(p)}\|_\infty &\le \kappa\left(\left[|\delta_s^{(p-1)}| \|\tilde{y}^{(p)}\|_\infty + |y_s^{(p-1)}| \|\tilde{\delta}^{(p-1)}\|_\infty \right] \right) \\ &\le \kappa\left(\left[\|\delta^{(p-1)}\|_\infty \|y^{(p)}\|_\infty + \|y^{(p-1)}\|_\infty \|\delta^{(p-1)}\|_\infty \right] \right) \\ &= \left(\kappa \|\delta^{(p-1)}\|_\infty \left[\|y^{(p)}\|_\infty + \|y^{(p-1)}\|_\infty \right] \right) \\ &\le \kappa \|\delta^{(p-1)}\|_\infty (2\varepsilon + 2\alpha\varepsilon^2\kappa) \\ &= 2\varepsilon\kappa \|\delta^{(p-1)}\|_\infty (1 + \alpha\varepsilon\kappa) . \end{aligned}$$

Using the fact that $\alpha = (1 + \alpha\varepsilon\kappa)^2$, then

$$\|\delta^{(p)}\| \leq 2\varepsilon\kappa\alpha^{1/2} \|\delta^{(p-1)}\|_\infty .$$

Here we define $\gamma \equiv 2\varepsilon\kappa\alpha^{1/2}$

$$\|\delta^{(p)}\|_\infty \leq \gamma \|\delta^{(p-1)}\|_\infty .$$

The assumption that $\varepsilon\kappa < 1/4$ implies $\gamma < 1$ so we have a contraction.

Q.E.D.

The analysis in the previous chapter, which showed the equivalence between Newton's Method and the improvement algorithm, together with the convergence results shows that the Improvement Method is a special case of Newton's Method. The Newton-Kantorovich theorem gives the general conditions for convergence of Newton's Method.

For Newton's Method, we would like to investigate the conditions under which a sequence $\{v_i\}$ will converge to a solution $v = v^*$ of $f(v) = 0$ for the eigenvalue problem.

We will now state the Kantorovich theorem on convergence of Newton's Method in a Banach space.

Theorem (Kantorovich)[18]:

If $\|f''(v)\| \leq k$, $\|v_1 - v_0\| \leq \eta_0$, $\|[f'(v_0)]^{-1}\| \leq \beta_0$ and $h_0 = \beta_0\eta_0 k \leq 1/2$ in some closed ball $\bar{u}(v_0, r)$ with $r \geq r_0 = \dfrac{1-\sqrt{1-2h_0\eta_0}}{h_0}$, then the Newton sequence $\{v_i\}$ starting from v_0 will converge to a solution v^* which exists in $\bar{u}(v_0, r)$.

The rate of convergence for Newton's Method can be stated as

$$\|v^* - v_i\| \leq \frac{1}{2^{i-1}} (2h_0)^{2^{i-1}} \eta_0 .$$

Note that if $h_0 < 1/2$, then we will get convergence and when we have convergence it is quadratic. Looking at the i^{th} iteration, we have

$$h_i = \beta_i \eta_i k ,$$

where k is some bound on the second derivative,

and
$$\|[f'(v_i)]^{-1}\| \leq \beta_i$$
$$\|v_{i+1} - v_i\| \leq \eta_i .$$

For the eigenvalue problem we have

$$f(v) = \begin{pmatrix} (A-\lambda I)x \\ e_s^T x - 1 \end{pmatrix}$$

$$f'(v) = \begin{pmatrix} (A-\lambda I) & -x \\ e_s^T & 0 \end{pmatrix}$$

where
$$f''(v) = \left(\begin{array}{cc|cc} 0 & -I & I & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) .$$

The term f''(v) is a bilinear operator and constant.

We can examine the quantities from the Kantorovich theorem;

$$\|f''(v)\| \leq k$$
$$\|[f'(v_0)]^{-1}\| \leq \beta_0$$
$$\|v_1 - v_0\| \leq \eta_0 \qquad \text{and}$$
$$h_0 = \beta_0 \eta_0 k .$$

For the eigenvalue problem k is constant and equal to 2, because f"(v) is a constant operator. We will have convergence when $\beta_0\eta_0 < 1/4$.

In the Newton-Kantorovich theorem the quantity $\beta_0\eta_0$ is closely related to $\varepsilon\kappa$ used in the proof of contraction for the Improvement Method. In Newton's Method we have

$$\|f'(v_0)^{-1}\| \leq \beta_0 .$$

For the eigenvalue problem

$$f'(v_0) = B, \quad \text{so}$$
$$\|f'(v_0)^{-1}\| = \|B^{-1}\| \leq \beta_0 .$$

The value η_0 is related to the quantity v_1-v_0 by

$$\|v_0-v_1\| \leq \eta_0 .$$

The quantity v_0-v_1 corresponds to $y^{(1)}$ from the Improvement Method. Using the value for $y^{(1)}$ from the Improvement Method, we have

$$v_0-v_1 = y^{(1)} = \varepsilon b = B^{-1}r , \quad \text{or}$$
$$\|v_0-v_1\| = \|B^{-1}r\| \leq \eta_0 .$$

So that $\|B^{-1}r\|\|B^{-1}\|$ from Newton's Method are bounded by

$$\|B^{-1}r\|\|B^{-1}\| \leq \beta_0\eta_0 ,$$

and for convergence of Newton's Method we require $\beta_0\eta_0 < 1/4$. From the Improvement Method, we have

$$\varepsilon = \|B^{-1}r\| \quad \text{and} \quad \kappa = \|B\|\|B^{-1}\| , \qquad \|B\| = 1 .$$

So $$\varepsilon\kappa = \|B^{-1}r\|\|B^{-1}\| ,$$

or $$\|B^{-1}r\|\|B^{-1}\| = \varepsilon\kappa ,$$

and for convergence we require

$$\varepsilon\kappa < 1/4 .$$

In conclusion, we have shown that the convergence conditions for the Improvement Method and those of the Newton-Kantorovich theorem are the same.

Error Analysis

We will examine the nature of the roundoff errors for the improvement algorithm which updates λ and x every iteration; this corresponds to Algorithm 4. The method involves four steps:

1) Calculate the residual $r^{(p)} = \lambda^{(p)}x^{(p)} - Ax^{(p)}$.

2) Let $B^{(p)}$ be $A-\lambda^{(p)}I$ with the s column replaced by $-x^{(p)}$.

3) Solve the system $B^{(p)}y^{(p+1)} = r^{(p)}$.

4) Update the approximation $x^{(p+1)} = x^{(p)} + \tilde{y}^{(p+1)}$.

We will assume that calculations will be performed in working precision, using normalized floating point arithmetic unless otherwise specified. By working precision we mean arithmetic is carried out with t_1 digits of base β. We will also use the term extended precision to describe arithmetic that is carried out with t_2 digits of base β, where $t_2 > t_1$.

The residual calculation is a critical part of the improvement process. If λ and x have been produced by a stable algorithm, such as the QR algorithm using working precision, then the true residual, $\bar{r} = \lambda x - Ax$ satisfies

$$\|\bar{r}\| \leq \beta^{-t_1}\|A\|g(n) \quad \text{where } g(n) \text{ is a modest function of } n .$$

If we now compute the residual in working precision, we will produce roundoff errors of the order β^{-t_1}, which is the same order of magnitude as the true residual. Consequently, unless some precaution is taken,

the computed residual may differ quite substantially from the true residual.

To demonstrate the need for carrying out the residual calculations in extended precision we will show an example.

If we consider the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix},$$

then using single precision as our working precision one of the eigenvalues is

$$\lambda = -.623475976$$

and its corresponding eigenvector is given by

$$x = \begin{pmatrix} -.827670515 \\ -.142414615 \\ .542843938 \end{pmatrix} .$$

If the residual calculation, $(A-\lambda I)x$, is carried out totally in working precision then the residual r_s is

$$r_s = \begin{pmatrix} -.0536 \\ -.1192 \\ .0060 \end{pmatrix} * 10^{-5} .$$

If the subtraction, $A-\lambda I$, is performed in working precision and the resulting matrix multiplied by x in extended precision the result is

$$r_{sd} = \begin{pmatrix} -.0557 \\ -.1186 \\ -.0389 \end{pmatrix} * 10^{-5} .$$

If the operations Ax and λx are carried out in extended precision and

the subtraction $Ax-\lambda x$ performed in working precision, the result is

$$r_{ds} = \begin{pmatrix} -.0596 \\ -.1207 \\ -.0179 \end{pmatrix} * 10^{-5} .$$

Finally, if all operations are carried out in extended precision the result is

$$r_{d} = \begin{pmatrix} -.0606 \\ -.1211 \\ -.0163 \end{pmatrix} * 10^{-5} .$$

If Ax is subtracted in working precision from λx, regardless of the precision of each quantity, errors may be introduced of order β^{-t_1}. The same is true in the term $(A-\lambda I)x$, where $A-\lambda I$ is subtracted in working precision. This error, β^{-t_1}, is precisely the order of magnitude of the true residual. Thus, the computed residual can be arbitrarily different from the true residual. In the example, we see that r_s is completely different than r_d, even to the point of having a different sign.

If, on the other hand, the subtraction of Ax from λx is performed in extended precision the computed residual will differ from the true residual by order of magnitude β^{-t_2}.

This example also points out the need to carry out the residual calculation with the original matrix. If the matrices produced from the factorization were used in calculating the residual there would be an additional source of error from roundoff generated during the decomposition. Therefore, we see the need to carry out the entire computation in extended precision using A, λ and x. This can be done as an $n+1$ inner product, with the first term λx and the remaining n terms $-Ax$, using extended precision accumulation. The form of the expression is

$$r_i = \lambda x_i - \sum_{j=1}^{n} a_{ij} x_j .$$

Subroutine DQQDOT, found in the appendix, carries out this operation using extended double precision accumulation for the double precision quantities λ, x and A.

We can express the improvement algorithm by

$$\begin{pmatrix} A-\lambda^{(p)} & -x^{(p)} \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} \tilde{y}^{(p)} \\ \mu^{(p)} \end{pmatrix} = \begin{pmatrix} \lambda^{(p)} x^{(p)} - Ax^{(p)} \\ 0 \end{pmatrix}$$

where $x^{(p+1)} = x^{(p)} + \tilde{y}^{(p)}$ and $\lambda^{(p+1)} = \lambda^{(p)} + \mu^{(p)}$. If we define v and f as in chapter 6 we have

$$v_r = \begin{pmatrix} x^{(r)} \\ \lambda^{(r)} \end{pmatrix} \quad \text{and} \quad f(v_r) = \begin{pmatrix} \lambda^{(r)} x^{(r)} - Ax^{(r)} \\ 0 \end{pmatrix} ,$$

then we can write $f'(v_r)\ (v_{r+1}-v_r) = f(v_r)$. Let $\bar{v}_r$ denote the r^{th} iterate computed in the absence of floating point roundoff errors, let v_r denote the r^{th} iterate actually computed and v^* denote a solution to $f(v) = 0$.

We want to know how v_r compares with $\bar{v}_r$ for every r. In particular if $\bar{v}_r \to v^*$ as $r \to \infty$, what can be said about v_r as $r \to \infty$.

In the absence of roundoff errors, the algorithm is equivalent to Newton's method applied to the eigenvalue problem. We can therefore apply results about the error analysis for Newton's method [12] to our improvement method and obtain the desired error estimates.

The Newton-Kantorovich theorem tells us that under certain conditions the iterates, $\bar{v}_r$, lie in a neighborhood of $\bar{v}_0$. The neighborhood is

$$K_0 = \{\bar{v} \in V \mid \|\bar{v}-\bar{v}_0\| \leq \bar{\alpha}_0\}$$

where $$\bar{\alpha}_r = \|\bar{v}_{r+1}-\bar{v}_r\|$$
$$= \|(f'(\bar{v}_r))^{-1} f(\bar{v}_r)\| \ .$$

When we compute the iterates v_r we can expect them to deviate from $\bar{v}_r$ to some degree; in particular, they may not belong to K_0. We need to expand the neighborhood K_0 to $K_\delta = \{v \in V \mid \|v-v_0\| \leq \bar{\alpha}_0 + \delta\}$ for some $\delta > 0$.

The computation of v_{r+1} involves an equation of the form

$$\bar{v}_{r+1}-\bar{v}_r = \left(f'(\bar{v}_r)\right)^{-1} f(\bar{v}_r) \ .$$

In the presence of floating point arithmetic we will actually compute

$$v_{r+1}-v_r = \left(f'(v_r)+E_r\right)^{-1}\left(f(v_r)+e_r\right) + g_r \ . \qquad (16)$$

There are three sources of errors: e_r, E_r and g_r. The error e_r comes from errors made during construction of the right hand side, which is the residual calculation for the eigenvalue problem and is carried out in extended precision. The error E_r comes from solving the system of equations. The error g_r is derived from adding the correction to the

previously computed $x^{(r)}$ and $\lambda^{(r)}$. We will assume that there exist ε_1, ε_2 and ε_3 such that for every $v_r \in K_\delta$,

$$\|e_r\| \leq \varepsilon_1$$
$$\|E_r\| \leq \varepsilon_2$$
$$\|g_r\| \leq \varepsilon_3 .$$

The analysis also involves

$$\overline{\kappa}_\delta = \sup_{v \in K_\delta} \|(f'(v))^{-1}\| .$$

Note that if $f'(v_r)$ has been scaled to have a norm of order unity, then $\overline{\kappa}_\delta$ bounds the condition number of the matrix B_r involved in the linear system of equations solved at the r^{th} step.

Our objective is to show that the roundoff errors of size ε_2 introduced in the linear system solution may affect the rate of convergence of the algorithm, but do not significantly affect the accuracy of the final result. The other roundoff errors, which involve $\kappa\varepsilon_1$ and ε_3, do affect the final accuracy.

The following theorem is a restatement of Lancaster's Theorems 2A and 4A [12].

Theorem: Assume there exists a $\delta > 0$ and an iteration index n so that

a) $\overline{\beta} = 2\overline{\kappa}_\delta (\overline{\alpha}_n + \varepsilon_2) < 1 .$

Let $\kappa = \dfrac{\overline{\kappa}_\delta}{1-\overline{\beta}}$ and

$$\beta = \kappa\,(\varepsilon_1 + \overline{\alpha}_n\varepsilon_2) + \frac{1}{1-\beta}\,\varepsilon_3 \ .$$

Assume

b) $2\beta < \delta$

c) $4\kappa\beta < 1$.

Then the computed iterates v_r remain in the neighborhood K_δ for all $r > n$ and

$$\|v_r - \overline{v}_r\| \leq \frac{2\beta}{1 + \sqrt{1-4\kappa\beta}} \ .$$

Moreover, let

$$\kappa_n = \frac{2\|(f'(v_{n-1}))^{-1}\|}{1 - 2\alpha_{n-1}\|(f'(v_{n-1}))^{-1}\|} \ .$$

Then

$$\|v_r - v^*\| \leq \beta\left[1 + 4\kappa_n(\beta + \|v_n - v_{n-1}\|)\right] + \kappa_n\|v_n - v_{n-1}\|^2 \ .$$

The hypotheses involves three inequalities. Inequalities a) and c) are precise versions of the statement "The matrix B is not too badly conditioned." Inequality b) relates to the expansion of the Newton-Kantorovich neighborhood required to include effects of roundoff. For all but the most badly conditioned problems we expect these inequalities

to be satisfied by a margin wide enough to justify the following estimates.

$$1 - \overline{\beta} \approx 1$$
$$\kappa \approx \overline{\kappa}_{\delta}$$
$$1 + \sqrt{1-4\kappa\beta} \approx 2 \quad .$$

Moreover, when the process converges, $\overline{\alpha}_n \to 0$. Consequently we can expect to ultimately obtain accuracies of the order

$$\|v_r - v_*\| \lesssim \beta \approx \kappa\varepsilon_1 + \varepsilon_3 \quad .$$

Note that ε_2 is involved in the hypotheses, but does not affect the ultimate accuracy in an essential way. If we use extended precision for the residual calculation then

$$\varepsilon_1 = \varepsilon_2^2 \quad \text{and} \quad \varepsilon_2 = \varepsilon_3 \quad .$$

Hypothesis a) implies $\kappa\varepsilon_2 < 1$, hence $\kappa\varepsilon_1 < \varepsilon_2$ and the limiting accuracy is order ε_3. In other words, if the problem is not pathologically badly conditioned, it converges to working precision accuracy.

The error ε_3 is derived from the roundoff errors made in adding the correction to the approximate solution. If on the last iteration the correction is added in extended precision then ε_3 will be the same size as ε_1 and the final accuracy will be controlled by $\kappa\varepsilon_1$ solely.

This points out that when the matrix B is ill conditioned with respect to linear systems the accuracy attainable by the method will degrade, since $\kappa\varepsilon_1$ controls the accuracy. The condition under which B

is singular or ill conditioned can be described in the following theorem.

Theorem (Wilkinson): B is singular iff λ is a multiple eigenvalue of A.

Proof: If λ and x are the exact eigenpair and λ is a multiple eigenvalue of A, then there exists at least one left eigenvector y belonging to λ which is orthogonal to x. (This is true whatever the nature of the elementary divisors associated with λ.) Hence

$$y^T(A-\lambda I) = 0 \quad \text{and} \quad y^T x = 0 \ .$$

We have

$$B = A - \lambda I + ce_s^T \quad \text{with} \quad c = -a_{\lambda s} - x \ .$$

When y^T is multiplied into B, it follows that

$$y^T B = y^T(A-\lambda I+ce_s^T) = 0 \ .$$

So y is a null vector of B, and B is singular.

Conversely, if B is singular then there exists a $z \neq 0$ such that

$$Bz = 0 \quad \text{or}$$
$$B(\tilde{z}+z_s e_s) = 0$$

and hence

$$B\tilde{z} = z_s x \quad \text{because} \quad Be_s = -x \ .$$

But $B\tilde{z} = (A-\lambda I)\tilde{z}$ since $\tilde{z}_s = 0$ (by definition of the tilde notation).

Hence,

$$(A-\lambda I)\tilde{z} = z_s x \ .$$

If $z_s = 0$, then $\tilde{z}$ is therefore an eigenvector of A corresponding to λ. Because $x_s = 1$ and $\tilde{z}_s = 0$, the vectors x and $\tilde{z}$ are independent. If $z_s \neq 0$, then because $(A-\lambda I)\tilde{z} = z_s x$, x is an eigenvector and $\tilde{z} \neq 0$. Thus we have

$$(A-\lambda I)^2\tilde{z} = 0 \ .$$

Hence $\tilde{z}$ is a principal vector of grade 2. In either case λ is an eigenvalue of multiplicity at least two.

Q.E.D.

We can make a somewhat stronger statement on how the conditioning of B is related to the sensitivity of the eigenvalues.

Theorem (Wilkinson): Let y and x be the left and right eigenvectors of A corresponding to the eigenvalue λ. Let $B = A-\lambda I+ce_s^T$ where $c = -a_{\lambda s}-\epsilon$. Then

$$\text{cond}(B) \geq \frac{\|y\|\|x\|}{|y^T x|} \ .$$

Proof: Consider

$$y^T B = y^T(A-\lambda I+ce_s^T) \ ,$$

we know that y^T is orthogonal to the column of $A-\lambda I$ so that

$$y^T B = (0,\ldots,0,-y^T x,0,\ldots,0) \ .$$

We have that $\|y^T B\| = |y^T x|$, from above we see

$$\|B^{-1}\| \geq \frac{\|y\|}{|y^T x|} \ .$$

Since x is a column of B, $\|B\| \geq \|x\|$ and we have

$$\text{cond}(B) \geq \frac{\|y\|\|x\|}{|y^T x|} \ .$$

Q.E.D.

We expect the method to have trouble when the eigenvalues are sensitive, corresponding to an ill conditioned eigenvalue problem. The method as stated has a defect in that any type of multiplicity in the eigenvalues gives rise to ill conditioning in the matrix B. In a paper by Wilkinson [26] the case where multiple eigenvalues exist is handled and an algorithm is given to compute the invariant subspace associated with multiple eigenvalues.

Complex Eigenvalues for Real Matrices

Where there is a real matrix and complex eigenvalues, we have a computed eigenvalue $\lambda = \lambda_r \pm \lambda_i i$ and a computed eigenvector $x = x_r \pm x_i i$. If we use the strategy developed earlier for the complex case, we run into a problem. The original formulation replaced a column of $A-\lambda I$ with $-x$, the eigenvector. While we could do this by using complex arithmetic and doubling the storage, that procedure is not very attractive. We would like to maintain real arithmetic and conserve storage as much as possible.

Consider the following:

$$A(x_r,x_i) = (x_r,x_i)\begin{pmatrix} \lambda_r & \lambda_i \\ -\lambda_i & \lambda_r \end{pmatrix} .$$

Here, as before, (λ_r,λ_i) and (x_r,x_i) have been arrived at through some computation, and we wish to improve their accuracy. As before, we write

$$A\lfloor (x_r,x_i)+(y_r,y_i)\rfloor = \lfloor (x_r,x_i)+(y_r,y_r)\rfloor \begin{pmatrix} \lambda_r & \lambda_i \\ -\lambda_i & \lambda_r \end{pmatrix} + \begin{pmatrix} \mu_r & \mu_i \\ -\mu_i & \mu_r \end{pmatrix}$$

or $AY - Y\Lambda - XM = X\Lambda - AX + YM$ where $Y = (y_r,y_i)$

$X = (x_r,x_i)$

$$\Lambda = \begin{pmatrix} \lambda_r & \lambda_i \\ -\lambda_i & \lambda_r \end{pmatrix}$$

and $$M = \begin{pmatrix} \mu_r & \mu_i \\ -\mu_i & \mu_r \end{pmatrix}$$

which is

$$A(y_r,y_i)-(y_r,y_i)\begin{pmatrix}\lambda_r & \lambda_i\\ -\lambda_i & \lambda_r\end{pmatrix} - (x_r,x_i)\begin{pmatrix}\mu_r & \mu_i\\ -\mu_i & \mu_r\end{pmatrix} =$$

$$(x_r,x_i)\begin{pmatrix}\lambda_r & \lambda_i\\ -\lambda_i & \lambda_r\end{pmatrix} - A(x_r,x_i) + (y_r,y_i)\begin{pmatrix}\mu_r & \mu_i\\ -\mu_i & \mu_r\end{pmatrix} . \quad (17)$$

We will now rewrite this in a simple matrix notation as

$$\begin{pmatrix} A-\lambda_r I & \lambda_i I & -x_r & x_i \\ -\lambda_i I & A-\lambda_r I & -x_i & -x_r \\ e_s^T & 0\cdots 0 & 0 & 0 \\ 0\cdots 0 & e_s^T & 0 & 0 \end{pmatrix} \begin{pmatrix} y_r \\ y_i \\ \mu_r \\ \mu_i \end{pmatrix} = \begin{pmatrix} \text{r.h.s. 1} \\ \text{r.h.s. 2} \\ 0 \\ 0 \end{pmatrix} , \quad (18)$$

where r.h.s. 1 and r.h.s. 2 are the right hand side of equation (17) when expanded. We now have a $(2n+2)\times(2n+2)$ system of equations to solve. With a little care, this system can be solved without actually storing all the $(2n+2)\times(2n+2)$ elements and without the additional cost in storage brought on by a matrix of order $2n+2$.

We want to solve a system of equations based on (18), but do not explicitly want to form such a matrix because of the additional storage and computational requirements. Instead we will develop an approach that implicitly solves such a system and requires only the original quantities to be used. We will pre- and post-multiply the matrix in (18) by

$$\begin{pmatrix} Q^T & & & \\ & Q^T & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \text{ and its transpose.} \qquad (19)$$

The submatrix Q is the orthogonal matrix from the original eigenvalue decomposition of the matrix A, which reduced A to a quasi-triangular matrix. A quasi-triangular matrix is triangular except for some (2×2) blocks on the diagonal. These blocks correspond to complex conjugate eigenvalue pairs for the real matrix A.

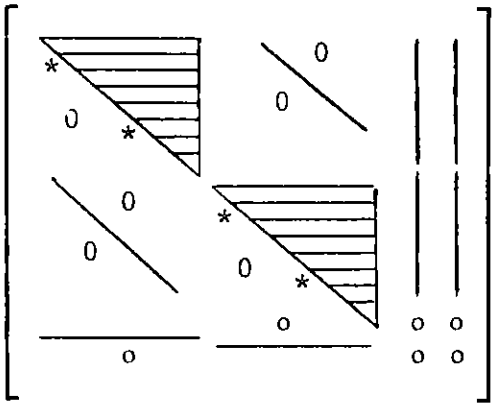
When (19) is applied to the matrix, the result is

$$\begin{pmatrix} Q^T(A-\lambda_r I)Q & \lambda_i I & -Q^T x_r & Q^T x_i \\ -\lambda_i I & Q^T(A-\lambda_r I)Q & -Q^T x_i & -Q^T x_r \\ e_s^T Q & 0\ldots 0 & 0 & 0 \\ 0\cdots 0 & e_s^T Q & 0 & 0 \end{pmatrix} .$$

Because Q^TAQ is the quasi-triangular matrix formed during the initial eigenvalue decomposition, we have

$$\begin{pmatrix} T-\lambda_r I & \lambda_i I & -w & v \\ -\lambda_i I & T-\lambda_r I & -v & -w \\ q_s^T & 0\cdots 0 & 0 & 0 \\ 0\cdots 0 & q_s^T & 0 & 0 \end{pmatrix} , \qquad (20)$$

where T is quasi-triangular, $q_s^T = e_s^T Q$, $w = Q^T x_r$, and $v = Q^T x_i$. This matrix is of the form



The * in the off diagonal positions of the triangular matrix represent elements arising from complex conjugate pairs of eigenvalues.

If the last two rows of the matrix are replaced with e_{2n+1}^T and e_{2n+2}^T, we can easily solve systems of equations based on this modified matrix. This is best seen by looking at an example. Let us assume we have an order 10 matrix, C, which originated from an eigenvalue problem of order 4 having one complex conjugate pair located in a 2×2 block in position 2,2 of the original matrix, so that

$$C = \begin{bmatrix} x & x & x & x & x & & & & x & x \\ & x & x & x & & x & & & x & x \\ & * & x & x & & & x & & x & x \\ & & & x & & & & x & x & x \\ x & & & & x & x & x & x & x & x \\ & x & & & & x & x & x & x & x \\ & & x & & & * & x & x & x & x \\ & & & x & & & & x & x & x \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{bmatrix} .$$

This is the structure that is obtained after removing the 9^{th} and 10^{th} rows and replacing them with e_9^T and e_{10}^T.

The matrix C, if formed, will unfortunately be singular. This feature is best seen by looking at the complex counterpart of C, which

has the form

$$\overline{C} = \begin{pmatrix} \overline{T}-\lambda I & -x \\ 0 & 1 \end{pmatrix} ,$$

where $\overline{T}$ is a complex upper triangular matrix and λ and x represent the complex eigenpair. $\overline{C}$ is singular because it is upper triangular and has a zero on the diagonal (namely, where $\lambda = \overline{T}_{ii}$) and has a null vector of the form $\binom{x}{0}$, where x is the eigenvector. If we assume λ is a simple eigenvalue, then $\overline{C}$ can be modified to remove the singularity and allow the upper triangular matrix to be used in solving a system of equations. The modification will be a rank one change of the form $e_i e_i^T$, where i is the position on the diagonal of T where λ occurs. Then $\overline{C}$ is upper triangular and does not have a zero on the diagonal and therefore is not singular. The chapter on eigenvalue clusters describes the situation when λ is not a simple eigenvalue and when $\overline{C}$ is near singular.

If we now go back to the real matrix form of $\overline{C}$, the $(2n+2)\times(2n+2)$ matrix C, it will be necessary to make two rank one corrections in C. These corrections will place 1 and -1 in the blocks containing $\lambda_i I$ and $-\lambda_i I$ and will occur on the subdiagonal position corresponding to the 2×2 block of the conjugate pair being corrected for the triangular matrix $T-\lambda_r I$. Thus, if the 2×2 block has its nonzero on the subdiagonal at position (i,i-1) and (i+n, i-1+n) the two rank one changes are of the form $-e_i e_{i-1+n}^T$ and $e_{i+n} e_{i-1}^T$. The matrix will then be of the form

$$C = \begin{bmatrix} x & x & x & x & x & & & & x & x \\ & x & x & x & & x & & & x & x \\ & * & x & x & & 1 & x & & x & x \\ & & & x & & & & x & x & x \\ x & & & & x & x & x & x & x & x \\ & x & & & & x & x & x & x & x \\ & -1 & x & & & * & x & x & x & x \\ & & & x & & & & x & x & x \\ & & & & & & & & 1 & 0 \\ & & & & & & & & 0 & 1 \end{bmatrix}$$

This form of the matrix will be used to solve a system.

The first step in solving a system based on C is to determine values for unknowns x_9 and x_{10}. This is straightforward. The next step is to use information in rows 4 and 8 to determine values for unknowns x_4 and x_8. This step is equivalent to solving a 2×2 system. We then proceed to rows 3 and 7. They contain information from the complex conjugate eigenpair. If we expand the scope to look at rows 2, 3, 6 and 7 we can then determine unknowns x_2, x_3, x_6 and x_7 by solving a 4×4 system. Finally, using rows 1 and 5 determine x_1 and x_5.

Another way to view this process is to carry out row and column permutations on the matrix to isolate 2×2 or 4×4 blocks on the diagonal. Again following the example above for a 10×10 case we have,

$$C = \begin{bmatrix} x & x & x & x & x & & & & x & x \\ & x & x & x & & x & & & x & x \\ & + & x & x & & 1 & x & & x & x \\ & & & x & & & & x & x & x \\ x & & & & x & x & x & x & x & x \\ & x & & & & x & x & x & x & x \\ & -1 & x & & & + & x & x & x & x \\ & & & x & & & & x & x & x \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{bmatrix} .$$

Permutation will be performed on rows and columns 4 and 7 to isolate a 2×2 block. After the permutations the matrix has the form:

$$\begin{bmatrix}
x & x & x & & x & & x & & x & x \\
 & x & x & & & x & x & & x & x \\
 & + & x & x & & 1 & x & & x & x \\
 & -1 & x & x & & + & & x & x & x \\
x & & & x & x & x & & x & x & x \\
 & x & & x & & x & & x & x & x \\
 & & & & & & x & x & x & x \\
 & & & & & & x & x & x & x \\
 & & & & & & & & 1 & \\
 & & & & & & & & & 1
\end{bmatrix}$$

A 2×2 block is now isolated in rows and columns 7 and 8. A second set of permutations will be applied to isolate a 4×4 block. The permutations will interchange rows and columns 2 and 5. After the permutations the matrix has the form:

$$\begin{bmatrix}
x & x & x & & x & & x & & x & x \\
x & x & & x & & x & & x & x & x \\
 & & x & x & + & 1 & x & & x & x \\
 & & x & x & -1 & + & & x & x & x \\
 & & x & & x & x & x & & x & x \\
 & & & x & x & x & & x & x & x \\
 & & & & & & x & x & x & x \\
 & & & & & & x & x & x & x \\
 & & & & & & & & 1 & \\
 & & & & & & & & & 1
\end{bmatrix}$$

The permutations have isolated 2×2 and 4×4 blocks on the diagonal.

Solving a system of this form reduces to 2×2 or 4×4 systems of equations, where the 4×4 systems arise from complex conjugate eigenpairs. The amount of work needed to solve a system based on a matrix of this structure is $O(n^2)$.

To get matrix (20) into the correct form, we will use the generalized form of the Sherman-Morrison-Woodbury update [10], which is

$$A^{-1} = (C-UV^T)^{-1} = C^{-1} + C^{-1}U(I-M)^{-1}V^TC^{-1}$$
$$\text{where} \quad M = V^TC^{-1}U \ .$$

The matrices U and V are of dimension 2n+2 by 4 in this case and are of the form

$$U = \left[-e_{2n+1},\ -e_{2n+2},\ -e_i,\ -e_{i+n}\right]$$

$$V = \begin{pmatrix} Q^T e_s & \overline{0} \\ \overline{0} & Q^T e_s \\ -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad -e_{i-1+n},\ e_{i-1}$$

where $(i,i-1)$ is the position of the nonzero element in the 2×2 block.

To solve systems we need to apply the update to a right hand side, so

$$C^{-1}b + C^{-1}U(I-M)^{-1}V^TC^{-1}b\ .$$

Determining $C^{-1}b$ and $C^{-1}U$ is just a matter of applying what was outlined above to the vector b and matrix U. I-M is a 4×4 system and can easily be used to solve systems.

The fact that the eigenvalues were complex conjugate is really quite incidental. The method can be easily extended to cover the case where we wish to improve several distinct eigenvalues simultaneously.

Multiple Eigenvalues

Up to this point, the discussion of improving eigenvalues has dealt with the case of a simple eigenvalue eigenvector pair or a group of distinct eigenvalues and eigenvectors. We will now consider the problem where there are multiple eigenvalues.

Multiple eigenvalues can occur with linear or nonlinear elementary divisors. Either situation is relatively easy to detect, but a nonlinear elementary divisor, by its nature, is ill-conditioned and very sensitive to small perturbations.

In the case where multiple eigenvalues exist, the algorithm described appears to break down. We are interested in solving

$$B\delta^{(p)} = y_s^{(p)}\tilde{y}^{(p)} - y_s^{(p-1)}\tilde{y}^{(p-1)} ,$$

where B is formed by taking $A-\lambda I$ and replacing a column by $-x$. The theorem from chapter 8 states the matrix B is singular if and only if λ is a multiple eigenvalue of A. Because we wish to solve systems of equations based on B, and B is ill-conditioned with respect to linear systems for the case being considered, we appear to have a problem. It is fairly easy to detect when this condition arises. The technique described by Cline, Moler, Stewart and Wilkinson [3] and implemented in LINPACK [5] can effectively be used to signal ill-conditioning in B, and some action can be taken. This will be discussed in more detail in chapter 13, which deals with implementation.

We would like to analyze the case where there are multiple roots corresponding to linear elementary divisors. This is a well conditioned eigenproblem, and therefore we would like to improve the accuracy without any additional complication. The theorem states that B is ill-conditioned with respect to linear systems in this case, but we will look at the consequences of solving the special system we have in mind.

We start out by assuming $\|A-\lambda I\| = O(1)$ and $\|B\| = O(1)$. If the system $Bz = c$ is solved, then the computed solution is exact for $(B+E)z = c$ where $\|E\| = O(\beta^{-t})$. Thus we can write

$$c-Bz = Ez$$

or $$\|c-Bz\| \leq \|E\|\|z\| = O(\beta^{-t})\|z\| \ .$$

The residual is related to the size of z, the computed solution. We wish to solve

$$By = -r \quad \text{where} \quad r = (\lambda I-A)x \ ,$$

and we want to show that y is $O(\|r\|)$ and not contaminated with B^{-1}. B is close to being singular when there are multiple eigenvalues. We will say that the computed y is of the same order of magnitude as the exact y unless B is singular to working precision. This can be seen by looking at

$$\|B^{-1}\| = \beta^{t-k} \ .$$

The computed y will have about k correct digits. Even one correct digit

implies that the computed and the exact y have the same order of magnitude.

We begin the analysis by looking at a closely related method, inverse iteration. For inverse iteration we have

$$(A-\lambda I)w = x_i \ ,$$

with $x_{i+1} = \frac{w}{w_s}$ being the next iterate. We also note that $\frac{w}{w_s} = x_i + \tilde{y}$ with $\delta\lambda = \frac{1}{w_s}$, where $\tilde{y}$ is the correction to x_i at this step and $\delta\lambda$ is the correction to λ. Now let us assume that we have a double eigenvalue corresponding to a linear elementary divisor with $\lambda_1 = \lambda_2$ and $\lambda = \lambda_1 - \varepsilon$, i.e., that λ is the true eigenvalue which differs from λ_1, the computed eigenvalue by ε, where ε is small in a relative sense. Dropping subscripts from x, we can expand x in the true eigenvectors of A so that

$$x = av_1 + bv_2 + \sum_{i=3}^{n} \varepsilon_i v_i \ , \qquad \text{where } \{v_i\} \text{ is a complete system of eigenvectors.}$$

The next iterate, w, can also be expanded in a similar fashion so that

$$w = \frac{a}{\lambda-\lambda_1} v_1 + \frac{b}{\lambda-\lambda_1} v_2 + \sum_{i=3}^{n} \frac{\varepsilon_i v_i}{\lambda_i-\lambda_1+\varepsilon} \ .$$

We know that $\lambda_1 = \lambda_2$ and that $\lambda = \lambda_1 - \varepsilon$, so

$$w = \frac{a}{\varepsilon} v_1 + \frac{b}{\varepsilon} v_2 + \sum_{i=3}^{n} \frac{\varepsilon_i}{\lambda_i-\lambda_1+\varepsilon} v_i \ .$$

Here ε and ε_i are small, and a and b are order unity. We see that $\|w\| = O(1/\varepsilon)$ with $1/w_s = O(\varepsilon)$. If we consider $w/w_s - x$, the difference in the

two iterates, we see that it has components of the same order of magnitude as ε.

Theorem (Wilkinson): If B is not exactly singular, then By = r has a solution of size $O(\|r\|)$.

Proof: We know from above that $w/w_s = x+\tilde{y}$.

This implies that the exact solution, $\bar{y}$, of By = r is the order of magnitude of the ε's.

The computed y satisfies $(B+E)y = r$ or

$$By - r = -Ey \ .$$

Going back to the definition of B and r we have

$$B = [\tilde{A}-\lambda\tilde{I} \ : \ -x] \quad \text{and} \quad r = \lambda x - Ax \ , \quad \text{so that}$$
$$By = A\tilde{y} - \lambda\tilde{y} - y_s x \quad \text{and}$$
$$By - r = A\tilde{y} - \lambda\tilde{y} - y_s x + \lambda x - Ax = -Ey \ .$$

Then by adding $-y_s\tilde{y}$ to both sides we get

$$Ax - \lambda x - y_s x + A\tilde{y} - \lambda\tilde{y} - y_s\tilde{y} = -Ey - y_s\tilde{y} \quad \text{or}$$
$$(A-(\lambda+y_s)I)(x+\tilde{y}) = -Ey - y_s\tilde{y} \ .$$

We know that $\|E\| = O(\beta^{-t})$ and $y = O(\varepsilon)$, so

$$\|Ey+y_s\tilde{y}\| = O(\beta^{-t}e+e^2) \quad \text{where} \quad e = \max(\varepsilon,\varepsilon_i) \ .$$

Hence the residual corresponding to the improved eigenpair is far smaller than r. When A has a full set of eigenvector this means $\lambda+y_s$ is a much improved eigenvalue of A.

Q.E.D.

As far as $x+\tilde{y}$ is concerned we can say only that it is a much improved vector in 2 space. However, we have shown that $\|y\| = O(e)$, so that x+y cannot change too much in the subspace.

For the case corresponding to nonlinear elementary divisors or ill conditioned eigenvalues, things are not as simple. Because of their nature, these problems may be considered not well posed. By this we mean that the matrix has a poorly conditioned eigensystem in the sense that small perturbations in the matrix can cause large changes in the eigensystem. In such a case we are interested in finding an invariant subspace spanned by the eigenvectors corresponding to the ill-conditioned eigenvalues. For a given matrix A we would like to solve

$$AX = X\Lambda \ , \quad \text{where X is } n\times k \text{ and } \Lambda \text{ is } k\times k \ .$$

Note that because we are dealing with an ill-conditioned eigensystem, Λ will not in general be diagonal. We would like to improve a subspace of eigenvectors, X, and the associated eigenvalues, instead of a single

eigenvalue and eigenvector pair as was done earlier. The subspace is chosen because the eigenvalues form a cluster, and the cluster itself is usually associated with "close" eigenvalues. Here we are being a little vague about the terms cluster and close.

By the term close we do not necessarily mean recognizably close, but that the eigenvalues are sensitive to small changes in the matrix A. This sensitivity can be measured by

$$s_i = \frac{v_i^T x_i}{\|v_i\| \|x_i\|}$$

where x_i is the right eigenvector associated with λ_i and v_i is the left eigenvector associated with λ_i. Golub and Wilkinson [8] have shown that when s_i is small, A is necessarily relatively close to a matrix with multiple eigenvalues.

By using the triangular matrix and the approximate eigenvector [2] we can compute the sensitivity information, s_i, very inexpensively in the update version of the algorithm. (This will be discussed later in the implementation chapter). By using the s_i information we can signal the case where there is an ill-conditioned eigenvalue. Corrective action can then be taken to improve the eigenvalue cluster and the invariant subspace spanned by the associated eigenvectors.

One approach that can be used to determine an invariant subspace associated with an ill-conditioned eigensystem has been developed by Varah [21]. In that approach, inverse iteration is used to determine the size of the subspace. For a given eigenvalue λ_1 an approximate eigenvector is found by means of inverse iteration $(A-\lambda_1 I)x^{(1)} = x^{(0)}$, with $x^{(0)}$ chosen so that $\|x^{(1)}\|$ is close to η_1^{-1} where $\eta_1 = \beta^{-t}$. To

determine whether λ_1 is associated with a one-dimensional subspace, $(A-\lambda_1 I)y_1 = x^{(1)}$ is solved and $\|y\|/\|x^{(1)}\|$ is examined. If λ_1 corresponds to a semisimple root, the ratio will be close to η_1^{-1}. However, if λ_1 corresponds to a multiple eigenvalue given a numerically defective matrix, then the increase in norm will be about $\eta_1^{-1/k}$, where k is the order of the root λ. The basis vectors can then be found by

$$
\begin{aligned}
(A-\lambda_1 I)x_1 &= x_0 \\
(A-\lambda_2 I)x_2 &= x_1 \\
(A-\lambda_3 I)x_3 &= x_2 \\
&\vdots \\
(A-\lambda_k I)x_k &= x_{k-1} \ .
\end{aligned}
$$

Once the invariant subspace has been determined, the improvement algorithm can be used to improve it. If we look at an example where there is a cubic elementary divisor, then there exists x_1, x_2, x_3 and λ_1, λ_2, λ_3 such that,

$$
\begin{aligned}
Ax_1 &= \lambda_1 x_1 \\
Ax_2 &= \lambda_2 x_2 + x_1 \\
Ax_3 &= \lambda_3 x_3 + x_2 \ ,
\end{aligned}
$$

or

$$
\begin{aligned}
(A-\lambda_1 I)x_1 &= 0 \\
(A-\lambda_2 I)x_2 &= x_1 \\
(A-\lambda_3 I)x_3 &= x_2 \ ,
\end{aligned}
$$

where x_1 is a principal vector of grade 1, x_2 is a principal vector of grade 2, and x_3 is a principal vector of grade 3. We can then define

corrections to the principal vectors and eigenvalues in the same manner as was done earlier. Namely,

$$A(x_1+y_1) = (\lambda_1+\mu_1)(x_1+y_1)$$
$$A(x_2+y_2) = (\lambda_2+\mu_2)(x_2+y_2) + (x_1+y_1)$$
$$A(x_3+y_3) = (\lambda_3+\mu_3)(x_3+y_3) + (x_2+y_2)$$

where y_i and μ_i are corrections to x_i and λ_i, for i=1,2,3. Given approximations to the eigenvalues, λ_i, and approximations to the principal vectors, x_i, we can solve for corrections y_i and μ_i by considering the following equations,

$$(A-\lambda_1 I)y_1 - \mu_1 x_1 = \lambda_1 x_1 - Ax_1 + \mu_1 y_1$$
$$(A-\lambda_2 I)y_2 - \mu_2 x_2 - y_1 = \lambda_2 x_2 - Ax_2 + x_1 + \mu_2 y_2$$
$$(A-\lambda_3 I)y_3 - \mu_3 x_3 - y_2 = \lambda_3 x_3 - Ax_3 + x_2 + \mu_3 y_3 \ .$$

If expressed in matrix notation we have,

$$\begin{pmatrix} A-\lambda_1 I & -x_1 & & & & & \\ e_r^T & 0 & & & & & \\ & & A-\lambda_2 I & -x_2 & & & \\ -I & & e_s^T & 0 & & & \\ & & & & A-\lambda_3 I & -x_3 \\ & & -I & & e_t^T & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ \mu_1 \\ y_2 \\ \mu_2 \\ y_3 \\ \mu_3 \end{pmatrix} = \begin{pmatrix} \lambda_1 x_1 - Ax_1 + \mu_1 y_1 \\ 0 \\ \lambda_2 x_2 - Ax_2 + x_1 + \mu_2 y_2 \\ 0 \\ \lambda_3 x_3 - Ax_3 + x_2 + \mu_3 y_3 \\ 0 \end{pmatrix}$$

This system will then be solved in the same manner as was done earlier.

Extensions to the Generalized Eigenvalue Problem

The methods described up to this point have all applied to the standard eigenvalue problem $Ax = \lambda x$. The generalization of these ideas to handle the generalized eigenvalue problem $Ax = \lambda Bx$ presents one minor difficulty in the residual calculation. As before, we have an approximate eigenpair λ and x, and we would like to find μ and y such that

$$A(x+y) = (\lambda+\mu)B(x+y) \ .$$

Then expanding and grouping terms as before, we have

$$Ay-\lambda By-\mu Bx = \lambda Bx-Ax+\mu By \ . \tag{21}$$

Equation (21) is analogous to the corresponding one for the standard problem where B is replaced by I. If we define $r = \lambda Bx-Ax$ and C as $A-\lambda B$ with the s column replaced by $-Bx$, and the same conventions on y_s and $\tilde{y}$, we obtain

$$Cy = r + y_s B\tilde{y} \ . \tag{22}$$

The basic iteration has the form

$$Cy^{(p+1)} = r + y_s^{(p)} B\tilde{y}^{(p)} \ ,$$

and when successive iterates are subtracted,

$$C\delta^{(p)} = y_s^{(p)}\tilde{By}^{(p)} - y_s^{(p-1)}\tilde{By}^{(p-1)} .$$

The complete iteration would be:

$$\begin{aligned}
C\delta^{(0)} &= r & y^{(1)} &= \delta^{(0)} \\
C\delta^{(1)} &= y_s^{(1)}\tilde{By}^{(1)} & y^{(2)} &= y^{(1)} + \delta^{(1)} \\
C\delta^{(2)} &= y_s^{(2)}\tilde{By}^{(2)} - y_s^{(1)}\tilde{By}^{(1)} & y^{(3)} &= y^{(2)} + \delta^{(2)} \\
&\vdots & &\vdots \\
C\delta^{(p)} &= y_s^{(p)}\tilde{By}^{(p)} - y_s^{(p-1)}\tilde{By}^{(p-1)} & y^{(p+1)} &= y^{(p)} + \delta^{(p)}
\end{aligned}$$

which is analogous to (5).

An additional complication is incurred by the residual calculation $\lambda Bx-Ax$, which now involves the triple product λBx. In forming the components of the residual vector r_1, Bx should be formed in double precision then multiplied by λ and the n+1 inner product found. This procedure requires a double precision and single precision operation to occur, but at no stage is the multiplication of two double precision operands required.

In trying to extend the ideas in the updating form of the improvement algorithm to the generalized problem, a difficulty arises. For the generalized eigenproblem proiblem, the QZ algorithm is the generalization of the QR algorithm used for the standard eigenvalue problem. In the QZ algorithm, orthogonal matrices Q and Z are determined to transform

$$Ax = \lambda Bx \quad \text{into}$$

$$QAZx = \lambda QBZx \ ,$$

where QAZ and QBZ are upper triangular. Because of the nature of the transformations used to generate Q, Q is not accumulated or stored but applied in the reduction process at each step. Only the transformations which go into making up Z are actually accumulated and stored. In order to apply the updating technique, both Q and Z must be accumulated and stored for use during the updated form of the improvement algorithm. For the update form to be effective, Q must be accumulated in less than $1/3\ n^3$ operations. If this is not the case, then it would be cheaper to form $A-\lambda B$ with the s column replaced with $-Bx$ and use the generalized form of Algorithm 1 to update λ and x, and ignore the eigenvalue factorization. It turns out that the accumulation and storage of the matrix Q requires more than $1/3\ n^3$ operations. Thus the updated form of the improvement algorithm cannot be effectively used to correct a single eigenpair. If many eigenpairs are to be improved then the updating will be advantageous. This will depend on the order of the matrix and the number of eigenpairs to be improved.

Numerical Results

The following section displays some results for improving eigenvalues and eigenvectors by one of the approaches described earlier. The method uses Algorithm 4 which applies the updating technique, computes an eigenvector and corrects the eigenpair at each iteration.

The experiments were carried out as follows: a matrix with known eigenvalues was generated and the eigenvalues computed using a modification to the EISPACK routine HQR2 called HQRORT. The eigenvalues, one at a time, were then passed to the improvement algorithms for correcting. In Algorithm 4, a "random" initial guess for the eigenvector is made.

All runs were made on an IBM 3033 using the H extended enhanced compiler in double precision. The word size is approximately 15 decimal digits in this case. The results as displayed were generated by taking the approximate eigenvalue and adding the correction in extended precision and printing the extended results.

The first matrix is a "magic square" of order four. A magic square is a matrix whose rows, columns, or diagonal elements add to the same number. For this example the magic square has the form

$$\begin{pmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{pmatrix}$$

and has eigenvalues 34, 8.94427..., -8.94427..., and 0 with eigenvectors

$$x_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad x_2 = \begin{pmatrix} 1 \\ -5.14... \\ -2.86... \\ -4.57... \end{pmatrix} \quad x_3 = \begin{pmatrix} -4.57... \\ -2.86... \\ -5.14... \\ 1 \end{pmatrix} \quad x_4 = \begin{pmatrix} 1/3 \\ 1 \\ -1 \\ -1/3 \end{pmatrix}$$

This matrix is fairly easy for the improvement method. It has distinct eigenvalues that are well conditioned.

Typical behavior for the improvement algorithm can be seen here. The first iteration is spent in calculating the eigenvector from the initial random vector, and the last iteration is performed to determine convergence.

```
INITIAL
     EIGENVALUE                                  EIGENVECTOR
 3.3999999999999876D+01
ITERATION  1
 3.4000000000000034117673963773853 5Q+01  9.9999999999999863390526266826626 0Q-01
                                           9.9999999999999800159855567472092 9Q-01
                                           1.0000000000000000000000000000000 0Q+00
                                           9.9999999999999970856645603589680 2Q-01
ITERATION  2
 3.4000000000000000000000000006122Q+01  9.9999999999999999999999999897448Q-01
                                           9.9999999999999999999999999948724Q-01
                                           1.0000000000000000000000000000000Q+00
                                           9.9999999999999999999999999921903Q-01
ITERATION  3
 3.3999999999999999999999999999985Q+01  9.9999999999999999999999999999930Q-01
                                           9.9999999999999999999999999999977Q-01
                                           1.0000000000000000000000000000000Q+00
                                           9.9999999999999999999999999999976Q-01
ITERATION  4
 3.3999999999999999999999999999165Q+01  1.0000000000000000000000000000000Q+00
                                           9.9999999999999999999999999995415Q-01
                                           9.9999999999999999999999999993738Q-01
                                           9.9999999999999999999999999999112Q-01
ITERATION  5
 3.3999999999999999999999999998493Q+01  9.9999999999999999999999999975890Q-01
                                           1.0000000000000000000000000000000Q+00
                                           9.9999999999999999999999999993837Q-01
                                           9.9999999999999999999999999998351Q-01



INITIAL
     EIGENVALUE                                  EIGENVECTOR
 8.9442719099991024D+00
ITERATION  1
 8.9442719099991546752304039102909 9Q+00   1.0000000000000000000000000000000Q+00
                                          -5.1433135193545678340064852739700 3Q-01
                                          -2.8662703870913173886429746047193 8Q-02
                                          -4.5700594419362999153857918523999 3Q-01
ITERATION  2
 8.9442719099991587856366946747614 4Q+00   1.0000000000000000000000000000000Q+00
                                          -5.1433135193545673502053853790020 3Q-01
                                          -2.8662703870913470041077075795771 4Q-02
                                          -4.5700594419362979493838438629633 4Q-01
ITERATION  3
 8.9442719099991587856366946749244 0Q+00   1.0000000000000000000000000000000Q+00
                                          -5.1433135193545673502053853790082 3Q-01
                                          -2.8662703870913470041077075801626 1Q-02
                                          -4.5700594419362979493838438629759 2Q-01
ITERATION  4
 8.9442719099991587856366946749244 1Q+00   1.0000000000000000000000000000000Q+00
                                          -5.1433135193545673502053853790082 6Q-01
                                          -2.8662703870913470041077075801621 8Q-02
                                          -4.5700594419362979493838438629759 5Q-01
```

```
INITIAL
     EIGENVALUE                                  EIGENVECTOR
-8.9442719099991237D+00
ITERATION  1
-8.9442719099991507408775603948925OQ+00  4.5700594419363106644436341467737 5Q-01
                                          2.8662703870912977006870240876001 6Q-02
                                          5.1433135193545601798913419860360 1Q-01
                                         -1.0000000000000000000000000000000 0Q+00
ITERATION  2
-8.9442719099991587856366946746991 2Q+00  4.5700594419362979493838438631038 6Q-01
                                          2.8662703870913470041077075795130 5Q-02
                                          5.1433135193545673502053853790079 5Q-01
                                         -1.0000000000000000000000000000000 0Q+00
ITERATION  3
-8.9442719099991587856366946749247 2Q+00  4.5700594419362979493838438629756 6Q-01
                                          2.8662703870913470041077075801613 7Q-02
                                          5.1433135193545673502053853790080 2Q-01
                                         -1.0000000000000000000000000000000 0Q+00
ITERATION  4
-8.9442719099991587856366946749247 4Q+00  4.5700594419362979493838438629756 7Q-01
                                          2.8662703870913470041077075801611 0Q-02
                                          5.1433135193545673502053853790080 7Q-01
                                         -1.0000000000000000000000000000000 0Q+00



INITIAL
     EIGENVALUE                                  EIGENVECTOR
-1.4328549785478851D-15
ITERATION  1
 6.9493288573720468274761274122006OQ-15  3.3333333333333375284729394039171 6Q-01
                                          1.0000000000000000000000000000000 0Q+00
                                         -9.9999999999999965825947523256856 8Q-01
                                         -3.3333333333333313846606286527175 5Q-01
ITERATION  2
 2.6032409872293389578587042654331 9Q-29  3.3333333333333333333333333337097Q-01
                                          9.9999999999999999999999999992752Q-01
                                         -1.0000000000000000000000000000000 0Q+00
                                         -3.3333333333333333333333333338822Q-01
ITERATION  3
-3.8518598887744717061119558851698 5Q-33  3.3333333333333333333333333333348Q-01
                                          9.9999999999999999999999999999992Q-01
                                         -1.0000000000000000000000000000000 0Q+00
                                         -3.3333333333333333333333333333348Q-01
ITERATION  4
-2.6963019221421301942783691196189 0Q-33  3.3333333333333333333333333333350Q-01
                                          9.9999999999999999999999999999992Q-01
                                         -1.0000000000000000000000000000000 0Q+00
                                         -3.3333333333333333333333333333349Q-01
ITERATION  5
-2.6963019221421301942783691196189 0Q-33  3.3333333333333333333333333333351Q-01
                                          9.9999999999999999999999999999991Q-01
                                         -1.0000000000000000000000000000000 0Q+00
                                         -3.3333333333333333333333333333350Q-01
```

The next example is a 3×3 matrix

$$\begin{pmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{pmatrix} .$$

The matrix has eigenvalues $\lambda_1 = 1$, $\lambda_2 = 2$ and $\lambda_3 = 3$. The eigenvalues are fairly sensitive to small changes in the matrix. The sensitivity, s_i, for all the eigenvectors is $O(10^4)$. A detailed discussion of this example can be found in [4].

```
INITIAL
     EIGENVALUE                                  EIGENVECTOR
 1.0000000000142959D+00
ITERATION  1
 1.000000000014182988117283912288260+00  3.333333333337856339531310723617000-01
                                        -1.000000000000000000000000000000000+00
                                        -4.703216982544220995461222433326840-13
ITERATION  2
 9.999999999999999999998055461150290-01  3.333333333333333333333218392386690-01
                                        -1.000000000000000000000000000000000+00
                                         1.166812369065812876100038472315490-23
ITERATION  3
 1.000000000000000000000000000061166Q+00  3.333333333333333333333333333552210-01
                                        -1.000000000000000000000000000000000+00
                                        -2.265432874983817820671882466176280-29
ITERATION  4
 1.000000000000000000000000000000310+00  3.333333333333333333333333333333440-01
                                        -1.000000000000000000000000000000000+00
                                        -1.117039370209480809021855787195970-32
ITERATION  5
 1.000000000000000000000000000000460+00  3.333333333333333333333333333333490-01
                                        -1.000000000000000000000000000000000+00
                                        -1.675559015880052323203227904132890-32



INITIAL
     EIGENVALUE                                  EIGENVECTOR
 1.9999999999901883D+00
ITERATION  1
 1.999999999990077312328473624347680+00  4.444444444408010827690203023842710-01
                                        -1.000000000000000000000000000000000+00
                                        -1.111111111075091621679835357456570-01
ITERATION  2
 2.000000000000000000000018975252641Q+00  4.444444444444444444444335222003090-01
                                        -1.000000000000000000000000000000000+00
                                        -1.111111111111111111111007240302140-01
ITERATION  3
 1.999999999999999999999999999961020+00  4.444444444444444444444444443049700-01
                                        -1.000000000000000000000000000000000+00
                                        -1.111111111111111111111111110973270-01
ITERATION  4
 1.999999999999999999999999999930800+00  4.444444444444444444444444443825600-01
                                        -1.000000000000000000000000000000000+00
                                        -1.111111111111111111111111110498100-01
ITERATION  5
 1.999999999999999999999999999933000+00  4.444444444444444444444444443910300-01
                                        -1.000000000000000000000000000000000+00
                                        -1.111111111111111111111111110579800-01
```

```
INITIAL
     EIGENVALUE                              EIGENVECTOR
 2.9999999999956521D+00
ITERATION  1
 2.99999999999554799179346531445844Q+00  1.42857142856110241557563602056358Q-01
                                        -1.00000000000000000000000000000000Q+00
                                         1.83673469388779290658086312175022Q-01
ITERATION  2
 2.99999999999999999999998953584213Q+00  1.42857142857142857142862711151298Q-01
                                        -1.00000000000000000000000000000000Q+00
                                         1.83673469387755102040810866818749Q-01
ITERATION  3
 2.99999999999999999999999999993310Q+00  1.42857142857142857142857142823945Q-01
                                        -1.00000000000000000000000000000000Q+00
                                         1.83673469387755102040816326563306Q-01
ITERATION  4
 2.99999999999999999999999999993313Q+00  1.42857142857142857142857142823135Q-01
                                        -1.00000000000000000000000000000000Q+00
                                         1.83673469387755102040816326564119Q-01
```

The initial eigenvalues are accurate to about 11 digits. This is predicted by the sensitivity information of the eigensystem. After 4 or 5 iterations of improvement, both the eigenvalues and eigenvectors are accurate to 29 digits.

The third example is W_{21}^{+} which is tridiagonal and symmetric with $w_{ii} = |11-i|$, $w_{i,i+1} = w_{i+1,i} = 1$. The two largest eigenvalues λ_1, λ_2 differ only in their 16^{th} decimal. There is a complete set of eigenvectors associated with this matrix. Because the matrix has such close eigenvalues we expect the improvement algorithm to have trouble in correcting the eigenvalues and eigenvectors. The original eigenvalue corresponding to λ_1 had 14 digits of accuracy and after 8 iterations of the improvement method the eigenvalue has 18 digits correct. The same behavior is observed when correcting the eigenvalue corresponding to λ_2.

```
INITIAL EIGENVALUE          1.0746194182903373D+01
ITERATION 1                 1.0746194182903438071874369086344 8Q+01
ITERATION 2                 1.0746194182903381603155779089320 3Q+01
ITERATION 3                 1.0746194182903388222860563416816 1Q+01
ITERATION 4                 1.0746194182903390953315314604310 5Q+01
ITERATION 5                 1.0746194182903392242214857255078 2Q+01
ITERATION 6                 1.0746194182903392862378499916786 7Q+01
ITERATION 7                 1.0746194182903393159883576046809 1Q+01
                FINAL EIGENVECTOR
    9.8862919084613306751699842850703 2Q-01
    7.3770935125787369530140778089637 7Q-01
    2.9955458699380240694741076006864 0Q-01
    8.4925713006535129673895912109513 8Q-02
    1.8593625050202660468344572497552 7Q-02
    3.3232420458236068431590605705030 2Q-03
    5.0236906188892842236335809072918 0Q-04
    6.5837197162154420676151504267186 4Q-05
    7.6186517872021193810716039266221 6Q-06
    7.9701078069141820386164871823579 9Q-07
    1.4917004730969128320559048790503 0Q-07
    8.0599951394965738193271143616266 1Q-07
    7.7062577269826484959625664678589 6Q-06
    6.6594426989713581403640963099258 2Q-05
    5.0814710523454064713208793184597 4Q-04
    3.3614646184026687173559264078295 4Q-03
    1.8807481331066441176704800113573 8Q-02
    8.5902493870169086919389402628141 2Q-02
    3.0299994150225542037827887460064 8Q-01
    7.4619418290339309098531555839975 7Q-01
    1.0000000000000000000000000000000 0Q+00
```

```
INITIAL EIGENVALUE          1.0746194182903271D+01
ITERATION 1                 1.0746194182903343397605944176120 8Q+01
ITERATION 2                 1.0746194182903304012444145598692 6Q+01
ITERATION 3                 1.0746194182903291161612635562505 6Q+01
ITERATION 4                 1.0746194182903326189149062486194 5Q+01
ITERATION 5                 1.0746194182903320901711907708886 4Q+01
ITERATION 6                 1.0746194182903319160916899566160 5Q+01
ITERATION 7                 1.0746194182903321561774190318061 5Q+01
ITERATION 8                 1.0746194182903321732590242593152 8Q+01
ITERATION 9                 1.0746194182903321803659694999577 6Q+01
```

FINAL EIGENVECTOR

```
    9.9976708237355899747613359902333 1Q-01
    7.4602038112537592638909297588716 0Q-01
    3.0292936747489151499465413785031 2Q-01
    8.5882485664753885675493999674199 7Q-02
    1.8803100735687416497455639041961 1Q-02
    3.3606816675109011623805225489411 1Q-03
    5.0802871275355938292136867231364 9Q-04
    6.6578679215023905391173752018898 7Q-05
    7.7026648872451575884372781216887 1Q-06
    7.9032361465338351214417528111924 5Q-07
   -1.7471499219482585860360395309999 6Q-11
   -7.9051136677666092897022031184317 1Q-07
   -7.7044598128984252299887925714042 0Q-06
   -6.6594190231207951015163281710373 0Q-05
   -5.0814706917124171395608908379728 6Q-04
   -3.3614646118711944265476574581192 4Q-03
   -1.8807481329598386102713392564703 2Q-02
   -8.5902493869731493136872720282315 7Q-02
   -3.0299994150207798731091943676304 6Q-01
   -7.4619418290332186966359089614903 9Q-01
   -1.0000000000000000000000000000000 0Q+00
```

The poor performance in this case is a result of the closeness of λ_1 and λ_2. Even though W_{21}^{+} has a well conditioned eigensystem and the initial eigenvalues are fairly accurate, the improvement algorithm converges very slowly as a result of the matrix B having condition of $O(10^{16})$. We do converge to working precision but cannot do much better. This example points out the deficiency of the method for matrices which have multiple well conditioned eigenvalues.

The next case involves a Frank matrix of order 16, F_{16}. For modest orders, some of the eigenvalues and eigenvectors are very ill conditioned. The Frank matrix of order 5 has the form

$$F_5 = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ & 3 & 3 & 2 & 1 \\ & & 2 & 2 & 1 \\ & & & 1 & 1 \end{pmatrix} .$$

The eigenvalues as returned by the EISPACK routines have only 3 or 4 correct digits. After the improvement method is used they are correct to at least 16 digits.

```
INITIAL EIGENVALUE   4.5088136657577553D-02
ITERATION 1          4.3218438981915759285190592464687 1Q-02
ITERATION 2          4.5169811871221355128753344176928 4Q-02
ITERATION 3          4.5164097839108366963316413015758 2Q-02
ITERATION 4          4.5164092593465581917771857379089 9Q-02
ITERATION 5          4.5164092592824355155070319842070 6Q-02
ITERATION 6          4.5164092592824291939114768491833 0Q-02
ITERATION 7          4.5164092592824291926201401334253 9Q-02
ITERATION 8          4.5164092592824291926201401334250 2Q-02
               FINAL EIGENVECTOR
    4.5529585808434741853381332948663 0Q-15
   -9.6256297610586173866581591524832 4Q-14
    5.2286206152537648989812452338260 0Q-13
    1.8783521308200092241042218946839 9Q-11
   -5.4761174181935521864090544749753 1Q-10
    6.5865842006481575499571514232516 8Q-09
   -5.8758612581736369878861499402607 2Q-09
   -1.3341430156355693616506367632955 8Q-06
    2.9376664781696417465552977903888 6Q-05
   -3.8474702469952418552536247175193 0Q-04
    3.5810222611447455153508343657036 6Q-03
   -2.4594903344626920515082022181291 8Q-02
    1.2352701508324541504684779517386 8Q-01
   -4.3327375874063016514680803584790 0Q-01
    9.5483590740717570807379859866574 9Q-01
   -1.0000000000000000000000000000000 0Q+00


INITIAL EIGENVALUE   6.7142228388109168D-02
ITERATION 1          6.6664313769819544264194632887665 6Q-02
ITERATION 2          6.7116986001095227214185978681371 1Q-02
ITERATION 3          6.7117818380623328998973487205732 0Q-02
ITERATION 4          6.7117819051259600801556536541658 0Q-02
ITERATION 5          6.7117819051830838251461937681921 4Q-02
ITERATION 6          6.7117819051831324884680957654965 7Q-02
ITERATION 7          6.7117819051831325293671516500583 5Q-02
ITERATION 8          6.7117819051831325292329271860822 2Q-02
               FINAL EIGENVECTOR
    1.5152605742876002413138544192081 4Q-14
   -2.1060868919989463912139053355392 9Q-13
   -4.5913280960321201373905937085878 0Q-13
    5.0312100620526522178258738161904 2Q-11
   -6.1036750312882752854904380878973 3Q-10
   -5.1170077384498064030589064141323 4Q-10
    1.0714589910558565721762583810084 5Q-07
   -1.4130002082693375868071707227692 6Q-06
    5.2720667777805337639525432430568 4Q-06
    9.5142908434837985262524949595316 6Q-05
   -1.8722537940314466111664324400198 6Q-03
    1.7517475517903167800809720659567 9Q-02
   -1.0400325717228410983992960403656 4Q-01
    4.0157567223939019807235001350665 3Q-01
   -9.3288218094816867470767072813912 2Q-01
    1.0000000000000000000000000000000 0Q+00
```

```
INITIAL EIGENVALUE    2.1728546966020205D-02
ITERATION 1           2.21275683536895041367276171229150Q-02
ITERATION 2           2.17670940262182380990456376235187Q-02
ITERATION 3           2.17643091864742214663175329898356Q-02
ITERATION 4           2.17643101462820506272220231819728Q-02
ITERATION 5           2.17643101463078914989925208745560Q-02
ITERATION 6           2.17643101463078921699435252926719Q-02
ITERATION 7           2.17643101463078921797841271304686Q-02
               FINAL EIGENVECTOR
    6.09604200702384038532704455957581Q-15
   -2.73997467322881376572083933142223Q-13
    8.11390162023412987354355092524343Q-12
   -1.88443538011708528389774685030064Q-10
    3.62343087431551569843124637939496Q-09
   -5.89610668202315394084972015216677Q-08
    8.18775331638580358500033096711219Q-07
   -9.71060335740480132731427431287424Q-06
    9.78795456947034517572640469492097Q-05
   -8.30002785502956068363334204862331Q-04
    5.82520313034380206421780033863721Q-03
   -3.30083924049340976490747364589400Q-02
    1.45374323477916687230415429288618Q-01
   -4.67590377378610502061874206437384Q-01
    9.78235689853692107820215872869586Q-01
   -1.00000000000000000000000000000000Q+00


INITIAL EIGENVALUE    3.1423815001609419D-02
ITERATION 1           2.98920447725064585345333689758718Q-02
ITERATION 2           3.13342874872101858794645745653895Q-02
ITERATION 3           3.13342893197246380510016435612399Q-02
ITERATION 4           3.13342893484457732139412910378051Q-02
ITERATION 5           3.13342893485027760813686373410242Q-02
ITERATION 6           3.13342893485028892142856233977159Q-02
ITERATION 7           3.13342893485028894440674471172407Q-02
ITERATION 8           3.13342893485028894453348926301873Q-02
ITERATION 9           3.13342893485028894453348926270434Q-02
               FINAL EIGENVECTOR
    3.17169022957775830179754362566001Q-15
   -9.80493776651468891135863511564213Q-14
    1.51277458773913301139471741752716Q-12
   -2.95783263966778301976680283763098Q-12
   -5.36183169749131968412696656076112Q-10
    1.77082759614888112651544066577968Q-08
   -3.59203450450862012845803461668147Q-07
    5.45298168674534454190165501668829Q-06
   -6.54007277430382373383086756256789Q-05
    6.29584692442211664012753484966195Q-04
   -4.85257564412694696433944370298184Q-03
    2.94570420885963346982494016083846Q-02
   -1.36309087476099540516468783438076Q-01
    4.53489484821733516647939966262720Q-01
   -9.68665710651497110554665107372980Q-01
    1.00000000000000000000000000000000Q+00
```

Implementation

In this chapter we describe some of the details skipped over in the earlier chapters. The Fortran programs that implement the algorithms given earlier are listed in the appendix. The routines are written in double precision, and trivial changes are needed in obvious places to convert to single precision. Extended double precision variables, REAL*16, are used in routine DQQDOT for the residual calculation. It should be noted that REAL*16 is not "standard" and cannot be supplied in a portable fashion easily.

In the improvement algorithm which uses updates and handles the case where there are real eigenvalues, the triangular form of the matrix is necessary as well as the orthogonal matrix that generated the triangular form. These matrices make up the real Schur form of the original matrix. The EISPACK [19] routines ORTHES, ORTRAN and a modification to HQR2, say HQRORT, can be used to generate both the triangular and orthogonal matrices. The calls to the EISPACK routines are of the form

```
CALL ORTHES(LDA,N,1,N,A,FV1)
CALL ORTRAN(LDA,N,1,N,A,FV1,Q)
CALL HQRORT(LDA,N,1,N,A,WR,WI,Q,IERR)
```

where

LDA is the leading dimension of the array A.

N is the order of the matrix A.

A contains the original matrix to start and, after the sequence of calls, contains the triangular matrix.

FV1 is a scratch array.

Q will contain the orthogonal matrix that reduced A to triangular form.

WR is an array that contains the real part of the eigenvalues.

WI is an array that contains the imaginary part of the eigenvalues.

IERR is an error flag.

If the matrix A has complex conjugate eigenpairs, then the matrix returned by HQRORT is not a triangular matrix but a quasi-triangular matrix which has 2×2 blocks on the diagonal which correspond to the complex conjugate pairs. The modified version of HQR2, HQRORT, is easily generated by changing a GO TO statement of HQR2

```
from  60 IF (EN .LT. LOW) GO TO 340
to         60 IF (EN .LT. LOW) GO TO 1001
```

Parlett and Feldman [16] describe another approach to generate the real Schur form of the original matrix. Parlett's routines, called ORTHAN and HQR3, have an advantage in that there are fewer operations required to generate T and Q than what was described for the modified EISPACK routine above. In the case where the improvement algorithm is used to calculate an eigenvector with no approximation given, Parlett's routine is advised. The calls are of the form

```
CALL ORTHAN(LDA,N,1,N,A,Q,RV1)
CALL HQR3(LDA,N,1,N,A,Q,RV1,IERR)
```

Algorithm 4 actually computes an eigenvector corresponding to an approximate eigenvalue. For the initial approximation to the eigenvector, the k^{th} column of the elementary reflector which annihilates the last n-1 components of a vector with all ones is chosen. Here, k refers to the index on the diagonal of the tridiagonal matrix T where the approximate eigenvalue is found. The k^{th} column is used so as to have a better chance of resolving eigenvalues corresponding to multiple eigenvalues. If this initial approximation fails to produce an approximate eigenvector, the next column of the elementary reflector, k+1, is chosen. This will be repeated until n attempts are made; then an error exit is taken. To determine failure of an initial approximate eigenvector, a check is made on the correction to the eigenvalue. If the correction is too large, then a new initial vector is tried.

The improvement algorithm can get into trouble when used on ill-conditioned eigenvalues. This situation can be detected fairly simply by computing the sensitivity for the eigenvalues. Chan, Feldman, and Parlett [2] point out that the sensitivity, s_i, can be computed very inexpensively from the triangular matrix of the eigenvalue decomposition. The left eigenvector of the triangular matrix requires a back substitution using T^T beginning at the k^{th} position of T, where k is the position where the eigenvalue is found on the diagonal of T.

The routine EIGCND can be used to compute the sensitivity for the eigenvalue. If the sensitivity is small in some sense, then it is

advisable to compute the invariant subspace. To detect smallness, the following test can be made

```
IF ( 1.0 + SI .EQ. 1.0 ) GO TO ...
```

Here SI is the sensitivity as returned from EIGCND, and the GO TO branches to a place where corrective action can be taken. Corrective action in this case would be to compute the invariant subspace by some method, perhaps using Varah's approach, then invoke the routine implementing the method outlined in chapter 10.

References

[1] P. M. Anselone and L. B. Rall, The Solution of Characteristic Value-Vector Problems by Newton's Method, Numer. Math. 11 (1968) 38.

[2] S. P. Chan, R. Feldman and B. N. Parlett, A Program for Computing the Condition Numbers of Matrix Eigenvalues Without Computing Eigenvectors, Algorithm 517, ACM TOMS 2 (1977) 189.

[3] A. K. Cline, C. B. Moler, G. W. Stewart and J. H. Wilkinson, An Estimate for the Condition Number of a Matrix, SIAM Num. Anal. 8 (1979) 639.

[4] G. J. Davis and C. B. Moler, Sensitivity of Matrix Eigenvalues, Int. J. Num. Meth. Engng. 12 (1978) 1367.

[5] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, LINPACK User's Guide, SIAM Pub., Philadelphia (1979).

[6] G. E. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall, Englewood Cliffs, N.J. (1967).

[7] P. E. Gill, G. H. Golub, W. Murray and M. A. Saunders, Methods for Modifying Matrix Factorizations, Math. Comp. 126 (1974) 505.

[8] G. H. Golub and J. W. Wilkinson, Ill-Conditioned Eigensystems and the Computation of the Jordan Canonical Form, SIAM Review 4 (1976) 578.

[9] W. B. Gragg and R. A. Tapia, Optimal Error Bounds for the Newton-Kantorovich Theorem, SIAM Num. Anal. 1 (1974) 10.

[10] A. S. Householder, The Theory of Matrices in Numerical Analysis, (Blaisdell), Boston, Mass. (1964).

[11] L. V. Kantorovich, Functional Analysis and Applied Mathematics, Uspehi Math. Nauk. 3 (1968) 89 (Russian).

[12] P. Lancaster, Error Analysis for the Newton-Raphson Method, Numer. Math. 9 (1966) 55.

[13] C. B. Moler, Iterative Refinement in Floating Point, J. Assoc. Comp. Mach. 2 (1967), 316.

[14] J. M. Ortega, The Newton-Kantorovich Theorem, Amer. Math. Month. (1968) 658.

[15] J. M. Ortega and W. C. Rheinboldt, Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, New York (1970).

[16] B. N. Parlett and R. Feldman, A Program to Compute the Real Schur Form of a Real Square Matrix, Memo No. ERL-M526, Electronics Research Laboratory, Univ. of Calif., Berkeley (1975).

[17] G. Peters and J. H. Wilkinson, Inverse Iteration, Ill-Conditioned Equations and Newton's Method, SIAM Review 3 (1979) 339.

[18] L. B. Rall, Computational Solution of Nonlinear Operator Equations, John Wiley and Sons, New York (1969).

[19] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler, Lect. Notes in Comp. Sci. Vol. 6, Ed. 2, Matrix Eigensystem Routines - EISPACK Guide, Springer-Verlag, Berlin (1976).

[20] G. W. Stewart, Introduction to Matrix Computation, Academic Press, New York (1973).

[21] J. M. Varah, The Computation of Bounds for the Invariant Subspaces of a General Matrix Operator, Stanford University Computer Science Department Tech. Report No. CS66, May 1967.

[22] J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, London (1965).

[23] J. H. Wilkinson, Rounding Errors in Algebraic Processes, Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, Prentice-Hall, New Jersey (1963).

[24] J. H. Wilkinson, Realistic Error Bounds for a Simple Eigenvalue and its Associated Eigenvector, Unpublished (1979).

[25] J. H. Wilkinson, Private Communication (1979).

[26] J. H. Wilkinson, Error Bounds for Computed Invariant Subspaces, unpublished (1980).

A P P E N D I X

```
      SUBROUTINE IMPRV1(A,LDA,N,X,W,SAVE,Y1,Y2,Z,R,IPVT)
C
      INTEGER LDA,N,IPVT(1)
      DOUBLE PRECISION A(LDA,1),SAVE(LDA,1),X(1),W,Y1(1),Y2(1),R(1),Z(1)
C
C     IMPRV1 GIVEN THE MATRIX A AND AN INITIAL ESTIMATE OF AN
C     EIGENVALUE AND EIGENVECTOR PAIR WILL IMPROVE THE PAIR.
C
C     ON ENTRY
C
C        A        DOUBLE PRECISION (LDA,N)
C                 THE ORIGINAL MATRIX.
C
C        LDA      INTEGER
C                 THE LEADING DIMENSION OF THE ARRAY A.
C
C        N        INTEGER
C                 THE ORDER OF THE MATRIX A.
C
C        X        DOUBLE PRECISION (N)
C                 THE EIGENVECTOR TO BE IMPROVED.
C
C        W        DOUBLE PRECISION
C                 THE EIGENVALUE TO BE IMPROVED.
C
C        SAVE     DOUBLE PRECISION (LDA,N)
C                 SAVE IS A WORK ARRAY.
C
C        Y1       DOUBLE PRECISION (N)
C                 Y1 IS A WORK ARRAY.
C
C        Z        DOUBLE PRECISION (N)
C                 Z IS A WORK ARRAY.
C
C     ON RETURN
C
C        A        HAS BEEN DESTROYED.
C
C        Y2       DOUBLE PRECISION (N)
C                 CONTAINS THE INFORMATION ON THE CORRECTION
C                 TO THE EIGENPAIR W AND X.  THE S ELEMENT OF
C                 Y2 CONTAINS THE CORRECTION TO W AND THE REST
C                 OF Y2 CONTAINS THE CORRECTION TO X.  S IS THE
C                 POSITION IN X WHICH HAS BEEN NORMALIZED TO HAVE
C                 VALUE 1.0 AND IS CORRECT.
C
C        IPVT     INTEGER (N)
C                 AN INTEGER VECTOR OF PIVOT INDICES.
C
C     THIS VERSION DATED 3/80.
C     J.J. DONGARRA ARGONNE NATIONAL LABORATORY AND UNIVERSITY
C                   OF NEW MEXICO.
C
C     FORTRAN DABS
```

```
C     BLAS DSCAL,IDAMAX,DCOPY
C     LINPACK DGECO,DGESL,(DGEFA)
C     AUXILARY DQQDOT,DQADD
C
      DOUBLE PRECISION RCOND,DQQDOT,DQADD
      DOUBLE PRECISION ST(2)
      INTEGER IMAX,IDAMAX,I
C
C
C     NORMALIZE THE EIGENVECTOR.
C
      IMAX = IDAMAX(N,X,1)
      CALL DSCAL(N,1.0D0/DABS(X(IMAX)),X,1)
C
C     CALCULATE THE RESIDUAL WITH EXTENDED PRECISION
C     ACCUMULATION OF INNER PRODUCT.
C
      DO 10 I = 1,N
         ST(1) = -W
         ST(2) = X(I)
         Z(I) = -DQQDOT(N,A(I,1),LDA,X,1,ST)
   10 CONTINUE
C
C     FORM B FROM A - LAMDA*I WITH THE IMAX-TH COLUMN
C     REPLACED BY MINUS THE EIGENVECTOR. IMAX IS THE
C     INDEX OF THE LARGEST COMPONENT OF X.
C
      DO 20 I = 1,N
         A(I,I) = A(I,I) - W
         A(I,IMAX) = -X(I)
   20 CONTINUE
C
C     SAVE THE B MATRIX.
C
      DO 30 I = 1,N
         CALL DCOPY(N,A(1,I),1,SAVE(1,I),1)
   30 CONTINUE
C
C     FACTOR THE B MATRIX IN PREPARATION TO ITERATE.
C
      CALL DGECO(A,LDA,N,IPVT,RCOND,R)
      DO 40 I = 1,N
         Y2(I) = 0.0D0
   40 CONTINUE
C
C     START THE ITERATION.
C
      DO 100 L = 1,5
         CALL DCOPY(N,Z,1,R,1)
         CALL DGESL(A,LDA,N,IPVT,Z,0)
         DO 50 I = 1,N
            Y1(I) = Y2(I)
            Y2(I) = Z(I) + Y1(I)
   50    CONTINUE
```

```
            DO 60 I = 1,N
               IF( Y1(I) .NE. Y2(I) ) GO TO 70
   60       CONTINUE
            GO TO 110
   70       CONTINUE
C
C           CALCULATE RESIDUAL TO DO ONE STEP OF ITERATIVE IMPROVEMENT.
C
            DO 80 I = 1,N
               ST(1) = -R(I)
               ST(2) = 1.0D0
               R(I) = -DQQDOT(N,SAVE(I,1),LDA,Z,1,ST)
   80       CONTINUE
C
C           CALCULATE THE NEW RIGHT HAND SIDE.
C
            DO 90 I = 1,N
               IF( IMAX .EQ. I ) GO TO 90
               Z(I) = Y1(IMAX)*Z(I) + Z(IMAX)*Y1(I) + Z(IMAX)*Z(I)
   90       CONTINUE
            Z(IMAX) = R(IMAX)
  100    CONTINUE
  110    CONTINUE
C
C        CORRECT THE EIGENVALUE AND EIGENVECTOR.
C
         CALL DCOPY(N,Y2,1,Z,1)
         Z(IMAX) = 0.0
         W = DQADD(W,Y2(IMAX))
         DO 120 I = 1,N
            X(I) = DQADD(X(I),Z(I))
  120    CONTINUE
         RETURN
C
         END
```

```
      SUBROUTINE IMPRV2(T,LDT,N,X,W,Q,A,Y1,Y2,Z,R,Z1,Z2)
C
      INTEGER LDT,N
      DOUBLE PRECISION T(LDT,1),X(1),W,Q(LDT,1),A(LDT,1),Y1(1),Y2(1)
      DOUBLE PRECISION Z(1),R(1),Z1(1),Z2(1)
C
C     THIS SUBROUTINE WILL IMPROVE A GIVEN EIGENVALUE AND
C     EIGENVECTOR PAIR. THE METHOD IS ITERATIVE AND REQUIRES
C     ORDER N**2 WORK. THE IMPROVEMENT IS EQUIVALENT TO
C     CARRYING OUT THE EIGENVALUE COMPUTATION IN EXTENDED PRECISION
C     AND THEN TRUNCATING THE RESULTS TO WORKING PRECISION.
C
C     ON ENTRY
C
C        T       DOUBLE PRECISION(LDT,N)
C                CONTAINS THE TRIANGULAR MATRIX.
C
C        LDT     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  T.
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  T.
C
C        X       DOUBLE PRECISION(N)
C                CONTAINS THE APPROXIMATE EIGENVECTOR
C                TO BE IMPROVED.
C
C        W       DOUBLE PRECISION
C                CONTAINS THE APPROXIMATE EIGENVALUE
C                TO BE IMPROVED.
C
C        Q       DOUBLE PRECISION(LDT,N)
C                CONTAINS THE TRANSFORMATIONS NEEDED TO
C                REDUCE THE ORIGINIAL MATRIX TO TRIANGULAR FORM.
C
C        A       DOUBLE PRECISION(LDT,N)
C                CONTAINS THE ORIGINAL MATRIX.
C
C     ON RETURN
C
C        X       CONTAINS THE IMPROVED EIGENVECTOR.
C
C        W       CONTAINS THE IMPROVED EIGENVALUE.
C
C        Y1,Y2,Z,R,Z1,Z2
C                DOUBLE PRECISION(N)
C                WORK VECTORS.
C
C     THIS VERSION DATED 3/80
C     JACK DONGARRA, ARGONNE NATIONAL LABORATORY, AND
C                    UNIVERSITY OF NEW MEXICO.
C
C     FORTRAN DABS,DSQRT
C     BLAS DSCAL,IDAMAX,DCOPY,DDOT,DROTG,DROT
```

```
C     LINPACK DTRSL
C     AUXILARY DQQDOT,DQADD
C
      INTEGER I,IB,IMAX,IDAMAX
      DOUBLE PRECISION C,S,DQQDOT,DQADD,T1,T2
      DOUBLE PRECISION ST(2)
C
C     NORMALIZE THE EIGENVECTOR.
C
      IMAX = IDAMAX(N,X,1)
      CALL DSCAL(N,1.0D0/DABS(X(IMAX)),X,1)
C
C     CALCULATE THE RESIDUAL, FOR THE EIGENVALUE PROBLEM,
C     WITH EXTENDED PRECISION ACCUMULATION OF INNER PRODUCT.
C
      DO 10 I = 1,N
         ST(1) = -W
         ST(2) = X(I)
         R(I) = -DQQDOT(N,A(I,1),LDT,X,1,ST)
   10 CONTINUE
C
C     FORM  T - LAMDA*I  WITH THE IMAX-TH COLUMN
C     REPLACED BY MINUS THE EIGENVECTOR. IMAX IS THE
C     INDEX OF THE LARGEST COMPONENT OF X.
C
      DO 20 I = 1,N
         T(I,I) = T(I,I) - W
   20 CONTINUE
C
C     FORM C = - X - T(,IMAX), WHERE IMAX IS THE LARGEST COMPONENT
C     OF X.
C
      DO 30 I = 1,N
         Z2(I) = - X(I) - A(I,IMAX)
   30 CONTINUE
      Z2(IMAX) = Z2(IMAX) + W
C
C     FORM D = TRANS(Q)*C
C
      DO 40 I = 1,N
         Y1(I) = DDOT(N,Q(1,I),1,Z2,1)
   40 CONTINUE
C
C     RESTORE MATRIX TO TRIANGULAR FORM AFTER RANK ONE UPDATE,
C     AND APPLY TRANSFORMATIONS AND RANK ONE UPDATE TO
C     THE RIGHT HAND SIDE.
C
      DO 50 I = 1,N
         Z(I) = DDOT(N,Q(1,I),1,R,1)
   50 CONTINUE
      T2 = Y1(N)
      DO 60 IB = 2,N
         I = N - IB + 2
         T1 = Y1(I-1)
```

```
         CALL DROTG(T1,T2,C,S)
         CALL DROT(IB,T(I-1,I-1),LDT,T(I,I-1),LDT,C,S)
         CALL DROT(1,Z(I-1),1,Z(I),1,C,S)
         Z1(I) = T2
         T2 = T1
   60 CONTINUE
C
C     ADD T + D*TRANS(F), WHERE F = TRANS(Q)*E(S).
C
      DO 70 I = 1,N
         T(1,I) = T(1,I) + T1*Q(IMAX,I)
   70 CONTINUE
      DO 80 I = 2,N
         T1 = T(I-1,I-1)
         T2 = T(I,I-1)
         CALL DROTG(T1,T2,C,S)
         T(I-1,I-1) = T1
         Z2(I) = T2
         CALL DROT(N-I+1,T(I-1,I),LDT,T(I,I),LDT,C,S)
         T(I,I-1) = 0.0D0
         CALL DROT(1,Z(I-1),1,Z(I),1,C,S)
   80 CONTINUE
      DO 90 I = 1,N
         Y2(I) = 0.0D0
   90 CONTINUE
C
C     START THE ITERATION.
C
      DO 180 INFO = 1,5
C
C        SOLVE TRIANGULAR SYSTEM.
C
         CALL DCOPY(N,Z,1,R,1)
         CALL DTRSL(T,LDT,N,Z,1,INF)
C
C        CALCULATE RESIDUAL USING EXTENDED PRECISION ACCUMULATION
C        OF INNER PRODUCT.
C
         DO 110 I = 1,N
            ST(1) = -R(I)
            ST(2) = 1.0D0
            R(I) = -DQQDOT(N-I+1,T(I,I),LDT,Z(I),1,ST)
  110    CONTINUE
C
C        APPLY RANK ONE UPDATES TO SOLUTION.
C
         DO 120 IB = 2,N
            I = N - IB + 2
            IF( DABS(Z1(I)) .EQ. 1.0D0 ) C = 0.0D0
            IF( DABS(Z1(I)) .EQ. 1.0D0 ) S = 1.0D0
            IF( DABS(Z1(I)) .LT. 1.0D0 ) C = DSQRT(1.0D0 - Z1(I)*Z1(I))
            IF( DABS(Z1(I)) .LT. 1.0D0 ) S = Z1(I)
            IF( DABS(Z1(I)) .GT. 1.0D0 ) C = 1.0D0/Z1(I)
            IF( DABS(Z1(I)) .GT. 1.0D0 ) S = DSQRT(1.0D0 - C*C)
```

```
            CALL DROT(1,Z(I-1),1,Z(I),1,C,S)
  120     CONTINUE
          DO 130 I = 2,N
             IF( DABS(Z2(I)) .EQ. 1.0D0 ) C = 0.0D0
             IF( DABS(Z2(I)) .EQ. 1.0D0 ) S = 1.0D0
             IF( DABS(Z2(I)) .LT. 1.0D0 ) C = DSQRT(1.0D0 - Z2(I)*Z2(I))
             IF( DABS(Z2(I)) .LT. 1.0D0 ) S = Z2(I)
             IF( DABS(Z2(I)) .GT. 1.0D0 ) C = 1.0D0/Z2(I)
             IF( DABS(Z2(I)) .GT. 1.0D0 ) S = DSQRT(1.0D0 - C*C)
             CALL DROT(1,Z(I-1),1,Z(I),1,C,S)
  130     CONTINUE
C
C         FORM NEW CORRECTIONS.
C
          DO 140 I = 1,N
             Y1(I) = Y2(I)
             Y2(I) = Z(I) + Y1(I)
  140     CONTINUE
C
C         TEST IF CONVERGED.
C
          DO 150 I = 1,N
             IF( Y1(I) .NE. Y2(I) ) GO TO 160
  150     CONTINUE
          GO TO 185
  160     CONTINUE
C
C         CALCULATE THE NEW RIGHT HAND SIDE.
C
          DO 170 I = 1,N
             IF( IMAX .EQ. I ) GO TO 170
             Z(I) = R(I) + Y1(IMAX)*Z(I) + Z(IMAX)*Y1(I) + Z(IMAX)*Z(I)
  170     CONTINUE
          Z(IMAX) = R(IMAX)
  180 CONTINUE
  185 CONTINUE
C
C     UN-TRANSFORM THE CORRECTIONS.
C
      DO 190 IB = 2,N
         I = N - IB + 2
         IF( DABS(Z2(I)) .EQ. 1.0D0 ) C = 0.0D0
         IF( DABS(Z2(I)) .EQ. 1.0D0 ) S = 1.0D0
         IF( DABS(Z2(I)) .LT. 1.0D0 ) C = DSQRT(1.0D0 - Z2(I)*Z2(I))
         IF( DABS(Z2(I)) .LT. 1.0D0 ) S = Z2(I)
         IF( DABS(Z2(I)) .GT. 1.0D0 ) C = 1.0D0/Z2(I)
         IF( DABS(Z2(I)) .GT. 1.0D0 ) S = DSQRT(1.0D0 - C*C)
         CALL DROT(1,Y2(I-1),1,Y2(I),1,C,-S)
  190 CONTINUE
      DO 200 I = 2,N
         IF( DABS(Z1(I)) .EQ. 1.0D0 ) C = 0.0D0
         IF( DABS(Z1(I)) .EQ. 1.0D0 ) S = 1.0D0
         IF( DABS(Z1(I)) .LT. 1.0D0 ) C = DSQRT(1.0D0 - Z1(I)*Z1(I))
         IF( DABS(Z1(I)) .LT. 1.0D0 ) S = Z1(I)
```

```
            IF( DABS(Z1(I)) .GT. 1.0D0 ) C = 1.0D0/Z1(I)
            IF( DABS(Z1(I)) .GT. 1.0D0 ) S = DSQRT(1.0D0 - C*C)
            CALL DROT(1,Y2(I-1),1,Y2(I),1,C,-S)
  200    CONTINUE
         DO 210 I = 1,N
            Z(I) = DDOT(N,Q(I,1),LDT,Y2,1)
  210    CONTINUE
C
C        CORRECT THE EIGENVALUE AND EIGENVECTOR.
C
         CALL DCOPY(N,Z,1,Y2,1)
         Z(IMAX) = 0.0D0
         W = DQADD(W,Y2(IMAX))
         DO 220 I = 1,N
            X(I) = DQADD(X(I),Z(I))
  220    CONTINUE
         RETURN
C
         END
```

```
      SUBROUTINE IMPRV4(T,LDT,N,X,W,K,Q,A,Y1,Y2,Z,R,Z1,Z2,INFO)
C
      INTEGER LDT,N,K,INFO
      DOUBLE PRECISION T(LDT,1),X(1),W,Q(LDT,1),A(LDT,1),Y1(1),Y2(1)
      DOUBLE PRECISION Z(1),R(1),Z1(1),Z2(1)
C
C     THIS SUBROUTINE WILL IMPROVE A GIVEN EIGENVALUE AND
C     EIGENVECTOR PAIR. THE METHOD IS ITERATIVE AND REQUIRES
C     ORDER N**2 WORK. THE IMPROVEMENT IS EQUIVALENT TO
C     CARRYING OUT THE EIGENVALUE COMPUTATION IN EXTENDED PRECISION
C     AND THEN TRUNCATING THE RESULTS TO WORKING PRECISION.
C     THIS ROUTINE PERFORMS A FULL NEWTON LIKE ITERATION AND
C     CORRECTS THE EIGENVALUE AND EIGENVECTOR EVERY ITERATION.
C
C     ON ENTRY
C
C        T       DOUBLE PRECISION(LDT,N)
C                CONTAINS THE TRIANGULAR MATRIX.
C
C        LDT     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  T.
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  T.
C
C        X       DOUBLE PRECISION(N)
C                CONTAINS THE APPROXIMATE EIGENVECTOR
C                TO BE IMPROVED.
C
C        W       DOUBLE PRECISION
C                CONTAINS THE APPROXIMATE EIGENVALUE
C                TO BE IMPROVED.
C
C        K       INTEGER
C                IS THE INDEX OF THE EIGENPAIR.
C
C        Q       DOUBLE PRECISION(LDT,N)
C                REDUCE THE ORIGINIAL MATRIX TO TRIANGULAR FORM.
C
C        A       DOUBLE PRECISION(LDT,N)
C                CONTAINS THE ORIGINAL MATRIX.
C
C     ON RETURN
C
C        X       CONTAINS THE IMPROVED EIGENVECTOR.
C
C        W       CONTAINS THE IMPROVED EIGENVALUE.
C
C        Y1,Y2,Z,R,Z1,Z2
C                DOUBLE PRECISION(N)
C                WORK VECTORS.
C
C     THIS VERSION DATED 3/80.
C     JACK DONGARRA, ARGONNE NATIONAL LABORATORY, AND
```

```
C                    UNIVERSITY OF NEW MEXICO.
C
C     FORTRAN DABS,DSQRT
C     BLAS DSCAL,IDAMAX,DCOPY,DROTG,DROT,DDOT
C     LINPACK DTRSL
C     AUXILIARY DQQDOT,DQADD
C
      INTEGER I,IB,IMAX,IDAMAX
      DOUBLE PRECISION C,S,DQQDOT,LAMDAR
      DOUBLE PRECISION NEWZ,OLDZ,TZ,DDOT,SI,T1,T2,ANORM
      DOUBLE PRECISION ST(2)
      IDIF = K
      OLDZ = 1.0D10
      ANORM = 0.0D0
      DO 1 I = 1,N
         S = DASUM(N,A(1,I),1)
         IF( S .GT. ANORM ) ANORM = S
    1 CONTINUE
      IRESET = 0
      LAMDAR = W
      ITER = 0
      GO TO 130
    5 CONTINUE
C
      INFO = 0
C
C     NORMALIZE THE EIGENVECTOR.
C
      IMAX = IDAMAX(N,X,1)
      CALL DSCAL(N,1.0D0/DABS(X(IMAX)),X,1)
C
C     CALCULATE THE RESIDUAL, FOR THE EIGENVALUE PROBLEM,
C     WITH DOUBLE PRECISION ACCUMULATION OF INNER PRODUCT.
C
      DO 10 I = 1,N
         ST(1) = -W
         ST(2) = X(I)
         R(I) = -DQQDOT(N,A(I,1),LDT,X,1,ST)
   10 CONTINUE
C
C     FORM  T - LAMDA*I  WITH THE IMAX-TH COLUMN
C     REPLACED BY MINUS THE EIGENVECTOR. IMAX IS THE
C     INDEX OF THE LARGEST COMPONENT OF X.
C
      DO 20 I = 1,N
         T(I,I) = T(I,I) - W
   20 CONTINUE
C
C     FORM C = - X - T(,IMAX), WHERE IMAX IS THE LARGEST COMPONENT
C     OF X.
C
      DO 30 I = 1,N
         Z2(I) = - X(I) - A(I,IMAX)
   30 CONTINUE
```

```
      Z2(IMAX) = Z2(IMAX) + W
C
C     FORM D = TRANS(Q)*C
C
      DO 40 I = 1,N
         Y1(I) = DDOT(N,Q(1,I),1,Z2,1)
   40 CONTINUE
C
C     RESTORE MATRIX TO TRIANGULAR FORM AFTER RANK ONE UPDATE,
C     THE RIGHT HAND SIDE.
C
      DO 50 I = 1,N
         Z(I) = DDOT(N,Q(1,I),1,R,1)
   50 CONTINUE
      T2 = Y1(N)
      DO 60 IB = 2,N
         I = N - IB + 2
         T1 = Y1(I-1)
         CALL DROTG(T1,T2,C,S)
         CALL DROT(IB,T(I-1,I-1),LDT,T(I,I-1),LDT,C,S)
         CALL DROT(1,Z(I-1),1,Z(I),1,C,S)
         Z1(I) = T2
         T2 = T1
   60 CONTINUE
C
C     ADD T + D*TRANS(F), WHERE F = TRANS(Q)*E(S).
C
      DO 70 I = 1,N
         T(1,I) = T(1,I) + T1*Q(IMAX,I)
   70 CONTINUE
      TZ = T1
      DO 80 I = 2,N
         T1 = T(I-1,I-1)
         T2 = T(I,I-1)
         CALL DROTG(T1,T2,C,S)
         T(I-1,I-1) = T1
         Z2(I) = T2
         CALL DROT(N-I+1,T(I-1,I),LDT,T(I,I),LDT,C,S)
         T(I,I-1) = 0.0D0
         CALL DROT(1,Z(I-1),1,Z(I),1,C,S)
   80 CONTINUE
C
C     SOLVE TRIANGULAR SYSTEM.
C
      CALL DTRSL(T,LDT,N,Z,1,INF)
C
C     APPLY RANK ONE UPDATES TO SOLUTION.
C
      CALL DCOPY(N,Z,1,Y2,1)
C
C     UN-TRANSFORM THE CORRECTIONS.
C
      DO 90 IB = 2,N
         I = N - IB + 2
```

```
         IF( DABS(Z2(I)) .EQ. 1.0D0 ) C = 0.0D0
         IF( DABS(Z2(I)) .EQ. 1.0D0 ) S = 1.0D0
         IF( DABS(Z2(I)) .LT. 1.0D0 ) C = DSQRT(1.0D0 - Z2(I)*Z2(I))
         IF( DABS(Z2(I)) .LT. 1.0D0 ) S = Z2(I)
         IF( DABS(Z2(I)) .GT. 1.0D0 ) C = 1.0D0/Z2(I)
         IF( DABS(Z2(I)) .GT. 1.0D0 ) S = DSQRT(1.0D0 - C*C)
         CALL DROT(N-I+1,T(I-1,I),LDT,T(I,I),LDT,C,-S)
         T1 = T(I-1,I-1)
         T(I,I-1) = S*T1
         T(I-1,I-1) = C*T1
   90 CONTINUE
      DO 100 I = 1,N
         T(1,I) = T(1,I) - TZ*Q(IMAX,I)
  100 CONTINUE
      DO 110 I = 2,N
         IF( DABS(Z1(I)) .EQ. 1.0D0 ) C = 0.0D0
         IF( DABS(Z1(I)) .EQ. 1.0D0 ) S = 1.0D0
         IF( DABS(Z1(I)) .LT. 1.0D0 ) C = DSQRT(1.0D0 - Z1(I)*Z1(I))
         IF( DABS(Z1(I)) .LT. 1.0D0 ) S = Z1(I)
         IF( DABS(Z1(I)) .GT. 1.0D0 ) C = 1.0D0/Z1(I)
         IF( DABS(Z1(I)) .GT. 1.0D0 ) S = DSQRT(1 0D0 - C*C)
         CALL DROT(N-I+2,T(I-1,I-1),LDT,T(I,I-1),LDT,C,-S)
  110 CONTINUE
      DO 120 I = 1,N
         T(I,1) = T(I,I) + W
         Z(I) = DDOT(N,Q(I,1),LDT,Y2,1)
  120 CONTINUE
C
C     TEST IF RESTART NEEDED
C
      S = ANORM*DSQRT(DFLOAT(N))*10.0D6*16.0D0**(-13)
      IF( DABS(Z(IMAX)) .LT. S ) GO TO 150
      IRESFT = IRESET + 1
      ITER = 0
      IF( IRESET .GT. N ) RETURN
      W = LAMDAR
C
C     PICK A NEW STARTING VECTOR.
C
  130 CONTINUE
      S = DSQRT(DFLOAT(N))
      C = 1.0D0/(DFLOAT(N) + S)
      X(1) = (1.0D0 + S)/C
      DO 140 I = 2,N
         X(I) = C
  140 CONTINUE
      X(IDIF) = X(IDIF) - 1.0D0
      IDIF = IDIF + 1
      IDIF = MOD(IDIF,N) + 1
      GO TO 5
C
C     CORRECT THE EIGENVALUE AND EIGENVECTOR.
C
  150 CONTINUE
```

```
      W = QEXTD(W) + QEXTD(Z(IMAX))
      NEWZ = Z(IMAX)
      Z(IMAX) = 0.0D0
      DO 160 I = 1,N
         X(I) = QEXTD(X(I)) + QEXTD(Z(I))
  160 CONTINUE
C
C     TEST FOR CONVERGENCE.
C
      IF( DABS(OLDZ) .LE. DABS(NEWZ) ) GO TO 180
      OLDZ = NEWZ
      ITER = ITER + 1
      IF(ITER .LT. 10 ) GO TO 170
      INFO = 10
      GO TO 180
  170 CONTINUE
      GO TO 5
  180 CONTINUE
      RETURN
C
      END
```

```
      SUBROUTINE IMPRVC(A,T,Q,LDA,N,XR,XI,WR,WI,Y1,Y2,Z,R,K)
C
      INTEGER LDA,N
      DOUBLE PRECISION A(LDA,1),XR(1),XI(1),W,Y1(1),Y2(1),R(1),Z(1)
      DOUBLE PRECISION T(LDA,1),Q(LDA,1),QXR(40),QXI(40)
      DOUBLE PRECISION WR,WI
C
C     THIS SUBROUTINE WILL IMPROVE A GIVEN COMPLEX CONJUGATE
C     EIGENVALUE AND ITS ASSOCIATED EIGENVECTOR. THE METHOD IS
C     ITERATIVE AND REQUIRES ORDER N**2 WORK PER ITERATION. THE
C     IMPROVEMENT IS EQUIVALENT TO CARRYING OUT THE EIGENVALUE
C     COMPUTATION IN EXTENDED PRECISION AND THEN TRUNCATING THE
C     RESULTS TO WORKING PRECISION.
C
C     ON ENTRY
C
C        A       DOUBLE PRECISION(LDA,N)
C                CONTAINS THE ORIGINAL MATRIX.
C
C        T       DOUBLE PRECISION(LDA,N)
C                CONTAINS THE QUASI-TRIANGULAR MATRIX.
C
C        Q       DOUBLE PRECISION(LDA,N)
C                CONTAINS THE ORTHOGONAL MATRIX USED TO REDUCE
C                A TO QUASI-TRIANGULAR FORM.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAYS A,T AND Q.
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A.
C
C        XR      DOUBLE PRECISION(N)
C                CONTAINS THE APPROXIMATE DOUBLE PRECISION PART OF
C                THE EIGENVECTOR TO BE IMPROVED.
C
C        XI      DOUBLE PRECISION(N)
C                CONTAINS THE APPROXIMATE IMAGINARY PART OF THE
C                EIGENVECTOR TO BE IMPROVED.
C
C        WR      DOUBLE PRECISION
C                CONTAINS THE APPROXIMATE DOUBLE PRECISION PART OF
C                THE EIGENVALUE TO BE IMPROVED.
C
C        WI      DOUBLE PRECISION
C                CONTAINS THE APPROXIMATE IMAGINARY PART OF THE
C                EIGENVALUE TO BE IMPROVED.
C
C        K       INTEGER
C                IS THE POSITION ON THE DIAGONAL OF T WHERE THE
C                2X2 BLOCK ASSOCIATED WITH (WR,WI) STARTS.
C
C     ON RETURN
C
```

```
C        XR      CONTAINS THE IMPROVED REAL PART OF
C                THE EIGENVECTOR.
C
C        XI      CONTAINS THE IMPROVED IMAGINARY PART OF THE
C                EIGENVECTOR.
C
C        WR      CONTAINS THE IMPROVED REAL PART OF
C                THE EIGENVALUE.
C
C        WI      CONTAINS THE IMPROVED IMAGINARY PART OF THE EIGENVALUE.
C
C        Y1,Y2,Z,R1,Z2
C                DOUBLE PRECISION(N)
C                WORK VECTORS.
C
C     THIS VERSION DATED 3/80.
C     JACK DONGARRA, ARGONNE NATIONAL LABORATORY, AND
C                    UNIVERSITY OF NEW MEXICO.
C
C
C     BLAS DCOPY,DDOT
C     AUXILIARY DQQDOT,DQADD,HESSL(CSOLVE)
C
      DOUBLE PRECISION RCOND,DQQDOT,DQADD,DDOT
      DOUBLE PRECISION ST(2),ST1,ST2
      INTEGER I
      DOUBLE PRECISION R1,RES
C
C
C     CALCULATE THE RESIDUAL WITH EXTENDED PRECISION
C     ACCUMULATION OF INNER PRODUCT.
C
      DO 10 I = 1,N
         ST(1) = -WR*XR(I)
         ST(2) = WI*XI(I)
         Z(I) = -DQQDOT(N,A(I,1),LDA,XR,1,ST)
         ST(1) = -WI*XR(I)
         ST(2) = -WR*XI(I)
         Z(I+N) = -DQQDOT(N,A(I,1),LDA,XI,1,ST)
   10 CONTINUE
      N2P1 = 2*N + 1
      NSQ = 2*N + 2
      DO 20 I = N2P1,NSQ
         Z(I) = 0.0D0
   20 CONTINUE
C
C     INITIALIZE Y2 AND MULTIPLE THE EIGENVECTOR BY TRANS(Q).
C
      NBIG = 2*N+2
      DO 30 I = 1,NBIG
         Y2(I) = 0.0D0
   30 CONTINUE
      DO 40 I = 1,N
         QXR(I) = DDOT(N,Q(1,I),1,XR,1)
```

```
         QXI(I) = DDOT(N,Q(1,I),1,XI,1)
   40 CONTINUE
C
C     START THE ITERATION.
C
      DO 120 INFO = 1,5
         CALL DCOPY(NBIG,Z,1,R,1)
         DO 50 I = 1,N
            Z(I) = DDOT(N,Q(1,I),1,R,1)
            Z(I+N) = DDOT(N,Q(1,I),1,R(N+1),1)
   50    CONTINUE
         CALL HESSL(T,LDA,N,Z,QXR,QXI,WR,WI,K+1)
         CALL DCOPY(NBIG,Z,1,Y1,1)
         DO 60 I = 1,N
            Z(I) = DDOT(N,Q(I,1),LDA,Y1,1)
            Z(I+N) = DDOT(N,Q(I,1),LDA,Y1(N+1),1)
   60    CONTINUE
         DO 70 I = 1,NBIG
            Y1(I) = Y2(I)
            Y2(I) = Z(I) + Y1(I)
   70    CONTINUE
         DO 80 I = 1,NBIG
            IF( Y1(I) .NE. Y2(I) ) GO TO 90
   80    CONTINUE
         GO TO 130
   90    CONTINUE
C
C        CALCULATE THE NEW RIGHT HAND SIDE.
C
         DO 100 I = 1,N
            Z(I) = Y2(2*N+1)*Y2(I) - Y2(2*N+2)*Y2(I+N)
     $                 - Y1(2*N+1)*Y1(I) + Y1(2*N+2)*Y1(I+N)
            Z(I+N) = Y2(2*N+2)*Y2(I) + Y2(2*N+1)*Y2(I+N)
     $                 - Y1(2*N+2)*Y1(I) - Y1(2*N+1)*Y1(I+N)
  100    CONTINUE
         DO 110 I = N2P1,NSQ
            Z(I) = 0.0D0
  110    CONTINUE
  120 CONTINUE
  130 CONTINUE
      WR = DQADD(WR,Y2(2*N+1))
      WI = DQADD(WI,Y2(2*N+2))
      DO 140 I = 1,N
         XR(I) = DQADD(XR(I),Y2(I))
         XI(I) = DQADD(XI(I),Y2(I+N))
  140 CONTINUE
      RETURN
C
      END
      SUBROUTINE HESSL(T,LDT,N,B,XR,XI,WR,WI,K)
C
C     HESSL WILL SOLVE A*X=B, WHERE A IS ORDER 2*N+2, MADE
C     UP OF THE MATRIX CALLED SCRIPT A IN THESIS.
C
```

```
C     ON ENTRY
C
C        T       DOUBLE PRECISION(LDT,N)
C                CONTAINS THE QUASI-TRIANGULAR MATRIX.
C
C        LDT     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY T.
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  T.
C
C        B       DOUBLE PRECISION(2*N+2)
C                THE RIGHT HAND SIDE.
C
C        XR      DOUBLE PRECISION(N)
C                CONTAINS THE APPROXIMATE DOUBLE PRECISION PART OF
C                THE EIGENVECTOR TO BE IMPROVED.
C
C        XI      DOUBLE PRECISION(N)
C                CONTAINS THE APPROXIMATE IMAGINARY PART OF THE
C                EIGENVECTOR TO BE IMPROVED.
C
C        WR      DOUBLE PRECISION
C                CONTAINS THE APPROXIMATE DOUBLE PRECISION PART OF
C                THE EIGENVALUE TO BE IMPROVED.
C
C        WI      DOUBLE PRECISION
C                CONTAINS THE APPROXIMATE IMAGINARY PART OF THE
C                EIGENVALUE TO BE IMPROVED.
C
C        K       INTEGER
C                IS THE POSITION ON THE DIAGONAL OF T WHERE THE
C                2X2 BLOCK ASSOCIATED WITH (WR,WI) STARTS.
C
C     ON RETURN
C
C        B       CONTAINS THE SOLUTION.
C
C
      DOUBLE PRECISION T(LDT,1),B(1),XR(1),XI(1),WR,WI
C
      DOUBLE PRECISION U(42,4),A(4,4),Y(4),DDOT
      INTEGER IPVT(4)
C
      LDU = 42
C
C     CSOLVE SOLVES THE SPECIAL STRUCTURE MATRIX OF ORDER 2*N+2
C     CALLED C IN THESIS.
C
C
C     SOLVE INV(C)*B
C
      CALL CSOLVE(T,LDT,N,WR,WI,XR,XI,B,K)
C
```

```
C     UPDATE FORMULA, GENERALIZED S-M-W, IS APPLIED TO SOLUTION
C
C      INV(C)*B+INV(C)*U*INV(I-TRANS(V)*INV(C)*U)*TRANS(V)*INV(C)*B
C
C
C     FORM U
C
      NBIG = 2*N + 2
      DO 20 J = 1,4
         DO 10 I = 1,NBIG
            U(I,J) = 0.0D0
   10    CONTINUE
   20 CONTINUE
      U(2*N+1,1) = -1.0D0
      U(2*N+2,2) = -1.0D0
      U(K,3) = -1.0D0
      U(K+N,4) = -1.0D0
C
C     SOLVE INV(C)*U
C
      DO 30 I = 1,4
         CALL CSOLVE(T,LDT,N,WR,WI,XR,XI,U(1,I),K)
   30 CONTINUE
C
C     FORM TRANS(V)*INV(C)*B
C
      Y(1) = DDOT(N,XR,1,B(1),1) - B(2*N+1)
      Y(2) = DDOT(N,XI,1,B(N+1),1) - B(2*N+2)
      Y(3) = -B(K-1+N)
      Y(4) = B(K-1)
C
C     FORM (I - TRANS(V)*(INV(C)*U))
C
      DO 40 I = 1,4
         A(1,I) = -(DDOT(N,XR,1,U(1,I),1) - U(2*N+1,I))
         A(2,I) = -(DDOT(N,XI,1,U(N+1,I),1) - U(2*N+2,I))
         A(3,I) = U(K-1+N,I)
         A(4,I) = -U(K-1,I)
         A(I,I) = A(I,I) + 1.0D0
   40 CONTINUE
C
C     SOLVE INV(I-TRANS(V)*INV(C)*U))*(TRANS(V)*INV(C)*B)
C
      CALL DGEFA(A,4,4,IPVT,INFO)
      CALL DGESL(A,4,4,IPVT,Y,0)
C
C     FORM INV(C)*B + INV(C)*U*(ABOVE QUANTITY)
C
      DO 50 I = 1,NBIG
         B(I) = B(I) + DDOT(4,U(I,1),LDU,Y,1)
   50 CONTINUE
      RETURN
      END
      SUBROUTINE CSOLVE(T,LDT,N,WR,WI,XR,XI,B,K)
```

```
C
C     SPECIAL STRUCTURE SOLVER, MATRIX SOLVING FOR IS ORDER 2*N+2
C     CALLED C IN THESIS.
C
      DOUBLE PRECISION T(LDT,1),WR,WI,XR(1),XI(1),B(1)
C
      DOUBLE PRECISION A(4,4),Y(4)
      INTEGER IPVT(4)
C
      LDA = 4
      NBIG = 2*N + 2
C
C     IN SOLVING FOR THE MATRIX C, THERE ARE 2 CASES, 2X2 CASE
C     OR 4X4 CASE. THE 4X4 CASE CORRESPONDS TO A 2X2 BUMP IN THE
C     QUASI-TRIANGULAR MATRIX T.
C
      I = N
   10 CONTINUE
      IF( I .EQ. 1 ) GO TO 20
      IF( T(I,I-1) .NE. 0.0D0 ) GO TO 50
C
C     2X2 CASE (NO BUMP)
C
   20 CONTINUE
      A(1,1) = T(I,I) - WR
      A(2,1) = -WI
      A(1,2) = WI
      A(2,2) = T(I,I) - WR
      Y(1) = B(I)
      Y(2) = B(N+I)
      IF( I .EQ. N ) GO TO 40
      IP1 = I + 1
      DO 30 J = IP1,N
         Y(1) = Y(1) - T(I,J)*B(J)
         Y(2) = Y(2) - T(I,J)*B(N+J)
   30 CONTINUE
   40 CONTINUE
      Y(1) = Y(1) + XR(I)*B(2*N+1) - XI(I)*B(2*N+2)
      Y(2) = Y(2) + XI(I)*B(2*N+1) + XR(I)*B(2*N+2)
C
C     SOLVE 2X2 SYSTEM
C
      CALL DGEFA(A,LDA,2,IPVT,INFO)
      CALL DGESL(A,LDA,2,IPVT,Y,0)
      B(I) = Y(1)
      B(N+I) = Y(2)
      I = I - 1
      GO TO 90
C
C     4X4 CASE (BUMP)
C
   50 CONTINUE
      A(1,1) = T(I-1,I-1) - WR
      A(2,1) = T(I,I-1)
```

```
      A(3,1) = -WI
      A(4,1) = 0.0D0
      A(1,2) = T(I-1,I)
      A(2,2) = T(I,I) - WR
      A(3,2) = 0.0D0
      A(4,2) = -WI
      A(1,3) = WI
      A(2,3) = 0.0D0
      A(3,3) = T(I-1,I-1) - WR
      A(4,3) = T(I,I-1)
      A(1,4) = 0.0D0
      A(2,4) = WI
      A(3,4) = T(I-1,I)
      A(4,4) = T(I,I) - WR
      IF( I .NE. K ) GO TO 60
      A(2,3) = 1.0D0
      A(4,1) = -1.0D0
   60 CONTINUE
      Y(1) = B(I-1)
      Y(2) = B(I)
      Y(3) = B(N+I-1)
      Y(4) = B(N+I)
      IF( I .EQ. N ) GO TO 80
      IP1 = I + 1
      DO 70 J = IP1,N
         Y(1) = Y(1) - T(I-1,J)*B(J)
         Y(2) = Y(2) - T(I,J)*B(J)
         Y(3) = Y(3) - T(I-1,J)*B(N+J)
         Y(4) = Y(4) - T(I,J)*B(N+J)
   70 CONTINUE
   80 CONTINUE
      Y(1) = Y(1) + XR(I-1)*B(2*N+1) - XI(I-1)*B(2*N+2)
      Y(2) = Y(2) + XR(I)*B(2*N+1) - XI(I)*B(2*N+2)
      Y(3) = Y(3) + XI(I-1)*B(2*N+1) + XR(I-1)*B(2*N+2)
      Y(4) = Y(4) + XI(I)*B(2*N+1) + XR(I)*B(2*N+2)
C
C     SOLVE 4X4 SYSTEM
C
      CALL DGEFA(A,LDA,4,IPVT,INFO)
      CALL DGESL(A,LDA,4,IPVT,Y,0)
      B(I-1) = Y(1)
      B(I) = Y(2)
      B(N+I-1) = Y(3)
      B(N+I) = Y(4)
      I = I - 2
   90 CONTINUE
      IF( I .GT. 0 ) GO TO 10
      RETURN
      END
```

```
      SUBROUTINE EIGCND(T,LDT,N,W,X,K,Y,SI)
C
      INTEGER LDT,N
      DOUBLE PRECISION T(LDT,1),W,X(1),Y(1),SI
C
C     THIS SUBROUTINE COMPUTES THE SENSITIVITY FOR THE
C     EIGENVALUE PROBLEM, GIVEN A TRIANGULAR MATRIX AND
C     THE RIGHT EIGENVECTOR.
C
C     ON ENTRY
C
C        T     DOUBLE PRECISION(LDT,N)
C              IS THE UPPER TRIANGULAR MATRIX WHICH
C              PRODUCED THE EIGENVALUE W.
C
C        LDT   INTEGER
C              THE LEADING DIMENSION OF THE ARRAY T.
C
C        N     INTEGER
C              THE ORDER OF THE MATRIX T.
C
C        W     DOUBLE PRECISION
C              THE EIGENVALUE
C
C        X     DOUBLE PRECISION(N)
C              THE EIGENVECTOR CORRESPONDING TO W, MAY NOT
C              BE THE RIGHT EIGENVECTOR OF THE MATRIX T,
C              BUT OF A MATRIX WHICH IS SIMILAR TO T.
C
C        K     INTEGER
C              THE INDEX OF WHERE W OCCURS IN THE MATRIX T.
C
C     ON RETURN
C
C        Y     DOUBLE PRECISION(N)
C              IS THE LEFT EIGENVECTOR OF THE MATRIX T.
C
C        SI    DOUBLE PRECISION
C              IS THE SENSITIVITY OF THE EIGENVALUE W.
C
C     THIS VERSION DATED 3/80
C     JACK DONGARRA, ARGONNE NATIONAL LABORATORY AND THE
C                    UNIVERSITY OF NEW MEXICO.
C
C     FORTRAN DABS
C     BLAS DSCAL,DDOT,IDAMAX
C
      INTEGER I,IDAMAX,KM1,KP1
      DOUBLE PRECISION DDOT
C
      DO 10 I = 1,N
         T(I,I) = T(I,I) - W
   10 CONTINUE
C
```

```
C     COMPUTE THE SENSITIVITY OF THE EIGENPAIR.
C
      IF( K .EQ. 1 ) GO TO 30
      KM1 = K - 1
      DO 20 I = 1,KM1
         Y(I) = 0.0D0
   20 CONTINUE
   30 CONTINUE
      Y(K) = 1.0D0
      SI = 1.0D0
      IF( K .EQ. N ) GO TO 50
      KP1 = K + 1
      DO 40 I = KP1,N
         Y(I) = -DDOT(I-K,T(K,I),1,Y(K),1)/T(I,I)
   40 CONTINUE
   50 CONTINUE
      J = IDAMAX(N,Y,1)
      CALL DSCAL(N,1.0D0/DABS(Y(J)),Y,1)
      SI = DDOT(N,X,1,Y,1)
      RETURN
C
      END
```

```
      DOUBLE PRECISION FUNCTION DQQDOT(N,X,INCX,Y,INCY,ST)
      DOUBLE PRECISION X(1),Y(1),ST(2)
      REAL*16 T
C
C     THIS ROUTINE FORMS AN INNER PRODUCT OF THE
C     VECTORS X AND Y AND ADDS IN EXTENDED PRECISION
C     THE QUANTITY ST.  THE ACCUMULATION OF INNER PRODUCT
C     IS DONE USING EXTENDED PRECISION.
C
      T = QEXTD(ST(1))*QEXTD(ST(2))
      IF( N .LE. 0 ) GO TO 20
      IX = 1
      IY = 1
      DO 10 I = 1,N
         T = T + QEXTD(X(IX))*QEXTD(Y(IY))
         IX = IX + INCX
         IY = IY + INCY
   10 CONTINUE
   20 CONTINUE
      DQQDOT = T
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION DQADD(A,B)
C
C     THIS ROUTINE ADD TWO VARIABLES TOGETHER IN EXTENDED
C     PRECISION AND TRUNCATES THE RESULT TO WORKING PRECISION.
C
      DOUBLE PRECISION A,B
C
      DQADD = QEXTD(A) + QEXTD(B)
      RETURN
      END
```