Algorithm XXX: Fortran Subroutines for Computing the Eigenvalues and Eigenvectors of a General Matrix by Reduction to General Tridiagonal Form*

J. J. Dongarra G. A. Geist C. H. Romine†

October 7, 1993

Abstract. This paper describes programs to reduce a nonsymmetric matrix to tridiagonal form, compute the eigenvalues of the tridiagonal matrix, improve the accuracy of an eigenvalue, and compute the corresponding eigenvector. The intended purpose of the software is to find a few eigenpairs of a dense nonsymmetric matrix faster and more accurately than previous methods. The performance and accuracy of the new routines are compared to two EISPACK paths: `RG` and `HQR-INVIT`. The results show that the new routines always more accurate and also faster if less than 20% of the eigenpairs are needed.

Categories and Subject Description. F.2.1 *Analysis of Algorithms and Problem Complexity*: Numerical Algorithms and Problems — *computations on matrices*; G.1.3 *Numerical Analysis*: Numerical Linear Algebra — *eigenvalue problems*; G.4 *Mathematical Software* — algorithm analysis

General Terms. Algorithms, eigenvalues, nonsymmetric

Additional Key Words and Phrases. Condensed form, numerical algorithms

1 Introduction and Objectives

A standard approach in computing the eigenvalues of a general square matrix is to reduce the matrix first to Hessenberg form by a sequence of orthogonal transformations, and then determine the eigenvalues of the Hessenberg matrix

*This work was supported in part by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under Contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

†The work of this author was also supported in part by the U.S. Navy Space and Naval Warfare Systems (SPAWAR) Contract N00039-89-C-0001.

through an iterative process known as the QR algorithm [2]. The reduction to Hessenberg form requires $O(n^3)$ operations, where n is the order of the matrix, and the subsequent iterative phase also requires $O(n^3)$ operations. The subroutine package EISPACK [8] uses this scheme to compute all the eigenvalues and eigenvectors of a general matrix.

If the original matrix is symmetric, then that symmetry can be preserved in the initial reduction, so that the reduced matrix is tridiagonal. Although the reduction to tridiagonal form still requires $O(n^3)$ operations, the subsequent iterations preserve the tridiagonal form and hence are much less expensive, so that the total cost of the iterative phase is reduced to $O(n^2)$ operations. Again, standard software is available in EISPACK for implementing this two-phase approach for the symmetric case.

The attractively low operation count obtained when iterating with a tridiagonal matrix suggests that the tridiagonal form would be extremely beneficial in the nonsymmetric case as well. Such an approach presents two difficulties, however. First, QR iteration does not preserve the structure of a nonsymmetric tridiagonal matrix. This problem can be overcome by using LR iteration [7] instead, which preserves the tridiagonal form. Second, it is difficult to reduce a nonsymmetric matrix to tridiagonal form by similarity transformations in a numerically stable manner. Methods to improve the stability can be found in [3]. Here, we describe the software available to reduce the matrix to tridiagonal form and to compute the eigenvalues and eigenvectors of the resulting tridiagonal matrix.

2 Initial Approximation to Eigenvalues

2.1 Reduction to Tridiagonal Form

The algorithm used in the direct reduction to tridiagonal form is discussed in detail in [4]. The algorithm alternately eliminates columns and rows of the matrix, preserving the form shown in Figure 1. Retaining this matrix structure allows us to improve the overall stability of the algorithm by pivoting at each step.

$$\left(\begin{array}{cc|cc} T_{k-1} & & & \\ & & \times & \\ \hline & \times & \alpha & w^T \\ & & v & B_{n-k} \end{array}\right)$$

Figure 1: Partially reduced matrix.

At the kth stage (see Figure 1), the algorithm applies the permutation that minimizes the maximum multiplier in the transformation matrix $N_r^{-1}N_c$. Here, N_r and N_c are elementary matrices such that $N_c A N_c^{-1}$ reduces column k and $N_r^{-1}(N_c A N_c^{-1})N_r$ reduces row k. This minimization can be performed in $O(n-k)$ time because of the special structure of $N_r^{-1}N_c$. Details of this minimization algorithm can be found in [?].

The reduction algorithm may encounter a pivot that is zero (or unacceptably small) regardless of the permutation. When this occurs, the original reduction is said to have broken down, and the subprogram `NEWSTR` is called. `NEWSTR` applies a random Householder similarity transformation to the original matrix. The original matrix must therefore be saved, but this is already necessary in order to apply the iterative refinement method described in Section 3.2. `NEWSTR` is only applied once in our scheme. If the reduction of the second matrix also breaks down, then the algorithm returns an error code. This occurrence has not yet been observed in practice.

The transformations used in annihilating each column and row are saved in the locations made available by the eliminations at each step. These transformations are needed for the calculation of the eigenvectors during the iterative refinement step.

2.2 Eigenvalues of a Tridiagonal Matrix

One of the most efficient methods of calculating all the eigenvalues of a nonsymmetric tridiagonal matrix is LR iteration. Most of the improvements that have been incorporated into QR iteration over the years [9], such as implicit double-shift iterations, deflation, splitting, and arbitrary shifts, can also be used in the context of LR iteration.

An implementation of LR iteration that is specifically tailored for tridiagonal matrices, called `TLR`, has been developed. The user supplies the tridiagonal matrix as three vectors. In the first step, the matrix is scaled so that its superdiagonal elements are equal to one. This reduces the operation count, since LR iteration preserves the form of such a tridiagonal matrix. Moreover, the superdiagonal vector is now free for use as a working array. Implicit double-shift iterations and splitting are implemented just as they are in EISPACK for `HQR`. Splitting due to either negligable subdiagonal elements or to two consecutive small subdiagonal elements are implemented. The criteria we use for negligable and small entries are the same as in `HQR`.

Implicit arbitrary shifts are invoked in two different contexts in `TLR`. First, if an eigenvalue has not converged in 20 iterations, then the iteration is assumed to be stuck in a cycle, and one arbitrary (random) double-shift is applied. Second, if the LR iteration encounters a zero (small) diagonal element, then the iteration breaks down, and one arbitrary shift is applied to change the values of the diagonal elements. (Another obvious method for avoiding a zero diagonal is to pivot inside LR, but this has the major drawback of destroying the tridiagonal structure of the matrix). Up to ten consecutive arbitrary shifts are tried if the breakdown condition persists, after which the algorithm aborts with an error condition. However, a single arbitrary shift proved sufficient during all our tests.

Because of the potential for breakdown and the need to restart an iteration with a different shift, a copy of the matrix is made before the start of each iteration. This requires at most $2n$ storage. One n vector must be supplied by the user for this purpose. The second n vector initially holds the superdiagonal, but this space is reclaimed after the matrix is scaled.

3 Improving the Accuracy of the Eigenvalue and Computing the Eigenvector

Approximations to the eigenvalues of A are obtained by reducing the matrix to tridiagonal form T (with `ATOTRI`) and then calculating the eigenvalues of T (with `TLR`). In many cases, particularly for small matrices, these computed eigenvalues closely approximate the eigenvalues of A. However, for larger matrices, or for matrices whose eigenvalues are ill-conditioned, the rounding errors introduced during the reduction of A to tridiagonal form, coupled with the errors introduced by LR iteration, can induce significant errors in the computed eigenvalues. Hence, we regard the reduction to tridiagonal form T and the subsequent calculation of the eigenvalues of T as yielding approximations to the eigenvalues of A that are then improved in a subsequent phase of the computation.

3.1 Inverse Iteration with Rayleigh Quotients

One standard technique for improving the accuracy of an eigenvalue and at the same time computing the associated eigenvector is to apply inverse iteration coupled with calculating the Rayleigh quotient. If only a few eigenpairs are desired, then inverse iteration is fairly attractive, since it is accurate and reasonably rapid. The EISPACK routine `INVIT` performs inverse iteration (without Rayleigh quotients) to a Hessenberg matrix. Each iteration requires $O(n^2)$ operations, since solving a linear system with a new right-hand side is required for each iteration. If the complete eigensystem of a dense matrix is required, then the EISPACK routine `RG` is recommended because it is robust, highly accurate, and requires only $O(n^3)$ operations for the full eigensystem.

Another alternative is to apply inverse iteration with Rayleigh quotients to the tridiagonal matrix T obtained from A by `ATOTRI`. Again, the solution of a different linear system for each iteration is required, but the linear systems now have a tridiagonal coefficient matrix and therefore can be solved in only $O(n)$ steps. Thus, inverse iteration with Rayleigh quotients applied to the matrix T is a very fast means of obtaining accurate approximations to the eigensystem of T. Unfortunately, to obtain the eigenvectors of the original matrix A, one must apply the inverse of the transformations that reduced A to tridiagonal form to the computed eigenvectors of T, and the eigenvectors of A will suffer from any resulting roundoff error. Moreover, the eigenvalues of T may differ from those of A for the same reason. The results given in Section 4 indicate the degree of inaccuracy stemming from these roundoff errors.

In summary, inverse iteration on T can give a useful rapid initial approximation to the eigensystem of A. But there may be inaccuracies introduced by rounding error either in calculating the eigenvalues or in obtaining the eigenvectors of A from the eigenvectors of T.

3.2 Iterative Refinement

It has long been known that Newton's method for the solution of nonlinear systems can be applied to the problem of calculating the eigensystem of a matrix [6]. Moreover, in [1], Dongarra *et al.* describe an algorithm for improving the accuracy of an eigenpair based on Newton's method. The main drawback of their approach is that it is too costly, in general. In this section, we describe a less costly variant of the algorithm given in [1] that takes advantage of the tridiagonalization of A while still obtaining a high degree of accuracy.

Assume that (λ, x) is an approximate eigenpair of the matrix A and that $\lambda + \delta\lambda$ and $x + \delta x$ are a neighboring eigenpair such that the relationship

$$A(x + \delta x) = (\lambda + \delta\lambda)(x + \delta x),$$

is exact. Thus, $\delta\lambda$ and δx represent the errors associated with the computed values λ and x, respectively.

Rearranging the equation, we have

$$(A - \lambda I)\delta x - \delta\lambda x = \lambda x - Ax + \delta\lambda\,\delta x,$$

where the last term on the right will be of second order in the errors in λ and x.

If we let $r = \lambda x - Ax$ and assume that the second-order term $\delta\lambda\,\delta x$ is negligible, we can rewrite the equation in the form

$$\begin{pmatrix} A - \lambda I & -x \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta\lambda \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix},$$

where $e_s^T \delta x = 0$ is a normalization applied to x such that the s component of x equals one, implying $\delta x_s = 0$ (see [1] for details).

When the original approximate eigenvalue is found by using the reduction to tridiagonal form, this yields a matrix N such that

$$A = N^{-1} T N.$$

Using the transformations from the reduction to tridiagonal form, we have

$$\begin{pmatrix} N & \\ & 1 \end{pmatrix} \begin{pmatrix} A - \lambda I & -x \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} N^{-1} & \\ & 1 \end{pmatrix} \begin{pmatrix} N & \\ & 1 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta\lambda \end{pmatrix}$$

$$= \begin{pmatrix} N & \\ & 1 \end{pmatrix} \begin{pmatrix} r \\ 0 \end{pmatrix},$$

which can be rewritten as

$$\begin{pmatrix} T - \lambda I & -Nx \\ e_s^T N^{-1} & 0 \end{pmatrix} \begin{pmatrix} \overline{\delta x} \\ \delta\lambda \end{pmatrix} = \begin{pmatrix} \overline{r} \\ 0 \end{pmatrix},$$

where $\overline{r} = Nr$ and $\overline{\delta x} = N\delta x$. The solution to the resulting linear system will produce approximations to the errors $\delta\lambda$ and δx, yielding new approximations to the eigenpair. The linear system is easily solved by transforming it into a tridiagonal system of equations by a rank-one modification. The software we have implemented applies the Sherman-Morrison formula [5] to solve the system of equations.

Given the original matrix A, the tridiagonal matrix T, the transformations N that reduced A to T ($A = N^{-1}TN$), and the approximate eigenvalue λ_0, the refinement algorithm can be described as follows.

v is an initial guess for the eigenvector;
perform one step of inverse iteration, $(T - \lambda_0 I)v_0 = v$;
for $i = 0, 1, 2, \ldots$
  $x_i = Nv_i$;
  $r_i = Ax_i - \lambda_i x_i$;
  $g_i^T = (e_s^T N, -1)^T$;
  solve $\begin{pmatrix} T - \lambda_i I & -N^{-1}x_i \\ g_i^T & 0 \end{pmatrix} \begin{pmatrix} y_i \\ \mu_i \end{pmatrix} = \begin{pmatrix} N^{-1}r_i \\ 0 \end{pmatrix}$;
  $w_i = Ny_i$;
  $x_{i+1} = x_i + w_i$;
  $\lambda_{i+1} = \lambda_i + \mu_i$;
  check if converged, $\frac{||Ax_{i+1} - \lambda_{i+1}x_{i+1}||}{10||A||\epsilon} \leq 1$;
end.

Note that the eigenpair is refined relative to the original matrix, *i.e.*, the residual is computed with the original data A and the improvement is being made to the eigenvector of A. The tridiagonal matrix T and the transformations N are used solely to simplify solving the system of equations. Hence the convergence will be to the eigensystem of the original matrix A, not the tridiagonal matrix T.

Because of the relationship with Newton's method, convergence will be guaranteed when $\beta\eta\kappa < 1/2$, where

$$\beta = \left|\left| \begin{pmatrix} A - \lambda_i I & -x_i \\ e_s^T & 0 \end{pmatrix} \right|\right|^{-1},$$

$\eta = ||x_{i+1} - x_i||$, and $\kappa = 2$ (κ is a bound on the second derivative). As can be seen, the convergence rate and error bound depend on the condition of the matrix, (see [?] for additional information).

The approach described here will not only improve the accuracy of the approximate eigenvalue λ but will also compute the eigenvector. The convergence theorem for this iterative procedure can be found in [1].

During the improvement phase, the subprogram named `REFINE` is called with the original data matrix `A`, the reduced tridiagonal matrix `T`, the transformation `N`, and an approximate eigenvalue `(WR,WI)` as parameters. A single inverse iteration step is performed with the tridiagonal matrix `T` (using `GTINIT`) to obtain an initial approximation to the eigenvector associated with the given eigenvalue. On return from `REFINE` the improved eigenvalue is stored in `(WR,WI)` and the improved eigenvector in `(XR,XI)`.

4 Examples and Performance

We present two test suites to illustrate the speed and accuracy of the new algorithms. The performance of `RG` is included for comparison. All experiments were executed on an IBM RS/6000 model 530, using the Fortran compiler `xlf` with optimization.

Tables 1 and 2 show the results for random matrices with the entries uniformly distributed in $[-1.0, 1.0]$. Table 1 shows the maximum difference between the eigenvalues computed by `ATOTRI-TLR` and those calculated by `RG`. The maximum difference of the improved eigenvalues is also given. Finally, the residual is given for the results from inverse iteration, iterative refinement, and `RG`. The inverse iteration results are obtained by calling the EISPACK routines `ELMHES` and `HQR`, followed by `INVIT` for every eigenpair. The maximum residual over all the eigenpairs is reported in the table. Similarly, `REFINE` was called for every eigenpair, and the maximum residual is reported. In every case the smallest maximum residual was generated with the new iterative refinement routines.

Table 1: Comparison of the accuracy of the new routines to the EISPACK routine `RG`. Residual is $\max ||Ax - \lambda x||_\infty$ and e_λ is $\max |\lambda_i - \overline{\lambda_i}|$. ($\lambda_i$ is the eigenvalue obtained from `RG`, and $\overline{\lambda_i}$ is the computed eigenvalue.

Accuracy of Routines on dense random matrices					
	`ATOTRI-TLR`	`RG`	`INVIT`	`REFINE`	
problem		(EISPACK)	(EISPACK)		
size	e_λ	residual	residual	e_λ	residual
10	8.7E-14	1.8E-14	2.0E-14	4.4E-15	3.7E-16
100	7.2E-06	5.3E-12	2.9E-12	2.7E-13	5.1E-13
500	1.2E-02	2.5E-09	3.0E-10	4.3E-12	2.3E-12

Table 2 compares the execution times of three methods of calculating eigenpairs for a nonsymmetric matrix. For reference the time required for `RG` to

Table 2: Comparison of execution times in seconds of the new routines to the EISPACK routine RG. Time for INVIT and REFINE are per eigenpair.

Performance of Routines on dense random matrices					
problem size	RG all (λ, x)	ATOTRI TLR	ELMHES HQR	INVIT (per λ, x)	REFINE (per λ, x)
10	.024	.004	.030	.0004	.0036
100	1.71	.293	.796	.0053	.0332
300	78.3	5.56	34.5	.1490	.4342
500	459	22.2	202	.5911	1.867

calculate all the eigenpairs is given. RG does not allow the user to calculate selected eigenpairs. If selected eigenpairs are desired, then the user can call the EISPACK path ELMHES, HQR, INVIT, ELMBAK. The table shows the time to reduce the matrix to Hessenberg form and calculate all its eigenvalues. In a separate column, the average time to calculate an eigenpair is given. (The time for ELMBAK is divided among the n eigenpairs calculated). The table also shows the time to reduce the matrix to tridiagonal form and calculate its eigenvalues. This operation is amazingly fast on a cache-based machine like the RS/6000. The average time per eigenpair for improving the eigenvalue and calculating the corresponding eigenvector with iterative refinement is about four times more than using INVIT. But because the routines ATOTRI and TLR are so fast, the total time for calculating up to 20% of the eigenpairs is smaller (and the results more accurate) using the new routines.

The results of running the EISPACK general matrix test suite [8] are shown in Table 3. This test suite consists of 35 small matrices (none exceeding 20×20) that are designed to be pathological with respect to their eigenvalues and eigenvectors. Most of the matrices are ill-conditioned, some are defective, some are derogatory, and some are all three. The accuracy and robustness of the new algorithms are displayed by this test, where we compare the residual from RG to GTINIT and REFINE. GTINIT applies inverse iteration with Rayleigh quotients to the tridiagonal matrix T until convergence to the desired eigenpair is achieved. The eigenvectors of A are then obtained by applying the inverse of the transformation matrix N. For reference the maximum error (with respect to RG) of the initial eigenvalue estimate from TLR is given for each matrix.

5 Results

¿From the examples in the previous section and numerous other tests, these new routines appear superior in both speed and accuracy to existing methods (EISPACK) for calculating a few eigenpairs of a dense nonsynnetric matrix. The speed advantage is clear if less than 20% of the eigenpairs are required.

Table 3: Maximum residual for the three methods of calculating eigenvalue/eigenvector pairs for dense matrices.

EISPACK Test Suite of Real General Matrices				
$\max \lvert(\lambda - \overline{\lambda}_{RG})/\overline{\lambda}_{RG}\rvert$		$\max \lvert\lvert Ax - \lambda x\rvert\rvert_{\infty}$		
problem number	`ATOTRI` `TLR`	inverse iteration	iterative refinement	EISPACK (`RG`)
1	1E-13	2.5E-12	2.9E-13	1.2E-12
2	1E-11	9.2E-07	2.1E-07	6.3E-06
3	8E-07	9.0E-13	1.3E-14	4.6E-06
4	5E-15	1.8E-14	2.7E-13	1.0E-13
5	1E-15	1.7E-07	9.4E-09	9.4E-07
6	1E-13	1.5E-07	1.2E-09	2.4E-08
7	8E-10	3.8E-08	2.9E-10	8.5E-09
8	0E-00	0.0E-00	0.0E-00	0.0E-00
9	2E-06	2.9E-15	1.7E-13	5.3E-09
10	5E-15	1.2E-10	9.5E-11	1.8E-08
11	7E-16	1.7E-14	1.3E-14	1.7E-13
12	3E-00	2.9E-15	1.7E-15	2.4E-14
13	7E-15	1.7E-13	9.2E-16	1.7E-14
14	3E-15	3.3E-12	1.9E-16	2.4E-14
15	7E-16	5.2E-14	4.8E-16	1.6E-14
16	0E-00	7.5E-15	0.0E-00	1.1E-49
17	0E-00	4.4E-15	0.0E-00	1.2E-30
18	0E-00	6.3E-15	0.0E-00	0.0E-00
19	8E-07	9.0E-15	8.8E-09	2.7E-08
20	1E-16	1.4E-14	1.0E-15	9.7E-15
21	1E-16	6.3E-15	2.2E-16	6.0E-15
22	5E-14	1.0E-13	7.1E-16	2.1E-14
23	6E-14	2.0E-10	3.2E-17	2.9E-14
24	1E-10	2.5E-06	6.2E-09	1.1E-02
25	1E-16	8.7E-07	2.2E-15	6.0E-14
26	1E-16	4.3E-13	3.6E-14	2.2E-15
27	6E+07	3.6E-01	9.0E-10	2.6E-06
28	1E-16	4.8E-14	1.2E-14	5.7E-14
29	1E-16	2.4E-14	2.8E-14	4.0E-12
30	1E-13	5.2E-14	2.3E-13	4.2E-13
31	2E-12	5.7E-14	1.8E-15	5.6E-14
32	1E-03	1.4E-14	1.4E-05	4.4E-07
33	1E-16	5.4E-01	1.9E-04	1.1E-08
34	1E-16	4.4E-02	9.1E-14	1.5E-08
35	1E-01	1.8E-12	1.8E-05	2.7E-13

The demand for routines to solve just such problems is growing rapidly in the areas of quantum chemistry and materials science.

It is also clear from our tests that if all the eigenpairs are required, then the EISPACK routine `RG` is the recommended approach.

6 Description of the Software and Programming Details

In this section we show the software implementation of the new algorithms.

```
      SUBROUTINE ATOTRI( LDA, A, N, PIVOTS, INFO )
c  Purpose:
c    This subroutine reduces an n-by-n real general matrix A to
c    tridiagonal form using elementary similarity transformations.
c
c    At each step k the permutation that minimizes the maximum entry
c    in the transformation matrix which reduces column k then row k
c    is applied.
c
c  Arguments:
c   LDA      -integer
c             leading dimension of A
c
c   A        -double precision array of dimension (LDA,N)
c             On entry A contains the matrix being reduced.
c             On exit A is overwritten by its tridiagonal form.
c
c   N        -integer
c             N specifies the order of the matrix A.
c             N must be nonnegative.
c             not modified.
c
c   PIVOTS   -integer vector of dimension (LDA)
c             On exit pivots contains the pivot sequence used during
c             the reduction (permutation vector).
c
c   INFO     -integer
c             On exit, INFO is set to
c             0  normal return.
c             1  if NEWSTR should be executed before ATOTRI.
c
      SUBROUTINE NEWSTR( A, LDA, N, W, IFLAG )
c  Purpose:
c     Subroutine to generate a random Householder transformation to
c     apply to the matrix A to scramble it.  The matrix A is
c     assumed to be in dense format.
c
c  Arguments:
```

```
c   A        -double precision array of dimension (LDA,N)
c             On entry A contains the original matrix.
c             On exit, A contains QAQ, where Q is defined by W below.
c
c   LDA      -leading dimension of A
c
c   N        -N specifies the order of the matrix A.
c             N must be nonnegative.
c             not modified.
c
c   W        -double precision vector of length N.
c             On exit, contains a random Householder vector defining
c             a Householder transformation Q=I-2WW'.
c
c   IFLAG    -integer
c             On exit, IFLAG is set to one, indicating that NEWSTR
c             has been called.
c
      SUBROUTINE TLR( N, DIAG, SUB, SUP, SAV, INFO )
c  Purpose:
c    This subroutine determines the eigenvalues of a general
c    tridiagonal matrix stored in three vectors of length N
c    by applying implicit double-shift LR iterations.
c
c    The eigenvalues are returned with the real part on the
c    diagonal and the imaginary part on the subdiagonal.
c    INFO equals 1 on exit if TLR is unable to determine
c    the eigenvalues.
c
c  Arguments:
c   N     -integer
c          N specifies the order of the tridiagonal matrix.
c          not modified
c
c   DIAG  -double precision array of dimension N
c          On entry DIAG contains the diagonal of the tridiagonal
c          matrix.
c          On exit DIAG contains the real part of the eigenvalues.
c
c   SUB   -double precision array of dimension N
c          On entry SUB contains the sub-diagonal of the
c          tridiagonal matrix.
c          On exit SUB contains the imaginary part of the eigenvalues.
c
```

```
c   SUP   -double precision array of dimension N
c          On entry SUP contains the super-diagonal of the
c          tridiagonal matrix. It is used as a work array
c          in the iteration.
c
c   SAV   -double precision array of dimension N
c          SAV is a work array used along with SUP to save a copy
c          of the previous iteration matrix in case the present
c          iteration breaks down and an arbitrary shift is required.
c
c   INFO  -integer
c          On exit, INFO is set to
c           0  normal return.
c           1  failure to converge to one or more eigenvalues.
c              User should revert to HQR from EISPACK.
c
      SUBROUTINE REFINE( N, A, LDA, AORG, WR, WI, XR, XI, IPVT, W,
     $                   IFLAG, WORK, LDWORK )
c  Purpose:
c     This routine uses an iterative refinement technique to
c     improve the accuracy of the eigenvalue approximation
c     (WR,WI) and to compute the corresponding eigenvector
c     (XR,XI).  It is assumed that the user has reduced the
c     matrix to tridiagonal form (see routines ATOTRI and TLR
c     for details).  The matrix A contains information about
c     the reduction to tridiagonal form.  AORG is the original
c     matrix, required in the residual computation for the
c     refinement.
c
c  Arguments:
c    N     -integer
c      N specifies the order of the matrix A.
c      N must be nonnegative.
c      not modified.
c
c    A     -double precision array, dimension (LDA,N)
c      A contains information about the reduction to
c      tridiagonal form.
c
c    LDA    -integer
c      The leading dimension of the array A.
c      LDA >= max(1,N).
c
c    AORG   -double precision array, dimension (LDA,N)
```

```
c      AORG contains the original matrix.
c
c    WR     -double precision
c      On entry, is the real part of the approximate
c      eigenvalue.
c      On exit, is the improved real part of the
c      approximate eigenvalue.
c
c    WI     -double precision
c      On entry, is the imaginary part of the
c      approximate eigenvalue.
c      On exit, is the improved imaginary part of the
c      approximate eigenvalue.
c
c    XR     -double precision array, dimension (N)
c      The real part of the computed eigenvector.
c
c    XI     -double precision array, dimension (N)
c      The imaginary part of the computed eigenvector.
c
c    IPIV   -integer array, dimension (N)
c      Contains the pivot sequence used during the
c      reduction to tridiagonal form.
c
c    W     -double precision array, dimension (N)
c      May contain information if a restart was
c      performed in the tridiagonal process, as
c      indicated by IFLAG.
c
c    IFLAG  -integer
c      Signals if a restart was required during reduction
c      to tridiagonal form. IFLAG = 1 signals a restart
c      was taken.
c
c    WORK   -double precision array, dimension (LDWORK,19)
c      Used for workspace.
c
c    LDWORK -integer
c            The leading dimension of the array WORK.
c            LDWORK >= max(1,N+1).
c
      SUBROUTINE APPLY1( N, Z, LDZ, XRE, XIM, IPVT, W, IFLAG )
c
c
```

```
c  Purpose:
c     This routine applies the accumulated elementary
c     transformations inv(Z) to the vector.  The matrix
c     Z is the product of the elementary transformations
c     and permutation matrices that tridiagonalize the
c     original matrix A.  The computation is of the form
c         inv(Z)*A*Z = T, where
c     inv(Z) = inv(Lr(n-2)) * Lc(n-2) * P(n-2) * inv(Lr(n-3)) *
c              Lc(n-3) * P(n-3) * ... * inv(Lr(1))*Lc(1)*P(1)
c     Each of the elementary transformations is stored as a row or
c     column of the supplied array in this routine.
c
c  Arguments:
c     N       -integer
c              N specifies the order of the matrix Z.
c              N must be nonnegative.
c              not modified.
c
c     Z       -double precision array, dimension (LDZ,N)
c              Z contains information about the reduction to
c              tridiagonal form.
c
c     LDZ     -integer
c              The leading dimension of the array Z.
c              LDZ >= max(1,N).
c
c     XRE     -double precision array, dimension (N)
c              The real part of the vector used in applying the
c              transformations.
c
c     XIM     -double precision array, dimension (N)
c              The imaginary part of the vector used in applying
c              the transformations.
c
c     IPIV    -integer array, dimension (N)
c              Contains the pivot sequence used during the reduction
c              to tridiagonal form.
c
c     W       -double precision array, dimension (N)
c              May contain information if a restart was performed
c              in the tridiagonal process, as flagged by IFLAG.
c
c     IFLAG   -integer
c              Signals if a restart was required during reduction
```

```
c              to tridiagonal form.  IFLAG = 1 signals that a
c              restart was taken.
c
      SUBROUTINE APPLY2( N, Z, LDZ, X, IPVT, W, IFLAG )
c
c
c  Purpose:
c     This routine is used to multiply the row vector (xre,xim)
c     by the accumulated elementary transformations stored in
c     the array Z.  See routine APPLY1 for further comments about Z.
c     This routine assumes that the vector is some column of the
c     identity matrix.
c
c  Arguments:
c     N       -integer
c              The number of rows and columns in the matrix Z.
c              N >= 0.
c
c     Z       -double precision array, dimension (LDZ,N)
c              Z contains information about the reduction to
c              tridiagonal form.
c
c     LDZ     -integer
c              The leading dimension of the array Z.
c              LDZ >= max(1,N).
c
c     X       -double precision array, dimension (N)
c              The vector used in applying the transformations.
c
c     IPIV    -integer array, dimension (N)
c              Contains the pivot sequence used during the
c              reduction to tridiagonal form.
c
c     W       -double precision array, dimension (N)
c              May contain information if a restart was performed
c              in the tridiagonal process, as flagged by IFLAG.
c
c     IFLAG   -integer
c              Signals if a restart was required during reduction to
c              tridiagonal form. IFLAG = 1 signals a restart was taken.
c
      SUBROUTINE APPLY3( N, Z, LDZ, XRE, XIM, IPVT, W, IFLAG )
c
c  Purpose:
```

```
c     This routine used to multiply the accumulated elementary
c     transformations stored in the array Z by the column vector
c     (XRE,XIM).  See the routine APPLY1 for comments about the
c      array Z.
c
c  Arguments:
c     N       -integer
c              The number of rows and columns in the matrix Z.
c              N >= 0.
c
c     Z       -double precision array, dimension (LDZ,N)
c              Z contains information about the reduction to
c              tridiagonal form.
c
c     LDZ     -integer
c              The leading dimension of the array Z.
c              LDZ >= max(1,N).
c
c     XRE     -double precision array, dimension (N)
c              The real part of the vector used in applying the
c              transformations.
c
c     XIM     -double precision array, dimension (N)
c              The imaginary part of the vector used in applying the
c              transformations.
c
c     IPIV    -integer array, dimension (N)
c              Contains the pivot sequence used during the reduction
c              to tridiagonal form.
c
c     W       -double precision array, dimension (N)
c              May contain information if a restart was performed in
c              the tridiagonal process, as flagged by IFLAG.
c
c     IFLAG   -integer
c              Signals if a restart was required during reduction to
c              tridiagonal form. IFLAG = 1 signals a restart was taken.
c
      SUBROUTINE CXDIV( AR, AI, BR, BI, CR, CI )
      SUBROUTINE CXMULT( AR, AI, BR, BI, CR, CI )
      DOUBLE PRECISION FUNCTION CXABS( AR, AI )
      INTEGER FUNCTION ICXMAX( N, XRE, XIM, INCX )
      SUBROUTINE CXCOPY( N, XRE, XIM, YRE, YIM )
      SUBROUTINE CXAXPY( N, ARE, AIM, XRE, XIM, YRE, YIM )
```

```
      DOUBLE PRECISION FUNCTION CXNRM2( N, XRE, XIM )
      SUBROUTINE CXDOTU( N, XRE, XIM, YRE, YIM, ZRE, ZIM )
      SUBROUTINE CXDOTC( N, XRE, XIM, YRE, YIM, ZRE, ZIM )
      SUBROUTINE CXSCL( N, ARE, AIM, XRE, XIM )
      SUBROUTINE CXDSCL( N, A, XRE, XIM )
      SUBROUTINE CXSQRT( XRE, XIM, YRE, YIM )
      DOUBLE PRECISION FUNCTION CXASUM( N, XRE, XIM )
c
c     The above routines are support routines only, provided since
c     double precision complex arithmetic is not portable in FORTRAN77.
c     CXDIV and CXMULT are complex division and multiplication;
c     CXABS returns the modulus of the supplied number;
c     ICXMAX returns the index of the complex entry of largest modulus
c     in the arrays XRE and XIM;
c     CXCOPY returns a copy of the supplied complex number; CXAXPY
c     is a complex version of SAXPY; CXNRM2 returns the two-norm of
c     the supplied complex array; CXDOTU sums the term-by-term
c     product of the two supplied complex vectors; CXDOTC computes
c     the complex inner-product of the two supplied vectors; CXSCL
c     scales the supplied complex vector (XRE,XIM) by the complex
c     number (ARE,AIM); CXDSCL scales (XRE,XIM) by the double
c     precision number A; CXSQRT is the complex square-root;
c     CXASUM computes the sum of the moduli of the supplied complex
c     vector.
c
      DOUBLE PRECISION FUNCTION RANDOM( )
c
c     Returns a pseudo-random number on the interval [0,1]
c
      SUBROUTINE CXGECO(ARE,AIM,LDA,N,IPVT,RCOND,ZRE,ZIM)
c
c  Purpose:
c     Complex version of Gaussian elimination with forward
c     and back solves, in which the real and imaginary parts
c     of the complex arrays are separate double precision
c     arrays, for portability.
c
c  Arguments:
c     ARE      -double precision array, dimension (LDA,N)
c     AIM      -double precision array, dimension (LDA,N)
c               These contain the real and imaginary parts of
c               the matrix A.
c     LDA      -integer
c               The leading dimension of the array A.
```

```
c                LDA >= max(1,N).
c     N         -integer
c                The number of rows and columns in the matrix A.
c                N >= 0.
c     IPVT      -integer array, dimension (N)
c                Contains the pivot sequence used during the
c                factorization.
c     RCOND     -double precision
c                Returns an estimate of the condition number of the
c                matrix A.
c     ZRE       -double precision, dimension (N)
c     ZIM       -double precision, dimension (N)
c                These contain the solution to the linear system.
c
      SUBROUTINE CXGEFA(ARE,AIM,LDA,N,IPVT,INFO)
c
c  Purpose:
c     This routine performs LU factorization on the matrix whose
c     real and imaginary entries are contained in ARE and AIM.
c
c  Arguments:
c     ARE       -double precision, dimension (LDA,N)
c     AIM       -double precision, dimension (LDA,N)
c                These contain the array A.
c     LDA       -integer
c                The leading dimension of the array A.
c     N         -integer
c                The number of rows and columns of the matrix A.
c     IPVT      -integer array, dimension (N)
c                Contains the pivot sequence used during the
c                factorization.
c     INFO      -integer
c                Contains a nonzero error condition on error.
c
      SUBROUTINE CXGESL(ARE,AIM,LDA,N,IPVT,BRE,BIM,JOB)
c
c  Purpose:
c     Solves the linear system A X = B, where the real and imaginary
c     parts of A and B are contained in (ARE,AIM) and (BRE,BIM).
c
c  Arguments:
c     ARE       -double precision, dimension (LDA,N)
c     AIM       -double precision, dimension (LDA,N)
c                Contain the real and imaginary parts of the matrix A.
```

```
c     LDA      -integer
c               Leading dimension of the array A.
c     N        -integer
c               Number of rows and columns of the matrix A.
c     IPVT     -integer, dimension (N)
c               Contains the pivot sequence used during the
c               factorization of A.
c     BRE      -double precision, dimension (N)
c     BIM      -double precision, dimension (N)
c               Contain the real and imaginary parts of the vector B.
c               On exit, contain the solution.
c     JOB      -integer
c               Determines whether to operate on A or A'.
c
      SUBROUTINE GTINIT( N, A, LDA, WR, WI, XRE, XIM, WORK, LDWORK )
c
c  Purpose:
c     This routine performs one step of inverse iteration with the
c     matrix from the tridiagonal step and the approximate eigenvalue.
c
c  Arguments:
c     N       -integer
c              The number of rows and columns in the matrix A.
c              N >= 0.
c
c     A       -double precision array, dimension (LDA,N)
c              A contains information about the reduction to
c              tridiagonal form.
c
c     LDA     -integer
c              The leading dimension of the array A.
c              LDA >= max(1,N).
c
c     WR      -double precision
c              The real part of the approximate eigenvalue.
c
c     WI      -double precision
c              The imaginary part of the approximate eigenvalue.
c
c     XRE     -double precision array, dimension (N)
c              On exit, contains the real part of the
c              computed eigenvector.
c
c     XIM     -double precision array, dimension (N)
```

```
c              On exit, contains the imaginary part of the
c              computed eigenvector.
c
c    WORK   -double precision array, dimension (LDWORK,19)
c      Used for workspace.
c
c    LDWORK -integer
c            The leading dimension of the array WORK.
c            LDWORK >= max(1,N+1).
c
      SUBROUTINE MLU( N, WORK, LDWORK )
c
c  Purpose:
c     Form the LU decomposition of a matrix of the form:
c
c    (    d    sup   col  )
c    (  sub  \  \     |   )
c    (     \  \  \    |   )
c    (      \  \  \   |   )
c    (       \  \     |   )
c    (        \  \    |   )
c    (       sub  d  col  )
c
c     On return the U part of the decomposition replaces d, sup, and col
c               and (-L) is stored in sub.
c
c  Arguments:
c     N      -integer
c             number of rows and columns in the above matrix.
c     WORK   -double precision array, dimension (LDWORK,N)
c             Contains sub, d, sup, anc col in packed form.
c     LDWORK -integer
c             Leading dimension of the array WORK.
c
      SUBROUTINE MSOL( N, WORK, LDWORK, XRE, XIM )
c
c  Purpose:
c     This routine solves a system based on the factored
c     matrix stored in WORK as returned by MLU.  The complex rhs
c     is in (XRE,XIM) and the solution is returned in (XRE,XIM).
c
c  Arguments:
c
c     N      -integer
```

```
c             Size of the matrix.
c
c     WORK   -double precision array, dimension (LDWORK,N)
c             Contains the matrix in packed, factored form.
c
c     LDWORK -integer
c             Leading dimension of the array WORK.
c
c     XRE    -double precision array, dimension (N)
c     XIM    -double precision array, dimension (N)
c             On entry, contain the right hand side, and on exit,
c             contain the solution.
c
c
      SUBROUTINE RESID( N, A, LDA, XRE, XIM, WR, WI, RRE, RIM, RNORM )
c
c  Purpose:
c     This routine computes the residual for the eigenpair
c     (wr,wi) and x with the matrix A.
c             r = (wr,wi)*x - A*x
c     and returns the vector r and the norm of r.
c
c  Arguments:
c     N       -integer
c             The number of rows and columns in the matrix A.
c             N >= 0.
c
c     A       -double precision array, dimension (LDA,N)
c             A contains the matrix.
c
c     LDA     -integer
c             The leading dimension of the array A.
c             LDA >= max(1,N).
c
c     XRE     -double precision array, dimension (N)
c             The real part of the computed eigenvector.
c
c     XIM     -double precision array, dimension (N)
c             The imaginary part of the computed eigenvector.
c
c     WR      -double precision
c             The real part of the computed eigenvalue.
c
c     WI      -double precision
```

```
c             The imaginary part of the computed eigenvalue.
c
c     RRE     -double precision array, dimension (N)
c             The real part of the computed residual vector.
c
c     RIM     -double precision array, dimension (N)
c             The imaginary part of the computed residual vector.
c
c     RNORM   -double precision
c             The norm of the residual (RRE, RIM).
c
      SUBROUTINE SHEMOR( N, WORK, LDWORK, IFLAG )
c
c  Purpose:
c     This routine applies the Sherman-Morrison update formula to
c     the matrix:
c
c             ( T     col )
c             ( v'     0  ).
c
c     Then solves the simpler matrix problem
c
c             ( T     col )
c             (        1  )
c
c     and transforms the solution back to the original matrix problem.
c     The tridiagonal matrix and col are stored in packed form in WORK.
c
c  Arguments:
c     N       -integer
c             The number of rows and columns in the original matrix A.
c             N >= 0.
c
c     WORK    -double precision array, dimension (LDWORK,19)
c             Is used for a work space.
c
c     LDWORK  -integer
c              Leading dimension of the array WORK.
c
c     IFLAG   -integer
c             Normal return is zero.
c             If the modified matrix is singular then IFLAG = 1 and
c             the process fails.
```

References

[1] J. J. Dongarra, C. B. Moler, and J. H. Wilkinson. Improving the accuracy of computed eigenvalues and eigenvectors. *SIAM J. Numer. Anal.*, 20:23–45, February 1983.

[2] J. G. F. Francis. The QR transformation – Part 2. *The Computer Journal*, 4:332 – 345, 1961.

[3] G. A. Geist. Reduction of a general matrix to tridiagonal form. Technical report, Oak Ridge National Laboratory, February 1989. ORNL/TM-10991.

[4] G. A. Geist, A. Lu, and E. L. Wachspress. Stabilized Gaussian reduction of an arbitrary matrix to tridiagonal form. Technical report, Oak Ridge National Laboratory, February 1989. ORNL/TM-11089.

[5] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, 1983.

[6] L. B. Rall. *Computational Solution of Nonlinear Operator Equations.* Wiley, 1969.

[7] H. Rutishauser. Solution of eigenvalue problems with the LR transformation. *Nat. Bur. Standards Appl. Math. Ser.*, 49:47–81, 1958.

[8] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garabow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines - EISPACK Guide.* Springer, Heidelberg, 1974.

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