NASA TECHNICAL NOTE

NASA TN D-7032

ALLMAT: A TSS/360 FORTRAN IV SUBROUTINE FOR EIGENVALUES AND EIGENVECTORS OF A GENERAL COMPLEX MATRIX

by Gale Fair

Lewis Research Center

Cleveland, Ohio 44135

1. Report No. NASA TN D-7032	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ALLMAT: A TSS/360 FORTRAN IV SUBROUTINE FOR EIGENVALUES AND EIGENVECTORS OF A GENERAL COMPLEX MATRIX		5. Report Date January 1971
		6. Performing Organization Code
7. Author(s) Gale Fair		8. Performing Organization Report No. E-5885
9. Performing Organization Name and Address Lewis Research Center National Aeronautics and Space Administration Cleveland, Ohio 44135		10. Work Unit No. 129-02
		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Note
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Subroutine ALLMAT is described and listed. ALLMAT computes the eigenvalues and eigenvectors of a general (non-Hermitian) complex matrix. The program uses the complex QR algorithm to compute eigenvalues and inverse iteration to compute eigenvectors. The user has the option of computing only the eigenvalues, if desired. An entry point EVDATA is available to provide the user with timing and accuracy information, as well as the number of iterations necessary for each eigenvalue and eigenvector.

17. Key Words (Suggested by Author(s)) Matrix algebra; Computer program, FORTRAN; Eigenvalue; Eigenvector; Hermitian matrix; QR Transformation; Inverst iteration, Wielandt	18. Distribution Statement Unclassified - unlimited		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 29	22. Price* $3.00

ALLMAT: A TSS/360 FORTRAN IV SUBROUTINE FOR EIGENVALUES

AND EIGENVECTORS OF A GENERAL COMPLEX MATRIX

by Gale Fair

Lewis Research Center

SUMMARY

A subroutine is described and listed that computes the eigenvalues and eigenvectors of a general (non-Hermitian) complex matrix. The program, ALLMAT, uses the complex QR algorithm to compute eigenvalues and inverse iteration to compute eigenvectors. The user has the option of computing only the eigenvalues, if desired. An entry point EVDATA is available to provide the user with timing and accuracy information, as well as the number of iterations necessary for each eigenvalue and eigenvector.

INTRODUCTION

Many areas of physics, mathematics, statistics, and engineering require the eigenvalues and eigenvectors of square matrices. This area of numerical analysis, sometimes called the algebraic eigenvalue problem, holds a place that is as important as the more familiar areas such as numerical integration, curve fitting, and numerical integration of differential equations. A general library of subroutines for a computer installation is commonly limited to one such program, and quite often this one subroutine is of only limited applicability.

The ideal subroutine for the algebraic eigenvalue problem should have many features: it should be fast and accurate; it should give a matrix of eigenvectors that are linearly independent; it should be capable of computing only eigenvalues at a corresponding increase in speed; it should have minimal storage requirements; and it must be able to treat all matrices, real or complex, symmetric or nonsymmetric, regardless of the condition of the matrix. Unfortunately, probably no such procedure exists. The usefulness of any subroutine may be judged on the basis of how many of these criteria are fulfilled, as balanced against the needs of the individual user.

There are many different techniques for diagonalizing a square matrix. The treatise by Wilkinson (ref. 1) is evidence for this. Reference 1 describes the state-of-the-art for the algebraic eigenvalue problem as of 1965. Typically, one chooses a particular method because he believes that his matrix has some feature that requires special handling, because a subroutine is conveniently at hand, or because he knows only that one method. Two of the most commonly used procedures are the power method and the Jacobi transformation.

The power method is a special-purpose procedure that computes the largest eigenvalue of a matrix by the formation of a sequence of powers of the matrix acting upon an arbitrary vector. This procedure is useful for the computation of a few eigenvalues (the largest in magnitude) and their eigenvectors. The computation of a full set of eigenvalues and eigenvectors is both time consuming and inaccurate.

The Jacobi transformation, as it applies to a complex Hermitian matrix, consists of a sequence of unitary transformations that diagonalize 2×2 submatrices of the full matrix. This procedure generates the eigenvectors along with the eigenvalues and is particularly useful when the eigenvectors are required to be orthogonal to a high degree of accuracy. The limitations of the Jacobi transformation are that the accuracy of the eigenvectors is usually limited, and as yet no extension to nonsymmetric or non-Hermitian matrices has been made. The most common computer library subroutine for the algebraic eigenvalue problem is a real-symmetric version of the Jacobi method (ref. 2).

For a general (i.e., nonsymmetric or non-Hermitian) matrix, two procedures have been derived to compute the eigenvalues and eigenvectors, respectively. The input matrix is reduced to a Hessenberg form (ref. 1), and the QR transformation of Francis (refs. 3 and 4) is used to compute the eigenvalues. With a knowledge of the eigenvalues, the Wielandt inverse iteration method (ref. 1) generates the eigenvectors. The QR transformation and inverse iteration appear to be the best currently available for their respective tasks (ref. 1) in terms of accuracy and speed. This combined procedure has been coded at the Oak Ridge National Laboratory (ref. 5) for an IBM 360/50 using the H-level FORTRAN compiler and COMPLEX*16 arithmetic. This program was used as the basis for the subroutine to be described in this report.

In order to make the subroutine as general as possible, some modifications and additions were made to the ORNL program, as follows. The fact that for some matrices the Hessenberg form may be decomposed into disjoint submatrices is incorporated in both the QR transformation and the inverse iteration to reduce computational time. A perturbation method is used to obtain linearly independent eigenvectors when eigenvalues are either degenerate or very nearly the same value (meaning that the matrix itself may be ill-conditioned (ref. 1)). An auxiliary entry point is provided to give the user in-

formation about the number of iterations required, timing data (measured as central processor or CPU time elapsed in the computation), and error data for the resulting eigenvalues and eigenvectors. Finally, a flag has been provided to allow the user to compute only the eigenvalues, with the use of the QR transformation. The relative contribution made by the present work is seen from the observation that approximately 60 percent of the coding of the current form of the subroutine ALLMAT is the ORNL coding while the remaining 40 percent is new.

The end result of the work described here is a subroutine for the IBM/360 to compute the eigenvalues and eigenvectors of a square matrix. Certainly, this is not meant to be the final work in such procedures, the algebraic eigenvalue problem is an area of extensive research in numerical analysis. On the other hand, this subroutine does satisfy most of the criteria mentioned earlier for the ideal subroutine, at least to some degree. The criterion that is least satisfied is minimal storage. Because ALLMAT is written with COMPLEX*16 arithmetic and has some large scratch-pad arrays, the subroutine uses a large amount of storage. On a TSS/360 system this storage requirement is not a basic limitation on the subroutine, but it does imply that the CPU time is affected.

This brief mention of storage requirements is an opportunity to interpose a slight warning to the prospective user of ALLMAT. If only a small number of the (largest) eigenvalues of a matrix are desired, the power method is more efficient than ALLMAT. For a real, symmetric matrix, a problem that requires eigenvectors along with the eigenvalues would be better suited to a real Jacobi subroutine. On the other hand, for the computation of eigenvalues alone, or for the eigenvalues and eigenvectors of a real, nonsymmetric matrix or for a complex matrix, ALLMAT seems to be the best choice, at this time.

The next section of this report describes schematically the construction of the subroutine ALLMAT. This includes the information necessary for a programmer to use ALLMAT. Also included are brief descriptions of the mathematical procedures used in ALLMAT. The following section discusses the special features that have been incorporated in ALLMAT, including a description of the subsidiary ENTRY EVDATA that provides timing and accuracy information for the user. Finally, a number of test matrices are used as examples for ALLMAT. These examples give an indication of running times and accuracy obtainable with the program, even with some ill-conditioned input matrices. A FORTRAN listing of ALLMAT is given in the appendix.

This report is intended to be used as a user's manual for the subroutine ALLMAT, and as such it described the call vector for the subroutine and the rules for usage. In addition, enough information is provided the prospective user to allow an intelligent application of this program to his particular problem. The prospective user should not

apply this program to his problem without some understanding of the numerical methods involved and of the construction of the subroutine.

GENERAL CONSTRUCTION

Usage

The information to be discussed in this section is aimed at explaining the program as a FORTRAN subroutine, along with a description of the ENTRY EVDATA.

The user's access is through the statement (see the appendix for the complete FORTRAN listing of the subroutine):

$$\text{CALL ALLMAT (AA, LAMBDA, M, MM, EVECT, NCAL)}$$

where

AA	input COMPLEX*16 matrix, of dimension $M \leq MM$. Upon return from ALLMAT, i^{th} column of AA is i^{th} eigenvector, corresponding to i^{th} eigenvalue.
LAMBDA	COMPLEX*16 vector of length M that contains eigenvalues upon return from ALLMAT.
M	actual dimension of input matrix AA.
MM	dimension of AA as it appears in a dimension statement in the calling program. MM is the upper bound for the size of matrices used. As ALLMAT is currently written, MM must be no greater than 50.
EVECT	a logical switch. If EVECT = .TRUE., the eigenvectors of AA are calculated, and returned in the matrix AA. If EVECT = .FALSE., no eigenvectors are calculated and AA contains no useful information upon return from ALLMAT.
NCAL	number of eigenvalues successfully computed by ALLMAT. If NCAL < M some attempts of the QR transformation did not converge within 10 iterations. The value of the element of LAMBDA that corresponds to this eigenvalue has been set to zero by ALLMAT.

In addition to the primary entry point, a secondary ENTRY EVDATA is available to give the user information on the CPU time taken for the eigenvalue and eigenvector procedures. Also available are the number of QR iterations required for each eigen-

4

value, the number of inverse iterations required for each eigenvector, and the Euclidean norms of the residual vectors. A more complete description of these quantities is given later. The usage for this optional entry point is

CALL EVDATA (ITS, KTS, NCO, MCO, RNORM)

where

ITS   elapsed time for QR transformation for eigenvalues, including time to re-
           duce to upper Hessenberg form. ITS is an integer, in microminutes.

KTS   elapsed time for inverse iteration for eigenvectors. Does not include time
           represented by ITS. Also an integer in microminutes.

NCO   an integer vector of dimension MM that has as its ith element the number
           of QR iterations for the ith eigenvalue. NCO (i) ≤ 10. If NCO (i) = 0,
           this eigenvalue was obtained along with another, no separate QR iteration
           was required. If NCO (i) < 0, no convergence was obtained for this eigen-
           value within ten QR iterations.

MCO   integer vector of dimension MM that has as its ith element the number of
           inverse iterations necessary to obtain the ith eigenvector. MCO (i) ≤ 10.

RNORM   REAL*8 vector of the norms of the residual vectors of AA. See section
           SPECIAL FEATURES OF ALLMAT for a more complete description.
           RNORM also has a dimension MM.

As an example of the usage of ALLMAT, consider a 6×6 complex matrix AA that is to be diagonalized. Let us assume that the TYPE statement in the calling program that specifies the dimensions of AA and LAMBDA has the form

COMPLEX*16 AA (10,10), LAMBDA(10)

The arrays have been overdimensioned for more generality. Let us further assume that eigenvectors are desired from ALLMAT, so that EVECT has been assigned a value .TRUE.. Then the call to ALLMAT is

CALL ALLMAT (AA, LAMBDA, 6, 10, EVECT, NCAL)

Upon return from ALLMAT the integer variable NCAL contains the number of eigen-values that have been successfully computed by ALLMAT. The ith column of AA (I.E.

AA (1,1) to AA (6,1)) contains the ith eigenvector, corresponding to the eigenvalue LAMBDA (1).

If the timing and error information provided by EVDATA are desired by the user, then the statement

CALL EVDATA (ITS, KTS, NCO, MCO, RNORM)

is used, where NCO, MCO, and RNORM have been dimensioned at least six in the calling program. The conversion from ITS or KTS (in microminutes) to milliseconds is obtained by multiplying either integer by 0.06 and assigning the result to a floating-point variable.

QR TRANSFORMATION

The basis of the QR transformation is a theorem by Francis that states any non-singular matrix A has a unique decomposition into the product of a unitary matrix Q and an upper triangular matrix R (ref. 3), or

$$A = QR$$

The QR algorithm consists of forming a sequence of matrices similar to $A (=A_{(1)})$ such that

$$A_{(K)} = Q_{(K)} R_{(K)}$$

and then

$$A_{(K+1)} = R_{(K)} Q_{(K)}$$

where $A_{(K)}$ is the form of the matrix after the Kth decomposition. Francis (ref. 3) shows that this sequence of matrices has as its limit an upper triangular matrix, the diagonal elements of which are the eigenvalues of the original matrix A. Furthermore, even if the original matrix is singular, the algorithm still gives convergence to a unique triangular matrix, even though some of the intermediate Q and R may not be unique.

A full description of the QR transformation is certainly not relevant to this report. A detailed discussion of the convergence properties and the error analysis of the QR algorithm is given in references 1, 3, and 4. It is sufficient to note for our purpose

6

that the QR algorithm is an extremely stable, rapidly converging procedure to calculate the eigenvalues of a general matrix (ref. 1). The version of the QR transformation that is part of ALLMAT, one that includes origin shifts to accelerate convergence, is powerful enough to satisfy nearly all of the needs of the average user.
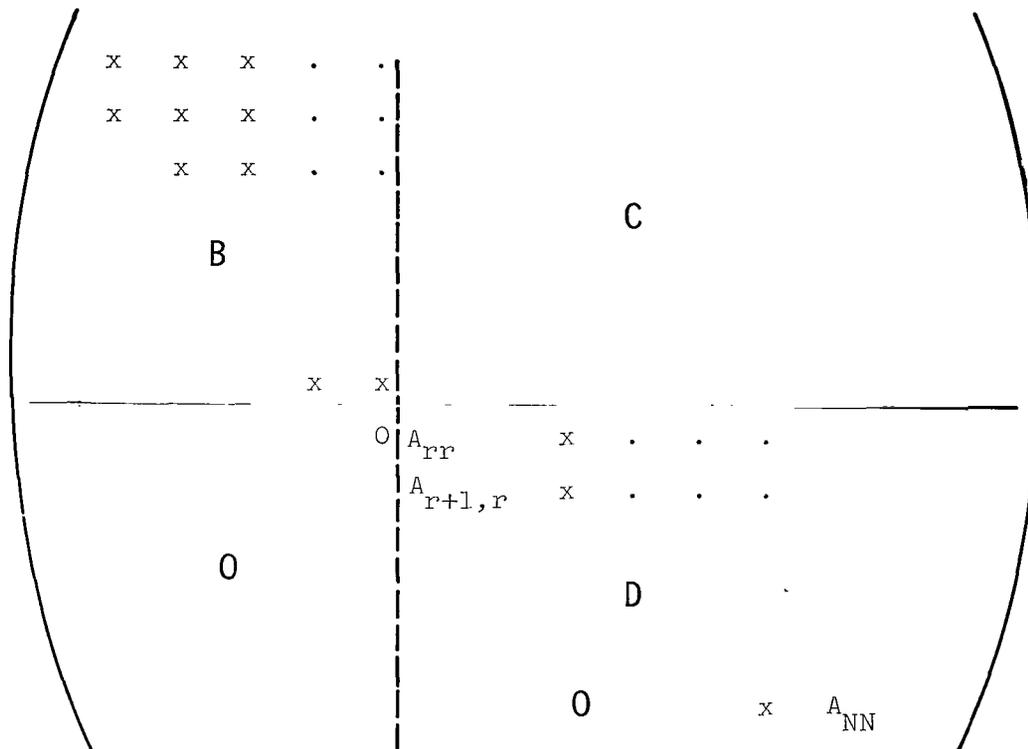
There is one unusual feature of the standard way in which the QR algorithm is employed that the prospective user should be aware of. A preliminary step in any implementation of the QR transformation is the reduction of the input matrix to Hessenberg form. An upper Hessenberg form (i.e., $A_{ij} = 0$ if $i > j + 1$) is used in ALLMAT. The reduction is accomplished by a sequence of elementary transformations (ref. 1). The elements of these elementary transformations are stored in the unused portion of A (the lower subtriangle of A) and in the integer vector JNT. This information is used at the end of the inverse iteration to recover the eigenvectors of the original matrix from the eigenvectors of the Hessenberg matrix. The point of caution for the user is that the working matrix for the subroutine is the Hessenberg form, which in general bears no simple relation to the input matrix. Thus, if the user attempts to debug this subroutine at an intermediate stage, the relation between the Hessenberg form and the original form must ge kept in mind.

The advantage of using the Hessenberg form is apparent in the time needed to complete the computation of the eigenvalues. Most methods that operate on the entire input matrix, such as the Jacobi method, require a number of operations that is approximately $30N^3$ (ref. 2), where N is the order of the matrix. The reduction to Hessenberg form is a one-pass operation and requires αN^3 operations, where α is of order unity. The QR algorithm applied to the Hessenberg form only requires something of the order of N^2 operations. One interesting result of this is the observation (ref. 5) that under many conditions the QR transform produces eigenvalues in less time than the Jacobi transformation.

Inverse Iteration

The basis of the inverse iteration procedure is the observation that, if λ is an eigenvalue of the matrix A, the quantity $(A - \lambda I)$, where I is the unit matrix, will be singular. Thus, if λ is a good approximation to an eigenvalue of A, the matrix $(A - \lambda I)^{-1}$ may be iterated to obtain an approximation Y to the eigenvector X. The iteration process is carried out until after the Kth iteration the norm of the iterated vector, $(A - \lambda I)^{-1} Y_K$ is greater than some preselected value (see the appendix). This procedure is equivalent to the power method, but in inverse powers of the matrix $(A - \lambda I)$. The speed with which this iteration produces an eigenvector depends on the accuracy of the estimate for the eigenvalue, but rarely does this procedure, combined

with the QR algorithm, require more than 2 iterations to produce eigenvectors to at least six or seven place accuracy. Again, the interested user is referred to Wilkinson (ref. 1) for a complete description of the method and the error analysis.

SPECIAL FEATURES OF ALLMAT

As mentioned in the introduction, the basic elements of ALLMAT, the reduction to Hessenberg form, the QR transformation, and the inverse iteration, are taken from an ORNL subroutine (ref. 5). There are several features that have been added to this basic program to either add effectiveness to the program or provide timing and accuracy information to the user. These special features will be discussed in this section, more or less in the order that they appear in the program.

Decomposed Hessenberg Form

The reduction of the original matrix to Hessenberg form is a procedure that decreases the number of operations necessary for the QR algorithm. In a large number of cases the nature of the Hessenberg form allows further simplifications. To illustrate this, sketch (a) shows an upper Hessenberg matrix, of order N. The X's in the sketch

(a)

indicate matrix elements, generally nonzero, whose values are unimportant. Now let one of the subdiagonal elements vanish, for example $A(R, R - 1) = 0$. Then the Hessenberg matrix may be decomposed into four submatrices as shown in sketch (b).

$$
\begin{pmatrix}
\begin{array}{cccccc}
x & x & x & \cdot & \cdot & \\
x & x & x & \cdot & \cdot & \\
 & x & x & \cdot & \cdot & \\
 & & & & & \\
 & & B & & & C \\
 & & & & & \\
 & & & x & x & \\
\hline
 & & & & 0 & A_{rr} \quad x \cdot \cdot \cdot \\
 & & & & & A_{r+1,r} \quad x \cdot \cdot \cdot \\
 & & 0 & & & D \\
 & & & & & 0 \qquad x \quad A_{NN}
\end{array}
\end{pmatrix}
$$

(b)

The submatrices B and D are upper Hessenberg matrices of order $R - 1$ and $N - R + 1$, respectively. Submatrix C is a nonzero matrix with $N - R + 1$ columns and $R - 1$ rows. The remaining submatrix of this partition of A is entirely filled with zeroes.

The result of this decomposition is that the problem of finding the eigenvalues of B and D becomes entire disjoint; that is, the eigenvalues of B and D, collectively, are the eigenvalues of A. The submatrix C plays no part in the eigenvalue problem. Thus, instead of the solution of a single matrix of order N, the problem has been reduced to the solution of two matrices, of order $N - R + 1$ and $R - 1$. Since $N^2 > (N - R + 1)^2 + (R - 1)^2$ for $N \geq 3$ and R that is not trivial, this decomposition implies a significant reduction in the total number of operations in the QR transforma-

tion. For many input matrices, particularly those matrices that are sparse, a number of such decompositions may be performed and the gain in machine time is important.

This decomposition is not as important for the calculation of the eigenvectors, although some improvement is made. The eigenvectors corresponding to eigenvalues of D (see sketch (b)) depend upon the submatrices B and C, so that the entire matrix must be used in the inverse iteration procedure. The eigenvectors corresponding to the eigenvalues of B, on the other hand, do not require matrices C or D, so that only B is used in the inverse iteration. Thus, some advantage is gained from the decomposition for the calculation of the eigenvectors. On the whole, though, the main advantage of the decomposition enters in the QR transformation.

Perturbation of Close Eigenvalues

One difficulty with the inverse iteration method arises when two or more eigenvalues are very nearly the same. Since every calculated eigenvalue differs from the "true" eigenvalue by an amount that depends on many factors, these eigenvalues may not produce linearly independent eigenvectors. The way chosen to resolve this accidental degeneracy was to perturb each successive close eigenvalue by an amount small enough to not disturb the convergence of the iterative procedure, but large enough to resolve the eigenvectors into linearly independent vectors (ref. 1). The choice of the perturbation, EPSIL, is arbitrary and a better choice could be made for particular types of matrices.

Since the existence of close but distinct eigenvalues implies that the matrix may be ill-conditioned (ref. 1), the accuracy of the calculated eigenvectors will be in doubt. In this sense, the use of a perturbation to separate the eigenvalues is an attempt to recover some useful information from a badly posed problem. Thus, for most matrices encountered, the existence of close but distinct eigenvalues should be rare. The occurrence of multiple eigenvalues is more common.

Multiple Eigenvalues

The existence of a set of multiple eigenvalue is a not uncommon occurrence in physical problems. The existence of such a set implies that there is a subspace of eigenvectors that one desires the basis vectors of. In this situation the perturbation is of some help. If, by the process of perturbing the degenerate eigenvalues within the inverse iteration process, one can obtain a set of distinct eigenvectors, even if they are not linearly independent, then there is a standard solution to the problem of determining

the basis vectors. For this purpose ALLMAT takes the set of distinct eigenvectors produced by the perturbation technique just discussed and uses a Gram-Schmidt (ref. 1) orthogonalization procedure to give a set of linearly independent eigenvectors. Since the Gram-Schmidt process involves taking the differences of nearly equal numbers in many cases, the accuracy of such a procedure is less than the accuracy of an inverse iteration vector for a distinct eigenvalue. Again, however, this represents an attempt to salvage as much information as one can from an undesirable situation. In practice, as shall be seen in the section TESTS, the results of this perturbation and orthogonalization procedure are good.

ENTRY EVDATA

The remaining special feature of ALLMAT is represented by the secondary entry point, EVDATA, as discussed in general construction. A typical user of an installation-supplied mathematical subroutine is usually blissfully unaware of any error considerations for his problem. Since the accuracy of any matrix eigenvalue evaluation strongly depends upon the properties of the input matrix, ignoring error information is equivalent to shutting one's eyes to avoid an oncoming truck. Additionally, since some eigenvalues and eigenvectors may in fact be absent due to nonconvergence either in QR or inverse iteration, the information provided by EVDATA is important to a user. The use of the TSS/FORTRAN multiple-data set capability means that this information is readily available to the user, without so much as the disturbance of an artistic output format.

The information available in EVDATA includes the number of iterations, the CPU time elapsed for the eigenvalue and the eigenvector computations, and an error estimate for each eigenvalue-eigenvector pair. The timing and counting variable provided in EVDATA were discussed sufficiently under usage, but the error information requires some further comment.

If λ and X are an exact eigenvalue and an exact eigenvector of the matrix A, then the vector $AX - \lambda X$ will be identically zero. Since neither λ nor X can ever be computed exactly, this vector ($AX - \lambda X$), called the residual vector, will be nonzero. The magnitude of this vector is then a measure of the error in λ and X. The length of a vector, as used in ALLMAT, is the Euclidean norm, $||X|| = \left(\sum |x(i)|^2 \right)^{1/2}$. The vector RNORM of EVDATA contains the norm of the residual vector for each eigenvalue-eigenvector pair, scaled to the Euclidean norm of the input matrix.

The data entry point EVDATA may be used even if no eigenvectors are computed (i.e., if EVECT = .FALSE.) In this case only ITS and NCO contain meaningful values.

TESTS

Seven matrices were chosen as examples for ALLMAT. The dimensions of these matrices vary from four to 19. All but two matrices are real but not symmetric, one of the remaining matrices is Hermitian, and the final example matrix is complex, but not Hermitian. Some of these matrices were chosen to illustrate ill-conditioning of one type or another. Since the numerical values of the eigenvectors are not of general use, they are not displayed.

Matrix 1

$$A_1 = \begin{pmatrix} 0.1 & -0.7 & -0.4 & -0.5 \\ -0.5 & 0.2 & -0.1 & -0.2 \\ 0.4 & 0.5 & 0.5 & 0.7 \\ 0.1 & 0.2 & 0.5 & 0.4 \end{pmatrix}$$

This real, but unsymmetric, matrix of order four has the exact eigenvalues 0.9, 0.6, -0.3, and 0. In addition, the computed eigenvalue corresponding to 0. is 0.14E-16. The information available from EVDATA on this test includes:

Eigenvalue	Number of QR iterations	Number of inverse iterations	RNORM
-0.3	8	3	0.65E-16
.14E-16	1	3	.43E-16
.6	1	3	.16E-16
.9	1	3	.91E-16

The total time for the QR transformation, including the initial reduction to Hessenberg form was 0.053 second, and the time for the inverse iteration was 0.046 second.

Matrix 2

$$A_2 = \begin{pmatrix} 0.25000025 & -1.0 & -0.49999975 & -0.99999975 \\ -0.50000050 & 0.5 & 0.24999950 & 0.25000025 \\ 1.00000025 & 1.25 & 1.00000025 & 1.25000025 \\ -0.4999975 & -0.25 & 0.25000025 & 0.50000025 \end{pmatrix}$$

This matrix has eigenvectors identical to those of matrix 1, but has a different set of eigenvalues. The exact eigenvalues of A_2 are given in the following table:

Eigenvalue	Number of QR iterations	Number of inverse iterations	RNORM
1.5	2	3	0.68E-16
.75000075	4	3	.10E-15
.75	0	3	.46E-14
-.75	1	3	.64E-15

The closeness of the second and third eigenvalues hint at some error problems with the eigenvectors. The time for QR transformation was 0.020 second, and the time for the inverse iteration was 0.090 second. The difficulties anticipated from the closeness of the eigenvalues are evidenced in the degradation of the third value of RNORM in the table.

Matrix 3

$$A_3 = \begin{pmatrix} 0.009 & 5.00101 & -8.999 & 3.999 \\ -0.001 & 5.01101 & -8.999 & 3.999 \\ -0.001 & 4.91101 & -8.899 & 3.999 \\ -0.001 & 4.96101 & -8.999 & 4.049 \end{pmatrix}$$

This matrix A_3 is an example of an ill conditioned matrix, in contrast with the previous example. Here, A_2 had two nearly alike eigenvalues even though the matrix is not mathematically ill conditioned (ref. 1).

Eigenvalue	Number of QR iterations	Number of inverse iterations	RNORM
0.01	3	3	0.25E-16
.01001	3	3	.14E-16
.1	3	3	.18E-16
.05	1	3	.23E-16

The time for QR transformation was 0.038 second; that for inverse iteration was 0.025 second. Apparently, the ill conditioning did not effect the inverse iterations, as all values of RNORM are satisfactory.

Matrix 4

$$A_4 = \begin{pmatrix} 6. & 5. & 4. & 3. & 2. & 1. \\ 1. & -2. & 1. & 6. & 3. & 2. \\ 2. & 3. & 2. & -2. & 4. & 3. \\ 3. & 1. & -3. & -1. & 5. & 5. \\ 4. & -4. & 2. & 0. & 1. & 4. \\ 5. & 0. & 1. & 3. & 6. & 6. \end{pmatrix}$$

Unlike the first three test matrices, A_4 has a pair of complex eigenvalues.

Eigenvalue	Number of QR iterations	Number of inverse iterations	RNORM
3.0929	6	3	0.78E-16
.1772+.95E-16i	6	3	.24E-15
.42295+4.3954i	5	3	.11E-15
.42295-4.3954i	4	3	.17E-15
15.247+.11E-14i	1	3	.56E-15
-7.3630	1	3	.47E-15

14

The time for QR transformation was 0.142 second; that for inverse iteration was 0.314 second. The imaginary part of the sum of the eigenvalues (which should be 0.) is 0.355E-14.

Matrix 5

This test matrix is a 19 by 19 real, unsymmetric matrix given by Francis (ref. 4) to demonstrate the QR transformation. The matrix is too complicated to list here, but the error information is informative. The time to produce the eigenvalues was 2 seconds, and the time to calculate the 19 eigenvectors was 22 seconds. Although this time is large when compared with the previous examples, it is quite reasonable when compared with other methods (ref. 5). Even with a matrix of this order, the residual vectors all had norms less than 1.E-16.

Matrix 6

```
        0.     -.913761031i  0.      0.       0.    0.           0.          0.    0.          0.
     .913761031i   0.        0.      0.       0.    0.           0.          0.    0.          0.
        0.        0.         0.      0.       0.    0.           0.          0.    0.          0.
        0.        0.         0.  -.38821306i  0.    0.      -.14433376i      0.    0.    -.946740090i
A    = 0.        0.    .38821306i   0.        0. -.14433376i      0.         0. -.946740090i    0.
 6      0.        0.         0.      0.        0.    0.           0.          0.    0.          0.
        0.        0.         0.  .14433376i   0.    0.     -1.021406483i     0.    0.     .03966810i
        0.        0.    .14433376i   0.   1.021406483i  0.       0.      -.03966810i  0.         0.
        0.        0.         0.  .946740090i  0.    0.      .03966810i       0.    0.     .747055401i
        0.        0.    .946740090i  0.   -.03966810i   0.       0.      -.747055401i  0.         0.
```

This matrix has several features that make it useful as an example. Each nonzero element of A_6 is a purely imaginary number and, in addition, A_6 is Hermitian. Thus, the eigenvalues of A_6 are real and, since the trace of A_6 vanishes, the eigenvalues occur in positive-negative pairs. There is a pair of degenerate eigenvalues with the value 0, so that the orthogonalization procedure must be used to obtain the eigenvectors. Finally, A_6 is sufficiently sparse that the decomposition of the Hessenberg form is effective in reducing the time required for the computations.

Eigenvalue	Number of QR iterations	Number of inverse iterations	RNORM
-0.91376103	3	3	0.26E-16
.91376103	3	3	.16E-16
-1.03872417	7	3	.31E-15
-.38452612	6	3	.96E-16
.38452612	5	3	.55E-15
1.03872417	4	3	.74E-15
-1.53711192	1	3	.15E-14
1.53711192	1	3	.21E-14
0	3	3	0
0	1	3	0

The time for QR transformation was 1.07 second; that for inverse iteration was 1.50 second. The eigenvalues appear in this table in the order in which they are calculated by the QR algorithm. Since the reduction to Hessenberg form and the use of the decomposed Hessenberg form rearrange the matrix, the eigenvalues are not computed in pairs, necessarily. This same effect caused the QR routine to take three iterations to compute a zero eigenvalue. The degenerate eigenvalues caused no loss of accuracy in the computation of the eigenvectors. Furthermore, the sum of the eigenvalues is purely imaginary, and has the magnitude 0.4E-14, reflecting the zero trace of A_6.

Matrix 7

The final example matrix was generated from matrix A_4 by taking each element of this 6 by 6 real, nonsymmetric matrix and multiplying by the imaginary unit i. The result, A_7, is a complex non-Hermitian matrix whose eigenvalues are the eigenvalues

Eigenvalue	Number of QR iterations	Number of inverse iterations	RNORM
3.0929i	6	3	0.77E-16
.95E-16+.1772i	6	3	.28E-15
4.3954+.42295i	5	3	.10E-15
-4.3954+.42295i	4	3	.16E-15
.44E-15-7.3630i	1	3	.48E-15
15.247i	1	3	.52E-15

of A_4 multiplied by i. The Qr transformation time was 0.140 second, and the inverse iteration time was 0.319 second.

A comparison of the results indicated in the preceding table with the results for example A_4 shows that ALLMAT handles the non-Hermitian form with comparable speed, at no loss of accuracy in the eigenvalues and eigenvectors. The sum of the eigenvalues is 0.31E-14+12.i.

These seven examples were chosen to be representative of the application of ALLMAT. Some matrices $(A_1, A_4, A_5,$ and $A_7)$ pose no particular problems, while the remaining $(A_2, A_3,$ and $A_6)$ were included to demonstrate one of more special characteristics of the program. It is seen from the results given above that ALLMAT had no difficulty with any of these test matrices. The norm of the residual vectors is typically less than 1.E-15, and all computed eigenvalues that were also known exactly were in agreement to at least 14 places. At no point did either the QR algorithm or the inverse iteration fail to give convergence within the allotted limit of 10 iterations. In fact, only once did the inverse iteration procedure require more than three iterations to satisfy the convergence criterion.

CONCLUDING REMARKS

This report is intended to be a user's guide for the prospective user of ALLMAT. The information presented here about the construction of ALLMAT should be considered a minimum for the use of this matrix eigenvector program. No program of the complexity of ALLMAT should be used without some understanding of the basic algorithms involved. Certainly, though, most users will apply ALLMAT without consideration of even the simplified discussion presented here. For these users the entry point EVDATA should be required usage as an indicator when ALLMAT does fail on a matrix.

Experience has shown that two inverse iterations are usually enough to give an eigenvector correct to sufficient accuracy. The current version of ALLMAT, however, iterates until the norm of the iterated vector, $(A - \lambda I)^{-1} X$, is greater than 1.E40. If computing time is at a premium, this criterion can be easily changed to a test on the number of iterations. The current limitation on ALLMAT is to matrices of dimension no larger than 50. This restriction may also be changed easily.

ALLMAT was designed to be a general purpose matrix eigenvalue and eigenvector subroutine. Almost any matrix, including the most general case of a complex, non-Hermitian matrix, is amenable to diagonalization by ALLMAT. Furthermore, timing

test (ref. 5) indicate that the QR transform may be preferred to the Jacobi method for the eigenvalues of real and symmetric matrices.

Lewis Research Center,
National Aeronautics and Space Administration,
Cleveland, Ohio, September 16, 1970,
129-02.

```
0000100        SUBROUTINE ALLMAT (AA,LAMBDA,M,MM,EVECT,NCAL)
0000200        IMPLICIT REAL*8 (A-H,O-Z)
0000300        COMPLEX*16 AA(MM,MM)
0000400C
0000500C       IF THE USER REQUIRES DIMENSION LARGER THAN 50, THE
0000600C        DIMENSIONS IN THE 2 LINES FOLLOWING THIS COMMENT MUST
0000700C       BE CHANGED FROM 50 TO A SIZE TO SUIT THE USER.
0000800C
0000900        COMPLEX*16 A(50,50),H(50,50),HL(50,50),LAMBDA(MM)
0001000        COMPLEX*16 VECT(50),MULT(50),SHIFT(3),TEMP,SINT,COST,TEMP1,TEMP2
0001100        COMPLEX*16 EIG,CONJI
0001200        LOGICAL EVECT,INTH(50)
0001300        DIMENSION NCOUNT(50),MCOUNT(50)
0001400        INTEGER JNT(50),R,RP1,RP2
0001500        DO 1000 J=1,M
0001600        DO 1000 I=1,M
0001700 1000   A(I,J) = AA(I,J)
0001800        CALL CPUTIM(ITIM)
0001900        ITSUM = 0
0002000        KTSUM = 0
0002100        CONJI = (0.,-1.)
0002200C
0002300C       THE CONSTANT EPSIL DETERMINES THE CONVERGENCE OF THE
0002400C       QR ALGORITHM, AND ALSO IS THE PERTURBATION PARAMETER
0002500C       FOR THE INVERSE ITERATION.
0002600C
0002700        EPSIL = 1.0D-12
0002800C
0002900C       THE CONSTANT EPSMAX DETERMINES THE CONVERGENCE OF THE
0003000C       INVERSE ITERATION.  THIS NUMBER IS THE LOGARITHM OF
0003100C       NORM OF THE ITERATED EIGENVECTOR THAT IS SUFFICIENT
0003200C       FOR CONVERGENCE.
0003300C
0003400        EPSMAX = 40.
0003500        NSTOP = M
0003600        N = NSTOP
0003700        NSTART = 1
0003800        MN1 = 1
0003900        NCAL = 0
0004000        IF (N.NE.1) GOTO 1
0004100        LAMBDA(1) = A(1,1)
0004200        A(1,1) = 1.0
0004300        GO TO 92
```

```
0004400      1 ICOUNT = 1
0004500        SHIFT(1) = 0.
0004600        IF (N.NE.2) GOTO 4
0004700      2 NSP1 = NSTART + 1
0004800        TEMP = (A(NSTART,NSTART)+A(NSP1,NSP1)+CDSQRT -
0004900      1((A(NSTART,NSTART)+A(NSP1,NSP1))**2-4.*(A(NSP1, -'
0005000      1NSP1)*A(NSTART,NSTART)-A(NSP1,NSTART)*A(NSTART, -'
0005100      1NSP1))))/2.
0005200        RELTEM = TEMP
0005300        AMGTEM = CONJI*TEMP
0005400        IF (RELTEM.NE.0..OR.AMGTEM.NE.0.) GOTO 3
0005500        LAMBDA(NSTOP) = SHIFT(1)
0005600        LAMBDA(MN1) = A(NSTART,NSTART)+A(NSP1,NSP1)+SHIFT(1)
0005700        NCOUNT(NSTOP) = ICOUNT
0005800        NCOUNT(MN1) = ICOUNT
0005900        GO TO 37
0006000      3 LAMBDA(NSTOP) = TEMP + SHIFT(1)
0006100        LAMBDA(MN1)=(A(NSTART,NSTART)*A(NSP1,NSP1)- -'
0006200      1A(NSP1,NSTART)*A(NSTART,NSP1))/(LAMBDA(NSTOP) -'
0006300      2-SHIFT(1))+SHIFT(1)
0006400        NCOUNT(NSTOP) = ICOUNT
0006500        NCOUNT(MN1) = ICOUNT
0006600        ICOUNT = 1
0006700        GO TO 37
0006800C
0006900C      REDUCE MATRIX A TO HESSENBERG FORM.
0007000C
0007100      4 NM2 = N-2
0007200        DO 15 R=1,NM2
0007300        RP1 = R+1
0007400        RP2 = R+2
0007500        ABIG = 0.
0007600        JNT(R) = RP1
0007700        DO 5 I=RP1,N
0007800        RELAIR = A(I,R)
0007900        AMGAIR = CONJI*A(I,R)
0008000        ABSSQ = RELAIR**2 + AMGAIR**2
0008100        IF (ABSSQ.LE.ABIG) GOTO 5
0008200        JNT(R) = I
0008300        ABIG = ABSSQ
0008400      5 CONTINUE
0008500        INTER = JNT(R)
0008600        IF (ABIG.EQ.0.) GOTO 15
0008700        IF (INTER.EQ.RP1) GOTO 8
0008800        DO 6 I=R,N
0008900        TEMP = A(RP1,I)
0009000        A(RP1,I) = A(INTER,I)
0009100      6 A(INTER,I) = TEMP
0009200        DO 7 I=1,N
0009300        TEMP = A(I,RP1)
0009400        A(I,RP1) = A(I,INTER)
0009500      7 A(I,INTER) = TEMP
0009600      8 DO 9 I=RP2,N
0009700        MULT(I) = A(I,R)/A(RP1,R)
0009800      9 A(I,R) = MULT(I)
0009900        DO 11 I=1,RP1
```

```
0010000        TEMP = 0.
0010100        DO 10 J=RP2,N
0010200    10 TEMP = TEMP + A(I,J)*MULT(J)
0010300    11 A(I,RP1) = A(I,RP1) + TEMP
0010400        DO 13 I=RP2,N
0010500        TEMP = 0.
0010600        DO 12 J=RP2,N
0010700    12 TEMP = TEMP + A(I,J)*MULT(J)
0010800    13 A(I,RP1) = A(I,RP1)+TEMP-MULT(I)*A(RP1,RP1)
0010900        DO 14 I=RP2,N
0011000        DO 14 J=RP2,N
0011100    14 A(I,J) = A(I,J) - MULT(I)*A(RP1,J)
0011200    15 CONTINUE
0011300C
0011400C        CALCULATE EPSILON.
0011500C
0011600        EPS = 0.
0011700        DO 16 I=1,N
0011800    16 EPS = EPS +CDABS(A(1,I))
0011900        DO 18 I=2,N
0012000        SUM = 0.
0012100        IM1 = I - 1
0012200        DO 17 J=IM1,N
0012300    17 SUM = SUM + CDABS(A(I,J))
0012400    18 IF(SUM.GT.EPS) EPS=SUM
0012500        EPS = DSQRT(DFLOAT(N))*EPS*1.D-20
0012600        IF (EPS.EQ.0.) EPS=1.D-20
0012700        EPSIL = DMAX1(EPS,EPSIL)
0012800C
0012900C        SAVE THE HESSENBERG FORM IN THE ARRAY H.
0013000C
0013100    20 DO 19 I=1,N
0013200        DO 19 J=1,N
0013300    19 H(I,J) = A(I,J)
0013400        NSM1 = NSTOP - 1
0013500        IF (NSM1.NE.0) GOTO 100
0013600        R = 1
0013700C
0013800C        START SCANNING FOR ZEROES IN THE SUB-DIAGONAL. THIS
0013900C        DEFINES THE SUB-BLOCKS OF THE DECOMPOSED HESSENBERG
0014000C        FORM.
0014100C
0014200        GO TO 102
0014300   100 DO 101 I=1,NSM1
0014400        R = NSTOP - I + 1
0014500        RELAM1 = A(R,R-1)
0014600        AMGAM1 = CONJI*A(R,R-1)
0014700        IF ((DABS(RELAM1)+DABS(AMGAM1)).LE.EPSIL) GOTO 102
0014800   101 CONTINUE
0014900        R = 1
0015000   102 NSTART = R
0015100C
0015200C        NSTART AND NSTOP ARE THE INDICES OF THE BEGINNING AND
0015300C        END OF A DECOMPOSED HESSENBERG BLOCK.
0015400C
0015500        NS = NSTOP - NSTART + 1
```

```
 0015600      NC = NS
 0015700      MN1 = NSTOP + NSTART - N
 0015800  103 IF (NS.NE.1) GOTO 21
 0015900      LAMBDA(MN1) = A(NSTART,NSTART) + SHIFT(1)
 0016000      NCOUNT(MN1) = ICOUNT
 0016100      GO TO 37
 0016200   21 IF(NS.EQ.2) GOTO 2
 0016300   22 RELANN = A(N,N)
 0016400      AMGANN = CONJI*A(N,N)
 0016500     -RLNNM1 = A(N,N-1)
 0016600     -AMNNM1 = CONJI*A(N,N-1)
 0016700      RLNDN1 = A(N,N-1)/A(N,N)
 0016800      AMNDN1 = CONJI*(A(N,N-1)/A(N,N))
 0016900      IF (RELANN.NE.0..OR.AMGANN.NE.0.) -
 0017000    - 1IF (DABS(RLNDN1) + DABS(AMNDN1)-1.D-18) 24,24,23
 0017100   23 IF (DABS(RLNNM1)+DABS(AMNNM1).GE.EPS) GOTO 25
 0017200   24 LAMBDA(MN1) = A(N,N) + SHIFT(1)
 0017300      NCOUNT(MN1) = ICOUNT
 0017400      ICOUNT = 1
 0017500      N = N - 1
 0017600      NS = NS - 1
 0017700      MN1 = MN1 + 1
 0017800      GO TO 21
 0017900C
 0018000C      DETERMINE SHIFT
 0018100C
 0018200   25 SHIFT(2)=(A(N-1,N-1)+A(N,N)+CDSQRT((A(N-1, -
 0018300    1N-1)+A(N,N))**2-4.*(A(N,N)*A(N-1,N-1)-A(N,N-1) -
 0018400    2*A(N-1,N))))/2.
 0018500      RELSHF = SHIFT(2)
 0018600      AMGSHF = CONJI*SHIFT(2)
 0018700      IF (RELSHF.NE.0..OR.AMGSHF.NE.0.) GOTO 26
 0018800      SHIFT(3) = A(N-1,N-1)+A(N,N)
 0018900      GO TO 27
 0019000   26 SHIFT(3)=(A(N,N)*A(N-1,N-1)-A(N,N-1)*A(N-1,N))/SHIFT(2)
 0019100   27 IF(CDABS(SHIFT(2)-A(N,N)).LT.CDABS(SHIFT(3) -
 0019200    1-A(N,N))) GO TO 28
 0019300      INDEX = 3
 0019400      GO TO 29
 0019500   28 INDEX = 2
 0019600   29 IF(CDABS(A(N-1,N-2)).GE.EPS) GOTO 30
 0019700      LAMBDA(MN1) = SHIFT(2) + SHIFT(1)
 0019800      LAMBDA(MN1+1) = SHIFT(3) + SHIFT(1)
 0019900      NCOUNT(MN1) = ICOUNT
 0020000      NCOUNT(MN1+1) = 0
 0020100      ICOUNT = 1
 0020200      N = N - 2
 0020300      NS = NS - 2
 0020400      MN1 = MN1 + 2
 0020500      GO TO 103
 0020600   30 SHIFT(1) = SHIFT(1) + SHIFT(INDEX)
 0020700      DO 31 I=NSTART,N
 0020800   31 A(I,I) = A(I,I) - SHIFT(INDEX)
 0020900C
 0021000C      PERFORM GIVENS ROTATIONS, QR ITERATES.
 0021100C
```

```
0021200        IF (ICOUNT.LE.10) GOTO 32
0021300        NCOUNT(MN1) = -ICOUNT
0021400        NC = NC - NS
0021500        GO TO 37
0021600     32 NM1 = N - 1
0021700        TEMP1 = A(NSTART,NSTART)
0021800        TEMP2 = A(NSTART+1,NSTART)
0021900        DO 36 R=NSTART,NM1
0022000        NN = R
0022100        RP1 = R + 1
0022200        RELTM1 = TEMP1
0022300        AMGTM1 = CONJI*TEMP1
0022400        RELTM2 = TEMP2
0022500        AMGTM2 = CONJI*TEMP2
0022600        RHO = DSQRT(RELTM1**2+AMGTM1**2+RELTM2**2+AMGTM2**2)
0022700        IF (RHO.EQ.0.) GOTO 36
0022800        COST = TEMP1/RHO
0022900        SINT = TEMP2/RHO
0023000        INDEX = MAXO(NN-1,NSTART)
0023100        DO 33 I=INDEX,N
0023200        TEMP = DCONJG(COST)*A(NN,I)+DCONJG(SINT)*A(RP1,I)
0023300        A(RP1,I) = -SINT*A(NN,I)+COST*A(RP1,I)
0023400     33 A(NN,I) = TEMP
0023500        TEMP1 = A(RP1,RP1)
0023600        TEMP2 = A(NN+2,R+1)
0023700        DO 34 I=NSTART,R
0023800        TEMP=COST*A(I,NN)+SINT*A(I,RP1)
0023900        A(I,RP1) = -DCONJG(SINT)*A(I,NN)+DCONJG(COST)*A(I,RP1)
0024000     34 A(I,NN) = TEMP
0024100        INDEX = MINO(NN+2,N)
0024200        DO 35 I=RP1,INDEX
0024300        A(I,NN) = SINT*A(I,RP1)
0024400     35 A(I,RP1)=DCONJG(COST)*A(I,RP1)
0024500     36 CONTINUE
0024600        ICOUNT = ICOUNT + 1
0024700        GO TO 22
0024800C
0024900C       CALCULATE VECTORS.
0025000C
0025100     37 IF (.NOT.EVECT) GOTO 64
0025200        CALL CPUTIM(JTIM)
0025300        ITSUM = ITSUM + (JTIM - ITIM)
0025400        IF (NC.EQ.0) GOTO 64
0025500        NPNCAL = NSTART + NC - 1
0025600        N = NSTOP
0025700        NS = NSTOP - NSTART + 1
0025800        NM1 = N - 1
0025900        IF (N.NE.2) GO TO 38
0026000        EPS = QMAX1(CDABS(LAMBDA(1)),CDABS(LAMBDA(2)))*1.D-16
0026100        IF (EPS.EQ.0.) EPS=EPSIL
0026200        H(1,1) = A(1,1)
0026300        H(2,1) = A(2,1)
0026400        H(1,2) = A(1,2)
0026500        H(2,2) = A(2,2)
0026600     38 DO 63 L=NSTART,NPNCAL
0026700        ABIG = 0.
```

```
0026800          EIG = LAMBDA(L)
0026900          IF (L.EQ.NSTART) GOTO 40
0027000          LM1 = L - 1
0027100          RELEIG = EIG
0027200          AMGEIG = CONJI*EIG
0027300          DO 39 I=NSTART,LM1
0027400          RELAMI = LAMBDA(I)
0027500          AMGAMI = CONJI*LAMBDA(I)
0027600          IF (DABS(RELEIG-RELAMI).GT.EPSIL) GOTO 39
0027700          IF (DABS(AMGEIG-AMGAMI).GT.EPSIL) GOTO 39
0027800          EIG = EIG + CONJI*EPSIL
0027900       39 CONTINUE
0028000       40 DO 42 I=1,N
0028100          DO 41 J=1,N
0028200       41 HL(J,I) = H(J,I)
0028300       42 HL(I,I) = HL(I,I) - EIG
0028400          DO 46 I=1,NM1
0028500          MULT(I) = 0.
0028600          INTH(I) = .FALSE.
0028700          IP1 = I + 1
0028800          IF (CDABS(HL(I+1,I)).LE.CDABS(HL(I,I))) GO TO 44
0028900          INTH(I) = .TRUE.
0029000          DO 43 J=I,N
0029100          TEMP = HL(I+1,J)
0029200          HL(I+1,J) = HL(I,J)
0029300       43 HL(I,J) = TEMP
0029400       44 RELHII = HL(I,I)
0029500          AMGHII = CONJI*HL(I,I)
0029600          IF (RELHII.EQ.0..AND.AMGHII.EQ.0.) GOTO 46
0029700          MULT(I) = -HL(I+1,I)/HL(I,I)
0029800          DO 45 J=IP1,N
0029900       45 HL(I+1,J)=HL(I+1,J) + MULT(I)*HL(I,J)
0030000       46 CONTINUE
0030100          DO 48 I=1,N
0030200       48 VECT(I) = 1.
0030300          IF (NSTOP.EQ.M) GOTO 110
0030400          NSTP1 = NSTOP + 1
0030500          DO 47 I=NSTP1,M
0030600       47 VECT(I) = 0.
0030700      110 ICOUNT = 1
0030800       49 RELHNN = HL(N,N)
0030900          AMGHNN = CONJI*HL(N,N)
0031000          IF (RELHNN.EQ.0..AND.AMGHNN.EQ.0.) HL(N,N)=EPS
0031100          VECT(N) = VECT(N)/HL(N,N)
0031200          DO 51 I=1,NM1
0031300          K = N-I
0031400          DO 50 J=K,NM1
0031500       50 VECT(K) = VECT(K) - HL(K,J+1)*VECT(J+1)
0031600          RELHKK = HL(K,K)
0031700          AMGHKK = CONJI*HL(K,K)
0031800          IF (RELHKK.EQ.0..AND.AMGHKK.EQ.0.) HL(K,K)=EPS
0031900       51 VECT(K) = VECT(K)/HL(K,K)
0032000          BIG = 0.
0032100          DO 52 I=1,N
0032200          RELVEC = VECT(I)
0032300          AMGVEC = CONJI*VECT(I)
```

24

```
0032400          SUM = DABS(RELVEC)+DABS(AMGVEC)
0032500          IF (SUM.LE.BIG) GOTO 52
0032600          BIG = SUM
0032700          II = I
0032800          RELV = RELVEC
0032900          AMGV = AMGVEC
0033000      52 CONTINUE
0033100          IF (BIG.EQ.0.) GOTO 155
0033200          IF (AMGV.EQ.0.) GOTO 135
0033300          IF (DABS(AMGV).GT.DABS(RELV)) GOTO 125
0033400          RAT = AMGV/RELV
0033500          DEN = RELV + RAT*AMGV
0033600          DO 120 I=1,N
0033700          IF (I.EQ.II) GOTO 120
0033800          RELVEC = VECT(I)
0033900    *     AMGVEC = CONJI*VECT(I)
0034000          RELVC = (RELVEC + RAT*AMGVEC)/DEN
0034100          AMGVC = (AMGVEC - RAT*RELVEC)/DEN
0034200          VECT(I) = DCMPLX(RELVC,AMGVC)
0034300     120 CONTINUE
0034400          VECT(II) = 1.
0034500          GO TO 150
0034600     125 RAT = RELV/AMGV
0034700          DEN = AMGV + RAT*RELV
0034800          DO 130 I=1,N
0034900          IF (I.EQ.II) GOTO 130
0035000          RELVEC = VECT(I)
0035100          AMGVEC = CONJI*VECT(I)
0035200          RELVC = (AMGVEC + RAT*RELVEC)/DEN
0035300          AMGVC = (RAT*AMGVEC - RELVEC)/DEN
0035400          VECT(I) = DCMPLX(RELVC,AMGVC)
0035500     130 CONTINUE
0035600          VECT(II) = 1.
0035700          GO TO 150
0035800     135 DO   53 I=1,N
0035900      53 VECT(I) = VECT(I)/BIG
0036000     150 ABIG = ABIG + DLOG10(BIG)
0036100          IF (ABIG.GT.EPSMAX) GOTO 55
0036200     155 IF(ICOUNT.GE.10) GOTO 55
0036300          DO 54 I=1,NM1
0036400          IF (.NOT.INTH(I)) GOTO 54
0036500          TEMP = VECT(I)
0036600          VECT(I) = VECT(I+1)
0036700          VECT(I+1) = TEMP
0036800      54 VECT(I+1) = VECT(I+1)+MULT(I)*VECT(I)
0036900          ICOUNT = ICOUNT + 1
0037000          GO TO 49
0037100      55 IF (M.LE.2) GOTO 69
0037200          MCOUNT(L) = ICOUNT
0037300          MM2 = M-2
0037400          DO 57 I=1,MM2
0037500          M1I = M-1-I
0037600          MI1 = M-I+1
0037700          DO 56 J=MI1,M
0037800      56 VECT(J)=H(J,M1I)*VECT(M1I+1)+VECT(J)
0037900          INDEX = JNT(M1I)
```

```
0038000          TEMP = VECT(M1I+1)
0038100          VECT(M1I+1) = VECT(INDEX)
0038200       57 VECT(INDEX) = TEMP
0038300C
0038400C        NORMALIZE EIGENVECTOR.
0038500C
0038600       69 SUM = 0.
0038700          DO 58 I=1,M
0038800          RELVEC = VECT(I)
0038900          AMGVEC = CONJI*VECT(I)
0039000       58 SUM = SUM + RELVEC*RELVEC + AMGVEC*AMGVEC
0039100          SUM = DSQRT(SUM)
0039200          IF (SUM.EQ.0.) GO TO 60
0039300          DO 59 I=1,M
0039400       59 VECT(I) = VECT(I)/SUM
0039500       60 CONTINUE
0039600          DO 61 I=1,M
0039700       61 A(I,L) = VECT(I)
0039800          CALL CPUTIM(KTIM)
0039900          KTSUM = KTSUM + (KTIM - JTIM)
0040000       63 CONTINUE
0040100          NCAL = NCAL + NC
0040200       64 IF(NSTART.EQ.1) GOTO 70
0040300          SHIFT(1) = 0.
0040400          NSTOP = NSTART - 1
0040500          N = NSTOP
0040600          GO TO 20
0040700       70 DO 80 L=2,M
0040800          DO 79 I=1,M
0040900       79 JNT(I) = 0
0041000          RELAML = LAMBDA(L)
0041100          AMGAML = CONJI*LAMBDA(L)
0041200          LM1 = L - 1
0041300          R = 0
0041400          DO 71 I=1,LM1
0041500          RELAMI = LAMBDA(I)
0041600          AMGAMI = CONJI*LAMBDA(I)
0041700          IF (DABS(RELAML-RELAMI).GT.EPS) GOTO 71
0041800          IF (DABS(AMGAML-AMGAMI).GT.EPS) GOTO 71
0041900          JNT(I) = L
0042000          R = R + 1
0042100       71 CONTINUE
0042200          IF (R.EQ.0) GOTO 80
0042300          DO 72 I=1,M
0042400       72 VECT(I) = 0.
0042500          DO 75 I=1,LM1
0042600          IF (JNT(I).NE.L) GOTO 75
0042700          TEMP = 0.
0042800          DO 73 J=1,M
0042900       73 TEMP = TEMP + DCONJG(A(J,L))*A(J,I)
0043000          DO 74 J=1,M
0043100       74 VECT(J) = VECT(J) + TEMP*A(J,I)
0043200          IF (R.EQ.1) GOTO 76
0043300          R = R - 1
0043400       75 CONTINUE
0043500       76 SUM = 0.
```

```
0043600          DO 77 I=1,M
0043700          A(I,L) = A(I,L) - VECT(I)
0043800       77 SUM = SUM + A(I,L)*DCONJG(A(I,L))
0043900          IF (SUM.EQ.0.) GOTO 80
0044000          SUM = DSQRT(SUM)
0044100          DO 78 I=1,M
0044200       78 A(I,L) = A(I,L)/SUM
0044300       80 CONTINUE
0044400       92 DO 95 J=1,M
0044500          DO 95 I=1,M
0044600          TEMP = A(I,J)
0044700          A(I,J) = AA(I,J)
0044800       95 AA(I,J) = TEMP
0044900          RETURN
0045000          ENTRY EVDATA (ITS,KTS,NCO,MCO,RNORM)
0045100          DIMENSION MCO(1),NCO(1),RNORM(1)
0045200          DO 83 I=1,M
0045300       83 NCO(I) = NCOUNT(I)
0045400          ITS = ITSUM
0045500          IF (.NOT.EVECT) RETURN
0045600          DO 84 I=1,M
0045700       84 MCO(I) = MCOUNT(I)
0045800          ANORM = 0.
0045900          DO 85 I=1,M
0046000          DO 85 J=1,M
0046100       85 ANORM = ANORM + A(J,I)*DCONJG(A(J,I))
0046200          ANORM = DSQRT(ANORM)
0046300          IF (ANORM.EQ.0.) ANORM=1.
0046400          KTS = KTSUM
0046500          DO 90 L=1,M
0046600          VNORM = 0.
0046700          DO 89 I=1,M
0046800          TEMP = 0.
0046900          DO 82 J=1,M
0047000       82 TEMP = TEMP + A(I,J)*AA(J,L)
0047100          TEMP = TEMP - LAMBDA(L)*AA(I,L)
0047200       89 VNORM = VNORM + (CDABS(TEMP))**2
0047300       90 RNORM(L) = DSQRT(VNORM)/ANORM
0047400          RETURN
0047500          END
```

REFERENCES

1. Wilkinson, J. H.: The Algebraic Eigenvalue Problem. Clarendon Press, Oxford, 1965.

2. Greenstadt, John: The Determination of the Characteristic Roots of a Matrix by the Jacobi Method. Mathematical Methods for Digital Computers. Anthony Ralston and Herbert S. Wilf, eds., John Wiley & Sons, Inc., 1960, pp. 84-91.

3. Francis, J. G. F.: The QR Transformation: A Unitary Analogue to the LR Transformation. I. Computer J., vol. 4, Oct. 1961, pp. 265-271.

4. Francis, J. G. F.: The QR Transformation. II. Computer J., vol. 4, 1962, pp. 332- 345.

5. Funderlic, R. E., ed.: The Programmer's Handbook: A Compendium of Numerical Analysis Utility Programs. Rep. K-1729, Union Carbide Nuclear Co., Feb. 9, 1968.

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

— NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include conference proceedings, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Washington, D.C. 20546